Manipulation Task Planning with Constrained Kinematic Controller

Marcos S. Pereira * **Bruno V. Adorno** **

* *Programa de Pós-Graduação em Engenharia Elétrica, - Universidade Federal de Minas Gerais, MG, (e-mail: marcos.si.pereira@gmail.com)*
** *Departamento de Engenharia Elétrica, Universidade Federal de Minas Gerais, MG, (e-mail: adorno@ufmg.br)*

Abstract: This paper addresses the integration of task planning and motion control in robotic manipulation, where automatically generated feasible manipulation sequences are executed by a controller that explicitly accounts for the task geometric constraints. To cope with the high dimensionality of the manipulation problem and the complexity of specifying the tasks, we use a multi-layered framework for task and motion planning adapted from the literature. The adapted framework consists of a high-level planner, which generates task plans for linear temporal logic specifications, and a low-level motion controller, based on constrained optimization, that allows to define regions of interest instead of exact locations while being reactive to changes in the workspace. Thus, there is no low-level motion planning time added to the total planning time. Moreover, since there is no replanning phase due to motion planner failures, the robot actions are generated only once for each task because the search for a plan occurs on a static graph. We evaluated this approach with two pick-and-place tasks with similar complexity to the original framework and showed that the number of plan nodes generated is smaller than the one in the original framework, which implies less total planning time.

Resumo: Este artigo trata da integração de planejamento de tarefas e controle de movimento em robótica de manipulação. O objetivo é gerar automaticamente sequências factíveis de manipulação para serem executadas por um controlador que considera restrições geométricas impostas pela tarefa. Para lidar com a alta dimensionalidade do problema de manipulação e a complexidade de especificar tarefas, foi usado um arcabouço multicamadas para planejamento de tarefa e movimento adaptado da literatura. O arcabouço adaptado consiste em um planejador de alto nível, que gera planos de tarefa para especificações em lógica temporal linear, e um controlador de movimento de baixo nível baseado em otimização com restrições, que permite definir regiões de interesse ao invés de localidades exatas e é reativo a mudanças no espaço de trabalho. Logo, não há adição de tempo de planejamento de movimento ao tempo total de planejamento. Além disso, como não há fase de replanejamento devido a falhas em um planejador de movimento, as ações para o robô são geradas apenas uma vez para cada tarefa, portanto, a busca por plano de tarefas ocorre em um grafo estático. Essa abordagem foi testada com duas tarefas de *pick-and-place* com complexidade similar à das tarefas realizadas no arcabouço original. O número de nós de planejamento gerados foi reduzido, o que implica em menor tempo total de planejamento.

Keywords:
Linear temporal logic; task planning; constrained control.

Palavras-chaves:
Lógica temporal linear; planejamento de tarefa; controlador baseado em restrições.

1. INTRODUCTION

As robots have become physically capable of executing highly complex manipulation tasks, there is a need to define plans in a longer time horizon and with a considerable number of manipulable objects in cluttered environments. Those plans determine the task execution order and how to execute them. Furthermore, there is also a need to increase robot autonomy and enable the robot to automatically generate those plans, which motivates the integration of task and motion planning (ITMP). In this sense, this work focuses on solving the ITMP problem for manipulation tasks in a computational efficient manner by combining state of the art techniques including task planning and motion controllers instead of motion planning.

Motion planning and task planning are studied in robotics from different point of views. Task planners generate plans by processing a high number of action states, that is, states that describe what the robot must do regardless of where objects are located in the environment (Kaelbling and Lozano-Perez, 2011). On the other hand, a motion planner treats the geometry of the problem, that is, it must know the location of each object to plan how to manipu-

late them (Kaelbling and Lozano-Perez, 2011). Neither of the approaches alone is enough to do manipulation task planning. Thus, there is a need to do ITMP.

The goal of ITMP is to use a task planner to define a high-level manipulation task sequence, that is used to generate a trajectory in the task space for a low-level motion planner that plans the trajectories in the configuration space. This low-level trajectory is then executed by a motion controller. This way, continuous motion planning is combined with discrete task reasoning (He et al., 2015). Discrete high-level task plans are appropriately described by using an abstract discrete model of the robot and formal specifications and, then, task planning can be done through model-checking techniques (Bhatia et al., 2011).

Task planning appears in many guises and usually are dealt with by using Artificial intelligence (AI). Fikes and Nilsson (1971) propose a problem solver called STRIPS (Stanford Research Institute Problem Solver) that searches for a model in a space of world models to reach a desired goal. However, STRIPS does not regard temporal specifications, only uses linear sequences of operators, and requires lower layers to translate its very high-level plans to robot actions (Fikes and Nilsson, 1971). In this sense, Lana et al. (2015) propose the Manipulation Task Model (MTM) that is dedicated to the planning of robotic manipulation tasks in a bottom-up manner.

The MTM describes actions and their sequence to take an object from an initial state to a desired state. The task primitive actions are defined as elements of dual quaternion algebra while the sequence is described by a small subset of process algebra. One of the advantages of the MTM is that it considers the task's physical parameters and, thus, facilitates the use of motion controllers to execute the task. However, MTM is specific to manipulation tasks, which is a drawback when describing tasks such as navigation. Furthermore, the MTM also does not allow temporal specifications.

An interesting alternative for STRIPS and the MTM is linear temporal logic (LTL), which allows to specify Boolean and temporal constraints. Also, LTL has correctness and completeness guarantees (Baier and Katoen, 2008, ch. 5), but the number of states of the specified task has combinatorial growth, which is also known as the state-explosion problem (Wongpiromsarn et al., 2010). Therefore, frameworks based on LTL usually build a discrete abstraction of the robotic system (Kress-Gazit et al., 2007; He et al., 2015; Kloetzer and Belta, 2008; Bhatia et al., 2011), but such construction is usually non-trivial.

LTL has been used mainly in mobile robotics (Kress-Gazit et al., 2007; Kloetzer and Belta, 2008; Bhatia et al., 2011). Nonetheless, the abstraction techniques for mobile robotics depend on the discrete representation of the state-space, which make them intractable to manipulation planning due to the large dimensionality of the manipulation problem. To solve this problem, He et al. (2015) propose a framework that allows the planning of high-level manipulation tasks specified in LTL and also handles the ITMP problem.

To solve the ITMP problem, Kambhampati et al. (1991) investigated the use of a hybrid planning model that contributes with expressiveness and reasoning power to traditional hierarchical planners through the use of a set of specialists, which adapt the plan to satisfy unexpected additional constraints. Hence, the planner and the specialists must know the constraints imposed by their decisions to avoid inconsistent tasks. In order to add more flexibility to ITMP, some works started using multi-layered frameworks that use a task planner to generate a motion sequence for the motion planner, which checks the viability of executing the task (Erdem et al., 2011; Kaelbling and Lozano-Perez, 2011; Srivastava et al., 2014; Lozano-Perez and Kaelbling, 2014; He et al., 2015).

In addition to using LTL, He et al. (2015) also propose a multi-layered framework to solve the ITMP problem, which uses a coordinating layer between high- and low-level to decide about the difficulty of executing the task. However, it also uses a motion planner, which requires motion planning time, in addition to the high-level planning time. Therefore, we adapt the framework of He et al. to solve the ITMP challenge by using a new approach to replace the motion planning layer.

In order to replace the motion planning layer, our work proposes the use of a constrained motion controller (Marinho et al., 2019), which allows the definition of regions of interest instead of discrete poses for the end-effector and the system to be reactive. The idea is to specify geometrical constraints, based on those regions of interest, to accomplish a desired task generated by the high-level task-planner without resorting to low-level motion planning. Constrained motion controllers depend on the specification of a set of constraints that must be respected by the system. They can be described by minimization problems (Laumond et al., 2015) that make use of extra available degrees of freedom of the robot to achieve the task while respecting the constraints. This enables the generation of collision-free motions by using mathematical programming. Nonetheless, in the general case there are no analytical solutions and numeric solvers must be used (Goncalves et al., 2016; Escande et al., 2014).

The main contribution of this work is to adapt the framework of He et al. (2015), which allows the specification of manipulation tasks in LTL, to use a constrained motion controller (Marinho et al., 2019), which allows imposing joint limits and constraints to prevent collisions with the workspace. This strategy greatly reduces the computational burden of the overall method, with the additional advantages that there is no need of motion planning and the system is reactive. Moreover, the merging of both frameworks combines the expressivity and temporal reasoning of LTL at the task level with the closed-loop stability guarantees of low level constrained motion controllers.

2. PRELIMINARIES

2.1 Linear Temporal Logic

In LTL, a *proposition* is a statement that can be true or false, but not both, and *atomic propositions* are the ones that do not depend on the truth or falsity of any other proposition. Let $\mathbb{A} = \{a_0, a_1, ..., a_N\}$ be a set of atomic propositions. LTL semantics is defined over infinite traces, that is, words over the alphabet $2^{\mathbb{A}}$, where $2^{\mathbb{A}}$ is the

power set of $\mathbb{A}$. Given letters $A_i \in 2^{\mathbb{A}}$, with $i = 0, 1, 2, \ldots$, words are infinite sequences such as $\sigma = A_0 A_1 A_2 \cdots$. For example, consider $\mathbb{A} = \{a_0, a_1, a_2\}$, hence, a possible word would be $\sigma = A_0 A_1 A_0 A_1 A_0 A_1 \cdots$, with $A_0 = \{a_0, a_1\}$ and $A_1 = \{a_0\}$, which means that a_0 is always true as it belongs to all letters, a_1 alternates between true (when it appears in a letter) and false (when it does not appear in a letter), and a_2 is always false, because it does not belong to any letter in the word σ.

A LTL formula φ is composed of atomic propositions, Boolean operators, and basic temporal operators. More specifically, a formula φ over $\mathbb{A}$ results in

$$\varphi = a \,|\, \neg a \,|\, \varphi_1 \wedge \varphi_2 \,|\, \varphi_1 \vee \varphi_2 \,|\, \varphi_1 \mathcal{U} \varphi_2 \,|\, \mathcal{X}\varphi_1 \,|\, \mathcal{E}\varphi_1, \quad (1)$$

where $a \in \mathbb{A}$. The Boolean operators are "negation" ($\neg$), "and" ($\wedge$), and "or" ($\vee$). The temporal operators "until" ($\mathcal{U}$) is such that φ_1 is true until φ_2 becomes true. The temporal operator "next" ($\mathcal{X}$) means that φ will definitely be true at the next step, and "eventually" ($\mathcal{E}$) [1] means that φ will become true at some point in the future. It is also possible to include the operator "implication" ($\varphi_1 \rightarrow \varphi_2$) and "equivalence" ($\varphi_1 \leftrightarrow \varphi_2$), although we do not make use of them in our current work.

To clarify what is a formula φ and the operators mentioned above, let $\mathbb{A} = \{a_0, a_1\}$. Therefore,

(1) $\varphi = \mathsf{T}$ is always true;
(2) $\varphi = a_0$ is true if and only if a_0 is true;
(3) $\varphi = \neg a_0$ is true if and only if a_0 is false;
(4) if $\varphi_0 = a_0$, $\varphi_1 = a_1$, then $\varphi = \varphi_0 \wedge \varphi_1$ is true if and only if both a_0 and a_1 are true;
(5) if $\varphi_0 = a_0$, $\varphi_1 = a_1$, then $\varphi = \varphi_0 \vee \varphi_1$ is true if and only if a_0 is true or a_1 is true or both are true;
(6) if $\varphi_0 = a_0$, $\varphi_1 = a_1$, then $\varphi = \varphi_0 \mathcal{U} \varphi_1$ is true if and only if a_1 becomes true after a_0 was already true;
(7) if $\varphi_1 = a_1$, then $\varphi = \mathcal{X}\varphi_1$ will be definitely true if and only if a_1 becomes true in the next time step;
(8) if $\varphi_1 = a_1$, then $\varphi = \mathcal{E}\varphi_1$ will be eventually true since a_1 becomes true at some point in the future.

The formulas for φ, φ_1 and φ_2 could be any formula containing the operators in (1) with an arbitrary number of propositions. Although in the previous example we used simple formulas for the sake of clarity, they can be arbitrarily more complex. For instance, $\varphi_2 = \mathcal{E}(a_0 \wedge \mathcal{X}\mathcal{E}(a_1))$ means that eventually a_0 will be true and, afterwards, a_1 will eventually be true. More specifically, the operation $\mathcal{E}(a_0 \wedge \mathcal{X}\mathcal{E}a_1)$ implies that a_0 and $\mathcal{X}\mathcal{E}a_1$ will surely be true in an unknown time in the future. Let us assume, for the sake of example, that a_0 becomes true in time j. The operation $\mathcal{E}a_1$ implies that a_1 will surely be true in an unknown time k in the future and $\mathcal{X}\mathcal{E}a_1$ implies that $k \geq j + 1$. Therefore, eventually a_0 will be true, and, after that, eventually a_1 will be true. The formula φ_2 could be used, for instance, to specify a pick-and-place task that requires two objects to be eventually manipulated in a given order.

In our current work, manipulation tasks must be achieved over a finite time horizon. Hence, we use only co-safe LTL formulas, which are the ones that can be interpreted by considering finite words (Kupferman and Y. Vardi, 2001). Syntactically, co-safe LTL formulas contain only the temporal operators $\mathcal{X}$, $\mathcal{E}$, $\mathcal{U}$, and the negation operator is only allowed over atomic propositions, but not over temporal formulas.

See the works of Kupferman and Y. Vardi (2001); Baier and Katoen (2008) for more formal definitions of the syntax and semantics of LTL. Examples of LTL specifications for manipulation tasks are given in Section 3.2.

2.2 Constrained Kinematic Controller

The constrained controller (Marinho et al., 2019) is based on an optimization problem that minimizes the joint velocities, $\dot{\boldsymbol{q}} \in \mathbb{R}^n$, in the ℓ_2-norm sense while respecting hard constraints, such as obstacles in the workspace, joints limits, etc. The controller is derived from the desired closed-loop task-error dynamics. The differential kinematics, given by $\dot{\boldsymbol{x}} = \boldsymbol{J}\dot{\boldsymbol{q}}$, provides the relation between the joint velocity vector and the task-space velocity vector in which $\boldsymbol{q} \triangleq \boldsymbol{q}(t) \in \mathbb{R}^n$ is the robot configuration vector, $\boldsymbol{x} \triangleq \boldsymbol{x}(\boldsymbol{q}) \in \mathbb{R}^m$ is the task vector, and $\boldsymbol{J} \triangleq \boldsymbol{J}(\boldsymbol{q}) \in \mathbb{R}^{m \times n}$ is the task Jacobian. Given a desired task vector $\boldsymbol{x}_d \in \mathbb{R}^m$, we define the task error $\tilde{\boldsymbol{x}} \triangleq \boldsymbol{x} - \boldsymbol{x}_d$. Considering $\dot{\boldsymbol{x}}_d = \boldsymbol{0}$ for all t, the error dynamics is given by $\dot{\tilde{\boldsymbol{x}}} = \dot{\boldsymbol{x}}$. To drive $\tilde{\boldsymbol{x}}$ to zero with an exponential convergence rate, the closed-loop dynamics is given by $\dot{\tilde{\boldsymbol{x}}} + \eta\tilde{\boldsymbol{x}} = \boldsymbol{0}$ where $\eta \in (0, \infty)$ is the gain. Thus, the desired closed-loop dynamics is given by $\boldsymbol{J}\dot{\boldsymbol{q}} + \eta\tilde{\boldsymbol{x}} = \boldsymbol{0}$ and the control input $\boldsymbol{u}$ that minimizes the joint velocities $\dot{\boldsymbol{q}} \in \mathbb{R}^n$ is obtained as

$$\begin{aligned} \boldsymbol{u} \in \underset{\dot{\boldsymbol{q}}}{\operatorname{argmin}} \quad & \left\| \boldsymbol{J}\dot{\boldsymbol{q}} + \eta\tilde{\boldsymbol{x}} \right\|_2^2 + \lambda \left\| \dot{\boldsymbol{q}} \right\|_2^2 \\ \text{subject to} \quad & \boldsymbol{W}\dot{\boldsymbol{q}} \leq \boldsymbol{w} \end{aligned} \quad (2)$$

where $\lambda \in [0, \infty)$ is a damping factor and $\boldsymbol{W} \in \mathbb{R}^{l \times n}$ and $\boldsymbol{w} \in \mathbb{R}^l$ are used to impose linear constraints in the control inputs. In addition to describing poses, the task vector $\boldsymbol{x}_d$ can also be used to describe geometric primitives (e.g., points, lines, planes) in the workspace. In this sense, the end effector will converge to the specified desired task vector.

Constrained controllers allow the definition of regions of interest, eliminating the need of low-level motion planners, thus saving computational time and resulting in a reactive system. In order to prevent collisions with the workspace, we use the Vector Field Inequalities (VFI) framework (Marinho et al., 2019), which requires distance functions between two collidable entities and the corresponding Jacobian matrices, as better described in Section 3.3.

3. METHODOLOGY

The framework for integration of task planning and motion control that we adapt from the literature is composed of a high-level planning framework and a low-level constrained motion controller. We use the high-level planner proposed by He et al. (2015) and the constrained motion controller proposed by Marinho et al. (2019). Figure 1 shows both steps.

[1] In LTL literature, "eventually" is commonly represented as $\mathcal{F}$. Because in our works $\mathcal{F}$ denotes coordinate systems, we use $\mathcal{E}$ for "eventually".

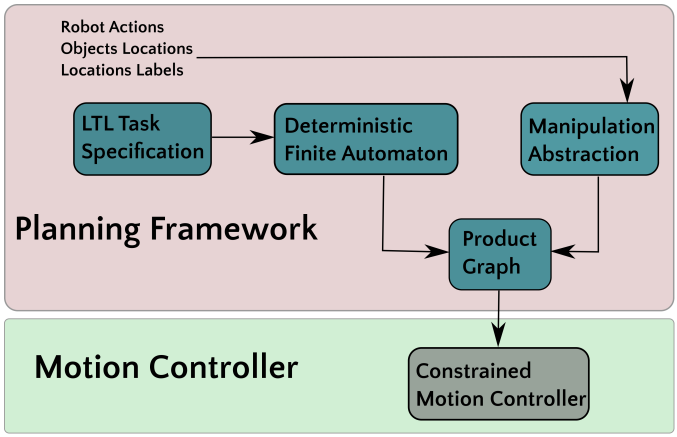


Figure 1. Adapted manipulation framework.

3.1 Problem Definition

Consider a robot in the workspace that can pick-and-place objects between locations. Given a finite set of objects O, a finite set of locations $\mathcal{L}$, and a finite set of actions $\mathcal{M}$, find a sequence of actions $\alpha_0, \alpha_1, \ldots, \alpha_m \in \mathcal{M}$ that manipulate the objects $o_0, o_1, \ldots, o_n \in O$ between locations $\text{loc}_0, \text{loc}_1, \ldots, \text{loc}_k \in \mathcal{L}$ to satisfy a linear temporal logic (LTL) specification φ. The sequence of actions is obtained by the planning framework that will be presented in Section 3.2 and the actions are executed by the constrained motion controller presented in Section 2.2 with the constraints that will be presented in Section 3.3.

3.2 Planning Framework

The planning framework of He et al. (2015) can be divided into the four steps depicted in Figure 1, which are described next.

Linear Temporal Logic Task Specification In the first step, a manipulation task φ is specified using co-safe LTL and it depends only on the objects and the locations. For instance, in a pick-and-place task we can specify where each object must be at the end without mentioning anything about the robot. Therefore, the propositions of the LTL formulas are defined as (o_i, l_j), which means that "object o_i is in location with the label l_j" (He et al., 2015).

Given a scene with a finite set of objects O, a finite set of labels Γ, and a finite set of locations $\mathcal{L}$, with a function $\mathfrak{L}$ that maps locations to labels, the atomic propositions of this scene are elements of $O \times \Gamma$. For instance, first, suppose a robot has to place an object $o_1 \in O$ at the location with label $l_1 \in \Gamma$. The specification for this task can be given by $\varphi_1 = \mathcal{E}(o_1, l_1)$, which means that "eventually object 1 is in location 1". For the sake of clarity, let us define $p_i = (o_i, l_i) \in O \times \Gamma$. We specify a second task "Keep object 1 at location 1 until object 2 and object 3 are sequentially placed at their locations", which is defined as $\varphi_2 = p_1 \mathcal{U} (p_2 \wedge \mathcal{X} p_3)$. More specifically, p_1 will be true until $(p_2 \wedge \mathcal{X} p_3)$ becomes true. If p_2 becomes true in time k, then $\mathcal{X} p_3$ will be true in time $k+1$. Therefore p_1 will be true, at least, until time k. If we had multiple objects to be sorted, we would define $\mathcal{E} p_1 \wedge \mathcal{E} p_2 \wedge \mathcal{E} p_3 \wedge \cdots \wedge \mathcal{E} p_n$ that reads "eventually place object 1 in location 1 and so on".

Manipulation Abstraction The next step is to define an abstraction graph $\mathcal{R} = (V, E, L)$ that captures all the ways the robot can manipulate the objects, in which the set of nodes V consists of the Cartesian product between the set $\mathcal{M}$ of motion primitives, the set $\mathcal{L}$ of locations, the set O of objects, and a set $O_\mathcal{L}$ of objects locations. The set E contains the edges of $\mathcal{R}$ and L is a labeling function that is defined later in this section when we introduce the product graph. The set of motion primitives contains the robot actions needed for pick-and-place tasks and is defined as $\mathcal{M} \triangleq \{\text{GRASP}, \text{PLACE}, \text{HOLD}, \text{MOVE}\}$. The motion graph $\mathscr{M} \triangleq (\mathcal{M}, E_\mathscr{M})$, with $E_\mathscr{M} \subseteq \{(e_i, e_j) \in \mathcal{M} \times \mathcal{M}\}$, which is shown in Figure 2a, defines the allowed sequence of motions.

The location set $\mathcal{L} \triangleq \{\text{loc}_1, ..., \text{loc}_k, \text{loc}_{\text{inter}}\}$ represents the locations of interest in the robot workspace where objects can be placed. Also, when an object is in the gripper, it is considered to be in the intermediate location $\text{loc}_{\text{inter}}$. Furthermore, the location graph $\mathscr{L} \triangleq (\mathcal{L}, E_\mathscr{L})$ with $E_\mathscr{L} \subseteq \{(e_i, e_j) \in \mathcal{L} \times \mathcal{L}\}$, which is shown in Figure 2b, represents how objects can be transferred between locations.

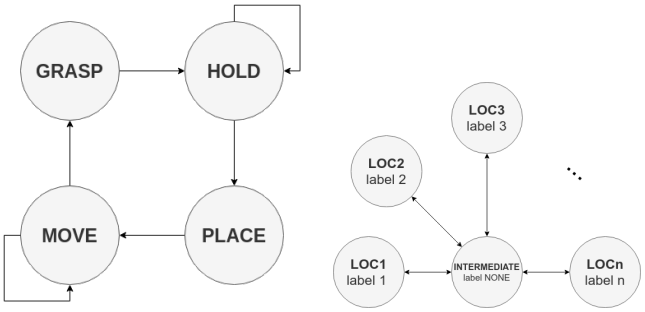
The set of objects is given by $O = \{o_1, ..., o_n\}$ and the set containing all the possibilities for objects locations is given by $O_\mathcal{L} = \{(\bar{o}_{1,\text{loc}}, \ldots, \bar{o}_{n,\text{loc}}) \in \mathcal{L}^n : i \neq j \iff \bar{o}_{i,\text{loc}} \neq \bar{o}_{j,\text{loc}}\}$, where $\bar{o}_{i,\text{loc}}$ is the location of the ith object. Finally, all sets are combined to obtain the set of abstraction nodes V by doing the Cartesian product $V = \mathcal{M} \times \mathcal{L} \times \{O \cup \{\varnothing\}\} \times O_\mathcal{L}$. Therefore, each node in V is given by the tuple $V \ni v = (\alpha, \text{loc}, o, \mathfrak{o}_\mathcal{L})$, where $\alpha \in \mathcal{M}$ is an action, $\text{loc} \in \mathcal{L}$ is the end-effector location, o is the gripper object and $\mathfrak{o}_\mathcal{L} \in O_\mathcal{L}$ is the tuple that indicates the location of each object in the world.

The construction rules of the set of edges $E \subseteq \{(e_i, e_j) \in V \times V\}$ that connect the nodes V of $\mathcal{R}$ can be found in the work of He et al. (2015). Generally speaking, at least each edge $(v, v') \in E$ in the abstraction must satisfy that $(\alpha, \alpha') \in E_\mathscr{M}$. For example, assume two nodes $v = (\text{HOLD}, \text{loc}, o, \mathfrak{o}_\mathcal{L})$ and $v' = (\text{PLACE}, \text{loc}', o', \mathfrak{o}_\mathcal{L})$. There will be an edge transition between them if $\text{loc} = \text{loc}' \neq \text{loc}_{\text{inter}}$, meaning that the robot will be holding the object at one of the k locations in v and placing it at that location in v'. On the other hand, if the action was MOVE in v', there would be no edge between v and v' because there is no edge connecting HOLD to MOVE in $\mathscr{M}$.

Invalid nodes may be generated by the Cartesian product (He et al., 2015). For instance, it is not possible that object 1 be at the gripper while at location 1. Therefore, in our work, the set of reachable nodes is incrementally constructed by starting from an initial node $v_0 \in V$ and following the construction rules of E that only point to valid nodes.

Deterministic Finite Automaton After constructing the abstraction graph $\mathcal{R}$, the formula φ is converted into a deterministic finite automaton (DFA) $\mathcal{A}_\varphi$ (Duret-Lutz and Poitrenaud, 2004) that specifies all the ways the robot can execute the task while fulfilling the formula φ (He et al., 2015).

The DFA is defined as $\mathcal{A}_\varphi = (Z, z_0, \Sigma, \delta, F)$, where Z is the finite set of states, z_0 is the initial state, Σ is the



(a) Motion graph $\mathscr{M}$. (b) Locations graph $\mathscr{L}$.

Figure 2. Abstraction graphs.

set of events that causes transitions in the automaton, $\delta : Z \times \Sigma \to Z$ is the transition function, and F is the set of final states (also known as accepting or marked states). The set of events Σ is also the alphabet of the LTL formula, that is, $\Sigma = 2^{\mathbb{A}}$. The automaton begins in the initial state z_0 and when an event represented by the letters from Σ occurs, there is a transition to the next state following the transitions in δ. This happens until a state in F is reached. The transitions between each state are represented by letters and the path that leads to a state in F represents the sequence of truth assignments of the propositions that satisfy the specification (He et al., 2015). In other words, a sequence of letters represents a word accepted by the automaton.

For example, consider the automaton shown in Figure 3,[2] with $\mathbb{A} = \{p_0, p_1, p_2\}$, where $p_i = (o, l_i)$ and $i \in \{0, 1, 2\}$. Therefore, $\Sigma = \{\varnothing, \{p_0\}, \{p_1\}, \{p_2\}, \{p_0, p_1\}, \{p_1, p_2\}, \{p_0, p_2\}, \{p_0, p_1, p_2\}\}$. Given $A_i = \{p_i\}$, an example of accepted word is $\sigma = A_0 A_1 A_2$. The letter $A_0 = \{p_0\}$ means that object o will be placed at location with label l_0, which causes the transition from state 3 to state 2 because the letter $\{p_0\}$ is equivalent to the logical condition $p_0 \wedge \neg p_1 \wedge \neg p_2$. Similarly, letters A_1 and A_2 cause the transitions to state 1 and 0, respectively. Again, the letter $\{p_1\}$ is equivalent to the logical condition $\neg p_0 \wedge p_1 \wedge \neg p_2$ and $\{p_2\}$ is equivalent to $\neg p_0 \wedge \neg p_1 \wedge p_2$. Therefore, the object o will be in locations with labels l_0, l_1, and l_2 in this sequence, but not simultaneously.

Nonetheless, there may be words or paths that do not respect the physical world. For instance, one transition may require that an object be at multiple locations at the same time, that is, $(o, l_j) \wedge (o, l_k)$. This is exemplified in Figure 3 by the one-letter word $\sigma = \{p_0, p_1, p_2\}$, which causes the transition from state 3 to the final state 0. This transition means that the object will be simultaneously in locations with labels l_0, l_1, and l_2.

Product Graph Last, the nodes in the set V of the abstraction graph $\mathcal{R}$ are combined with the states in the set Z of the automaton $\mathcal{A}_\varphi$ into a product graph $\mathcal{P} \triangleq (V_\mathcal{P}, E_\mathcal{P})$, where $V_\mathcal{P} = V \times Z$ and $E_\mathcal{P} \subseteq \{(e_i, e_j) \in V_\mathcal{P} \times V_\mathcal{P}\}$. This combination represents how the robot can move the objects to achieve the specified task (He et al., 2015). There is an edge $(\mathfrak{p}, \mathfrak{p}') \in E_\mathcal{P}$ from $\mathfrak{p} = (v, z)$ to $\mathfrak{p}' = (v', z')$, where $\mathfrak{p}, \mathfrak{p}' \in V_\mathcal{P}$, if and only if there exists an

[2] Generated at https://spot.lrde.epita.fr/app/ and then adapted.

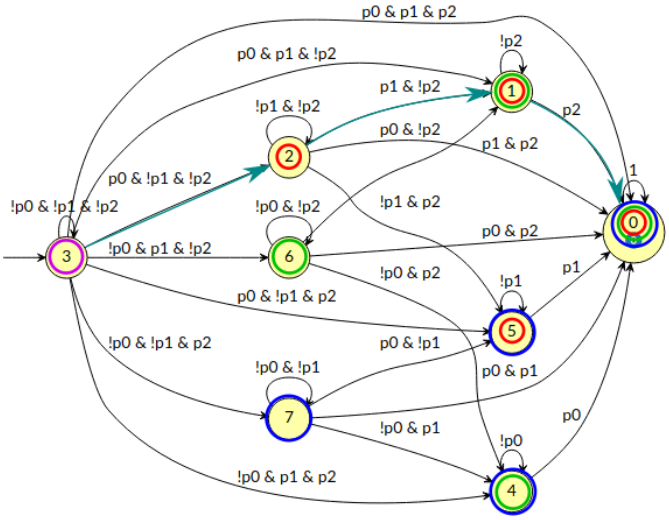


Figure 3. Automaton $\mathcal{A}_\varphi$ obtained from LTL specification $\varphi = \mathcal{E}p_0 \wedge \mathcal{E}p_1 \wedge \mathcal{E}p2$. State 3 is the initial state and 0 is the final state. In this diagram the negation operator is given by ! and the Boolean operator "and" is given by &. Recall that $p_i = (o, l_i)$ means that object o is in location with label l_i. States marked *red*, *green*, *blue*, and *magenta* represents the object in locations with labels l_0, l_1, and l_2 and l_{inter}, respectively. The *cyan* arrows represents an accepted path for an accepted word $\sigma = A_0 A_1 A_2$. States marked with multiple colors mean that the object can be at one of the locations indicated. For instance, in state 1 the object may be in location with label l_0 or l_1 depending on the path taken.

edge $(v, v') \in E$ and $\delta(z, \Pi(L(v'))) = z'$, with $L(v)$ being the labeling function

$$L(v) \triangleq \{(p, \mathfrak{P}(v, p)) \in \mathbb{A} \times \{\mathsf{T}, \mathsf{F}\} : \mathfrak{P}(v, p) = \mathsf{T}\},$$

where

$$\mathfrak{P}(v, p) \triangleq \begin{cases} \mathsf{T}, & \text{if } \mathfrak{L} \circ \mathfrak{F}(\mathfrak{o}_\mathcal{L}, o_p) = l_p, \\ \mathsf{F}, & \text{otherwise.} \end{cases}$$

The function $\mathfrak{P}(v, p)$ determines if the object label l_p in $p = (o_p, l_p)$ matches the label of the corresponding object in the tuple $\mathfrak{o}_\mathcal{L}$ of $v = (\alpha, \text{loc}, o, \mathfrak{o}_\mathcal{L})$. Furthermore, $\mathfrak{L} : \mathcal{L} \to \Gamma$ provides the corresponding label of a given location and $\mathfrak{F} : O^{|O|} \times O$ returns the object location in $\mathfrak{o}_\mathcal{L}$ corresponding to o_p. Lastly, $\Pi(L) \triangleq \{p : (p, \mathsf{T}) \in L\}$ generates the letter that causes the transition in the automaton.

For example, consider $O = \{o_1, o_2\}$, $\Gamma = \{l_1, l_2\}$, $\mathcal{L} = \{\text{loc}_1, \text{loc}_2, \text{loc}_3\}$, $\mathbb{A} = \{p_1, p_2, p_3\}$ with $p_1 = (o_1, l_1)$, $p_2 = (o_2, l_1)$, and $p_3 = (o_2, l_2)$; also, $\mathfrak{L}(\text{loc}_1) = l_1$, $\mathfrak{L}(\text{loc}_2) = l_1$, and $\mathfrak{L}(\text{loc}_3) = l_2$; lastly $v = (\text{MOVE}, \text{loc}_{\text{inter}}, \varnothing, \mathfrak{o}_\mathcal{L})$, with $\mathfrak{o}_\mathcal{L} = (\text{loc}_1, \text{loc}_2)$. In this case,

$$L(v) = \{(p_1, \mathsf{T}), (p_2, \mathsf{T}), (p_3, \mathsf{F})\}$$

because the label of $p_1 = (o_1, l_1)$ and the location label of o_1 (i.e., $\mathfrak{L}(\text{loc}_1)$) are l_1; hence, (p_1, T). Similarly, the label of $p_2 = (o_2, l_1)$ and the location label of o_2 (i.e., $\mathfrak{L}(\text{loc}_2)$) are l_1; therefore, (p_2, T). The label of $p_3 = (o_2, l_2)$ is different from the location label of o_2; thus, (p_3, F). Furthermore, the corresponding letter is $\Pi(L_{A_i}(v)) = \{p_1, p_2\}$.

As a result from the building process of $\mathcal{P}$, a path from the start node $\mathfrak{p}_0 = (v_0, z_0)$ to an accepting node $\mathfrak{p}_F = (v_F, z_F)$—which consists of a node in which z_F is an accepting state—on $\mathcal{P}$ induces a path on $\mathcal{R}$ and a run on $\mathcal{A}_\varphi$. Furthermore, the path on $\mathcal{R}$ considers that each object will be at exactly one location, at a given instant, and that objects can be moved only by the manipulator (He et al., 2015). Finally, each node $(v, z) \in \mathcal{P}$ contains the information v, which indicates what action the robot must do, where the end-effector should be located, the object that should be on the gripper, and where the objects should be placed in the world, and the state z that indicates the current state of the automaton. Moreover, if an accepting state is reached, then the task plan executes the task satisfying the specification. Therefore, a path is searched from the start node to the accepting node. For instance, assume that $(v, v') \in E$ with $v = (\text{HOLD}, \text{loc}_1, o_i, \mathfrak{o}_\mathcal{L}\}$ and $v' = (\text{HOLD}, \text{loc}_\text{inter}, o_i, \mathfrak{o}'_\mathcal{L}\}$. This means that object i is on the gripper, which will move from loc_1 to loc_inter and $\mathfrak{o}_\mathcal{L} = (\dots, \text{loc}_\text{inter}, \dots)$ and $\mathfrak{o}_\mathcal{L}{}' = (\dots, \text{loc}_\text{inter}, \dots)$, since the object begins in the gripper and stays in the gripper.

Since there may be more than one solution in $\mathcal{P}$, the Dijkstra's algorithm is used to search for the shortest accepting path on the product graph $\mathcal{P}$, that is, the shortest path from an initial node to an accepting one. In our work, differently from the work of He et al. (2015), the high-level plan is executed by the constrained motion controller (Marinho et al., 2019) defined by (2). More specifically, the controller is used to execute the actions of type HOLD and MOVE. The actions GRASP and PLACE are executed by closing and opening the end-effector gripper, respectively.

3.3 Constrained workspace manipulation

We use three planes in the environment to prevent end-effector collisions with two walls and the table in the workspace and four lateral planes to constrain the end effector inside a region of interest—the relaxed task region—as shown in Figure 4. Similarly to the work of Quiroz-Omana and Adorno (2019), these seven constraints can be written as

$$-\boldsymbol{J}_{p,n_{\underline{\boldsymbol{\pi}}_i}} \dot{\boldsymbol{q}} \leq \eta \tilde{d}_{p,n_{\underline{\boldsymbol{\pi}}_i}}, \tag{3}$$

where $i \in \{1, 2, ..., 7\}$ and $\tilde{d}_{p,n_{\underline{\boldsymbol{\pi}}_i}} = d_{p,n_{\underline{\boldsymbol{\pi}}_i}} - d_{\pi,\text{safe}}$, with $d_{\pi,\text{safe}}$ being the safe distance to each plane and $d_{p,n_{\underline{\boldsymbol{\pi}}_i}}$ and $\boldsymbol{J}_{p,n_{\underline{\boldsymbol{\pi}}_i}}$ are the point-static-plane distance and its Jacobian, respectively (Marinho et al., 2019). The error vector in the control law (2) is given by $\tilde{\boldsymbol{x}} \triangleq d_{\text{eff},n_{\underline{\boldsymbol{\pi}}}}$, where $d_{\text{eff},n_{\underline{\boldsymbol{\pi}}}}$ is the (signed) distance from the end-effector to the horizontal plane passing through the regions of interest shown in Figure 5. In addition to the plane constraints, we use joints velocities constraints to prevent saturation of actuators (Quiroz-Omaña and Adorno, 2018). Moreover, to prevent end effector collisions with non-manipulated objects while a given object is being manipulated, we add semi-infinite cylindrical constraints to each non-manipulated object (see Figure 4). Therefore, each object is constrained by a cylinder cut by a plane. Given k objects, to each one of them is associated a cylindrical and a plane constraint, which yields the following inequalities:

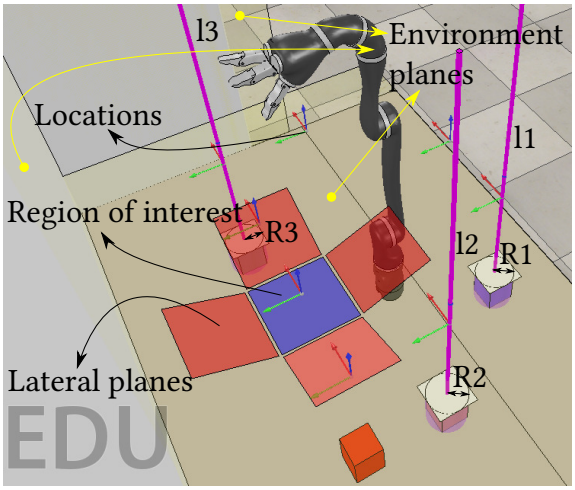


Figure 4. The *red* cuboid is the object to be manipulated, the other objects are already in their region of interest. The semi-infinite cylinders to prevent collisions with the non-manipulated objects are represented by the *light shaded purple* cylinder, with *pink* centerlines, around each object and the *white* planes that cut each cylinder. The coordinate frames indicate locations in the scene.

$$-\boldsymbol{J}_{\text{semi},j} \dot{\boldsymbol{q}} \leq \eta \tilde{d}_{p,\text{object}_j}, \tag{4}$$

$$-\boldsymbol{J}_{p,n_{\underline{\boldsymbol{\pi}}_j}} \dot{\boldsymbol{q}} \leq \eta d_{p,n_{\underline{\boldsymbol{\pi}}_j}}, \tag{5}$$

where $j \in \{1, \dots, k\}$, and $\tilde{d}_{p,\text{object}_j} = d_{p,\text{l}_j} - R_j$, with R_j being the radius of the cylinder around the j-th object; d_{p,l_j} and $\boldsymbol{J}_{\text{semi},j} \in \mathbb{R}^{1\times n}$, with n being the number of robot joints, are the point-static-line distance and its Jacobian, respectively (Marinho et al., 2019), and

$$\boldsymbol{J}_{\text{semi},j} = \begin{cases} \boldsymbol{J}_{p,l_j} & \text{if } d_{p,n_{\underline{\boldsymbol{\pi}}_j}} < 0, \\ \boldsymbol{0}^{1\times n} & \text{otherwise.} \end{cases} \tag{6}$$

4. SIMULATION & DISCUSSION

The implementation of the planning framework proposed by He et al. (2015) was done in C++ with the Boost Graph Library [3] and the automata utilities from the Open Motion Planning Library (OMPL). [4] The LTL task is processed using Spot (Duret-Lutz and Poitrenaud, 2004) and we performed simulations on CoppeliaSim [5] using ROS. [6] Furthermore, we used the DQ Robotics library (Adorno and Marinho, 2020) for robot modeling and control and to define the geometrical constraints, and constrained convex optimization was implemented using IBM ILOG CPLEX Optimization Studio. [7]

To test the planning framework, we created a simulation scene on CoppeliaSim where a Kinova JACO robot must execute assistive tasks for a seated person, with limited or no lower limbs mobility, who cannot reach farther objects in the scene. The robot is placed on a table and the

[3] https://www.boost.org/doc/libs/1_71_0/libs/graph/doc/index.html
[4] https://ompl.kavrakilab.org/
[5] http://www.coppeliarobotics.com/
[6] https://www.ros.org/
[7] https://www.ibm.com/products/ilog-cplex-optimization-studio

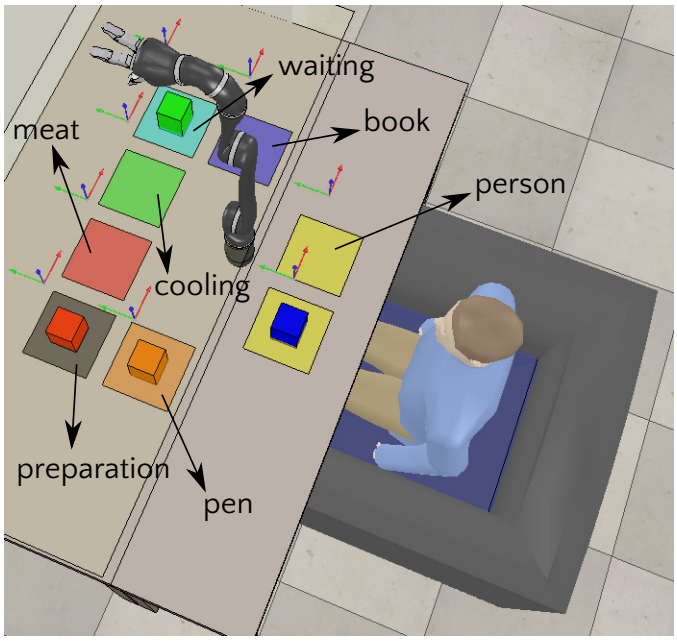


Figure 5. Scene for an LTL specification. The *red*, *green*, *blue* and *orange* cuboids represent the meat, salad, book, and pen, respectively. The regions of interest are indicated in the figure.

person is seated on a chair in front of the table. There are four colored cuboid objects o_{meat}, o_{salad} , o_{book}, o_{pen} representing, meat (red), salad (green), book (blue) and a pen (orange), respectively. In addition, eight colored regions of interest are depicted representing preparation area l_{prep} (dark gray), heating area l_{heat} (red), cooling area l_{cool} (green), waiting area l_{wait} (cyan), book area l_{book} (blue), two person-areas l_{pers} (yellow), and pen area l_{pen} (orange). Initially, the meat is in the preparation area, the salad is in the waiting area, the pen is in the pen area, and the book is in one of the person areas, as shown in Figure 5.

The planning framework generates all the graphs and searches the product graph $\mathcal{P}$ for an accepting path, that is, a task plan. Afterward, the robot iterates over the nodes of the accepting path executing the actions considering the scene locations. During HOLD, the robot holds an object from one location to another, and during MOVE, the robot moves the empty gripper from one location to another. Let us assume the robot is currently at node k with action MOVE and end-effector location loc_i. Assume that at the node $k+1$, the end-effector location is loc_j. Therefore, the end-effector will move from loc_i to loc_j.

Two tasks with increasing complexities were devised. Both tasks require the planner to identify that at least one location is occupied and an object must be removed from one location before continuing the task.

Task 1: "First, heat the meat and serve the salad, next serve the meat."

To express task 1 in co-safe LTL, we define $p_{m,h} = (o_{\text{meat}}, l_{\text{heat}})$, $p_{s,p} = (o_{\text{salad}}, l_{\text{pers}})$ and $p_{m,p} = (o_{\text{meat}}, l_{\text{pers}})$. Hence,

$$\varphi_1 = \mathcal{E}(p_{m,h} \wedge p_{s,p} \wedge \mathcal{X}\mathcal{E}(p_{m,p})). \tag{7}$$

The automaton generated from φ_1 has three states and the planner explores 16975 nodes in the product graph. The total planning time average is 1.65 seconds. Since task 1 is a sequential task, the planner generates a task plan that follows the specified order of manipulation. The formula for this task does not specify what should be done with the book in front of the person. Hence, the planner generates a task plan that heats the meat, then serves the salad and then serves the meat. However, after the salad is served, both locations in front of the person are occupied by the book and the salad. Thus, the planner decides to remove the salad instead of the book. Should both the meat and the salad be served together, it would be necessary to explicitly specify that.

Task 2: "First, move book to the book region. Next, serve the salad and heat the meat in any order. Afterward, serve the meat while the salad is being eaten. Keep the book on the book position during the whole task execution."

For task 2, we define $p_{o,b} = (o_{\text{book}}, l_{\text{book}})$, $p_{s,p} = (o_{\text{salad}}, l_{\text{pers}})$, $p_{m,h} = (o_{\text{meat}}, l_{\text{heat}})$, and $p_{m,p} = (o_{\text{meat}}, l_{\text{pers}})$. As a result φ_2 is given by

$$\begin{aligned}\varphi_2 = \mathcal{E}\Big(& p_{o,b} \wedge \neg p_{s,p} \wedge \neg p_{m,h} \wedge \\ & \mathcal{X}\mathcal{E}\Big(p_{o,b} \wedge \mathcal{E}(p_{s,p}) \wedge \mathcal{E}(p_{m,h}) \wedge \\ & \mathcal{X}\mathcal{E}\Big(p_{o,b} \wedge p_{s,p} \wedge \mathcal{X}\mathcal{E}(p_{s,p} \wedge p_{m,p})\Big)\Big)\Big)\end{aligned} \tag{8}$$

After moving the book to the book region, task 2 gives freedom to the robot to decide if the salad will be served first or, instead, if the meat should be heated first. Next, it will serve the meat while the salad is being eaten. In comparison to task 1, task 2 specifies that the salad and the meat must be placed in front of the person. Otherwise, the robot would be free to remove the salad and only then serve the meat. The automaton generated from φ_2 has eight states and the planner explores 25098 nodes in the product graph. The total planning time average is 2.41 seconds.

Table 1 shows the planning data for the two tasks, which have the same number of objects (four) and locations (eight). The number of states and edges in the DFA indicates the complexity of the task. The planning time T_{planning} with the associated standard deviation (s.d.) corresponds to an average of 50 runs. As expected, an increase in the task complexity increases the planning time because more nodes are explored in the product graph and then Dijkstra runs on a larger graph. In addition, Table 1 also shows the results for two tasks with similar complexities in the original framework of He et al. (2015).

5. CONCLUSION

This work presented an improvement of the manipulation framework proposed by He et al. (2015), which uses sampling-based motion planners as the low-level planner, to use instead a constrained motion controller that allows the definition of regions of interest. Our contribution is that the low-level layer from the original framework,

Table 1. Planning data for φ_1 and φ_2.

Task	$\lvert A_\varphi \rvert$	$\lvert E_{A_\varphi} \rvert$	$\lvert V_\mathcal{P} \rvert$	T_{planning}(s)
Framework with constrained motion controller				
φ_1	3	5	16975	1.65418 (0.152028)
φ_2	8	20	25098	2.40551 (0.203752)
Original framework				
φ_3	2	3	44100	2.76
φ_4	8	27	75511	4.48

$\lvert A_\varphi \rvert$ and $\lvert E_{A_\varphi} \rvert$ are the number of states and edges in the DFA, respectively, $\lvert V_\mathcal{P} \rvert$ is the total number of nodes in the product graph, T_{planning} (s.d.) is the total high-level planning time accounting for the generation of graphs and the Dijkstra search for a path in the product graph $\mathcal{P}$.

composed of a motion planner and a motion controller, is replaced by a single constrained motion controller. Hence, there is no motion planning time added to the total planning time and there is no need to generate more than one high-level plan due to changes in the scene, in terms of the modeled geometric primitives, as long as there are mechanisms to track their changes. As a consequence, there is no increase in the number of generated planning nodes during the task planning phase and the Dijkstra's algorithm searches for a task plan on a static graph. For tasks with similar complexity in the original framework, our approach has greatly reduced the number of generated task plan nodes, which also reduced the planning time. Since we define regions of interest instead of single locations of interest, the number of states in the planner is reduced thanks to a smaller discretization of the workspace. Future works will exploit the use of regions of interest to describe complex tasks using less locations, increasing the computational efficiency and demanding less degrees of freedom for a given motion task.

ACKNOWLEDGEMENTS

This work was supported by the Brazilian funding agencies CAPES and CNPq.

REFERÊNCIAS

Adorno, B.V. and Marinho, M.M. (2020). DQ Robotics: A Library for Robot Modeling and Control. *IEEE Robotics & Automation Magazine*, 0–0.

Baier, C. and Katoen, J.P. (2008). *Principles of model checking*. MIT Press.

Bhatia, A., Maly, M.R., Kavraki, L.E., and Vardi, M.Y. (2011). Motion Planning with Complex Goals. *IEEE Robot. Autom. Mag.*, 18(3), 55–64.

Duret-Lutz, A. and Poitrenaud, D. (2004). SPOT: an extensible model checking library using transition-based generalized buchi automata. In *IEEE Comput. Soc. 12th Annu. Int. Symp. Model. Anal. Simul. Comput. Telecommun. Syst. 2004. (MASCOTS 2004). Proceedings.*, 76–83. IEEE.

Erdem, E., Haspalamutgil, K., Palaz, C., Patoglu, V., and Uras, T. (2011). Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation. In *2011 IEEE Int. Conf. Robot. Autom.*, 4575–4581. IEEE.

Escande, A., Mansard, N., and Wieber, P.B. (2014). Hierarchical quadratic programming: Fast online humanoid-robot motion generation. *Int. J. Rob. Res.*, 33(7), 1006–1028.

Fikes, R.E. and Nilsson, N.J. (1971). Strips: A new approach to the application of theorem proving to problem solving. *Artif. Intell.*, 2(3-4), 189–208.

Goncalves, V.M., Fraisse, P., Crosnier, A., and Adorno, B.V. (2016). Parsimonious Kinematic Control of Highly Redundant Robots. *IEEE Robot. Autom. Lett.*, 1(1), 65–72.

He, K., Lahijanian, M., Kavraki, L.E., and Vardi, M.Y. (2015). Towards manipulation planning with temporal logic specifications. In *2015 IEEE Int. Conf. Robot. Autom.*, 346–352. IEEE.

Kaelbling, L.P. and Lozano-Perez, T. (2011). Hierarchical task and motion planning in the now. In *2011 IEEE Int. Conf. Robot. Autom.*, 1470–1477. IEEE.

Kambhampati, S., Cutkosky, M., Tenenbaum, M., and Hong Lee, S. (1991). Combining Specialized Reasoners and General Purpose Planners: A Case Study. In *Proc. 9th Natl. Conf. Artif. Intell.* Anaheim.

Kloetzer, M. and Belta, C. (2008). A Fully Automated Framework for Control of Linear Systems from Temporal Logic Specifications. *IEEE Trans. Automat. Contr.*, 53(1), 287–297.

Kress-Gazit, H., Fainekos, G.E., and Pappas, G.J. (2007). Where's Waldo? Sensor-Based Temporal Logic Motion Planning. In *Proc. 2007 IEEE Int. Conf. Robot. Autom.*, 3116–3121. IEEE.

Kupferman, O. and Y. Vardi, M. (2001). Model Checking of Safety Properties. *Form. Methods Syst. Des.*, 19(3), 291–314.

Lana, E.P., Adorno, B.V., and Maia, C.A. (2015). A new algebraic approach for the description of robotic manipulation tasks. In *2015 IEEE Int. Conf. Robot. Autom.*, 3083–3088. IEEE.

Laumond, J.P., Mansard, N., and Lasserre, J.B. (2015). Optimization as motion selection principle in robot action. *Commun. ACM*, 58(5), 64–74.

Lozano-Perez, T. and Kaelbling, L.P. (2014). A constraint-based method for solving sequential manipulation planning problems. In *2014 IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, 3684–3691. IEEE.

Marinho, M.M., Adorno, B.V., Harada, K., and Mitsuishi, M. (2019). Dynamic Active Constraints for Surgical Robots Using Vector-Field Inequalities. *IEEE Trans. Robot.*, 35(5), 1166–1185.

Quiroz-Omaña, J.J. and Adorno, B.V. (2018). Whole-Body Kinematic Control of Nonholonomic Mobile Manipulators Using Linear Programming. *J. Intell. Robot. Syst.*, 91(2), 263–278.

Quiroz-Omana, J.J. and Adorno, B.V. (2019). Whole-Body Control With (Self) Collision Avoidance Using Vector Field Inequalities. *IEEE Robot. Autom. Lett.*, 4(4), 4048–4053.

Srivastava, S., Fang, E., Riano, L., Chitnis, R., Russell, S., and Abbeel, P. (2014). Combined task and motion planning through an extensible planner-independent interface layer. In *2014 IEEE Int. Conf. Robot. Autom.*, 639–646. IEEE.

Wongpiromsarn, T., Topcu, U., and Murray, R.M. (2010). Receding horizon control for temporal logic specifications. In *Proc. 13th ACM Int. Conf. Hybrid Syst. Comput. Control - HSCC '10*, 101. ACM Press, New York, New York, USA.