A toolbox for multi-objective planning in non-deterministic environments with simulation validation

Seyedshams Feyzabadi Stefano Carpin

Abstract— **Mobile robots are often subject to several costs and tasked with multiple objectives with different priorities when operating in realistic situations. We recently proposed and theoretically analyzed a planner handling multiple costs and multiple tasks expressed as syntactically co-safe linear temporal logic formulas. In this paper, we present an easy-to-use toolbox which is capable of solving multi-objective planning in non-deterministic environments where multiple tasks have to be accomplished. We experimentally evaluate this planner in a simulation scenario, and we moreover analyze its robustness to modeling errors in the underlying motion models. We show how this theoretical framework can be instantiated in a practical scenario and demonstrate its robustness.**

I. Introduction

Autonomous ground vehicles are becoming a reality in our every day lives and will play an increasingly more important role in the future. While it may still take a few years before autonomous cars will be available to the general public, fleets of autonomous vehicles operating in industrial environments are already a reality, with the Kiva system being perhaps the most famous example [5], [6]. Notwithstanding, if robots are required to operate in environments not specifically engineered for them, or to co-work with humans, multiple challenges still need to be addressed. In particular, in so-called *low volume manufacturing* environments robots need to be frequently reconfigured to perform different jobs, often requiring to pursue multiple tasks at the same time. The ability to easily formulate task specifications is therefore instrumental to increase the usability of these systems. In recent years, the theory of formal languages, and in particular the theory of Linear Temporal Logic (LTL), has emerged as a promising instrument to tackle these challenges. While powerful, such theory can however be expanded in multiple directions. In our recent work [13] we proposed a planner combining the theory of Constrained Markov Decision Processes (CMDPs) [1] with a subset of LTL known as syntactically co-safe linear temporal logic (sc-LTL). The peculiar feature of this planner is that it produces a feedback control policy capable of tackling multiple objectives and tasks at once. In particular, it is applicable to scenarios where multiple cost functions are defined, and multiple tasks are specified as sc-LTL formulas. Each task is associated with a desired satisfaction probability, and the policy computed by the planner will minimize one of the cost functions, while obeying bounds on the other cost functions, and at the same time guaranteeing that all additional tasks are completed with probability exceeding the given desired satisfaction probability. If the constraints and objectives are incompatible, the planner recognizes this situation and reports failure.

In [13] we provided a thorough theoretical analysis of the problem and performed limited validation in a fairly abstract simulation setting. In this paper, we present a toolbox that can be easily reconfigured to add new costs or change existing cost functions to adapt to new environments. It also provides a very simple mechanism to define tasks and their assigned satisfaction rates. The toolbox can be downloaded from the Internet[1] and freely used.

In this paper, we also thoroughly evaluate the planner and its robustness to modeling errors, in particular for what concerns the transition probabilities characterizing the underlying CMDP. Robustness to modeling errors for CMDPs has been scarcely considered in the past. In our past work [4] we have provided some theoretical bounds with regard to bounded errors in the cost functions, but to the best of our knowledge robustness with regard to errors in transition probabilities in CMDPs is a scarcely investigated area. Our validation is performed in a factory-like environment where a UGV is tasked with various pick and place tasks. We rely on ROS/Gazebo [16] to perform a large number of trials, and we therefore also present a set of principles to turn high-level policies into actionable robot controllers ready to be executed on state of the art robotic platforms. The remainder of this paper is organized as follows. In Section II we discuss the state of the art. In Section III we shortly summarize the results presented in [13] and formalize the planning problem we consider. Section V describes the experimental setup and discusses implementation notes. Experimental results are presented in Section VI, and conclusions are offered in Section VII.

II. State of the Art

Autonomous and semi-autonomous robots have been extensively used in industrial applications. The challenge has been tackled from different aspects such as controller design [15] and trajectory planning [3]. In this paper we focus on motion planning with nondeterministic motions but observable states.

S. Feyzabadi and S. Carpin are with the School of Engineering, University of California, Merced, CA, USA.

This work is supported by the National Institute of Standards and Technology under cooperative agreement 70NANB12H143. Any opinions, findings, and conclusions or recommendations expressed in these materials are those of the authors and should not be interpreted as representing the official policies, either expressly or implied, of the funding agencies of the U.S. Government.

[1] `http://robotics.ucmerced.edu/software`

Lavalle's book provides an ample introduction to robot planning algorithms [14]. Amongst these, Markov Decision Processes (MDPs) have been extensively used when state transitions are uncertain but the state of the system is observable [10]. In the MDP formulation a single objective function is considered, but in many practical situations more than one cost function has to be taken into account. For instance, if the objective function is to minimize risk along a path, some other factors such as path length or energy consumption could be considered. Constrained Markov Decision Processes extend MDPs by considering multiple costs. Their solution produces a feedback policy optimizing one objective function while ensuring that the remaining ones are bounded (in expectation) by some user specified constants. A comprehensive description for CMDPs is presented in [1]. While MDPs have been extensively used for robot planning, less emphasis has been given to CMDPs, although this approach is not entirely unexplored [7], [11], [12].

LTL has emerged as a powerful tool to model reactive systems [2] and is enjoying growing popularity in the robotics community. Syntacticly co-safe LTL (sc-LTL) properties [8] have been introduced to model situations where systems operate for a finite amount of time, and therefore their objectives have to be fulfilled in finite time. Sc-LTL has been used to define various robotics tasks [9], [8], [17]. However, all of these methods focused on a single task and are not immediately usable when more tasks are simultaneously considered. In an effort to overcome these limitations, we recently proposed a planner [13] that can handle an arbitrary number of tasks where each of them has to be satisfied with a given probability.

III. MULTI-OBJECTIVE PLANNING WITH MULTIPLE TASKS

We introduce the problem solved by the planner we presented in [13] and briefly summarize its basic properties. The reader is referred to [13] for a thorough discussion about its theoretical foundations and related algorithmic details. Consider a scenario where an autonomous ground vehicle operates in an environment where it can reliably localize itself, e.g., an autonomous forklift operating on a factory floor where suitable unique markers have been installed. The vehicle is subject to various costs as it operates, and without loss of generality we assume that we prefer executions where these costs are low. Examples of such costs include the time spent to complete a task, traveled distance, consumed energy, and the amount of undertaken risk,[2] just to name a few. When multiple costs are considered, a possible approach to solve the corresponding multi objective optimization problem is to combine all cost functions into a single objective function, for example through a linear combination. However, with this approach the resulting objective function does not have a physical meaning because it combines together heterogeneous quantities. An alternative approach consists in minimizing one of the objective functions (called primary objective) while imposing bounds on the others. This is the approach we follow. For example, in our scenario we may aim at minimizing the time to complete the mission while having bounds on the consumed energy and traveled distance. The mission of the robot is to complete one or more tasks, where each task is expressed as a formula in sc-LTL, whose formalization is given later. In general it may be impossible for the robot to complete all tasks while satisfying all the constraints on the costs. Therefore we associate to each sc-LTL formula a desired completion probability, so that each task can be given a different weight. The objective is then to compute a policy that in expectation completes every task with its specific desired probability, while obeying to the constraints and minimizing the primary objective.

In the following, we briefly recap sc-LTL as a formalism to specify tasks. We then describe labeled constrained Markov Decision Processes, formulate the problem we aim to solve, and give a linear program that solves it.

A. Syntacticly Co-safe Properties

The formalism of LTL has emerged as one of the leading approaches to specify desired behaviors for reactive systems. A comprehensive introduction to the subject is provided in [2]. In LTL, a set of atomic propositions Π is defined, and a labeling function L is introduced to associate to each state a truth value to every proposition in Π. As the system evolves over time and its state changes, the values assumed by the atomic propositions change as well, as defined by the labeling function L. By concatenating the truth values assumed by the propositions, *words* of infinite length are obtained. LTL uses the usual logic operators ($\neg$, $\vee$, $\wedge$), and in addition it introduces the temporal operators *eventually* ($\Diamond$), *next* ($\bigcirc$), *until* (U) and *always* ($\Box$). LTL can be used to specify properties like *safety*, *liveness*, and others that are satisfied by words of infinite length. However, most robotic tasks have finite duration, and LTL is therefore not necessarily the most appropriate choice because it reasons about words of infinite length. To overcome this limitation, syntactically co-safe LTL (sc-LTL) properties have been introduced. A sc-LTL property is built using the operators eventually, next, and until. Furthermore, the operator $\neg$ can only be used in front of atomic propositions. Note that the operator *always* is not used. sc-LTL properties are verified by words of finite length and can therefore be used to specify properties for a robot operating for a finite amount of time.

B. Labeled Constrained Markov Decision Processes

A labeled CMDP (LCMDP) extends the classic definition of CMDP by adding atomic propositions and a labeling function associating to every state a truth value to the atomic propositions. An extensive introduction to CMDPs can be found in [1]. A finite CMDP is defined as follows.

Definition 1: A finite CMDP is given by the tuple $\mathcal{C} = (S, \beta, A, c_i, \Pr)$ where

- S is a finite set of states.

[2]We will introduce a risk measure later on.

- β is a probability mass distribution over S providing the initial distribution over S, i.e., $\beta(s_j)$ is the probability that the initial state of the CMDP is s_j.
- $A = \cup_{s \in S} A(s)$ is a finite set of actions, where $A(s)$ is the set of actions executable in state s. S and A induce the definition of the set K as follows: $K = \{(s,a) \in S \times A \mid s \in S \wedge a \in A(s)\}$.
- c_i: $K \to \mathbb{R}_{\geq 0}$, $i = 0, \dots, n$ are $n+1$ cost functions. When action a is executed in state s each of the costs $c_i(s,a)$ is incurred.
- Pr: $K \times S \to [0,1]$ is the transition probability function where $\Pr(s_1, a, s_2)$ is the probability of reaching state s_2 from s_1 after applying action a.

Labeled CMDPs extend CMDPs by adding a set of atomic propositions and a labeling function.

Definition 2: A LCMDP is given by the tuple $\mathcal{LC} = (S, \beta, A, c_i, \Pr, AP, L)$ where $S, \beta, A, c_i, \Pr$ are as in the CMDP definition and:

- AP is a finite set of binary atomic propositions.
- L: $S \to 2^{AP}$ is a labeling function that defines for each state which atomic propositions are true.

A *policy* π is a function defining which action should be taken in every state. Optimal policies for (L)CMDPs are Markovian, i.e., they depend on the current state only (and not on the past history), but they are in general randomized, i.e., they are defined as $\pi : S \to \mathbb{P}(A)$ where $\mathbb{P}(A)$ is the set of probability mass distributions over A. For a given state s, $\pi(s) \in \mathbb{P}(A(s))$, i.e., the action is chosen from the set $A(s)$ according to the mass distribution $\mathbb{P}(A(s))$.

Starting from a state $s_0 \in S$ and following a policy π, a sequence of states and actions $\omega = s_0, a_0, s_1, a_1, ...$ is generated, where $a_i \in A(s_i)$. In an LCMDP the sequence ω induces a sequence of atomic propositions that are true, as defined by the labeling function, i.e., $\mathcal{L}(\omega) = L(s_0)L(s_1)...$ The sequence ω satisfies an sc-LTL formula ϕ if and only if $\mathcal{L}(\omega)$ satisfies ϕ. The last element to be introduced to formalize the problem definition is a set of cost criteria for each of the c_i functions in the LCMDP definition. The *total cost* is the undiscounted cost accrued during a sequence ω. For this cost to be finite, it is necessary to impose that the LCMDP is absorbing.

Definition 3: An LCMDP is absorbing if its state set S can be partitioned into two subsets S' (transient states) and M (absorbing states) so that for each policy π:

1) for each $s \in S'$, $\sum_{t=0}^{+\infty} \Pr^{\pi}_{\beta}[S_t = s] < +\infty$ where $\Pr^{\pi}_{\beta}$ is the probability distribution induced by β and π.
2) for each $s \in S'$, $s_m \in M$ and $a \in A(s_m)$ we have $\Pr(s_m, a, s) = 0$.
3) $c_i(s,a) = 0$ for each $s \in M$ and each $0 \leq i \leq n$.

These conditions impose that under every policy the state eventually reaches the set M and remains there, where no more cost is accrued. In an absorbing LCMDP, the $n+1$ costs induce $n+1$ corresponding total cost functions defined as follows[3]

$$c_i(\pi, \beta) = \mathbb{E}\left[\sum_{t=0}^{+\infty} c_i(s_t, a_t)\right]$$

where the expectation is taken with respect to the probability distribution over trajectories induced by π and β. Note that, according to the theory of CMDPs, these costs depend both on the policy π and the initial mass distribution β.

C. Problem Formulation

At this point we have all the elements to introduce the multi-objective, multi task MDP problem defined in [13].

Multi-objective, multi-task MDP (MOMT-MDP) problem : Given

- an LCMDP $\mathcal{LC} = (S, \beta, A, c_i, \Pr, AP, L)$ with $n+1$ costs functions $c_0, c_1, \dots, c_n$;
- m sc-LTL formulas $\phi_1, \dots, \phi_m$ over AP;
- n cost bounds $B_1, \dots, B_n$;
- m probability bounds $P_{\phi_1}, \dots, P_{\phi_m}$;

determine a policy π for $\mathcal{LC}$ that:

- minimizes the cost $c_0(\pi, \beta)$;
- for each cost c_i, $(1 \leq i \leq n)$, $c_i(\pi, \beta) \leq B_i$;
- for every trajectory ω, each of the m formulas ϕ_i is satisfied with at least probability P_{ϕ_i}.

Note that the definition allows to consider tasks that must be completed for sure. This can be done by defining a suitable formula ϕ_i and setting the corresponding probability bound P_{ϕ_i} to 1. The following theorem, proven in [13], establishes that the solution to MOMT-MDP problem can be found solving a suitably defined linear program.

Theorem 1: Let $\mathcal{LC} = (S, \beta, A, c_i, \Pr, AP, L)$ be an LCMDP, and $\phi_1, \dots, \phi_m$, $B_1, \dots, B_n$ and $P_{\phi_1} \dots P_{\phi_m}$ as in the definition of the MOMT-MDP problem. The problem has a solution if and only if the following linear program is solvable:

$$\min_{\rho(x,a) \in K} \sum_{x \in S'_p} \sum_{a \in A(x)} c_0(x,a)\rho(x,a) \tag{1}$$

subject to

$$\sum_{x \in S'_p} \sum_{a \in A(x)} c_i(x,a)\rho(x,a) \leq B_i, i = 1, ..., n$$

$$\sum_{a \in A(\mathscr{S}_i)} \rho(\mathscr{S}_i, a) \leq 1 - P_{\phi_i}, i = 1, ..., m$$

$$\sum_{x' \in S'_p} \sum_{a \in A(x')} \rho(x,a)[\delta_x(x') - \Pr(x', a, x)] = \\ = \beta(x), \forall x \in S'_p$$

$$\rho(x,a) \geq 0, \forall x \in S'_p.$$

[3]Note that we use the symbol c_i both for the LCMDP costs and for the total costs because they are strictly related, but these are two different functions, as evident from their different domains.

where[4] S_p' is a new state spaces induced by S and the ϕ_is, and $\mathscr{S}_i$ are elements in this new state space (see [13] for the detailed definition).

In [13] we also show that if the linear program (1) has a solution, then the solution $\rho^*(x,a)$ defines the optimal policy $\pi(x,a)$ for each $(x,a) \in K$.

IV. MOMT-MDP Toolbox

The MOMT-MDP toolbox implements the aforementioned approach in Matlab. The input parameters of this toolbox are as following:

- An LCMDP $\mathcal{M}$ where $n+1$ cost functions are defined for each state. The toolbox provides a class for LCMDPs.
- m DFAs. The DFA class implemented in the toolbox is used for this purpose.
- n upper bounds for the non-primary cost functions, encoded as a vector $\mathcal{A}$.
- m satisfaction probabilities for each DFA. These values are appended to $\mathcal{A}$.

Upon proper setup, the toolbox calculates the optimal policy according to the linear program formerly presented. If a solution satisfying the constraints is found, the toolbox generates a feedback policy π specifying which action should be taken at any given state. Additionally, the planner simulates multiple executions of the generated policy to experimentally ensure that the policy satisfies the given constraints.

This toolbox comes with some auxiliary functions to convert an image into an LCMDP (see section V for examples), and also to verify that the user defined DFAs are correct and total. The class diagram of the toolbox is given in Figure 1.

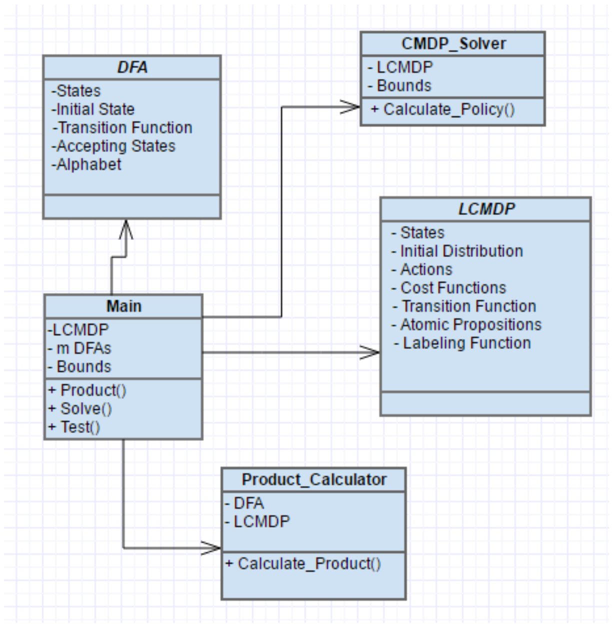

Fig. 1. Class Diagram for the MOMT-MDP Toolbox

[4]$\delta_x(x')$ equals to one, if $x = x'$ and zero otherwise

V. Experimental Setup

We describe various simulations aiming at assessing strengths and limits of the planner we just described. In particular, we aim at showing its ability to express suitable missions in scenarios relevant to manufacturing and automation, and its robustness to modeling errors. We also provide a detailed description on how the planner can be implemented using the mainstream ROS software platform. For all our simulations we rely Gazebo.

A. Environment Setup

We consider an autonomous forklift, i.e., an unmanned ground vehicle equipped with a forward facing gripper used to grab and release items. The Gazebo simulation environment does not provide a model for this vehicle, and therefore we developed one building upon the existing model for the Pioneer P3AT robot. Starting from the P3AT, we added four extra links and joints to enable the robot to restrain an object, and lift it. Figure 2 shows the model we developed. Note that on top of the gripper we placed a laser range finder to localize the robot inside the known map of the environment.

From a mobility perspective the P3AT is a differential drive platform, and one could argue that forklifts instead use Ackermann steering, where the rear wheels are used for steering. While this is true, this distinction is immaterial to the nature of the experiments we present, because, as explained in the next subsection, navigation is fully handled by the ROS navigation stack that can manage different mobility configurations. The robot operates in the known environment displayed in the rightmost panel of Figure 2. In our implementation a map of the environment is acquired upfront using one of ROS' built in SLAM algorithms. The map is then discretized into cells of 0.5×0.5 meters. With reference to the LCMDP formulation, the state of the robot is (x, y, c), where x, y identify the grid cell where the robot is located, and c is a binary variable indicating whether the robot is carrying something or not. The set of actions is $A = \{$*left*, *right*, *up*, *down*, *load*, *unload*$\}$. The first four actions describe motions in the grid[5], whereas the last two indicate the action on the gripper. Note that the orientation of the robot is missing from the state because thanks to the underlying motion controller the robot is capable of executing any of the four motion actions irrespectively of the direction it is facing. For what concerns the transition probabilities, when one of the *left*, *right*, *up*, *down* actions is executed, it succeeds with probability 0.63. This value was derived experimentally by simulating the actions multiple times and observing how often it succeeds. The action *load* succeeds with probability 0.6, whereas the action *unload* succeeds with probability 1.

B. Software architecture

From a software perspective we organized the system into the two-layer architecture shown in Figure 3. The top layer

[5]Actions that cannot be executed in a state s are removed from $A(s)$, e.g., motions that would move the robot into a wall.

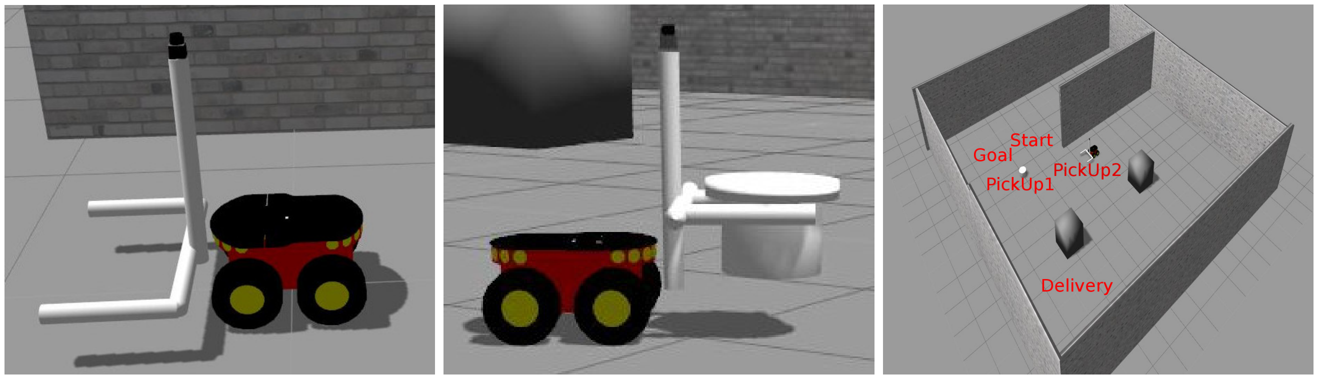

Fig. 2. On leftmost figure, the autonomous vehicle with its gripper fully opened at its lowest position. In the center figure, the vehicle carrying an object. Note that the fork as been closed and lifted. On the right, the environment used for testing.

(Layer 2) is in charge of generating the policy by solving the linear program in Eq. (1), and to determine the desired action $\pi(s)$ as a function of the current state. Hence, it receives from the lower layer (Layer 1) the robot pose and the grasp status. These values determine the current state $s = (x, y, c)$, and the next action $\pi(s) \in A(s)$ can therefore be identified. If the action is either *left*, *right*, *up* or *down*, it is sent to the *Move Base* node in Layer 1, whereas if the action is *load* or *unload* it is sent to the node *Load Controller*.

Layer 1 consists of a set of standard ROS nodes (*AMCL*, *Move Base*, *Map Server*) and of a custom developed node (*Load Controller*) in charge of the gripper device we introduced. AMCL handles the localization task using a particle filter. Given a map of the environment (provided by Map Server), and the data from the range finder and odometry, it provides an estimate of the robot pose. In our experiments, since the map is static, the estimate provided by AMCL is reliable and consistent with our hypothesis of state observability. Move Base implements the ROS mobility stack, i.e., drives the robot to a desired target location. Finally, Load Controller implements a PID algorithm to open/close the gripper and move it up or down.

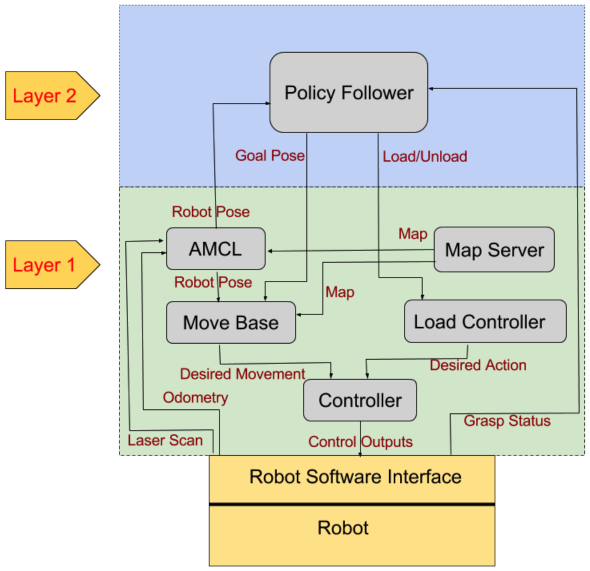

Fig. 3. Software architecture

C. Costs, Atomic Propositions, and Tasks

According to the formulation presented in Section III, we introduce two costs, i.e., risk (c_0) and traveled distance (c_1). The risk cost is defined by a risk map extracted from the environment, where locations near to obstacles are deemed more risky (see Figure 4). The overall risk along a path is the sum of the risk values of the cells traversed along the path. Next, we define two missions characterized by different atomic propositions and tasks.

Mission 1. For the first mission the set of atomic propositions is $\Pi = \{P_1, P_2, D, G, \mathcal{L}\}$, and the associated labeling function L is as follows (refer to the right panel in Figure 2).

- $L(P_1)$ is true when the robot is in the state corresponding to location *Pickup 1* and false otherwise.
- $L(P_2)$ is true when the robot is in the state corresponding to location *Pickup 2* and false otherwise.
- $L(D)$ is true when the robot is in the state corresponding to location *Delivery* and false otherwise.
- $L(G)$ true when the robot is in the state corresponding to location *Goal* and false otherwise.
- $L(\mathcal{L})$ is true if the robot is carrying something (loaded) and false otherwise.

The following two tasks are defined as sc-LTL formulas over Π. In both cases, we assume that the robot starts without carrying anything.

1) Task 1: $\phi_1 = \neg\mathcal{L} U P_1 \bigcirc \mathcal{L} U D \bigcirc \neg\mathcal{L} U G$. In plain words this task requires to go to location *Pickup 1*, retrieve the element, bring it to location *Delivery*, release it, and then terminate in location *Goal*.
2) Task 2: $\phi_2 = \neg\mathcal{L} U P_2 \bigcirc \mathcal{L} U D \bigcirc \neg\mathcal{L} U G$. This task requires to go to location *Pickup 2*, retrieve the element, bring it to location *Delivery*, release it, and then terminate in location *Goal*.

We associate to ϕ_1 a probability $P_{\phi_1} = 0.4$ and to ϕ_2 a value $P_{\phi_2} = 0.3$. When solving the linear program, we minimize the risk cost, while setting a bound $B_1 = 70$ on

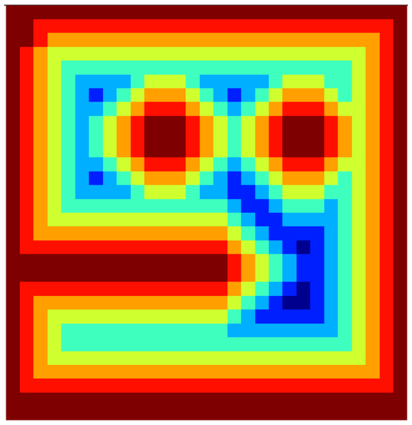

Fig. 4. Risk Map. Warmer colors indicate higher risk areas.

the cost c_1 (path length).

Mission 2. For the second mission the set of atomic propositions is $\Pi = \{S_1, S_2, P_1, P_2, D, G, \mathcal{L}\}$. The labeling function for $P_1, P_2, D, G, \mathcal{L}$ is as in Mission 1, whereas for S_1 and S_2 it is as follows (refer to Figure 5).

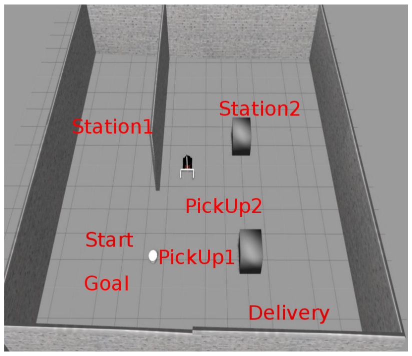

Fig. 5. Relevant locations for the tasks in the second mission.

- $L(S_1)$ is true when the robot is in the state corresponding to location *Station 1* and false otherwise.
- $L(S_2)$ is true when the robot is in the state corresponding to location *Station 2* and false otherwise.

We define two tasks also for the second mission, and we again assume that the robot starts without carrying anything.

1) Task 1: $\phi_1 = \neg\mathcal{L}US_1 \bigcirc \neg\mathcal{L}UP_1 \bigcirc \mathcal{L}UD \bigcirc \neg\mathcal{L}UG$. Informally speaking, the robot is required to go to *Station 1* while unloaded, receive some orders, then drive to *Pick Up 1* to pick up an object. Then it unloads the object at the *Delivery* location and terminates the task by going to the *Goal* location.
2) Task 2: $\phi_2 = \neg\mathcal{L}UP_2 \bigcirc \mathcal{L}US_2 \bigcirc \mathcal{L}UD \bigcirc \neg\mathcal{L}UG$. In this task the robot goes to *Pick Up 2* location, loads the object. Then it drives towards *Station 2* which can be a check point in order to verify the status of the object being carried. After the check point, it moves towards the *Delivery* location to drop it off and then drives to the *Goal* location.

We associate to ϕ_1 a probability $P_{\phi_1} = 0.4$ and to ϕ_1 a probability $P_{\phi_2} = 0.3$. When solving the linear program, we minimize the risk cost (c_0), while setting a bound $B_1 = 100$ on the path length (c_1).

VI. Results

Starting from the experimental setup we described in the previous section, we solve the linear program corresponding to the cases described in the first and second mission, and we then repeatedly executed the policy in Gazebo, logging the results in terms of traveled path, accrued risk, and percentage of accomplishments of the specified tasks. In every experiment the policy is executed 500 times. For both missions we first solve the linear program with values for the transition probability equal to the probabilities we experimentally determined, as described in section V-A. Next, to assess the robustness of the algorithm to modeling errors, we repeat the same experiments after having solved the linear program using transition probabilities that differ from the value we observed, and evaluate how the performance degrades.

A. *Mission 1*

In the first batch of experiments, we solve the linear program using our best estimates for the transition probabilities. Table I shows the results we obtained. It can be seen that over 500 runs the bound $B_1 = 70$ on the path is on expectation met with a rather small variance.

Quantity	Mean	Variance
Risk	362.9	64.2
Path Length	61.2	9.3

TABLE I

Mission 1 results in terms of total accrued cost and path length

Moreover, ϕ_1 was satisfied with probability 0.43 and ϕ_2 and with probability 0.25. The reason for the mismatch between $P_{\phi_2} = 0.3$ and the observed value of 0.25 is twofold. First, the number of samples is relatively small and then does not necessarily converge to the expected value. Second, as we will show in the next experiment, it seems like that the transition probability value we experimentally estimated is not extremely accurate.

Next, to experimentally evaluate the robustness of the algorithm to errors in the transition probabilities, we solved again the linear program using transition probability values different from the one we experimentally determined. Figure 6 shows the satisfaction rates for ϕ_1, ϕ_2 as the transition probability varies, whereas figure 7 shows the Kullback-Leibler divergence between the desired values for P_{ϕ_1} and P_{ϕ_2} and the observed values. From this second figure we can observe that 0.7 appears to be a better estimate for the transition probability.

B. *Mission 2*

Similar experiments were performed for mission 2. Table II confirms that the policy satisfies in expectation the bound $B_1 = 100$ for the expected path length.

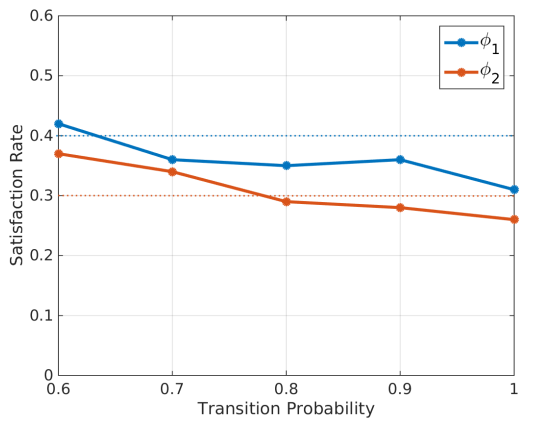

Fig. 6. Satisfaction rates of two tasks in mission 1.

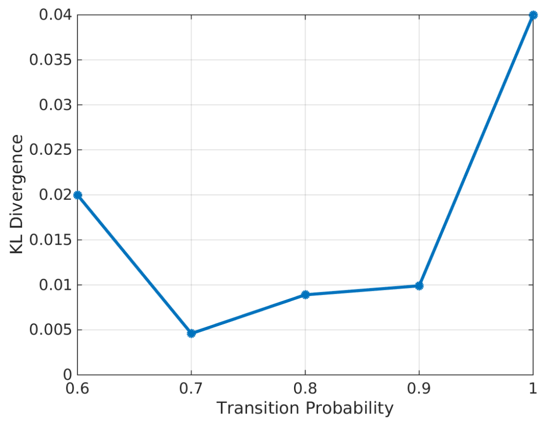

Fig. 7. KL Divergence for mission 1.

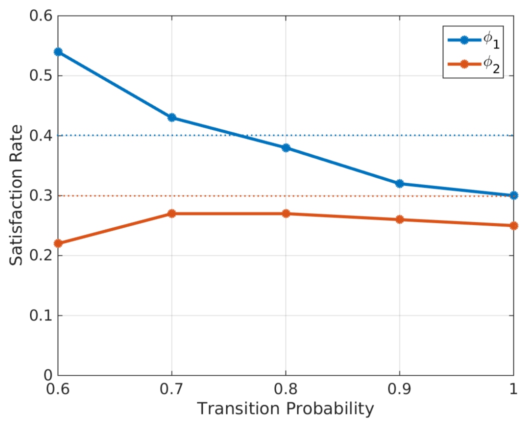

Fig. 8. Satisfaction rates of two tasks in mission 2.

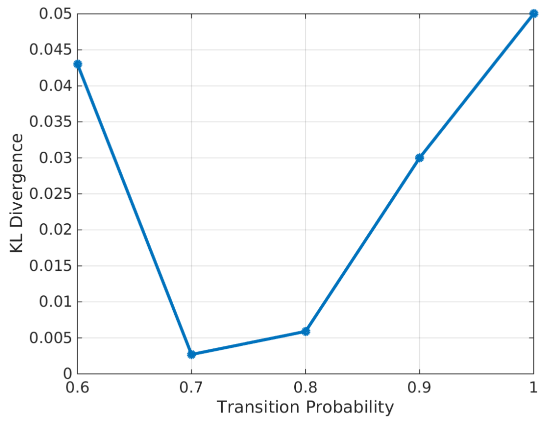

Fig. 9. KL Divergence for mission 2

In this case ϕ_1 was satisfied with probability of 0.49, and ϕ_2 with probability of 0.25. As compared to the values in Section V-C the desired value for P_{ϕ_2} is slightly off from the desired value. In our interpretation this is due to the fact that the transition probability value we used to solve the linear program is not necessarily the one best fitting the underlying model, as confirmed by Figure 8 and Figure 9 in which we observe the performance of the planner as the transition probability vary.

We conclude this section outlining that the presented results, although not completely aligned with the desired results, do not undermine the correctness of the planner. In fact, in [13] we showed that if the linear program in Eq. 1 is solved using transition probabilities accurately matching the underlying model, then all target probabilities are achieved.

Quantity	Mean	Variance
Risk	556.4	89.4
Path Length	102.6	18.2

TABLE II
MISSION 2 RESULTS IN TERMS OF TOTAL ACCRUED COST AND PATH LENGTH

VII. CONCLUSIONS AND FUTURE WORK

In this paper, we presented our toolbox to solve MOMT-MDP problems. In our former contribution we assessed the planner performance in an idealized abstract scenario in which all the parameters of the model were accurately provided. In this paper, instead, we implemented the planner and tested it on a realistic, noisy simulation environment in which not all parameters were precisely known. The first observation is that the multi-objective side of the problem seems to be less sensitive to inaccurate parameters, as evidenced by the fact that the bounds on the additional cost functions have always been met in expectation. On the contrary, the probability of satisfying the tasks seems to be less robust to modeling errors, as evidenced by small deviations from the desired values. Besides the experimental validation, we also illustrated a concrete architecture to instantiate the proposed planner on ROS, and we tested the system using a set of tasks inspired by manufacturing and logistics scenarios. In the future, an interesting research direction seems to be the development of formal bounds to assess the robustness of this planner to errors in the transition probabilities, similarly to what we did in [4] for errors in the cost functions.

References

[1] E. Altman. *Constrained Markov Decision Processes*. Stochastic modeling. Chapman & Hall/CRC, 1999.

[2] C. Baier and J.P Katoen. *Principles of model checking*. MIT press Cambridge, 2008.

[3] T. Chettibi, H. Lehtihet, M. Haddad, and S. Hanchi. Minimum cost trajectory planning for industrial robots. *European Journal of Mechanics-A/Solids*, 23(4):703–715, 2004.

[4] Y-L. Chow, M. Pavone, B.M. Sadler, and S. Carpin. Trading safety versus performance: Rapid deployment of robotic swarms with robust performance constraints. *ASME Journal of Dynamical Systems, Measurements and Control*, 137(3):031005, 2015.

[5] R. D'Andrea. Guest editorial: A revolution in the warehouse: A retrospective on kiva systems and the grand challenges ahead. *IEEE Transactions on Automation Science and Engineering*, 9(4):638–639, 2012.

[6] R. D'Andrea and P. Wurman. Future challenges of coordinating hundreds of autonomous vehicles in distribution facilities. In *IEEE International Conference on Technologies for Practical Robot Applications*, pages 80–83, 2008.

[7] X. Ding, B. Englot, A. Pinto, A. Speranzon, and A. Surana. Hierarchical multi-objective planning: From mission specifications to contingency management. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3735–3742, 2014.

[8] X. Ding, A. Pinto, and A. Surana. Strategic planning under uncertainties via constrained markov decision processes. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 4568–4575, 2013.

[9] K. Etessami, M. Kwiatkowska, M. Vardi, and M. Yannakakis. Multi-objective model checking of markov decision processes. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 50–65. Springer, 2007.

[10] E.A. Feinberg, editor. *Handbook of Markov Decision Processes: methods and applications*. Springer, 2012.

[11] S. Feyzabadi and S. Carpin. Risk-aware path planning using hierarchical constrained markov decision processes. In *Proceedings of the IEEE International Conference on Automation Science and Engineering*, pages 297–303, 2014.

[12] S. Feyzabadi and S. Carpin. HCMDP: a hierarchical solution to constrained markov decision processes. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3971–3978, 2015.

[13] S. Feyzabadi and S. Carpin. Multi-objective planning with multiple high level task specifications. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2016 (to appear).

[14] S. M. LaValle. *Planning algorithms*. Cambridge university press, 2006.

[15] J. Luh. Conventional controller design for industrial robots-a tutorial. *IEEE Transactions on Systems Man and Cybernetics*, 13:298–316, 1983.

[16] M. Quinley, B. Gerkey, and W.D. Smart. *Programming Robots with ROS*. O'Reilly, 2015.

[17] T. Wongpiromsarn, A. Ulusoy, C. Belta, E. Frazzoli, and D. Rus. Incremental synthesis of control policies for heterogeneous multi-agent systems with linear temporal logic specifications. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 5011–5018, 2013.