Partially Observable Monte Carlo Planning with State Variable Constraints for Mobile Robot Navigation

Alberto Castellini*, Enrico Marchesini, Alessandro Farinelli

Department of Computer Science, University of Verona, Verona, Italy, alberto.castellini@univr.it, enrico.marchesini@univr.it, alessandro.farinellin@univr.it

**corresponding author*

Final version available at
https://www.sciencedirect.com/science/article/pii/S095219762100230X

Abstract

Autonomous mobile robots employed in industrial applications often operate in complex and uncertain environments. In this paper we propose an approach based on an extension of Partially Observable Monte Carlo Planning (POMCP) for robot velocity regulation in industrial-like environments characterized by uncertain motion difficulties. The velocity selected by POMCP is used by a standard engine controller which deals with path planning. This two-layer approach allows POMCP to exploit prior knowledge on the relationships between task similarities to improve performance in terms of time spent to traverse a path with obstacles. We also propose three measures to support human-understanding of the strategy used by POMCP to improve the performance. The overall architecture is tested on a Turtlebot3 in two environments, a rectangular path and a realistic production line in a research lab. Tests performed on a C++ simulator confirm the capability of the proposed approach to profitably use prior knowledge, achieving a performance improvement from 0.7% to 3.1% depending on the complexity of the path. Experiments on a Unity simulator show that the proposed two-layer approach outperforms also single-layer approaches based only on the engine controller (i.e., without the POMCP layer). In this case the performance improvement is up to 37% comparing to a state-of-the-art deep reinforcement learning engine controller, and up to 51% comparing to the standard ROS engine controller. Finally, experiments in a real-world testing arena confirm the possibility to run the approach on real robots.

Keywords: Planning under uncertainty, POMDP, POMCP, Mobile robot planning, Industry 4.0, Explainable planning

1. Introduction

Planning under uncertainty is a key task for long-term robot autonomy. Despite recent advances of intelligent robotic systems in several contexts, such as, industrial robots, service robots, patrolling, search and rescue (Farinelli et al., 2017; Parker et al., 2016; Bevacqua et al., 2015; Orlandini et al., 2013; Basilico et al., 2012), the results of recent robot challenges (Krotkov et al., 2017; Correll et al., 2018) show that improvements are still needed to achieve a reliable management of uncertainty in unstructured environments. The risk introduced by uncertainty in these contexts can, in fact, make traditional planning methods impractical (Lanighan and Grupen, 2019). Novel approaches are needed to overcome these issues enabling robots to make effective decisions while managing risk (Wang et al., 2018; Laroche et al., 2019; Simao and Spaan, 2019).

In several application domains robots are required to execute series of tasks having similar properties. A typical example in Industry 4.0 concerns mobile robots involved in warehouse pick-and-place operations (Caccavale and Finzi, 2019). They traverse aisles possibly populated by people, other robots and obstacles with the goal of moving objects from one place of the warehouse to another. Tasks are represented by movements across aisles and a task property of interest is the traffic level in each aisle, that characterizes the difficulty of the aisle and of the entire path traversed by the robot, which affects the movement capability of the robot. Interestingly, some aisles have similar physical properties that make their difficulties also similar to each other. Another example is represented by flying drones involved in autonomous package delivery (Grippa et al., 2019). Their goal is to deliver as many packages as possible in the shortest possible time and using a fixed amount of energy due to battery limitations (Chen et al., 2019). Tasks in this context are movements through flight sections and a task property is the energy requirement of each flight section, which could depend on wind direction, presence of buildings and other features. Also in this case, similarities between pairs of flight sections can be identified, which characterize the similarities between the energy demand of these flight sections. Similar applications concern exploration and surveillance using unmanned aerial vehicles (UAV), where control strategies based on model predictive control (Altan and Hacolu, 2020), metaheuristic optimization (Altan and Hacolu,

2020) and other approaches (e.g., neural networks) have been integrated in the flight control for path planning. We instead focus on the integration of a probabilistic planner based on Markov decision processes. Another domain concerns aquatic drones involved in water monitoring, that traverse path segments with properties depending on water flow and waves (Castellini et al., 2020a).

The similarity structure of tasks involved in robot planning can provide useful information for improving planning performance. However, in the majority of cases this structure is only partially known in advance. In the warehouse pick-and-place application domain, for instance, it is typical to know in advance that two aisles have similar degree of difficulty (due to their position in the warehouse and physical properties) but this information could be available only for a subset of aisles or it could be uncertain, namely, only a probability that two aisles have the same degree of difficulty could be known.

In this paper we investigate the impact of prior knowledge about task similarity structure on planning performance of real mobile robotic platforms. We focus, in particular, on a problem concerning velocity regulation of a mobile robot following a pre-specified path in an environment with uncertain obstacle densities. The robot has to reach the end of the path in the shortest possible time and to avoid collisions with obstacles in the path to guarantee safety. This problem is used to show that the proposed approach can be applied to real robotic platforms. Similar problems were proposed in the literature to test planning methods in mobile robots (Yang et al., 2014). Real-world applications of this case study concern, for instance, safety management in Industry 4.0.

In our case study the path that has to be traveled is divided into segments and subsegments, and every segment is characterized by a *difficulty* that depends on the density of obstacles in the segment. A time penalty is imposed to the robot each time it collides. The real difficulty of segments is unknown to the robot in advance, and the robot has to reach the end of the path as quickly as possible, hence it should move slowly in difficult segments to avoid collisions, and fast in simple segments to minimize the traveling time. Since it is known in advance that some pairs of segments can (probabilistically) have the same difficulty (e.g., because they have similar properties), the information about (unknown) segment difficulties can be collected as the robot moves and then propagated to subsequent segments to improve planning performance. The problem has therefore a sequential structure, in which difficulties of previously traveled segments are used to infer the difficulty of subsequent segments known to have similar difficulty.

We use *Partially Observable Markov Decision Processes (POMDP)* (Russell and Norvig, 2003; Kaelbling et al., 1998) to formalize the problem. This framework allows to model dynamical processes in uncertain environments and to synthesize optimal policies in this context. To overcome scalability issues (Papadimitriou and Tsitsiklis, 1987), we use *Partially Observable Monte Carlo Planning (POMCP)* (Silver and Veness, 2010) as a solver. It is an approximate online algorithm (Ross et al., 2008) able to synthesize the planning policy step-by-step and without representing the overall state space. Then, we consider the POMCP extension proposed in (Castellini et al., 2019) to introduce and exploit prior knowledge about relationships between segment difficulties. In particular, we represent these relationships by probabilistic state-variable constraints using Markov Random Fields (MRFs). This prior knowledge yields a performance improvement in terms of expected return that we show to be related to the improvement of two external measures, namely, the distance between the real state and the belief, and the mutual information between segment difficulty and action taken in the segment. These two measures are defined and analyzed across the paper together with other informative measures that support the interpretability of the approach and the related results (Anjomshoae et al., 2019; Langley et al., 2017; Zhang et al., 2017). The novel contribution we propose in this work is the integration of the POMCP-based planner presented in (Castellini et al., 2019) into a real robotic task. This is not trivial and requires a specific formalization of the problem. We provide a two-layer control architecture in which the upper layer uses an extension of Partially Observable Monte Carlo Planning (POMCP) for regulating the velocity of a mobile robot, and the lower layer uses a standard engine controller for dealing with path planning. The integration of the two layers, with POMCP set on top of the engine controller and used to control high-level aspects of robot motion, is also not trivial. The proposed architecture allows to *i)* devise a probabilistically optimal strategy based on POMDPs for regulating the robot velocity, *ii)* integrate prior knowledge about the environment, *iii)* improve the efficiency of standard engine controllers for path planning, e.g., we present results using a Deep RL controller in the lower level of the architecture but a standard ROS controller can also be used. The interaction between the two controllers is explained in the following and the overall software/hardware architecture described. The results on simulated and a real Turtlebot are fully documented and comparisons with state-of-the-art control approaches are performed.

The proposed methodology is evaluated in four ways. First, we compute a statistical analysis of performance considering a large number of instances of our problem having different configurations of segment difficulties. This analysis is performed on two environments, namely, a rectangular path and a path that reproduces a real industrial environment. Experiments are performed using the standard C++ simulator provided by the POMCP software[1]. Such simulator does not consider the physical properties of the environment (e.g., frictions, etc.). Transition and observation models of the POMDP are learned from tests performed on Unity (Juliani et al., 2018), a simulation tool increasingly used in recent robotics works (Marchesini and Farinelli, 2020a; Yoon et al., 2018), since it represents a viable and faster alternative to other simulation tools such as Gazebo[2]. The native simulation time speed-up introduced by Unity (up to $\times 100$ times) enables quick data collection. Our experiments show that the planning approach based on prior knowledge (Castellini et al., 2019) outperforms the standard POMCP approach (Silver and Veness, 2010). The improvement is small (i.e., 0.7%) in the simple rectangular path and larger (i.e., 3.1%) in the more complex ICE path. Second, we perform tests on Unity simulators of the rectangular and industrial environments using a *TurtleBot3*[3] as an agent. In these experiments the simulator considers the physical properties of the environment. We use a state-of-the-art engine controller (Marchesini and Farinelli, 2020b) for path planning and collision avoidance in single subsegments, a localization algorithm for improving robot localization, inter-process communication between planner and robot, and other tools explained in the next sections. These tests show that our approach has good performance in physics-grounded simulations (OpenAI et al., 2018) of real-world environments. Third, we actually deploy our method in a real-world testing arena reproducing the rectangular path environment and using a real *TurtleBot3* as an agent. As expected, tests (displayed in an attached video) are faithful reproductions of the evaluations performed on Unity simulators. They therefore confirm the results obtained in simulations and the possibility to apply the two-layer planning approach to real-world robotic platforms. As a fourth experimental test, we provide a comparative analysis of performance between the proposed methodology (us-

[1]http://www0.cs.ucl.ac.uk/staff/D.Silver/web/Applications.html

[2]http://gazebosim.org/

[3]https://www.turtlebot.com/

ing POMCP for velocity regulation and a deep reinforcement learning - DRL - engine controller for path planning) and state-of-the-art engine controllers (without the POMCP layer). This analysis confirms that the proposed approach achieves a performance improvement up to 37% comparing to the DRL engine controller. The performance improvement reaches even 51% if the standard ROS[4] engine controller is used. In summary, analyzing advantages and disadvantages of the proposed approach we observe that it has an increased complexity than standard controllers, because it introduces a new POMCP-based layer for velocity regulation, but it also achieves higher planning performance exploiting prior knowledge about task relationships.

The main contributions of this paper to the state-of-the-art can be then summarized in the following four points:

- we formalize a problem of mobile robot velocity regulation in industrial-like environments, considering uncertain motion difficulties (e.g., due to clutterness or presence of moving obstacles) and prior knowledge about similarities in segment difficulties;
- we propose a two-layer approach in which an extended version of POMCP regulates the robot velocity considering prior knowledge about segment similarities, and a standard engine controller performs path planning and operates the robot considering the velocity selected by POMCP;
- we deploy the methodology on a real Turtlebot and on a Unity simulator of the Turtlebot, showing that it outperforms state-of-the-art controllers when prior knowledge about segment similarity is considered by POMCP;
- we introduce three measures that support the explainability of results achieved by the extended POMCP, providing insight about the relationship between prior knowledge and performance improvement.

The rest of the paper is organized as follows. Section 2 presents related work. In Section 3 we define our problem and in Section 4 formalize it as a POMDP. Section 5 describes the three versions of POMCP used to synthesize the policy and Section 6 defines three useful measures for explaining performance differences among policies. In Section 7 we provide full details on the

[4]https://www.ros.org/

experimental setup of simulated and real-world tests, and describe the main elements of the proposed architecture. Then, in Section 8 the three evaluation methods are described and results are analyzed providing in-depth interpretation of the internal mechanisms that allow the performance improvement. Finally, Section 9 draws conclusions and indicates future directions.

2. Related work

Planning under uncertainty dates back to the seventies (Feldman and Sproull, 1977; Russell and Norvig, 2003) when aspects of mathematical decision theory started to be incorporated into the predominant symbolic problem-solving techniques. The interest in this topic subsequently grew (Ratering and Gini, 1995; Kaelbling et al., 1998; Boutilier et al., 1999; Zhang et al., 2015; Godoy et al., 2016), since planning under uncertainty is a critical task for autonomous and intelligent agents based on currently available data-driven technologies. The most recent developments mainly concern the use of point-based value iteration (Spaan and Spaan, 2004; Spaan and Vlassis, 2005; Wang et al., 2019), Monte Carlo Tree Search (MCTS) based solvers (Coulom, 2006; Kocsis and Szepesvári, 2006; Browne et al., 2012; Beretta et al., 2019) and Deep Reinforcement Learning (DRL) methods (Silver et al., 2016, 2017; Sutton and Barto, 2018; Leonetti et al., 2016). The first two approaches aim to deal with very large state spaces and the third to learn the policy only from observations and without using a model of the environment dynamics.

Prior knowledge, in different forms, is used in the literature to improve robot navigation (Luperto et al., 2019). However, we have found only a few approximate (Hauskrecht, 2000) and online (Ross et al., 2008) planning approaches based on POMCP, such as (Amato and Oliehoek, 2015; Lee et al., 2018), in which prior knowledge about the domain is used to improve planning performance and scale to large problem instances. The main differences between those approaches and our work are threefold. First, we use a different method to introduce prior knowledge (Castellini et al., 2019). Namely, in our method constraints on the state space are used to refine the belief space and increase performance (in terms of shorter execution time), while in (Amato and Oliehoek, 2015) the multiagent structure of a problem is used to decompose the value function into a set of overlapping factors that enable scalability and performance improvements in POMCP. In (Lee et al., 2018) cost-constraints are used to solve problems with multiple objectives. Our method to define prior knowledge is also different to that of factored

POMDPs (Boutilier and Poole, 1996), since we constrain states instead of the transition model, hence the prior knowledge is expressed in a different way. Second, we focus on an original problem related to obstacle avoidance in industrial-like environments. Our problem has a strong sequential nature in the way in which the robot explores the environment and transfers the acquired knowledge to future exploration (see problem formalization in Sections 3 and 4). In particular, our goal is only velocity regulation, a high-level planning problem, and we delegate the low-level navigation (i.e., path planning) to a state-of-the-art controller (Marchesini and Farinelli, 2020b) realized with a DRL approach (i.e., Rainbow (Hessel et al., 2018)) using standard setup for navigation problems (Marchesini et al., 2021; Tai et al., 2017). This controller is independent to the POMCP planner used for velocity regulation and it was selected for its simplicity to directly control and modify the linear speed of the Turtlebot. The literature on obstacle avoidance for general robotic applications is very wide (Steccanella et al., 2020; Kumar and Kumar, 2018; Correll et al., 2018) but, to the best of our knowledge, it does not contain techniques as that proposed in this work. Third, one of our main goals is to actually deploy the proposed approach on a real robotic platform, therefore we provide in-depth technical details about how the planner is integrated with the engine controller, the localization module and all other modules to make the approach work in practice. A preliminary work towards the integration of the methodology introduced in (Castellini et al., 2019) into robotic platforms is (Castellini et al., 2020b) where the POMCP extension based on MRF prior knowledge is applied to a real-world implementation of the rocksample problem. Here we significantly extend that preliminary work providing an improved methodology and a thorough experimental setting with several tests performed also on an industrial-like environment.

Robotics literature provides extensions of motion planning techniques for velocity regulation (Huang, 2009; Zhong et al., 2014; Gopalakrishnan et al., 2017), but these methods have completely different assumptions than our method. In particular, these approaches use the position and velocity of the robot in the 2D environment as an input, and aim to plan the overall movement of the robot (e.g., direction, velocity and acceleration) considering also positions and velocities of moving obstacles in the 2D environment. To solve this problem they use potential field methods (Koren and Borenstein, 1991), velocity obstacles (Fiorini and Shiller, 1998) and other standard motion planning methods. We instead assume a hierarchical organization of the problem where the (low-level) motion planning is solved by a standard controller and the (high-level) velocity regulation is solved by POMCP. The baseline controllers for motion planning to which we compare are the DRL controller presented in (Marchesini and Farinelli, 2020b) and the standard ROS navigation stack[5]. Velocity regulation is, however, only an example of possible high-level decisions that can be taken by our approach. Other examples include, for instance, alerting operators, blocking the robot movement, and other high-level decisions that can affect motion planning. We also notice that the hierarchical organization of the controller can be applied to other problems to which POMCP has been recently applied, such as, active visual search (Wang et al., 2020; Wandzel et al., 2019; Lauri and Ritala, 2016).

Explainable planning (XAIP) (Miller, 2019; Fox et al., 2017; Langley et al., 2017) is a branch of eXplainable Artificial Intelligence (XAI) (Gunning and Aha, 2019) which is related to our work. Three main challenges of XAI are the development of methods for learning more explainable models, the designation of effective explanation interfaces, and the understanding of psychologic requirements for effective explanations (Gunning and Aha, 2019). The aim of XAIP is to create artificial intelligence systems whose models and decisions can be understood and trusted by end users. XAIP has a strong impact on safety-critical applications, such as industrial robotic ones (Anjomshoae et al., 2019; Sridharan and Meadows, 2019; Zhang et al., 2017), wherein people accountable to authorize the execution of a plan need complete understanding of the plan itself. First approaches of XAIP (Smith, 2012) focus on human-aware planning and model reconciliation and on data visualization (Chakraborti et al., 2018). The two measures defined in Section 6 and their analysis performed in Section 8 are the first steps towards the interpretability of decisions taken by POMCP-based planners.

3. Problem definition

Assume to have a pre-defined path in an industrial environment which must be traversed by a mobile robot. The path (a possible instance is displayed in Figure 1) is made of segments $s_i, i = 1, \ldots, m$, which are then split into subsegments s_{ij}. Each segment (and related subsegments) is characterized by a difficulty $f_i \in F$, related to the average density of obstacles, the type

[5]http://wiki.ros.org/navigation

of terrain (e.g., stairs, slopes, etc.) or luminosity (which makes perception more difficult). The robot has to reach the end of the path in the shortest possible time, regulating the speed in each subsegment to avoid obstacles, since the probability of collision depends on speed and segment difficulty, and each collision generates a time penalty. The robot cannot directly observe segment difficulties, which are hidden state variables, but it can only infer their values in two ways, namely, by observing the occupancy of the segment (detected by lasers on top of the robot), and sensing the angular velocity kept in the segment. The generality of the concept of difficulty lends itself to different contexts in which a mapping exists between some properties of the environment and the difficulty the agent experiences when it executes its tasks. Our method can work also if this mapping is only coarsely defined.

In this work we test our planning approach on two specific paths depicted in Figure 3. The first path has a *rectangular* shape with short sides of three meters and long sides of five meters. An external and an internal wall delimit the area accessible by the robot and some obstacles are arranged along the way. The second path is developed in a real environment for industrial research, namely, the Industrial Computer Engineering (ICE) lab[6] of the Verona University (Italy), which is a laboratory for Industry 4.0 with a modern production line, extended with equipment for augmented reality and digital production. Technical details about the two paths are reported in Section 7. These paths are only two possible instances of several possible environments that can be dealt with by our approach. Also the problem formulation can be extended to more complex scenarios considering larger sets of actions and observations, or more realistic tasks but here we aim to show the applicability of the proposed approach to real robotic platforms.

4. POMDP representation of the problem

A Partially Observable Markov Decision Process (POMDP) (Kaelbling et al., 1998) is defined as a tuple $(S, A, O, T, Z, R, \gamma)$, where S is a finite set of partially observable *states*, A is a finite set of *actions*, Z is a finite set of *observations*, T: $S \times A \to \Pi(S)$ is the *state-transition model* (where $\Pi(S)$ is the power set of S), O: $S \times A \to \Pi(Z)$ is the *observation model*, R: $S \times A \to \mathbb{R}$ is the *reward function* and $\gamma \in [0, 1)$ is a *discount factor*. The goal of an

[6]`http://www.di.univr.it/?ent=progetto&id=4935&lang=en`

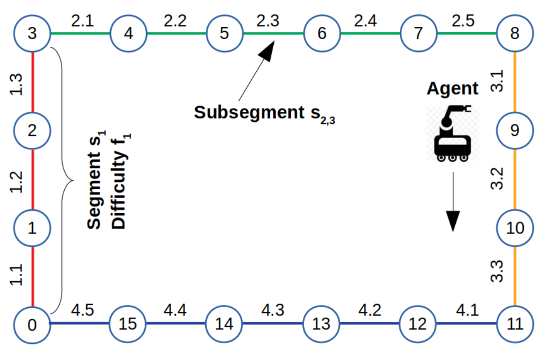


Figure 1: Problem definition: path travelled by the agent. Nodes are subsegment starting points.

agent operating a POMDP, is to maximize its expected total discounted reward (also called *discounted return*) $E[\sum_{t=0}^{\infty} \gamma^t R(q_t, a_t)]$, by choosing the best action a_t in each state q_t at time t; γ is used to reduce the weight of distant rewards and ensure the (infinite) sum's convergence. As mentioned above, the partial observability of the state is dealt with by considering at each time-step a probability distribution over states, called *belief*. The belief space is here represented by symbol B. POMDP *solvers* are algorithms that compute, in an exact or approximate way, a *policy* for POMDPs, namely a function π: $B \rightarrow A$ that provides an optimal action for each belief.

The problem described in the previous section can be formalized as a POMDP. The *state* contains *i)* the true configuration of segment difficulties $(f_1, \ldots, f_m)$, which is hidden, *ii)* the position $p = (i, j)$ of the robot in the path, where i is the index of the segment and j the index of the subsegment (notice that, saying that the agent is in position (i, j) we mean that it is at the beginning of subsegment $s_{i,j}$), *iii)* t is the time elapsed from the beginning of the path. *Actions* correspond to the speed the robot keeps in a subsegment, which may have three possible values, namely low (L), medium (M) or high (H).

Observations are related to subsegment *occupancy* and integral of robot *angular velocity*. The occupancy oc of a subsegment is computed from a laser rotating on top of the robot. Laser values in front of the robot (with a 30° angle) are averaged and thresholded obtaining a binary value where 0 means that the laser detects no obstacles in the next subsegment, and 1 that it detects some obstacle. The *occupancy model* provides the probability of occupancy given segment difficulties, namely $p(oc \mid f)$. The integral of robot *angular velocity* av is computed from the output signals (i.e., direction angle) of the low-level engine controller. In particular, the controller has 5 outputs meaning to go straight or to turn right/left with angular velocity of 45° or 90° deg/sec. In each subsegment the controller provides several low-level actions to the robot. We count the actions corresponding to turn right/left and threshold this count obtaining a binary signal av which is 0 if the robot performs few curves in the subsegment and 1 if it performs several curves. This method to gather information about segment difficulty based on angular velocity works because the low-level engine controller that drives the robot inside subsegments is completely independent from the POMCP-based planner that regulates the speed. The *angular velocity model* provides the probability of angular velocity given segment difficulties, namely $p(av \mid f)$. Notice that the discretization of occupancy and angular velocity used in our tests is coarse-grained because this degree of precision is enough for our tests but more precise discretizations can be used.

The final observation is a coding of both variables oc and av computed as $o = av + 2 \cdot oc$. Namely $o = 0$ if $av = 0$ and $oc = 0$, $o = 1$ if $av = 1$ and $oc = 0$, $o = 2$ if $av = 0$ and $oc = 1$, and $o = 3$ if $av = 1$ and $oc = 1$. The *observation model* provides the probability of observations given segment difficulties, namely $p(o \mid f)$. The parameters of the occupancy and angular velocity models for the rectangular and the ICE paths are displayed, respectively, in Tables 1.a,b (rectangular path) and Tables 2.a,b (ICE path). The methodology used to derive such parameters from real-world environments is described in Section 7.4.

The state transition model deals with the update of robot position and current time at each step. Position update is performed in a deterministic way since at each step the robot is assumed to reach the beginning of the next subsegment in the path. The current time is instead updated depending on both the action performed by the agent and (possibly) the collision penalty. The time spent to traverse a subsegment depends on the action performed by the robot, which is discretized as $t = \ell$ (where ℓ is the subsegment length) if action is H (namely the robot spends ℓ time units if the action is *high speed*), $t = 2\ell$ if action is M, and $t = 3\ell$ if action is L. The time penalty due to collision has been set to 40 time units in the rectangular path and 20 time units in the ICE path. The collision probability is governed by the *collision model* $p(c \mid f, a)$. Collision models of rectangular and ICE paths are displayed, respectively, in Table 1.c and Table 2.c., where $c = 0$ means no collision and $c = 1$ that a collision occurs. Notice, that the probability of not making a collision is one minus the probability of making a collision, since the collision value is binary. The reward function here used is $R = -(t_1 + t_2)$, where t_1 is the time depending on agent's action and t_2 is the penalty due to collisions. Finally the discount factor we used is $\gamma = 1$ because the paths have limited length and this allowed us to consider it the same way the time spent to move in different sections of the path. However, the proposed approach is general to problems with infinite horizon, that can be tackled using $\gamma < 1$.

Table 1: Probabilistic model for the rectangular path. (a) Occupancy model $p(oc \mid f)$. (b) Angular velocity model $p(av \mid f)$. (c) Collision model $p(c \mid f, a)$.

f	$p(oc = 1 \mid f)$
L	0.600
M	0.690
H	0.940

(a)

f	$p(av = 1 \mid f)$
L	0.170
M	0.240
H	0.530

(b)

f	a	$p(c = 1 \mid f, a)$
L	L	0.000
L	M	0.033
L	H	0.033
M	L	0.000
M	M	0.033
M	H	0.067
H	L	0.000
H	M	0.067
H	H	0.100

(c)

5. POMCP-based planners

Three planning strategies are used in our tests. The original implementation of POMCP, named *STD* in the following, is used as a baseline. An extended version of POMCP allowing the definition of state-variable constraints by Markov Random Fields (Castellini et al., 2019), named *MRF* in the following, is used to introduce prior knowledge about segment difficulty relationships. For example, in an instance of our problem we could know that the probability that segment s_0 and segment s_1 have same difficulty is 0.9. Planner *MRF* can use this information to improve the policy it generates and, consequently, the planning performance. Finally, we consider an oracle planner, named *ORC* in the following, in which perfect knowledge of segment difficulties is used. This planner performs the POMCP strategy using

Table 2: Probabilistic model for the ICE path. (a) Occupancy model $p(oc \mid f)$. (b) Angular velocity model $p(av \mid f)$. (c) Collision model $p(c \mid f, a)$.

f	$p(oc = 1 \mid f)$
L	0.65
M	0.83
H	0.93

(a)

f	$p(av = 1 \mid f)$
L	0.083
M	0.3
H	0.3

(b)

f	a	$p(c = 1 \mid f, a)$
L	L	0.0
L	M	0.0
L	H	0.0
M	L	0.0
M	M	0.056
M	H	0.14
H	L	0.028
H	M	0.11
H	H	0.25

(c)

only a unique particle corresponding to the true state (i.e., configuration of segment difficulties) hence it has exact and complete prior knowledge. In the following the standard POMCP algorithm (Silver and Veness, 2010) and its extension based on MRF (Castellini et al., 2019) are briefly described.

5.1. Standard POMCP

Partially Observable Monte Carlo Planning (POMCP) (Silver and Veness, 2010) is an online Monte-Carlo based algorithm for solving POMDPs. It uses *Monte-Carlo Tree Search* (MCTS) for selecting optimal actions at each time-step. The main elements of POMCP are a *particle filter*, which represents the belief state, and the *Upper Confidence Bound for Trees* (UCT) (Kocsis and Szepesvári, 2006) search strategy, that allows to select actions from the Monte Carlo tree. The particle filter contains, at each time-step, a sampling of the agent's belief at that step (the belief evolves over time). In particular, it contains k particles, each representing a specific state. At the beginning the particle filter is usually initialized following a uniform random distribution over states, if no prior knowledge is available about the initial state. Then, at each time-step the Monte Carlo tree is generated performing $nSim$ simulations from the current belief. In other words, for $nSim$ times a particle is randomly chosen from the particle filter and the related state is used as initial state to perform a simulation. Each simulation is a sequence of action-observation pairs that collect, altogether, a final return, where each action and observation brings to a new node in the tree. Rewards are then propagated upwards in the tree obtaining, for each action of the root node, an expected (approximated) value of the cumulative reward that this action can bring. The UCT strategy is then used to select actions considering both their expected cumulative reward and the necessity to explore new actions from time to time. The belief is finally updated, after performing the selected action a and getting a related observation o, by considering only the particles (i.e., states) in the new node. New particles can be generated through a *particle reinvigoration* procedure based on local transformation of available states, if the particle filter gets empty. A big advantage of POMCP is that it does not require a complete matrix-based definition of transition model, observation model and reward, but it only needs a black-box simulator of the environment.

5.2. Extended POMCP

The methodology we use to introduce prior knowledge in POMCP (Castellini et al., 2019) allows to define probabilistic relationships of equality between pairs of state-variables by means of Markov Random Fields (MRF). State variables in our application domain are segment difficulties and a relationship says that two segments have a certain relative *compatibility* to have the same difficulty. The MRF approach then allows to factorize the joint probability function of state-variable configurations and this probability is used to constrain the state space. In our application domain the state space is the space of all possible segment difficulty configurations and the constraints introduced by the MRF allow to (probabilistically) reduce the chance to explore states that have small probability to be the true state. The integration of MRF-based prior knowledge into POMCP is mainly performed in the particle filter initialization and in the reinvigoration phase (Castellini et al., 2019), where the constraints are used to optimize the management of the particle filter representing the agent belief. In this work the MRF is manually generated using expert knowledge about the application domain.

5.3. Complexity analysis

The complexity of the extended POMCP is the same as that of the standard POMCP (described in Section 5.1). Being Monte-Carlo methods, they have a sample complexity determined only by the underlying difficulty of the POMDP, rather than the size of the state space or observation space (Silver and Veness, 2010). This sample complexity, namely the complexity to perform a Monte-Carlo simulation in the known POMDP environment, is multiplied by the number of simulations, which is a constant parameter called $nSim$ in this paper. At the beginning of each subsegment of the path our approach performs $nSim$ simulations (in the POMCP context) to select the velocity, and it runs the standard engine controller to perform path planning in the next subsegment. The increase in complexity introduced by our approach compared to using a standard engine controller is therefore a constant, namely the time to perform $nSim$ simulations, for each subsegment. Since POMCP is an anytime algorithm, the time allotted for its execution can be also defined in advance to satisfy the requirements of specific applications.

6. Measures for policy explanation

To quantify the influence of prior knowledge on policy performance we introduce three measures, namely, the *belief-state distance*, the *mutual information between difficulty and action* and the *expected time to traverse the segment* (Castellini et al., 2020c). They strongly contribute to explain the mechanisms that affect the performance improvement and to improve the interpretability of results, as shown in Section 8. Moreover they represent a fundamental tool for explaining and improving planner performance since they allow to precisely identify undesired behaviours of the planner.

6.1. Belief-state distance

We define the belief-state distance at a certain instant as the weighted averaged Manhattan distance between the configuration of segment difficulties in the true (hidden) state and the configurations of segment difficulties in the belief at that instant. Mathematically, if we define the configuration of segment difficulties in the true state as $\boldsymbol{f} = (f_1, \ldots, f_m)$, where m is the number of segments, and we define the set of k possible configurations of segment difficulties in the belief as $\{(f_1^i, \ldots, f_m^i),\ i \in 1, \ldots, k\}$, where the probability of each difficulty configuration $(f_1^i, \ldots, f_m^i)$ in the belief is p_B^i (computed step-by-step by POMCP), then the belief-state distance is

$$d_{SB} = \sum_{i=1}^{k} \big(p_B^i \cdot \sum_{j=1}^{m} |f_j - f_j^i|\big). \tag{1}$$

since the belief is updated at each time-step, this measure can be computed at each time-step as well. It allows to quantify the discrepancy between what the agent believes about the real state of the environment and the real state itself, hence the addition of prior knowledge about segment difficulty relationships is expected to decrease this distance.

6.2. Mutual information (MI) between segment difficulty and action

In our problem the agent is expected to take actions that minimize both the time to reach the end of the path and the risk of collision. The quality of actions strongly depends on the degree of knowledge the agent has about the true configuration of segment difficulties. In fact, analyzing the collision model in Table 2.c, for instance, we observe that high speed (i.e., $a = H$) should be selected in segments with low difficulty (i.e., $f = L$) because the collision probability is always 0.0 in those segments, hence high speed should be preferred to reach earlier the end of the path. On the other hand, in segments with high difficulty (i.e., $f = H$) the collision probability is low (i.e., 0.028) if low speed (i.e., $a = L$) is kept, while the probability grows to 0.11 and 0.25, respectively, it if medium or high speed (i.e., $a = M$ or $a = L$) is kept. For this reason low speed should be preferred in these cases.

To check if the POMCP policy effectively generates actions related to segment difficulties we compute the mutual information between all actions taken in a run and the corresponding segment difficulties. In other words, given a run we consider the sequence of actions $\mathcal{A} = (a_{i,j})$, where i is the index of a segment and j is the index of a subsegment, and the sequence of related subsegment difficulties $\mathcal{F} = (f_{i,j})$. The mutual information (Bishop, 2006) between the two sequences, treated as random variables, is

$$I(\mathcal{A}, \mathcal{F}) = \sum_{a \in \mathcal{A}} \sum_{f \in \mathcal{F}} p_{(\mathcal{A},\mathcal{F})}(a, f) log\Big(\frac{p_{(\mathcal{A},\mathcal{F})}(a, f)}{p_{\mathcal{A}}(a) p_{\mathcal{F}}(f)}\Big), \tag{2}$$

where $p_{(\mathcal{A},\mathcal{F})}(a, f)$ is the joint probability mass function of $\mathcal{A}$ and $\mathcal{F}$, and $p_{\mathcal{A}}$ and $p_{\mathcal{F}}$ are the marginal probability mass functions of $\mathcal{A}$ and $\mathcal{F}$, respectively. Average MI values are computed on sets of runs. We notice that selecting actions with high difficulty-action MI is not trivial since the true configuration of segment difficulties is hidden. In Section 8 we experimentally analyze the trend of this measure depending on the prior knowledge provided in different planners.

6.3. Expected time to traverse a subsegment

The expected time to traverse a subsegment of unitary length and having difficulty $f \in F$ by performing action $a \in A$ is

$$\mathbb{E}[t]_{f,a} = t_a + p(c \mid f, a) \cdot w(c), \tag{3}$$

where t_a is the time due to the action (i.e., L, M or H) without considering any collision (see time t_1 in Section 4), $p(c \mid f, a)$ is the collision probability given action a and difficulty f, and $w(c)$ is the collision penalty, which depends on the collision value.

This measure is very useful to analyze and compare planner performance on different configurations of segment difficulties (see Section 8.1). In particular, in those experiments we want to compare the average reward (i.e., the opposite of time spent to traverse the path) of two planners where the average of each planner is computed on a set of runs having different difficulty configurations. The problem in this case is that the expected time to traverse the overall path is affected by strong randomness. One source of this randomness is the chance to get a collision penalty in each subsegment. Namely, two runs performed on the same configuration of difficulties, using the same actions, could get very different reward if a different number of collisions occurs, because the penalty of a single collision is much larger than the time needed to traverse a segment without collisions. The only way to get a statistically significant average performance in this case is to run a huge large number of tests, which is computationally infeasible.

The expected time to traverse a subsegment $\mathbb{E}[t]_{f,a}$ removes the randomness due to collisions because, for each subsegment, it considers the average time (over infinite tests) to traverse it, which is the sum of the time due to the selected action (without considering collisions) plus the fraction of penalty related to performing the action a in segment with difficulty f (i.e., $p(c \mid f, a) \cdot w(c)$). Notice that the randomness is removed by using the knowledge about the true segment difficulty, which is instead unknown by the planner.

7. Experimental setup

We describe here the experimental setup for our work. We first provide an overview of the complete architecture and then describe in detail all modules.

7.1. Overall architecture

Our experimental setup mainly involves the elements shown in Figure 2. The POMCP planner, explained in Section 5, is a C++ module that computes step-by-step the speed the robot must keep in each subsegment. This speed, which depends on the estimated difficulty configuration, is communicated to a low level *engine controller*, namely, the low-level engine controller that deals with path planning and obstacle avoidance in each subsegment. In particular, it allows the robot to reach the other side of the subsegment while keeping the linear velocity defined by the POMCP planner. The communication between planner and engine controller is managed by a communication layer described in Section 7.2.

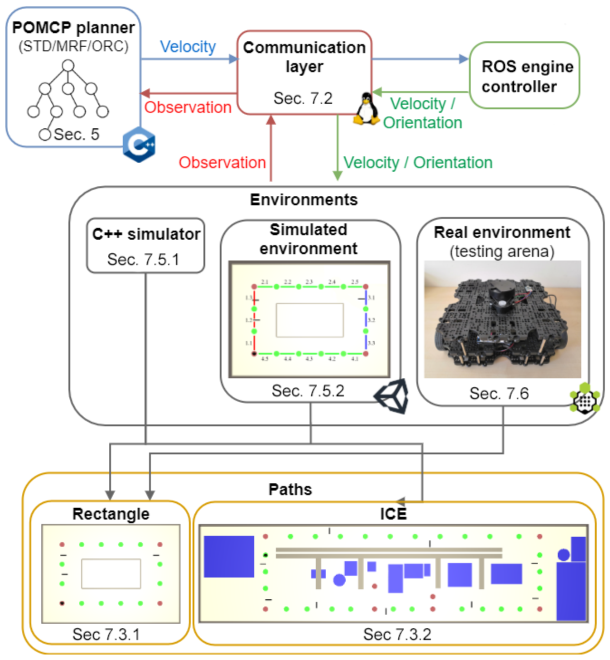


Figure 2: System overview: POMCP planner communicates actions to the Turtlebot in a simulated (Unity) or real (testing arena) environment. The environment returns observations and rewards to the planner which updates (online) its belief and policy.

The approach is then tested on two kinds of simulator and in a real-world testing arena. The first simulator is a C++ module (see Section 7.5.1) which does not consider physical properties of the environment, the second simulator is designed with Unity (see Section 7.5.2) which relies on the Havok physics engine[7] to realize realistic simulations and allows to use low-level engine controllers. The real-world testing arena (see Section 7.6) is a real environment wherein a Turtlebot3 was used as an agent to test the planning approach. As shown in the bottom of Figure 2, experiments are performed on two paths, described respectively in Section 7.3.1 and 7.3.2, namely, a rectangular path and a path defined in the ICE industrial research laboratory. Finally, we highlight that the transition and observation models of the POMDPs are trained on the Unity simulator following an approach described in Section 7.4.

7.2. Communication layer

The communication between the POMCP planner and the engine controller was implemented by inter-process communication via Unix named pipes. This is a common feature for both the Unity simulations and the experiments on the testing arena. Furthermore, Unity natively supports the communication between the low-level controller model (i.e., the engine controller) and the Unity environments. Communication among the controller and the robot is instead managed by a ROS node in the real scenario (i.e., the testing arena).

7.3. Paths

The two paths on which we developed our architecture are described in the following.

7.3.1. Synthetic rectangular path

The first path is a rectangle with short sides of 3 meters and long sides of 5 meters. We assume the robot performs two laps of the path, hence the total number of segments in this path is 8, the total number of subsegments is 32 and the total length is 32 meters. Figure 3.a shows the map of the path with also measures of passage widths and an example of obstacle disposal. Subsegments are all 1 meter long. Obstacles are objects with width of 2 centimeters and a length of 50 centimeters, that are randomly placed along the robot path. Their density depends on segment difficulty. This path was first implemented in Unity and then in a real world environment, considering the obstacle configuration and difficulties presented in Section 8.2.1.

[7]https://www.havok.com/products/havok-physics/

7.3.2. *Industrial computer engineering (ICE) lab*

The second path is located in the industrial computer engineering laboratory of the Verona University. As shown in Figure 3.b, the entire room is 4.55 meters large and 18.45 meters long. On the left hand wall there is a vertical warehouse from which raw materials can be taken and finished products stocked. Then a production line spreads over the whole room, from left to right, with a conveyor belt that transports items across the processing stations. The currently available stations are (from left to right) a milling machine, a 3D printer, an assembly/disassembly station, and a quality control station. The path contains 8 segments, the total number of subsegments is 36 and the total length 36.3 meters. The robot performs one lap starting and ending at the top-left corner. Subsegment lengths range from $0.6m$ to $1.4m$. In Figure 3.b a possible arrangement of obstacles is shown.

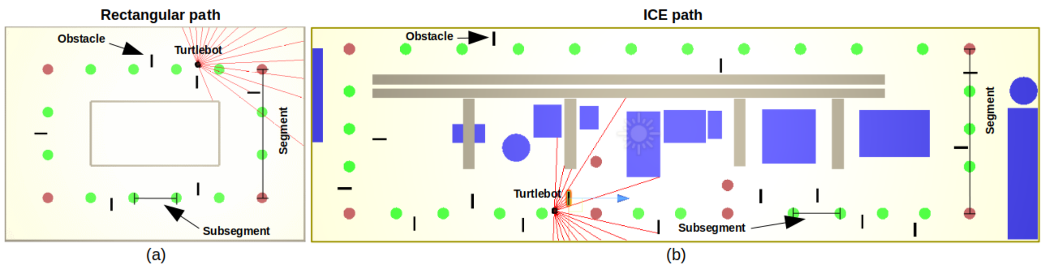


Figure 3: Unity environments. *(a)* Rectangular path. *(b)* ICE path. Red circles represent segment extremes, green circles represent subsegment extremes, small red lines represent obstacles, the black circle represents the Turtlebot, red lines are laser scans.

7.4. *POMDP model generation*

A data collection phase is initially performed on both the Unity environments, to compute the POMDP models described in Section 4. In particular, for each environment and action (i.e., velocity) we perform a complete run of the path keeping the action fixed in all subsegments. The low speed (i.e., $a = L$) corresponds to 0.07 m/s, medium speed (i.e., $a = M$) to 0.14 m/s and high speed (i.e., $a = H$) to 0.21 m/s. For each subsegment we collect: a binary value for marking collisions occurrence, five values with counts of actions (i.e., go straight, turn left/right with angular velocity of 45° or 90° deg/sec) taken by the engine controller along the subsegment, and five values for the frontal laser scans of the robot normalized in range (0, 1), where 1 corresponds to the presence of obstacles at a distance of 3.5 m and 0 corresponds to the presence of obstacles at a distance of 0 m. Values are then grouped by segment difficulty and action and probabilities are computed to make the POMCP collision and observation models. For instance, the collision probability for each difficulty-action pair is computed from the binary value accounting for collisions. Similar procedures are applied for occupancy and angular velocity models, using, respectively, the counts of laser scan and those of the controller actions.

We notice that POMCP assumes a full knowledge of the environment parameters. Since we have estimated these parameters from simulations an error in these parameters could bring a performance decrease as in all model-based planning methods. However, our tests showed that in practice also approximated model parameters can yield performance improvement. Bayesian reinforcement learning approaches, such as Bayesian Adaptive POMCP (Katt et al., 2017) have been recently introduced but their complexity is currently too high to be applied to real robotic planning tasks, hence we will consider them in future work.

7.5. Simulators

The two simulators used to test our approach are described in the following.

7.5.1. C++ simulations

The C++ simulator was developed by following the guidelines provided by the standard POMCP package[8]. Interfaces were implemented together with transition and observation models, and other specific features of the paths, such as number of segments, number of subsegments, subsegment lengths, and so on. The POMCP code was also extended to save to file, step-by-step, details of the simulation progress used to compute statistics and to plot charts shown in the result section.

7.5.2. Unity simulations

Our simulated Unity environment, displayed in Figure 3, is realized using primitive 3D objects of the Unity engine (e.g., cubes and cylinders) for walls and the occupancy of the processing stations of the ICE environment. For our

[8]http://www0.cs.ucl.ac.uk/staff/d.silver/web/Applications.html

robotic agent we use the manufacturer model of the Turtlebot3, simulating the laser sensor using Raycasts and the motors using Hinge joints. In detail, the former is used to create rays that detect collisions and the latter connects the 3D model of the wheels with the main robot components, simulating the rotation of the Turtlebot3 motors (previous work (Marchesini and Farinelli, 2020a) demonstrates that this is a realistic simulation of the behavior of the robot). The position, required by our controller, is returned by Unity as polar coordinates with respect to the modeled environment. Finally, default physics parameters are considered to simulate gravity and frictions.

7.6. Testing arena

We reproduced the rectangular path in Figure 3.a, considering the same setup presented in Section 7.3.1, to provide an explanatory evaluation of our POMCP planners (i.e., ORC, STD, MRF) in a real scenario. Specifically, the robot travels the path (a rectangle 3×5 m) two times, hence the path has 8 segments, 32 subsegments and its length is of 32 m. Figure 7 shows an overview of our scenario, where obstacles are wood panels $50 \times 25 \times 1.5$ cm, subsegments are represented with green marks and segments with red marks. The obstacle configuration and difficulties are the same presented in Section 8.2.1. The attached video provides full details about experiments performed in this real environment.

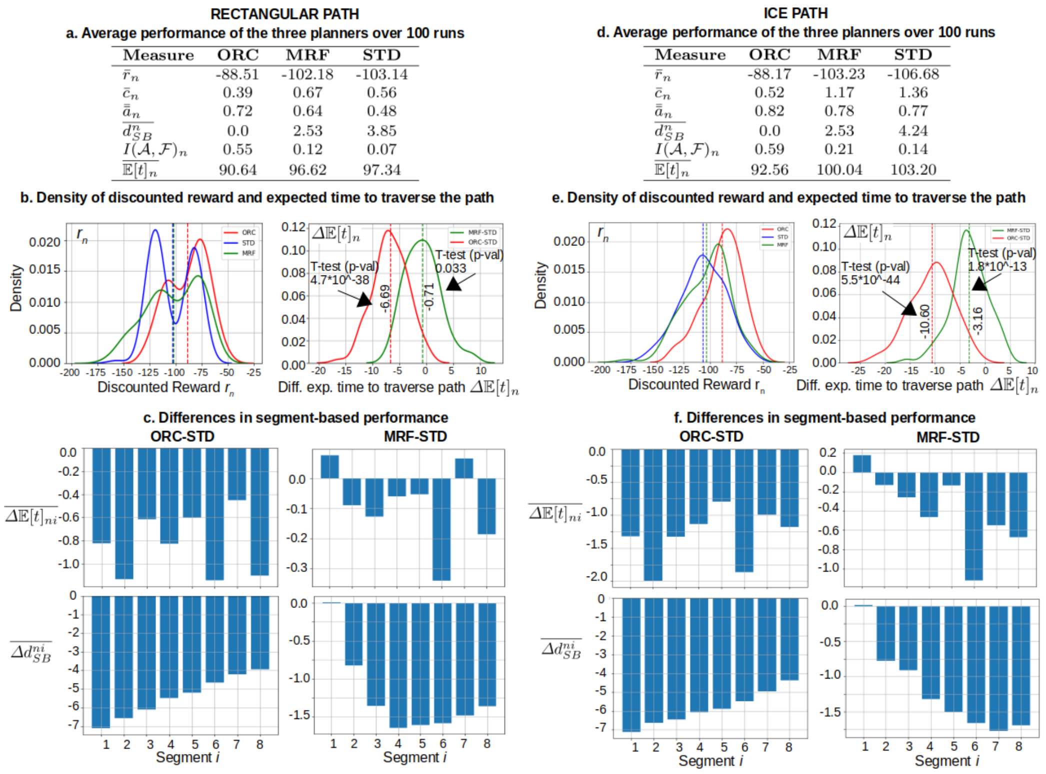
8. Results

We perform four kinds of test to evaluate our approach. First, we compare the average performance of the three planners on several difficulty configurations (Section 8.1). This test is performed on the C++ simulator described in Section 7.5.1 which enables massive testing since it does not consider any interaction with the physical (real or simulated) environment. Then, in Section 8.2 we select a specific configuration of difficulties and show how the three planners perform on the Unity simulator described in Section 7.5.2. The aim of this test is to show that our planner can be used to control the robot in a physical (although simulated) environment. Moreover, this experiment shows how the three planners make decisions, in a specific run, according to their knowledge of the environment, and how this knowledge influences the decisions. In Section 8.3, we provide results achieved in a real testing arena where a Turtlebot is used as an agent. This experiment shows that our architecture (i.e., planner, robot, communication layer, localization module, etc.) is able to work also in real world environments, with results very similar to those achieved in Unity simulations. Finally, in Section 8.4 we perform a comparative analysis of the proposed method against state-of-the-art engine controllers, showing that our method outperforms these controllers in terms of time spent by the robot to traverse the path. The time needed to perform a single run increases from the C++ simulator, to the Unity simulator, and from the Unity simulator to the real environment, therefore we perform statistical analyses of performance (that need several runs) using the C++ simulator, then we introduce the physical environment by Unity and make tests on a subset of runs, and finally display technical details of functioning in the real-world for a small number of (time consuming) tests.

8.1. Statistical analysis of planner performance

In this experiment we run each planner (i.e., ORC, STD and MRF) on 100 different configurations of segment difficulty. For each run $n \in \{1, \dots, 100\}$ we collect five parameters, namely, the discounted return $r_n \in \mathbb{R}$ of the run, the number of collisions $c_n \in \mathbb{N}$ of the run, the average action $\bar{a}_n \in \mathbb{R}^+$ of the run (where the average is computed over the actions taken in the run), the final belief-real state distance $d^n_{SB} \in \mathbb{R}^+$ of the run, and the normalized MI between difficulty and action $I(\mathcal{A}, \mathcal{F})_n \in \mathbb{R}^+$ in the run. Moreover, for each segment s_i of run n, we collect the discounted return $r_{ni} \in \mathbb{R}$ in the segment, the average action $\bar{a}_{ni} \in \mathbb{R}^+$ in the segment, the average belief-real state distance $d^{ni}_{SB} \in \mathbb{R}^+$ in the segment, and the expected time $\mathbb{E}[t]_{ni} \in \mathbb{R}^+$ to traverse the segment (see definition in Section 6.3). These parameters are then averaged over runs and compared between different planners to generalize on their performance.

Figure 4 summarizes the results of the overall analysis performed on both the rectangular path (sub-figures *a*, *b*, and *c*, analyzed in detail in Subsection 8.1.1) and the ICE path (sub-figures *d*, *e*, and *f*, analyzed in detail in Subsection 8.1.2). To have a fair comparison on how the different planners exploit the knowledge on the environment we compare their performance by fixing a difficulty configuration and a series of observations and running the three planners with this input. We repeat this process for 100 times. In this way we remove two sources of randomness, namely that coming from difficulty configuration and that coming from observations, and keep only two sources of randomness that cannot be removed, namely, that coming from policy generation (which is intrinsically related to the POMCP strategy) and that related to collision events (which depends on the actions selected by each planner).



RECTANGULAR PATH

Measure	ORC	MRF	STD
$\bar{r}_n$	-88.51	-102.18	-103.14
$\bar{c}_n$	0.39	0.67	0.56
$\bar{\bar{a}}_n$	0.72	0.64	0.48
$\overline{d^n_{SB}}$	0.0	2.53	3.85
$\overline{I(\mathcal{A},\mathcal{F})_n}$	0.55	0.12	0.07
$\overline{\mathbb{E}[t]_n}$	90.64	96.62	97.34

ICE PATH

Measure	ORC	MRF	STD
$\bar{r}_n$	-88.17	-103.23	-106.68
$\bar{c}_n$	0.52	1.17	1.36
$\bar{\bar{a}}_n$	0.82	0.78	0.77
$\overline{d^n_{SB}}$	0.0	2.53	4.24
$\overline{I(\mathcal{A},\mathcal{F})_n}$	0.59	0.21	0.14
$\overline{\mathbb{E}[t]_n}$	92.56	100.04	103.20

Figure 4: Average performance and properties of runs executed on the rectangular path (left) and the ICE path (right).

Furthermore, we provide run-based performance differences between *i)* ORC and STD, *ii)* MRF and STD. These differences are computed on a run basis and subsequently averaged. For instance, the difference of discounted return r_n in run n is computed as $\Delta r_n = r_n - r_n$. This value is then averaged over runs to obtain the *average difference of discounted return* $\overline{\Delta r_n}$. The same approach is used to compute the average difference of average action $\overline{\Delta \bar{a}_n}$, the average difference of final belief-state distance $\overline{\Delta d^n_{SB}}$, the difference of normalized MI between difficulty and action $\overline{I(\mathcal{A},\mathcal{F})_n}$, and the average difference of expected time to traverse the path $\overline{\Delta\mathbb{E}[t]_n}$. The last measure is particularly stable because it removes also the randomness due to collisions, since it considers, for each subsegment, only the fraction of time penalty due to the collision probability instead of real collision events. All measures described above are computed for both the entire path and single segments. In the second case we add subscript i to identify segment s_i obtaining, respectively, symbols $\overline{\Delta r_{ni}}$, $\overline{\Delta \bar{a}_{ni}}$, $\overline{\Delta d^{ni}_{SB}}$, and $\overline{\Delta \mathbb{E}[t]_{ni}}$. These measures are analyzed in the following to compare planner performance. In both cases we used a number of simulations per step $nSim = 10^{15}$.

8.1.1. *Discussion of results on rectangular path*

Average performance of the three planners over 100 runs in the rectangular path are displayed in Figure 4.a. ORC has the best average return $\bar{r}_n$ (i.e., -88.51), followed by MRF (i.e., -102.18) and then by STD (i.e., -103.14). These differences are generated by different decision strategies that are affected by the level of prior knowledge provided to the planners, which has direct effect on *i)* the average distance between real state and belief $\overline{\Delta d^{n}_{SB}}$, whose values at the end of the path are 0.0 for ORC, 2.53 for MRF and 3.85 for STD; *ii)* the MI between (hidden) segment difficulty and action $I(\mathcal{A}, \mathcal{F})_n$ (i.e., 0.55 for ORC, 0.12 for MRF and 0.07 for STD), which is higher for planners able to understand earlier the difficulty of the segment and to act accordingly. Finally, the average expected time to traverse the path $\overline{\mathbb{E}[t]_n}$ confirms the ranking of planners (respectively ORC, MRF, and STD) with a larger difference between ORC and MRF, and a smaller one between MRF and STD. Figure 4.b shows, on the left, the density of discounted return r_n for the three planners. The multi-modality of these curves are related to runs without and with collisions. As expected, these densities have large variance due to the strong randomness of the process generating the stochastic variable r_n.

The methodology exploiting run-based performance differences, described above, allows to remove some sources of randomness and point out the real statistically significant differences between the performance of the three planners in terms of average difference of expected time to traverse the path $\overline{\Delta \mathbb{E}[t]_n}$, which is our main result. The density plot on the right of Figure 4.b shows two charts, namely, the distribution of *differences* between expected time to traverse the path for ORC and STD (in red), and that between MRF and STD (in green). The two charts show that both ORC and MRF outperform STD, and their performance improvement is statistically significant. Focusing on the difference between ORC and STD (red line), the mean is -6.69, meaning that ORC needs on average 6.69 time units less than STD to reach the end of the path (average time to traverse the path is 97.34 for STD - see Figure 4.a - hence the improvement is of about 6.8%). Performing a *t-test* for the null hypothesis that the expected value (mean) of this distribution a is equal to zero we obtain a p-value $p = 4.7 \cdot 10^{-38}$, which proves the statistical significance of the result (since it is less than 0.05). A similar result is achieved by comparing MRF and STD. Their mean difference is lower (i.e., -0.71, meaning that MRF needs on average 0.71 time units less than STD to reach the end of the path, with an improvement of 0.7%) but still statistically significant, with a p-value $p = 0.033$.

Further details about differences in planner behaviors on specific path segments are displayed in Figure 4.c. From top to the bottom, we show the average difference of discounted return $\overline{\Delta r_{ni}}$ in each segment s_i (the segment index is in the x-axis in all the charts), the average difference of expected time $\overline{\Delta \mathbb{E}[t]_{ni}}$ to traverse the segment, the average difference of average action $\overline{\Delta \bar{a}_{ni}}$ in each segment, and the average difference of average belief-state distance $\overline{\Delta d_{SB}^{ni}}$ in each segment. The bar plots on the left show differences between ORC and STD, while those on the right display differences between MRF and STD. Interestingly, the average difference of expected time $\overline{\Delta \mathbb{E}[t]_{ni}}$ shows in which segments ORC and MRF outperform STD. In particular, the improvement of MRF with respect to STD in the expected time to traverse a segment increases in the second lap of the rectangle (see higher negative bars mainly in segments 6 and 8), which coincides with a smaller difference in average belief-state distance of MRF than STD (see negative $\overline{\Delta d_{SB}^{ni}}$). In other words, MRF and STD have almost the same behavior in the first segment, then MRF's belief gets closer to the real state than STD's belief (due to the use of prior knowledge to infer future segment difficulties) allowing MRF to outperform STD.

8.1.2. Discussion of results on ICE path

Tests performed on the ICE path, displayed in Figures 4.d, 4.e and 4.f, show even better performance. Focusing on discounted reward r_n (first row of the table in Figure 4.d and density function in Figure 4.e) we see that ORC has the best average performance (i.e., -88.17), followed by MRF (i.e., -103.23) and then by STD (i.e., -106.68). The average difference of expected time to traverse the path $\overline{\Delta \mathbb{E}[t]_n}$, shown on the right of Figure 4.e, are respectively -10.60 between ORC and STD, and -3.16 between MRF and STD. This means that ORC takes on average 10.60 time units (i.e., 10.2%) less than STD to traverse the path, and MRF takes on average 3.16 time unit (i.e., 3.1%) less than STD to traverse the path. In both cases the *t-test* for the null hypothesis that the expected value (mean) of this distribution is equal to zero is very close to zero (namely $p = 5.5 \cdot 10^{-44}$ for ORC-STD and $p = 1.8 \cdot 10^{-13}$ for MRF-STD), which proves the statistical significance of the performance difference. This difference can also increase for more complex and repeated paths.

The analysis of performance over single segments, displayed in Figure 4.f, enables to draw similar conclusions to those drawn for the rectangular path. A large part of performance improvement (in terms of both discounted reward and expected time to traverse the path) come from segments in the second part of the path, where MRF has already collected information about its difficulty from previous segments (see the negative difference of distance between real state and belief $\overline{\Delta d_{SB}^{ni}}$ which shows that MRF has a more precise belief than STD in the second part of the path).

8.2. Performance on Unity simulations

Tests on Unity aim at evaluating the performance of our approach on a simulated environment considering the physical properties of the environment, which are not considered by the C++ simulator analyzed in the previous subsection. The experimental setup for these tests is explained in Section 7.5.2.

8.2.1. Discussion of results on the rectangular path

We analyze here a single run performed on a Unity simulation of the rectangular path with difficulty configuration (H, L, M, L, H, L, M, L). The path is displayed in Figure 5.a, where segments with low difficulty (L) are colored in green, segments with medium difficulty (M) are colored in blue, and segments with high difficulty (H) are colored in red. The agent travels the rectangle twice and segment difficulties are fixed in the two laps.

The ORC planner, in Figure 5.b, performs an optimal strategy (in terms of expected reward), since it selects action L in subsegments with high or medium difficulty and action H in subsegments with low difficulty (green subsegments in Figure 5.b represent action L and red subsegments represent action H). The time to traverse the path (i.e., the opposite of the discounted return) is 56 time units, and no collision occurs. Under the rectangular path we display a heatmap of robot's belief per subsegment. Columns represent subsegments, rows represent difficulties, and the color of a general cell i, j represents the probability (from the current belief) that subsegment j has difficulty i. For instance, the robot's belief in all subsegments of the first segment has high probability (i.e., 1.0) for high difficulty (see the red cells

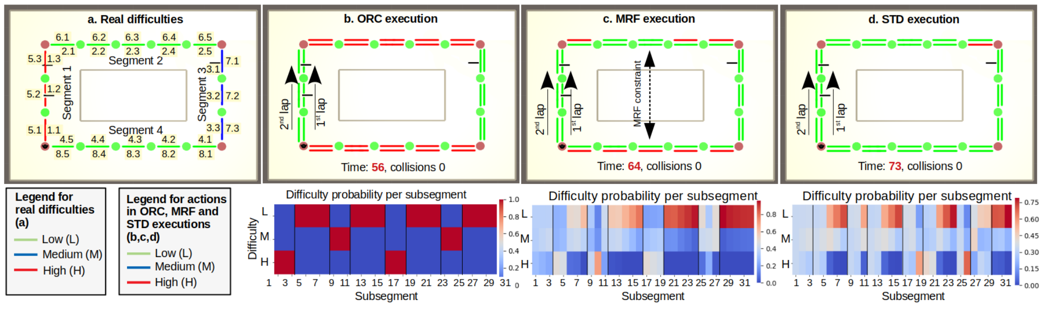


Figure 5: Performance comparison between the three planners on a specific configuration of difficulties in the rectangular path.

in the first three columns of the heatmap) and low probability (i.e., 0.0) for low and medium difficulty.

Comparing the plan generated by MRF (see Figure 5.c) with that generated by STD (see Figure 5.d) we observe that in the first lap they both select always action L (apart from a single action H chosen by STD at the end of the second segment). This is because their confidence to be in a segment with low difficulty is not high enough to choose action H (see the related heatmaps of belief probabilities), even in segments 2 and 4, where the difficulty is actually low. As expected, things change in the second lap, where MRF takes advantage of the knowledge about segment difficulties acquired in the first lap and it transfers this knowledge to subsequent segments connected by relationships of similarity between difficulties. For instance, the MRF edges (with probability 0.9) that generate a connected component containing segments 2, 4 (first lap), and 6, 8 (second lap) clearly improve the belief of the agent in segments 6 and 8. This is proved by the higher probabilities of low difficulty in segments 6 and 8 of the MRF heatmap (in the bottom of Figure 5.c) with respect to those of the same segments in the STD heatmap (in the bottom of Figure 5.d). Consequently, the MRF plan has action H in three subsegments of segment 6 and five subsegments of segment 8 (see red lines in the path of Figure 5.d), which improve the performance from the 73 time units required by STD to the 64 time units required by MRF. In both cases no collision occurs. In summary, the analysis shows that the MRF planner is able to propagate the knowledge about the hidden state acquired in initial segments to subsequent segments, with a consequent improvement of performance.

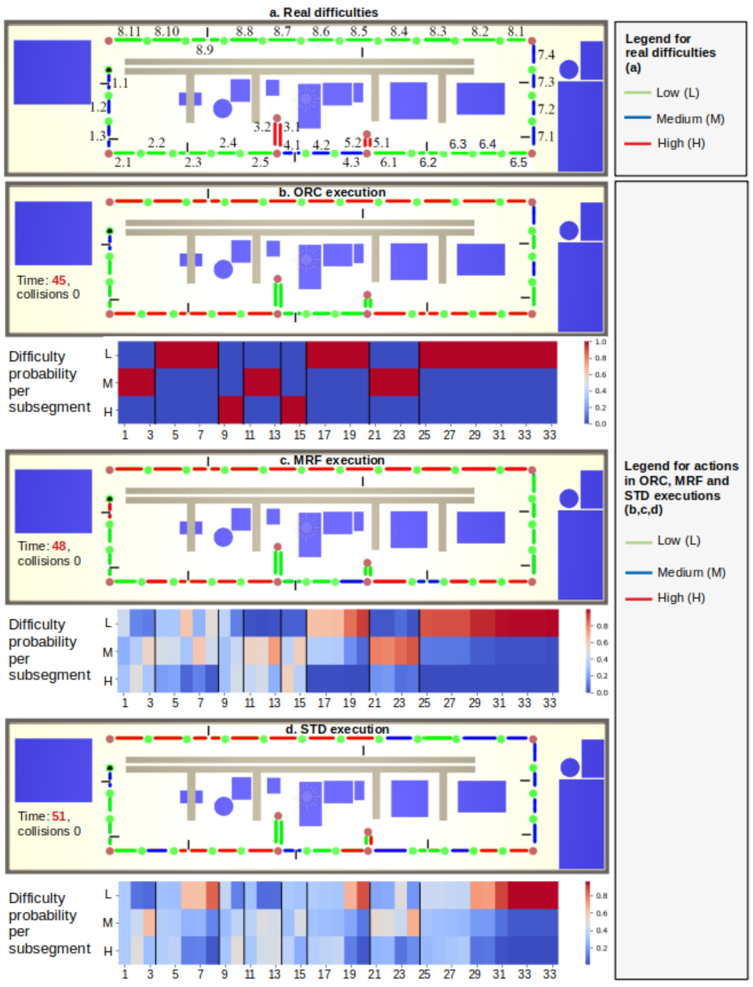


Figure 6: Performance comparison between the three planners on a specific configuration of difficulties in the ICE path.

8.2.2. Discussion of results on the ICE path

The test performed in the ICE path considers the difficulty configuration (M, L, H, M, H, L, M, L) (see Figure 6.a). The analyzed plans generated by ORC, MRF and STD take, respectively, 45, 48 and 51 time units, confirming the planner ranking observed in previous tests on the rectangular path.

The ORC planner selects almost always optimal actions (in terms of expected reward), namely, L in segments with high difficulty, and H in segments with low difficulty, and L or M in segments with medium difficulty (we separately checked the optimality of this relationship between difficulty and action). More interestingly, the comparison between MRF and STD confirms that MRF is able to get in advance higher confidence on the true configuration of difficulties with respect to STD and to use this knowledge to choose better actions. This capability is mainly shown in the second part of the path (from the fourth segment). For instance, in segment 6, MRF selects the optimal action (i.e., H) in four subsegments out of five while STD selects it in only two subsegments. The analysis of the related heatmaps shows that the reason of this difference is that MRF has a high confidence on the (true) low difficulty of segment 6 from the beginning of the segment itself (see red cells in difficulty L for segment 6 in the MRF heatmap) while STD reaches this confidence level only in the last two subsegments of segment 6 (see red cells in difficulty L for segment 6 in the STD heatmap). Accordingly, STD selects the optimal action only in the last two subsegments, while MRF selects them from the beginning. MRF has gathered this knowledge about the difficulty of segment 6 from segment 2, which has the same difficulty and it is connected to segment 6 by an edge in the MRF. A similar behaviour is observed in the (long) segment 8, where again MRF performs the optimal action (i.e., H) in all eleven subsegments while STD performs it only in the last seven subsegments.

8.3. Performance on the real world environment: testing arena

The explanatory run of our three planners in the real environment (testing arena) proves that the movements of the real robot have close correspondence to movements of the agent in the Unity simulation environment of the rectangular path. Figure 7 shows the starting point of a test performed in the arena. The video of the entire experiment is attached. It shows the Turtlebot moving in the real environment in three cases, namely, using the ORC, the MRF and the STD planner, respectively, with difficulty configuration shown in Figure 6.a. The video displays the evolution of the agent's belief, the sequence of actions selected by the planners and the time required to complete the path. Crucially, we were able to obtain similar results to those obtained in the Unity simulation, showing that a porting of our POMCP-based planners from the Unity simulation to the real robot is possible.

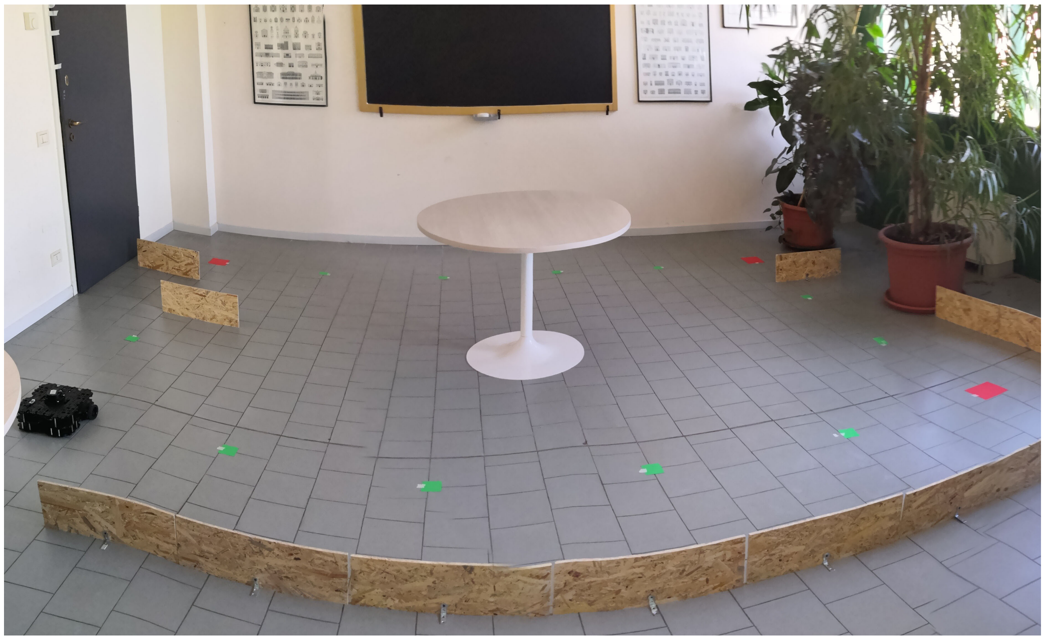


Figure 7: Overview of the testing arena.

8.4. Comparative analysis

To further investigate the proposed approach and highlight its benefits, we compare its performance with that of some state-of-the-art controllers for mobile robots. Our goal is to show that by introducing the POMCP-based planner with MRF on top of the engine controller we actually get a performance improvement over using only the engine controller. We perform this test using two state-of-the-art controllers as a baseline, namely, the DRL engine controller presented in (Marchesini and Farinelli, 2020b) and the ROS navigation stack[9]. For a fair comparison, we separately tuned the parameters for the navigation stack (e.g., for localization and sensing of the environment) in both the rectangular and ICE path. In detail, to balance the trade-off between performance and navigation, the difficulty of each segment was set to medium, and the default settings for the navigation stack provided good performance in both environments.

Table 3 shows the results of the comparative analysis. Tests were performed on the experimental setting described in Section 8.2, using Unity

[9]http://wiki.ros.org/navigation

simulators of the rectangular path (left column) and the ICE path (right column). We call our planner POMCP[MRF]+DRL as it uses the POMCP and the MRF to regulate velocity and the DRL controller for path planning. On the rectangular path, our planner requires 64 time units to complete the path, while the DRL engine controller alone requires 70 time units (about 9% more), showing that the velocity regulation performed by POMCP with MRF is important to improve navigation performance. On the other hand, the standard ROS controller manages to complete the path in 63 time units, one less than our approach. This difference is negligible, hence we consider the two controllers as having almost the same performance in this environment. The motivation for this result is that the environment is quite simple and there is no room to improve performance beyond that of ROS controller. On the ICE path instead POMCP[MRF]+DRL requires 48 time units to complete the path, the DRL engine controller alone requires 65.9 time units (about 37% more) and the ROS controller requires 72.3 time units (about 51% more). In this case our approach strongly outperforms both the DRL controller and the ROS controller. The reason for this good result is clearly related to the introduction of POMCP for velocity regulation, since also the standard planner POMCP[STD]+DRL outperforms both the DRL controller and the ROS controller with 51 time units required to complete the path. However, the introduction of prior knowledge about segment difficulty relationships via the MRF yields a further performance improvement of 6%, showing the importance of the MRF in the proposed approach. In a longer and more articulated environment, as the ICE path, our approach shows all its capability to outperform standard controllers. We finally notice that the performance improvement, in terms of time units, between POMCP[STD]+DRL and POMCP[MRF]+DRL in Table 3 is higher than the performance improvement, in terms of expected time to traverse the path $\overline{\mathbb{E}[t]_n}$, shown in Figure 4 for C++ simulations. This is because the expected time to traverse the path $\overline{\mathbb{E}[t]_n}$ considers also the expected time due to possible collisions (see Equation 3) while in the specific experiments considered in Table 3 no collision took place. Since the time related to collisions is much larger than that for traversing path segments the total times in Table 4 are smaller and the percentage of improvement increases. Furthermore, considering a single test the results of Table 3 also do not consider the variability over all possible configurations of segment difficulties.

	Time units	
Controller	**Rectangle**	**ICE**
POMCP[MRF]+DRL	**64**	**48**
POMCP[STD]+DRL	73 (+14%)	51 (+6%)
DRL (Hessel et al., 2018)	70 (+9%)	65.9 (+37%)
ROS[10]	**63 (-2%)**	72.3 (+51%)

Table 3: Performance comparison between the controller based on POMCP+DRL (proposed in this paper), the controller based only on DRL, and a standard ROS controller.

8.5. Final remarks

The experiments presented in this paper show that the extension of POMCP presented in (Castellini et al., 2019) can be applied to real robotic platforms. We used the extended POMCP for regulating mobile robot velocity in an industrial-like environment characterized by uncertain motion difficulties. The velocity selected by the planner was communicated to an engine controller which performs path planning. This two-layer control architecture was tested in four experimental settings always showing good performance. In particular, extensive tests performed on the C++ simulator shown that the extension of POMCP, which integrates prior knowledge on the environment, outperforms the standard POMCP from 0.7% to 3.1% depending on the complexity of the path. Experiments performed on a more realistic Unity simulator shown that also considering physical details of the environment and the (Turtlebot) agent the overall architecture properly works. Then, experiments in a real testing arena further confirmed the applicability of the approach to real robotic platforms. The last test concerned performance comparison between our (two-layer) approach and state-of-the-art (one-layer) approaches using only the engine controller for both velocity regulation and path planning. This test highlighted the contribution of the POMCP-based regulation of the velocity. The performance improvement was up to 37% against a DRL engine controller and up to 51% against a standard ROS engine controller.

All tests show that the performance improvement of the extended POMCP over the standard POMCP is due to the ability of the extended POMCP to propagate the knowledge about segment difficulty acquired in the first part of the path to subsequent segments via the MRF. This mechanism allows the robot to have better confidence in the estimation of the difficulties for the segments in the second half of the path. This allows to achieve a significant performance improvement, specifically in that part. The proposed measures show that the belief of the robot in the second half of the path is closer to the ground truth (in terms of belief-state distance) for the extended POMCP than for the standard POMCP. The improvement in the estimation of the ground truth is the cause of the planning performance improvement. We observed an increase of this effect in longer and more articulated paths because they give our approach more chance to exploit the similarity structure among segments. This happens even more if the robot repeats the same path several times.

9. Conclusion and future work

We presented an approach based on POMCP for improving the time performance of a mobile robot navigating in a path when prior knowledge is available about the similarity of path segments. Our results demonstrate the possibility to apply this planning approach to a real-world industrial-like environment characterized by uncertain motion difficulties. Moreover, we show that an extended version of POMCP considering prior knowledge about the environment is able to outperform the standard POMCP by improving the belief about the true state of the environment. This effect is obtained by propagating the knowledge on the environment acquired in the initial parts of the navigation to parts of the environment explored subsequently. Furthermore, we have shown that the addition of the POMCP-based layer for velocity regulation yields a consistent performance improvement compared to state-of-the-art engine controllers, such as DRL and ROS standard controllers. The approach can be applied to several domains in which series of tasks having similar properties are executed sequentially and some knowledge about task similarity is available. Performance measures introduced in the paper allow also to show that the belief improvement is transformed by the planner to improved action selection.

This work takes a first important step towards the use of advanced planning methods for mobile robots in industrial applications. This paves the way towards several interesting research directions. Specifically, our future work in this area includes three main topics: *i)* the development of methods for learning the MRF online, which is useful when the knowledge about task similarities is not available a-priori but it can be learnt; *ii)* the development of safe transfer learning methods for POMCP to allow imperfect transition and observation models trained on simulators (such as the Unity simulator we used in our experiments) to be used in real environments; *iii)* the development of new methods for improving the explainability of POMCP-based policies to allow their use in environments with safety requirements (and related human responsibilities). Some example applications regarding the last point can be found in Industry 4.0 where the recent request for a stronger human-robot interaction needs reliable and comprehensible policies to guarantee the safety. Some first results on this line have been recently published (Mazzi et al., accepted, 2021, 2020) and current work is focused on applying those results also to real robotic platform to further extend the work here presented.

10. Acknowledgements

The research is partially funded by project "Dipartimenti di Eccellenza 2018-2022", Italian Ministry of Education, Universities and Research, and the European Union's Horizon 2020 research and innovation programme under grant agreement No 689341.

11. References

Altan, A., Hacolu, R., 2020. Model predictive control of three-axis gimbal system mounted on uav for real-time target tracking under external disturbances. Mechanical Systems and Signal Processing 138, 106548.

Amato, C., Oliehoek, F. A., 2015. Scalable Planning and Learning for Multiagent POMDPs. In: Proceedings of the 29th AAAI Conference on Artificial Intelligence. AAAI Press, pp. 1995–2002.

Anjomshoae, S., Najjar, A., Calvaresi, D., Främling, K., 2019. Explainable agents and robots: Results from a systematic literature review. In: Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems. AAMAS '19. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, pp. 1078–1088.

Basilico, N., Gatti, N., Amigoni, F., Jun. 2012. Patrolling security games: Definition and algorithms for solving large instances with single patroller and single intruder. Artificial Intelligence 184–185, 78–123.

Beretta, C., Brizzolari, C., Tateo, D., Riva, A., Amigoni, F., 2019. A sampling-based algorithm for planning smooth nonholonomic paths. In: 2019 European Conference on Mobile Robots (ECMR). pp. 1–7.

Bevacqua, G., Cacace, J., Finzi, A., Lippiello, V., 2015. Mixed-initiative planning and execution for multiple drones in search and rescue missions. In: Proceedings of the 25th International Conference on Automated Planning and Scheduling. ICAPS'15. AAAI Press, pp. 315–323.

Bishop, C. M., 2006. Pattern Recognition and Machine Learning. Springer-Verlag New York.

Boutilier, C., Dean, T., Hanks, S., 1999. Decision-theoretic Planning: Structural Assumptions and Computational Leverage. JAIR 11 (1), 1–94.

Boutilier, C., Poole, D., 1996. Computing optimal policies for partially observable decision processes using compact representations. In: Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2. AAAI'96. AAAI Press, pp. 1168–1175.

Browne, C., Powley, E., Whitehouse, D., Lucas, S., Cowling, P., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S., 2012. A Survey of Monte Carlo Tree Search Methods. IEEE Trans. Comp. Intell. AI Games 4 (1), 1–43.

Caccavale, R., Finzi, A., 7 2019. Learning attentional regulations for structured tasks execution in robotic cognitive control. Autonomous Robots 43 (8), 2229–2243.

Castellini, A., Bicego, M., Masillo, F., Zuccotto, M., Farinelli, A., 2020a. Time series segmentation for state-model generation of autonomous aquatic drones: A systematic framework. Engineering Applications of Artificial Intelligence 90, 103499.

Castellini, A., Chalkiadakis, G., Farinelli, A., 2019. Influence of State-Variable Constraints on Partially Observable Monte Carlo Planning. In: Proc. 28th International Joint Conference on Artificial Intelligence (IJCAI 2019). pp. 5540–5546.

Castellini, A., Marchesini, E., Farinelli, A., 2020b. Online Monte Carlo planning for autonomous robots: Exploiting prior knowledge on task similarities. In: Proceedings of the 6th Italian Workshop on Artificial Intelligence and Robotics, AIRO@AI*IA 2019, CEUR Workshop Proceedings AI*IA Series 2594. Springer-Verlag, pp. 25–32.

Castellini, A., Marchesini, E., Mazzi, G., Farinelli, A., 2020c. Explaining the Influence of Prior Knowledge on POMCP Policies. In: Bassiliades, N., Chalkiadakis, G., de Jonge, D. (Eds.), Multi-Agent Systems and Agreement Technologies - 17th European Conference (EUMAS 2020). Vol. 12520 of Lecture Notes in Computer Science. Springer, pp. 261–276.

Chakraborti, T., Fadnis, K. P., Talamadupula, K., Dholakia, M., Srivastava, B., Kephart, J. O., Bellamy, R. K. E., 2018. Visualizations for an Explainable Planning Agent. In: Proceedings of the 27th International Joint Conference on Artificial Intelligence, IJCAI-18. pp. 5820–5822.

Chen, A., Harwell, J., Gini, M., 2019. Maximizing energy battery efficiency in swarm robotics. CoRR abs/1906.01957, 1–13.

Correll, N., Bekris, K. E., Berenson, D., Brock, O., Causo, A., Hauser, K., Okada, K., Rodriguez, A., Romano, J. M., Wurman, P. R., 2018. Analysis and observations from the first amazon picking challenge. IEEE Trans. Automation Science and Engineering 15 (1), 172–188.

Coulom, R., 2006. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In: Proceedings of the 5th International Conference on Computers and Games. CG06. Springer-Verlag, Berlin, Heidelberg, p. 7283.

Farinelli, A., Iocchi, L., Nardi, D., 2017. Distributed on-line dynamic task assignment for multi-robot patrolling. Autonomous Robots 41 (6), 1321–1345.

Feldman, J. A., Sproull, R. F., 1977. Decision theory and artificial intelligence II: The hungry monkey. Cognitive Science 1 (2), 158 – 192.

Fiorini, P., Shiller, Z., 1998. Motion planning in dynamic environments using velocity obstacles. The International Journal of Robotics Research 17 (7), 760–772.

Fox, M., Long, D., Magazzeni, D., 2017. Explainable Planning. CoRR abs/1709.10256, 1–7.

Godoy, J., Karamouzas, I., Guy, S. J., Gini, M., 2016. Moving in a crowd: Safe and efficient navigation among heterogeneous agents. In: Proceedings of the25th International Joint Conference on Artificial Intelligence. IJCAI'16. AAAI Press, pp. 294–300.

Gopalakrishnan, B., Singh, A. K., Kaushik, M., Krishna, K. M., Manocha, D., 2017. Prvo: Probabilistic reciprocal velocity obstacle for multi robot navigation under uncertainty. In: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 1089–1096.

Grippa, P., Behrens, D., Wall, F., Bettstetter, C., 2019. Drone delivery systems: job assignment and dimensioning. Autonomous Robots 43, 261–274.

Gunning, D., Aha, D., 2019. DARPAs Explainable Artificial Intelligence (XAI) Program. AI Magazine 40 (2), 44–58.

Hauskrecht, M., 2000. Value-Function Approximations for Partially Observable Markov Decision Processes. JAIR 13, 33–94.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., Silver, D., 2018. Rainbow: Combining improvements in deep reinforcement learning. In: Proceedings of the 32nd AAAI Conference on Artificial Intelligence.

Huang, L., 2009. Velocity planning for a mobile robot to track a moving target - a potential field approach. Robotics Autonomous Systems 57 (1), 5563.

Juliani, A., Berges, V., Vckay, E., Gao, Y., Henry, H., Mattar, M., Lange, D., 2018. Unity: A General Platform for Intelligent Agents. CoRR.

Kaelbling, L., Littman, M., Cassandra, A., 1998. Planning and Acting in Partially Observable Stochastic Domains. Artificial Intelligence 101 (1-2), 99–134.

Katt, S., Oliehoek, F. A., Amato, C., 2017. Learning in POMDPs with monte carlo tree search. In: Proc. ICML 2017. Vol. 70. PMLR, pp. 1819–1827.

Kocsis, L., Szepesvári, C., 2006. Bandit Based Monte-Carlo Planning. In: Proc. of European Conference on Machine Learning, ECML'06. Springer-Verlag, Berlin, Heidelberg, pp. 282–293.

Koren, Y., Borenstein, J., 1991. Potential field methods and their inherent limitations for mobile robot navigation. In: Proceedings. 1991 IEEE International Conference on Robotics and Automation. pp. 1398–1404 vol.2.

Krotkov, E., Hackett, D., Jackel, L., Perschbacher, M., Pippine, J., Strauss, J., Pratt, G., Orlowski, C., 2017. The DARPA robotics challenge finals: Results and perspectives. Journal of Field Robotics 34 (2), 229–240.

Kumar, N. V., Kumar, C. S., 2018. Development of collision free path planning algorithm for warehouse mobile robot. Procedia Computer Science - International Conference on Robotics and Smart Manufacturing (RoSMa2018) 133, 456 – 463.

Langley, P., Meadows, B., Sridharan, M., Choi, D., 2017. Explainable agency for intelligent autonomous systems. In: Proceedings of the 31st AAAI Conference on Artificial Intelligence. AAAI17. AAAI Press, pp. 4762–4763.

Lanighan, M. W., Grupen, R. A., 2019. Long-term autonomous mobile manipulation under uncertainty. In: Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems. AAMAS '19. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, pp. 2084–2086.

Laroche, R., Trichelair, P., Combes, R. T. D., 2019. Safe policy improvement with baseline bootstrapping. In: Proceedings of the 36th International Conference on Machine Learning. Vol. 97. PMLR, USA, pp. 3652–3661.

Lauri, M., Ritala, R., 2016. Planning for robotic exploration based on forward simulation. Robotics and Autonomous Systems 83, 15–31.

Lee, J., Kim, G.-H., Poupart, P., Kim, K.-E., 2018. Monte-Carlo Tree Search for Constrained POMDPs. In: 32nd Conference on Neural Information Processing Systems, NeurIPS 2018. pp. 1–17.

Leonetti, M., Iocchi, L., Stone, P., 2016. A synthesis of automated planning and reinforcement learning for efficient, robust decision-making. Artificial Intelligence 241, 103–130.

Luperto, M., Fusi, D., Borghese, N. A., Amigoni, F., 2019. Exploiting inaccurate a priori knowledge in robot exploration. In: Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems. AAMAS 19. IFAAMAS, pp. 2102–2104.

Marchesini, E., Corsi, D., Farinelli, A., 2021. Genetic soft updates for policy evolution in deep reinforcement learning. In: ICLR.

Marchesini, E., Farinelli, A., 2020a. Discrete deep reinforcement learning for mapless navigation. In: 2020 IEEE International Conference on Robotics and Automation, ICRA '20.

Marchesini, E., Farinelli, A., 2020b. Genetic deep reinforcement learning for mapless navigation. In: Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '20.

Mazzi, G., Castellini, A., Farinelli, A., 2020. Policy interpretation for partially observable monte-carlo planning: a rule-based approach. In: Proceedings of the 7th Italian Workshop on Artificial Intelligence and Robotics (AIRO 2020@AI*IA2020). Vol. 2806 of CEUR Workshop Proceedings. CEUR-WS.org, pp. 44–48.

Mazzi, G., Castellini, A., Farinelli, A., 2021. Identification of Unexpected Decisions in Partially Observable Monte Carlo Planning: A Rule-Based Approach. In: Proc. of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021). pp. 889–897.

Mazzi, G., Castellini, A., Farinelli, A., accepted. Rule-based Shielding for Partially Observable Monte-Carlo Planning. In: Proc. of the 31th International Conference on Automated Planning and Scheduling (ICAPS 2021).

Miller, T., 2019. Explanation in artificial intelligence: Insights from the social sciences. Artificial Intelligence 267, 1 – 38.

OpenAI, Andrychowicz, M., Baker, B., Chociej, M., Józefowicz, R., McGrew, B., Pachocki, J. W., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., Schneider, J., Sidor, S., Tobin, J., Welinder, P., Weng, L., Zaremba, W., 2018. Learning dexterous in-hand manipulation. CoRR.

Orlandini, A., Suriano, M., Cesta, A., Finzi, A., 2013. Controller synthesis for safety critical planning. In: 25th IEEE International Conference on Tools with Artificial Intelligence. pp. 306–313.

Papadimitriou, C., Tsitsiklis, J., 1987. The Complexity of Markov Decision Processes. Math. Oper. Res. 12 (3), 441–450.

Parker, J., Nunes, E., Godoy, J., Gini, M., 2016. Exploiting spatial locality and heterogeneity of agents for search and rescue teamwork. J. Field Robot. 33 (7), 877–900.

Ratering, S., Gini, M., 1995. Robot navigation in a known environment with unknown moving obstacles. Autonomous Robots 1 (2), 149–165.

Ross, S., Pineau, J., Paquet, S., Chaib-draa, B., 2008. Online Planning Algorithms for POMDPs. JAIR 32, 663–704.

Russell, S., Norvig, P., 2003. Artificial Intelligence: A Modern Approach, 2nd Edition. Pearson Education.

Silver, D., Huang, A., et. al., C. J. M., 2016. Mastering the Game of Go with Deep Neural Networks and Tree Search. Nature 529 (7587), 484–489.

Silver, D., Schrittwieser, J., et. al., K. S., 2017. Mastering the game of go without human knowledge. Nature 550, 354–359.

Silver, D., Veness, J., 2010. Monte-Carlo Planning in Large POMDPs. In: 22nd Conference on Neural Information Processing Systems, NIPS '10. pp. 2164–2172.

Simao, T. D., Spaan, M. T. J., 2019. Safe Policy Improvement with Baseline Bootstrapping in Factored Environments. In: Proceedings of the 33rd AAAI Conference on Artificial Intelligence, AAAI'19. AAAI Press, pp. 4967–4974.

Smith, D. E., 2012. Planning as an Iterative Process. In: Proc. 26th AAAI Conference on Artificial Intelligence. AAAI'12. AAAI Press, pp. 2180–2185.

Spaan, M. T. J., Spaan, N., 2004. A point-based pomdp algorithm for robot planning. In: Proc. IEEE International Conference on Robotics and Automation, 2004, ICRA '04. Vol. 3. pp. 2399–2404.

Spaan, M. T. J., Vlassis, N., Aug. 2005. Perseus: Randomized point-based value iteration for pomdps. J. Artif. Int. Res. 24 (1), 195–220.

Sridharan, M., Meadows, B., 2019. Towards a theory of explanations for humanrobot collaboration. Kunstl Intell 33, 331–342.

Steccanella, L., Bloisi, D., Castellini, A., Farinelli, A., 2020. Waterline and obstacle detection in images from low-cost autonomous boats for environmental monitoring. Robotics and Autonomous Systems 124, 103346.

Sutton, R., Barto, A., 2018. Reinforcement Learning, An introduction, 2nd Edition. MIT Press, Cambridge, MA, USA.

Tai, L., Paolo, G., Liu, M., 2017. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS '17. pp. 31–36.

Wandzel, A., Oh, Y., Fishman, M., Kumar, N., Wong, L. L., Tellex, S., 2019. Multi-object search using object-oriented pomdps. In: 2019 International Conference on Robotics and Automation (ICRA). pp. 7194–7200.

Wang, Y., Chaudhuri, S., Kavraki, L. E., 2018. Bounded policy synthesis for POMDPs with safe-reachability objectives. In: Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems. AAMAS '18. IFAAMAS, pp. 238–246.

Wang, Y., Chaudhuri, S., Kavraki, L. E., 2019. Point-based policy synthesis for pomdps with boolean and quantitative objectives. IEEE Robotics and Automation Letters 4 (2), 1860–1867.

Wang, Y., Giuliari, F., Berra, R., Castellini, A., Bue, A. D., Farinelli, A., Cristani, M., Setti, F., 2020. POMP: pomcp-based online motion planning for active visual search in indoor environments. In: 31st British Machine Vision Conference 2020, BMVC 2020, Virtual Event, UK, September 7-10, 2020. BMVA Press.

Yang, F., Khandelwal, P., Leonetti, M., Stone, P., 2014. Planning in answer set programming while learning action costs for mobile robots. In: AAAI 2014 Spring Symposia. AAAI, pp. 71–78.

Yoon, H., Chen, H., Long, K., Zhang, H., Gahlawat, A., Lee, D., Hovakimyan, N., 2018. Learning to communicate: A machine learning framework for heterogeneous multi-agent robotic systems.

Zhang, S., Sridharan, M., Wyatt, J. L., 2015. Mixed logical inference and probabilistic planning for robots in unreliable worlds. IEEE Transactions on Robotics 31 (3), 699–713.

Zhang, Y., Sreedharan, S., Kulkarni, A., Chakraborti, T., Zhuo, H. H., Kambhampati, S., 2017. Plan explicability and predictability for robot task planning. In: IEEE International Conference on Robotics and Automation, ICRA '17. pp. 1313–1320.

Zhong, X., Zhong, X., Peng, X., 2014. Velocity-change-space-based dynamic motion planning for mobile robots navigation. Neurocomputing 143, 153 – 163.