Task and Motion Planning for Manipulator Arms With Metric Temporal Logic Specifications

Sayan Saha and Anak Agung Julius

***Abstract*—The aim is to synthesize control inputs for robotic manipulator arms that ensure a desired task specification is executed while optimizing a desired performance objective. We use metric temporal logic (MTL) to express the task specifications defined in terms of manipulating objects and implemented a hierarchical method combining mixed integer-linear programming (at high-level) to obtain a candidate MTL task specification defined in terms of the location of the arm end-effector and gradient descent based optimization (at low-level) for automatic synthesis of the motion plans to execute such candidate task specifications. The gradient descent algorithm is performed over the feasible input space to optimize the manipulator arm trajectory from a given arbitrary initial condition to perform the task at hand. We demonstrate the efficacy of our method by simulating a task specification on a Baxter robot.**

***Index Terms*—Task planning, manipulation planning, motion and path planning, optimization and optimal control.**

I. Introduction

GIVEN a task specification in terms of moving manipulable objects in the workspace of a manipulator arm, the *task and motion planning* (TMP) problem for the arm deals with finding the control input signals to generate a collision-free trajectory of the arm in the obstacle filled workspace to complete the desired task. Motion planning for manipulator arms has been well researched in the robotics community [1]–[6]. Solving the task planning problem to accomplish the specification can be done independently of the motion planning problem [7], [8]. However, integration of TMP involves a hierarchical formulation of first proposing a candidate sequence of tasks to satisfy the desired task specification, followed by determining if a motion plan exists to achieve the candidate task sequence [9]–[14].

Temporal logics [15] allow us to express complex task specifications, by requiring the systems to satisfy timing constraints for correct behavior. The common controller synthesis approach to satisfy the linear variant of temporal logics, namely *Linear Temporal Logic* (LTL) is to follow a hierarchical procedure of creating a finite abstraction of the dynamical system, followed by synthesizing controllers using automata based techniques [16], [17]. This approach, however, results in high computational complexity due to the quantization of the finite abstraction model. Moreover, the size of the automaton can also be exponential in the length of the specification [18]. Since the state-space of the manipulator arm is high dimensional, this abstraction-based method might become intractable for a fine quantization method. Authors in [12] introduced a novel abstraction technique to obtain a coarse representation of all possible motions of the manipulator arm in its workspace to alleviate the state-space explosion problem. Iterative sampling-based approaches for path planning to satisfy temporal logic specifications have also been explored [19], [20]. However, these methods are not applied for manipulator arms with high dimensional state-space yet.

In this letter, we express the task specifications for manipulating objects in the workspace of the robot arm using *Metric Temporal Logic* (MTL) formulae, which extends LTL by augmenting the temporal operators (see Section II-D) with a time interval [21]. We propose a hierarchical control synthesis framework, where at the top level a Mixed Integer-Linear Program (MILP) is solved to find a candidate low-level MTL task specification describing how the end-effector of the robot arm moves in the workspace, based on a *knowledge map* of the workspace. At the lower level, we propose a method for computing descent (ascent) directions for minimizing (maximizing) a cost function while satisfying the candidate task specification. For this process, we use the ideas in [22], [23], where the desired MTL specifications are considered as constraints to an optimal control problem. This process of iteratively changing the motion plan of the manipulator arm in each step of the optimization algorithm, produces a optimal motion plan resulting in moving the arm through a feasible task sequence to satisfy the candidate task specification. If, however, the candidate specification is not satisfiable then the knowledge map is modified based on the cause of unsatisfiability of the specification and a new candidate specification is proposed. Thus, the low-level motion planner guides the high-level MILP problem to solve the task plan, similar to the approaches in [12]–[14]. The work presented in [11] is similar to ours in the sense that the authors use a SMT solver at a high-level to search over a *placement graph* to find a feasible path satisfying the task plan. At the low-level sampling-based motion planning is employed to create the placement graph representing all possible motions of the manipulator arm between locations of interest in the workspace. However, there is no feedback from the motion planning layer to the SMT solver to aid the search process unlike the proposed approach. Moreover the task specifications

Manuscript received June 26, 2017; accepted September 3, 2017. Date of publication September 21, 2017; date of current version October 5, 2017. This letter was recommended for publication by Associate Editor H. Kurniawati and Editor N. Amato upon evaluation of the reviewers comments. This work was supported by the National Science Foundation under Grants CAREER CNS-0953976, CNS-1218109, and CNS-1618369. *(Corresponding author: Sayan Saha.)*

The authors are with the Department of Electrical, Computer, and Systems Engineering, Rensselaer Polytechnic Institute, Troy, NY 12180 USA (e-mail: sahas3@rpi.edu; julia2@rpi.edu).

This letter has supplementary downloadable material available at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/LRA.2017.2755078

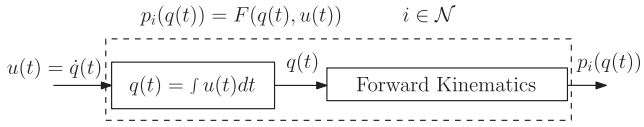

Fig. 1. Block diagram for system dynamics of the manipulator arm.

considered do not consider any temporal operators or timing constraints.

II. Preliminaries

A. System Dynamics Modeling

Consider an open kinematic chain manipulator arm with N rigid links, interconnected by N actuated revolute or prismatic joints. Let $\mathcal{N} = \{1, 2, \cdots, N, T\}$ denote the set of all the links, including the end-effector T. We denote the joint angles by $q(t) \in \mathbb{R}^N$ and the positions of the end-points of each link by $p_i(q) \in \mathbb{R}^3$ for $i \in \mathcal{N}$, where $p_T(q) \in \mathbb{R}^3$ denotes the end-effector position.

We consider the joint angle velocities $\dot{q}(t)$ to be the control input signal $u(t)$ to the manipulator arm system at time t. Let $\mathcal{H}$ denote the finite trajectory duration, which is fixed. We assume that the initial joint angles $q(0) = q^0$ are given. For the ith link, denoted as l_i, we denote the continuous state trajectory (change in link position with time) till time τ as $x_{\tau,i}$. We use $\mathbf{q}_\tau$ and $\mathbf{x}_\tau$ to denote the joint-angle trajectory and the entire trajectory of the arm till time τ, respectively. The control input signal applied to the entire arm till time τ is denoted by $\mathbf{u}_\tau$. Fig. 1 shows the block diagram for the system dynamics evolution of the manipulator arm under consideration.

B. Workspace

We consider a workspace consisting of a set of obstacle locations **Obs**, a set of manipulable objects **Obj**, and a set of object locations **Loc** where the objects can be placed. We assume that each location can hold at most one object and the end-effector can only grasp one object at a time. Let, $\lambda_o(t) \in \mathbf{Loc}$ denote the location of object o at time t.

C. Task Specification

Task specifications considered in this letter are manipulation of objects in **Obj** between locations in **Loc** by the manipulator arm. We assume that motion primitives for *grasp* and *place* are already available. Let Δt_{place} and Δt_{grasp} denote the maximum amount of time required to execute the motion primitives for *place* and *grasp* respectively. An estimate for the values of t_{place} and t_{grasp} can be based on the low-level implementation of the *place* and *grasp* primitives [24], [25]. Then the motion primitives perform the necessary actions if the following conditions hold: 1. If the end-effector is holding an object at an unoccupied location, then it can *place* the object at that location within Δt_{place} seconds; 2. If the end-effector is not holding any object and is at a location containing an object, then it can *grasp* that object within Δt_{grasp} seconds.

One example of a task specification that we consider is for the arm to place an object $o \in \mathbf{Obj}$ at some location $B \in \mathbf{Loc}$, which is currently unoccupied. To do so, the arm needs to pick o from its current location (say, $A \in \mathbf{Loc}$). In addition, the arm also needs to avoid the obstacles **Obs** present in the workspace. In the following, we assume $\Delta t_{\text{place}} = \Delta t_{\text{grasp}} = 0.5$ seconds.

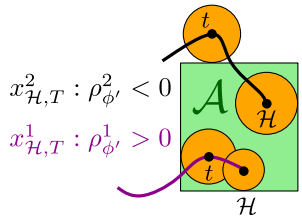

Fig. 2. Illustration of robustness degree for MTL specification [26].

D. Metric Temporal Logic Specification

We express the task specification for the manipulator arm in terms of Metric Temporal Logic (MTL) formulae [21]. We consider the temporal operators *eventually* ($\Diamond_{[\mathcal{I}]}$), *always* ($\Box_{[\mathcal{I}]}$) and *until* ($\mathcal{U}_{[\mathcal{I}]}$), and logical operators, such as, *conjunction* ($\wedge$), *disjunction* ($\vee$), *negation* ($\neg$), and *implication* ($\rightarrow$), that can be used to combine *atomic propositions* to form the MTL formula. We consider atomic propositions in terms of the location of the objects and the position of the end-effector. We associate a set $\mathcal{O}(\pi) \subseteq \mathbb{R}^3$ with an atomic proposition π, such that π is true at time t if and only if $p_T(q) \in \mathcal{O}(\pi)$ at time t.

In terms of MTL formula expressing manipulation of objects, the task specification example presented in Section II-C can be expressed as

$$\phi_o = \Diamond_{[0,\mathcal{H}]}\big(\lambda_o = B\big), \tag{1}$$

which reads "*at some point in the time interval* $[0, \mathcal{H}]$, *object* o *is in location* B". To accomplish ϕ_o, we need to translate this high-level task specification to a *motion specification* that describes the motion of the end-effector. In this case, the arm needs to grasp the object from location A, stay there for $\Delta t_{\text{grasp}} = 0.5$ seconds to grasp the object, and then within some time, say t_1, move to location B and place the object. Let π_A and π_B represent the 3D subsets corresponding to the locations A and B respectively. This specification can be expressed using an MTL formula in terms of end-effector position as

$$\phi^c = \Diamond_{[0,\mathcal{H}]}\big(\Box_{[0,0.5]}(p_T \in \pi_A) \wedge \Diamond_{[0,t_1]}\Box_{[0,0.5]}(p_T \in \pi_B)\big). \tag{2}$$

Utilizing the concept of robust satisfaction of MTL formulae, introduced in [21], we define the robustness radius ρ_ϕ for a system trajectory for an MTL formula ϕ as a measure of how robustly the system trajectory satisfies (falsifies) the MTL formula ϕ. This measure is positive for trajectories satisfying the MTL formula and negative otherwise. For the rest of the letter, we will denote a system trajectory satisfying the desired MTL formula to be a *feasible* system trajectory. The concept of (in)feasible system trajectories is explained using the Fig. 2 [26]. We consider the MTL specification

$$\phi' = \Box_{[t,\mathcal{H}]}(p_T \in \mathcal{A}),$$

which states that between times t and $\mathcal{H}$ (two consecutive time steps) the end-effector position should be inside set $\mathcal{A}$. Of the two trajectories shown in Fig. 2, $x^1_{\mathcal{H},T}$ is a feasible trajectory and $x^2_{\mathcal{H},T}$ is an infeasible trajectory. The trajectory $x^1_{\mathcal{H},T}$ has a positive robustness, denoted by $\rho^1_{\phi'}$, which is the distance between the trajectory at time $\mathcal{H}$ and the predicate $\mathcal{A}$. We call this time $\mathcal{H}$ the *critical time* and the predicate $\mathcal{A}$ the *critical predicate*. Since the trajectory at the critical time is closest to falsifying

the MTL formula, if the *critical point* is moved by an amount greater than $\rho^1_{\phi'}$, we can push it outside of $\mathcal{A}$ in order to falsify the specification. The other trajectory, $x^2_{\mathcal{H},T}$, has a negative robustness denoted by $\rho^2_{\phi'}$, which is the distance between the trajectory at time t and the predicate $\mathcal{A}$. This trajectory is farthest from satisfying the MTL formula at the critical time, and needs to be moved by at least $\rho^2_{\phi'}$ in order to try to satisfy the MTL formula. Given an end-effector trajectory $x_{\mathcal{H},T}$ and a desired MTL formula ϕ, we employ the `TaLiRo` tool [27] to compute the robustness radius, critical time, and critical predicate.

III. Task and Motion Planning

In order to achieve the MTL task specification ϕ_o defined in terms of the manipulable objects, we first find a candidate task specification ϕ^c defined explicitly in terms of the end-effector location by solving an MILP problem (high-level task planning). We then employ a gradient-descent based search in the control input space to satisfy ϕ^c (low-level motion planning problem). If ϕ^c is not satisfiable, we update the MILP problem to address the reason of unsatisfiability of ϕ^c and generate a new candidate task specification ϕ^c.

A. Generate Candidate MTL Task Specification

At the high-level, we are interested in solving the problem:

Problem 1. Given a desired task specification expressed by an MTL formula ϕ_o specifying manipulation of objects in **Obj** within the locations in **Loc**, find the location trajectories of the objects ($\lambda_{o,\mathcal{H}}, \forall o \in \mathbf{Obj}$), such that ϕ_o is satisfied.

To solve this problem, we formulate an MILP problem describing the evolution of positions of the objects as a discrete-time system, with a time-step Δt. We first consider that the manipulator arm moves over a location graph $\mathcal{L} = (\mathbf{Loc}, E)$, consisting of $L+1$ nodes corresponding to the L object locations in **Loc** and the initial location of the end-effector. Here we assume that the end-effector is not initialized to be at any of the locations in **Loc**. We also assume that the graph $\mathcal{L}$ is fully connected, and therefore has $M \triangleq 2\binom{L+1}{2}$ directed edges in E. We use and update a *knowledge map* of the workspace as we explore it. The knowledge map captures the number of time steps needed for the end-effector to traverse the edges of the graph. For an edge $j \in E$, we denote this quantity as δ_j.

We denote the presence of the end-effector at the node l at time-index k (corresponding to time $t = k\Delta t$) by $z_\ell(k) \in \{0,1\}$ taking the value `1` if the end-effector is present at the node l at time-index k, otherwise `0`. Denoting the motion of the end-effector through edge j at time-index k as $v_j(k) \in \{0,1\}$ taking the value `1` if the end-effector moves along edge j at time-index k and `0` otherwise, we can write

$$\begin{aligned} z_0(k) &= z_0(k-1) + \sum_{j\in \text{In}(0)} v_j(k-\delta_j) - \sum_{j\in \text{Out}(0)} v_j(k), \\ &\vdots \\ z_L(k) &= z_L(k-1) + \sum_{j\in \text{In}(L)} v_j(k-\delta_j) - \sum_{j\in \text{Out}(L)} v_j(k), \end{aligned} \tag{3}$$

where, we respectively denote the set of all edges that enter and come out of node l as $\text{In}(l)$ and $\text{Out}(l)$. (3) is initialized at $k=0$ using the initial position of the end-effector. The end-effector can only traverse edge j at time-index k if it starts at the origin node of edge j, denoted as j_s. Therefore we have the constraint

$$\forall j \in E,\ \forall k \geq 0,\ v_j(k) \leq z_{j_s}(k). \tag{4}$$

To ensure that there is sufficient time to execute the motion primitives (*grasp* or *place*) we add the constraints for any edge $j \in E$,

$$v_j(k) = 1 \Rightarrow \forall j, v_j(\bar{k}) = 0, \bar{k} \in \left[k+1, k+\delta_j + \left\lceil \frac{0.5}{\Delta t} \right\rceil\right] \tag{5}$$

implying once the end-effector reaches a certain node via the edge j, it stays there for at least 0.5 seconds to execute the desired motion primitive.

The locations of the objects are given by the Boolean/binary variables $P_{o,\ell}(k)$, where $P_{o,\ell}(k) = 1 = \texttt{true}$ if object o is at node ℓ at time-index k. Otherwise, $P_{o,\ell}(k) = 0 = \texttt{false}$. We also define the variables $V_{o,j}(k) \in \{0,1\}$, where $V_{o,j}(k)$ is `1` if the object o is moved across edge j at time-index k. We then formulate the following constraints $\forall \ell \in \mathbf{Loc}, o \in \mathbf{Obj}, k \geq 0$,

$$\begin{aligned} (P_{o,\ell}(k) \wedge P_{o,\ell}(k+1)) &\Rightarrow \bigwedge_{j\in \text{Out}(l)} (V_{o,j}(k) < 1), \\ (P_{o,\ell}(k) \wedge \neg P_{o,\ell}(k+1)) &\Rightarrow \bigvee_{j\in \text{Out}(l)} (V_{o,j}(k) > 0), \\ (\neg P_{o,\ell}(k) \wedge P_{o,\ell}(k+1)) &\Rightarrow \bigvee_{j\in \text{In}(l)} ((V_{o,j}(k) > 0) \\ &\qquad \wedge P_{o,j_s}(k-\delta_j)), \\ (\neg P_{o,\ell}(k) \wedge \neg P_{o,\ell}(k+1)) &\Rightarrow \bigwedge_{j\in \text{In}(l)} (V_{\ell,j}(k) < 1). \end{aligned} \tag{6}$$

Note that, since the end-effector can only grasp one object at a time,

$$\forall k \geq 0, \forall j \in E,\ v_j(k) \geq \sum_{o\in \mathbf{Obj}} V_{o,j}(k). \tag{7}$$

Additionally, since each location can hold at most one object, we have

$$\forall k \geq 0, \forall \ell \in \mathbf{Loc},\ \sum_{o\in \mathbf{Obj}} P_{o,\ell}(k) \leq 1. \tag{8}$$

The location of an object o at time $t \in [k\Delta t, (k+1)\Delta t]$ is given by:

$$\lambda_o(t) = \begin{cases} \ell, \text{ for } P_{o,\ell}(k) = 1, \\ 0, \text{ for } \sum_l P_{o,l}(k) = 0. \end{cases}$$

$\lambda_o(t) = 0$ implies that the object is grasped by the end-effector and is being moved between locations. We also minimize the total number of times the objects are manipulated by minimizing $\Sigma_{o,j,k} V_{o,j}(k)$ over the entire trajectory duration.

Finally we follow the approach detailed in our earlier work [26] to generate additional constraints such that the location trajectories of the objects $\lambda_{o,\mathcal{H}}$ satisfy the task specification ϕ_o.

If for satisfaction of ϕ_o, the object o is supposed to be at location ℓ at time-index k then we add constraint $P_{o,\ell}(k) = 1$. Conversely, if object o should not be at location ℓ then the constraint $P_{o,\ell}(k) = 0$ is added. Solving the MILP problem provides a candidate solution $z^c(k)$, $0 \leq k\Delta t \leq \mathcal{H}$ indicating the time intervals during which the end-effector is at any particular object location to pick or place the objects to satisfy ϕ_o. We utilized YALMIP [28] with the GUROBI solver to solve the MILP problem in MATLAB. Using this candidate solution we obtain a candidate motion specification ϕ^c defined in terms of end-effector location of the form (2). Observing that the end-effector can only sequentially visit the different object locations and is required to stay at that location for at least 0.5 seconds so that either the *grasp* or *place* primitive can be implemented, ϕ^c is of the form

$$\phi^c = \underbrace{\Diamond_{[t_1,t_2]}\Box_{[0,0.5]}(p_T \in \pi_1)}_{\phi_1^c} \wedge \ldots \wedge \underbrace{\Diamond_{[t_{2n-1},t_{2n}]}\Box_{[0,0.5]}(p_T \in \pi_n)}_{\phi_n^c}, \tag{9}$$

where $t_1 \leq t_2 < \cdots < t_{2n-1} \leq t_{2n} = \mathcal{H}$ and π_n is the predicate corresponding to the n-th location (node) to be reached by the end-effector.

We employ Algorithm 2 (detailed later) to find the control signal $\mathbf{u}$ to satisfy ϕ^c. Since our knowledge map (i.e., the time-steps $\delta_j, j \in E$) may not be accurate, ϕ^c may not be implementable. As we solve for the sequential motion of the end-effector from one node to another, if any ϕ_i^c, $i = 1, 2, \cdots, n$ becomes infeasible, we update the knowledge map by correcting the value of δ_j following Algorithm 1 in steps 19 to 24 and an updated MILP is solved. The knowledge map for an edge $\bar{j}$ is updated only if the corresponding joint-angle trajectory $Q_{\bar{j}}$ has not been determined yet or if a better trajectory (in terms of lesser time) is found for traversing the edge $\bar{j}$. If for the new motion specification ϕ^c, the end-effector is required to move along an edge $\bar{j}$ for which the value of joint-angle trajectory $Q_{\bar{j}}$ is already determined, then we can use Algorithm 2 initialized with that joint-angle trajectory $Q_{\bar{j}}$ to aid the local optimization process, thereby reducing time to find a trajectory satisfying ϕ_i^c. Incorrect value of some δ_j might also lead to an infeasible MILP, implying the timing constraints on the manipulation of the objects specified by the MTL task specification can not be satisfied. In such scenarios, the knowledge map is updated as in step 29 by halving all the δ_j, $j \in E$ and re-initializing their joint-angle trajectory to be empty. The MILP is updated accordingly and solved in step 30. This results in producing a new candidate solution ϕ^c comprising of possibly new edges to traverse to satisfy the given MTL task specification.

Algorithm 1: High-level task planning.

1: Input : ϕ_o, $\mathcal{H}$.
2: Initialize joint-angle trajectory as end-effector moves along edge j, denoted by $Q_j = [\,]$; solved = `False`.
3: Formulate and solve an MILP representing ϕ^o to obtain candidate specification ϕ^c.
4: **while** solved is `False` **do**
5: Initialize count $= 0$.
6: **while** MILP is feasible **do**
7: **for** each ϕ_i^c component of ϕ^c **do**
8: Use Alg. 2 to obtain control inputs satisfying ϕ_i^c, representing moving along edge $\bar{j}$ from node $\bar{i}$ to $\bar{i}$.
9: **if** ϕ_i^c is feasible **then**
10: Increment count by 1 and store $u_{\bar{j}}^*$.
11: **else**
12: break
13: **end if**
14: **end for**
15: **if** count $==$ n: number of components in ϕ^c **then**
16: Solved = `True`. (ϕ_o is feasible.)
17: Return $\mathbf{u}_\mathcal{H}$ by combining all the component control input signals $u_{\bar{j}}^*$.
18: **else**
19: **for** each ϕ_i^c component of ϕ^c **do**
20: Use bisection search with Alg. 2 to find the minimum time $D_{\min}$ to traverse edge $\bar{j}$ and corresponding joint-angle trajectory $Q_{\bar{j}}^*$.
21: **if** $Q_{\bar{j}}$ is empty **or** $\lceil \frac{D_{\min}}{\Delta t} \rceil \leq \delta_{\bar{j}}$ **then**
22: Update map: $\delta_{\bar{j}} = \lceil \frac{D_{\min}}{\Delta t} \rceil$, $Q_{\bar{j}} = Q_{\bar{j}}^*$.
23: **end if**
24: **end for**
25: **end if**
26: Reset count $= 0$.
27: Update MILP of ϕ^o using new knowledge map and resolve the MILP.
28: **end while**
29: Update knowledge map: $\delta_j = \max\big(1, \frac{\delta_j}{2}\big)$, $Q_j = [\,], \forall j \in E$.
30: Update MILP of ϕ^o using new knowledge map and resolve the MILP.
31: **end while**

B. Handling Candidate MTL Task Specifications

In this section, we explore the low-level motion planning approach to move the end-effector in order to satisfy the candidate MTL task specification obtained using the high-level MILP problem. We apply the optimization technique in [22], [23] to handle the nonlinear constraints in this letter, by formulating the following constrained optimization problem, based on the assumptions made in Problem 2:

Problem 2. Given a manipulator arm with dynamics as shown in Fig. 1, with initial joint angles $q^0 \in \mathbb{R}^N$, maximum and minimum joint limits ($q_{\max}, q_{\min} \in \mathbb{R}^N$), a candidate task specification expressed by an MTL formula ϕ^c, a desired robustness measure ρ_ϕ^d and and a performance objective $\mathcal{J}$ for a trajectory duration $\mathcal{H}$, find

$$\arg\min_{\mathbf{u}_\mathcal{H}} \mathcal{J}(x_{\mathcal{H},i}, \mathbf{u}_\mathcal{H}),\ i \in \mathcal{N}$$

$$\text{subject to } \begin{cases} \rho_{\phi^c}(x_{\mathcal{H},T}) \geq \rho_\phi^d, \\ \text{no collision of arm with obstacles} \in \mathbf{Obs}. \end{cases}$$

The objective of the proposed optimization problem is to generate control inputs that minimizes the chosen performance

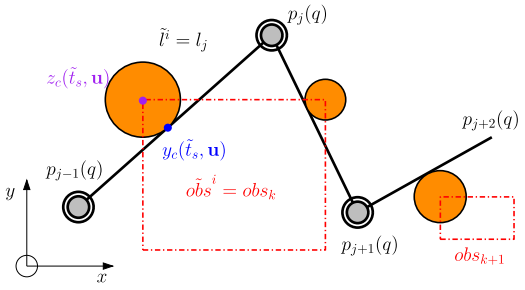

Fig. 3. Illustration of safety radius for collision with obstacles at $t = \tilde{t}_l$.

criteria $\mathcal{J}$, while satisfying the motion specification ϕ^c and avoiding all obstacles in the workspace.

Recall from Section II-D, that given an MTL specification ϕ, there exists a critical predicate $\mathcal{O}(\tilde{\pi})$ and a critical time $\tilde{t}_T \in [0, \mathcal{H}]$ that defines the robustness radius of a system trajectory with respect to the given specification ϕ. Following Proposition 3.1 in [23], the robustness radius can be expressed using critical predicate as

$$\rho_\phi(x_{\mathcal{H},T}) = \begin{cases} \inf\limits_{z \notin \mathcal{O}(\tilde{\pi})} \|p_T(q(\tilde{t}_T)) - z\|, \; p_T(q(\tilde{t}_T)) \in \mathcal{O}(\tilde{\pi}), \\ -\inf\limits_{z \in \mathcal{O}(\tilde{\pi})} \|p_T(q(\tilde{t}_T)) - z\|, \; p_T(q(\tilde{t}_T)) \notin \mathcal{O}(\tilde{\pi}). \end{cases} \tag{10}$$

The first constraint in Problem 2 corresponding to satisfaction of MTL formula ϕ can be re-written as

$$g_T(\mathbf{u}_\mathcal{H}) = \rho_\phi(x_{\mathcal{H},T}) - \rho^d_\phi, \; g_T(\mathbf{u}_\mathcal{H}) \geq 0, \tag{11}$$

since $u(\cdot)$ is the optimization variable. This function $g_T(\mathbf{u}_\mathcal{H})$ is in general non-differentiable and non-convex. Following the approach presented by [23], we propose an iterative gradient descent algorithm to solve Problem 2 as shown in Algorithm 2.

For each iteration i, from the knowledge of the critical proposition $\mathcal{O}(\tilde{\pi}^i)$ and critical time $\tilde{t}^i_T$, the corresponding critical point (closest point on the critical proposition from the system trajectory) on the critical proposition $z(\tilde{t}^i_T, \mathbf{u}^i)$ is computed as

$$z(\tilde{t}^i_T, \mathbf{u}^i) = \begin{cases} \arg\min\limits_{z \notin \mathcal{O}(\tilde{\pi}^i)} \|p_T(q(\tilde{t}^i_T)) - z\|, \; p_T(q(\tilde{t}^i_T)) \in \mathcal{O}(\tilde{\pi}^i), \\ \arg\min\limits_{z \in \mathcal{O}(\tilde{\pi}^i)} \|p_T(q(\tilde{t}^i_T)) - z\|, \; p_T(q(\tilde{t}^i_T)) \notin \mathcal{O}(\tilde{\pi}^i). \end{cases} \tag{12}$$

We now define a new constraint function

$$G_T(\mathbf{u}^i_\mathcal{H}) = \|z(\tilde{t}^i_T, \mathbf{u}^i) - p_T(q(\tilde{t}^i_T))\| \text{sign}\left(\rho_\phi(x^i_{\mathcal{H},T})\right). \tag{13}$$

Using (10)–(13) one can observe that [23], if $G_T(\mathbf{u}^2) > G_T(\mathbf{u}^1)$ holds, so does $g_T(\mathbf{u}^2) > g_T(\mathbf{u}^1)$.

C. Generating Collision-Free Trajectory

The second constraint in Problem 2 ensuring collision-free trajectory of the manipulator arm has to take into account all the link positions of the arm, not just the end-effector position. Similar to the robustness measure defined in Section II-D we use a safety radius to measure how *safe* the manipulator arm trajectory is with respect to the obstacles $obs_k \in \mathbf{Obs}$, in the workspace as illustrated in Fig. 3 for a time t. The safety radius for any $obs \in \mathbf{Obs}$ and any link l is defined as:

$$\rho_s(obs, l(t)) = \begin{cases} \min\limits_{z \in obs, y \in l(t)} \|y - z\|, obs \cap l(t) = \emptyset, \\ -(\min \|d\| : obs \cap (l(t) + d) = \emptyset), \text{otherwise}, \end{cases}$$

where, $l(t)$ denotes the space occupied by link l at time t. Note that $\rho_s(obs, l(t))$ is the separation distance for no collision between the obstacle-link pair, and penetration depth between the pair otherwise. In Fig. 3, both the l^{th}_j and l^{th}_{j+1} links of the manipulator arm has negative safety radii as the links collide with obs_k, whereas, the l^{th}_{j+2} link has a positive safety radius with respect to obs_{k+1}. The overall safety radius of the manipulator arm trajectory for this time t is then defined in terms of the l^{th}_j link (*critical link*) and obs_k (*critical obstacle*) as the ball formed by this link-obstacle pair is the largest and the l^{th}_j link has to be moved by at least the size of the corresponding ball in order to try to ensure the arm does not collide with any obstacle. The safety radius $\rho_s(\mathbf{x}_\mathcal{H})$ for the entire trajectory is then the minimum of all such radii over the entire time:

$$\rho_s(\mathbf{x}_\mathcal{H}) = \min_{0 \leq t \leq \mathcal{H}} \left(\min_{obs, l} \rho_s\big(obs, l(t)\big) \right). \tag{14}$$

For satisfaction of the second constraint we desire

$$g_s(\mathbf{u}_\mathcal{H}) = \rho_s(\mathbf{x}_\mathcal{H}) - \rho^d_\phi, \; g_s(\mathbf{u}_\mathcal{H}) \geq 0. \tag{15}$$

Let us denote the critical time, the obstacle, and the link that minimize (14) as $\tilde{t}_s$, $\tilde{obs}^i$, and $\tilde{l}^i$, respectively. The critical points on the critical obstacle and the critical link denoted as $z_c(\tilde{t}_s, \mathbf{u}) \in \tilde{obs}^i$ and $y_c(\tilde{t}_s, \mathbf{u}) \in \tilde{l}^i(\tilde{t}_s)$ are such that the following holds:

$$\rho_s(\mathbf{x}_\mathcal{H}) = \begin{cases} -\|y_c - z_c\|, \tilde{obs}^i \cap \tilde{l}^i(\tilde{t}_s) \neq \emptyset, \\ \|y_c - z_c\|, \tilde{obs}^i \cap \tilde{l}^i(\tilde{t}_s) = \emptyset. \end{cases} \tag{16}$$

Then we can re-write the second constraint function in a form similar to (13) as

$$G_s(\mathbf{u}_\mathcal{H}) = \|y_c(\tilde{t}_s, \mathbf{u}) - z_c(\tilde{t}_s, \mathbf{u})\| \text{sign}\left(\rho_s(\mathbf{x}_\mathcal{H})\right). \tag{17}$$

We use the Minkowski formulation, which is also used in the Gilbert-Johnson-Keerti (GJK) algorithm [29] in order to find the critical point corrsponding to the critical obstacle-link pair. As a result, we only consider convex shapes for the manipulator arm links and obstacles. Other specialized collision checker algorithms, such as Flexible Collision Library (FCL) [30], providing separation distances between non-overlapping objects and penetration depths between overlapping objects can also be used.

D. Satisfying Joint Limits

We express the requirement of keeping the joint angles within specified limits by formulating another MTL formula

$$\phi_{\text{limit}} = \Box_{[0,\mathcal{H}]} \pi_{\text{limit}}, \tag{18}$$

Algorithm 2: Low-level Motion Planning.

1: Input : ϕ^c, $\mathcal{H}$, $\mathbf{q}^0_{\mathcal{H}}$.
2: Initialize counter $i = 0$.
3: **if** $\mathbf{q}^0_{\mathcal{H}}$ is not provided **then**
4: Generate an initial joint angle trajectory $\mathbf{q}^i_{\mathcal{H}}$.
5: **end if**
6: Compute the link position trajectory $\mathbf{x}^i_{\mathcal{H}}$.
7: Run `TaLiRo` on the trajectories to determine robustness, critical times $\tilde{t}^i_j$, critical predicates and constraint functions $G_j(\mathbf{u}^i)$ corresponding to $\tilde{t}^i_j$ for $j = T, q$.
8: Determine the critical obstacle $\tilde{obs}^i$, the corresponding critical time $\tilde{t}^i_s$ and the second constraint $G_s(\mathbf{u}^i)$.
9: **while** $\min(G_j,\ j = \{T, s, q\}) < 0$ **do**
10: Compute objective function gradient $\frac{d\mathcal{J}(u)}{du}\Big|_{u=\mathbf{u}^i}$.
11: Compute constraint function gradients $\frac{dG_j(u)}{du}\Big|_{u=\mathbf{u}^i}$, $j = \{T, s, q\}$, corresponding to the critical times.
12: Employ a gradient descent (non)linear constrained optimization algorithm to generate next iteration's input $\mathbf{u}^{i+1}$. In this letter, *interior-point* method is used.
13: Generate next iteration's joint angle trajectory $\mathbf{q}^{i+1}_{\mathcal{H}}$ and then the link position trajectory $\mathbf{x}^{i+1}_{\mathcal{H}}$.
14: Rerun Steps 7 and 8.
15: **if** $\min(G_j)$ does not improve anymore **then**
16: Report task specification can not be satisfied and finish execution of the algorithm.
17: **end if**
18: Increase counter : $i = i + 1$.
19: **end while**
20: Report specification is satisfied; return $\mathbf{u}^i_{\mathcal{H}}$, $\mathbf{x}^i_{\mathcal{H}}$, $\mathbf{q}^i_{\mathcal{H}}$.

where, the predicate π_{limit} is defined as

$$\pi_{\text{limit}} : Aq \leq b,\ A = \begin{bmatrix} I_N \\ -I_N \end{bmatrix},\ b = \begin{bmatrix} q_{\max} \\ -q_{\min} \end{bmatrix}. \tag{19}$$

I_N is the identity matrix of order N. Unlike the previous MTL forumlae, this formula is to be evaluated with respect to the joint angle trajectory defined by $\mathbf{q}_{\mathcal{H}}$ since we want to ensure that the joint angles are within the joint angle limits defined by $q_{\max}, q_{\min} \in \mathbb{R}^N$ for the entire trajectory duration $\mathcal{H}$. Following, the method described in Section III-B, another constraint function $G_q(\mathbf{u}_{\mathcal{H}})$ can be added to the optimization algorithm to satisfy the formula ϕ_{limit}.

E. Optimization Problem Formulation for Controller Synthesis

We also consider optimizing the performance metric $\mathcal{J}$, so that the resulting sequence of control input signals should also satisfy $\mathcal{J}(\mathbf{u}^{i+1}) \leq \mathcal{J}(\mathbf{u}^i)$. Thus, we have a constrained optimization problem in hand, which can be solved by any optimization algorithm capable of solving constrained optimization problems with (non)linear objective and constraint functions. In this letter, we utilize the *interior-point* algorithm [31] to solve this optimization problem as detailed in Algorithm 2.

Our optimization technique utilizes the functional derivative of the constraint functions $G_j(\mathbf{u}^i)$, in the control input signal space to compute an ascent direction for increasing $G_j(\mathbf{u}^i)$, which in turn increases $g_j(\mathbf{u}^i)$ leading to a feasible trajectory of the system. Gradient of the first constraint function at the i^{th} iteration can be evaluated as

$$\frac{dG_1(\mathbf{u}^i)}{du} = \frac{\partial G_1}{\partial x}\bigg|_{p_T(q(\tilde{t}))} S(\tilde{t}), \tag{20}$$

where, $S(\cdot)$ is the sensitivity function of the end-effector trajectory with respect to the control input signal $u(\cdot)$ defined by the solution of the sensitivity equation [32] as

$$\dot{S}(t) = \frac{\partial F}{\partial x}\bigg|_{(t,p_T(q(t)),u)} S(t) + \frac{\partial F}{\partial u}\bigg|_{(t,p_T(q(t)),u)},\ S(0) = 0. \tag{21}$$

Gradients for the other two constraint function can be evaluated similar to (20)–(21) but with respect to position trajectory of the critical link and joint angle trajectory at that iteration, respectively. Any differentiable function can be chosen as the performance objective $\mathcal{J}$ for this approach. The performance objective that we consider is

$$\mathcal{J}(\mathbf{u}) = \frac{1}{2}\int_0^{\mathcal{H}} \|u(t)\|_2^2\, dt, \tag{22}$$

to minimize the overall control effort.

F. Completeness

At low-level the gradient-descent algorithm searches in the neighborhood of the initial trajectory to find a feasible trajectory that satisfies the candidate MTL task specification. One disadvantage is that the search process might get trapped at a local minimal in the neighborhood of the initial trajectory. To circumvent this problem, we therefore propose to execute the search process from different initial trajectories to broaden the search space of the optimization. Since the state-space is compact, the probability of failure to find a feasible trajectory at the low-level, approaches zero with time as more and more different new trajectories are explored. At the top-level, the MILP framework takes into account all possible arrangements of the objects as well as time required for the arm to move an object from one location to another. Therefore, if a feasible trajectory exists satisfying the MTL task specification ϕ_o, the MILP framework will find it since it considers all possible ways of manipulating the objects infinitely often. Consequently the gradient-descent optimizer also searches for the low-level implementations of all possible ways of manipulating objects infinitely often. Therefore, the proposed hierarchical structure, combining the MILP framework and gradient-descent optimization algorithm, is *probabilistically complete*.

IV. Examples

The motion planning framework is currently entirely implemented in *MATLAB*. We consider an object manipulation MTL task specification for motion planning of a 7-DOF dual arm

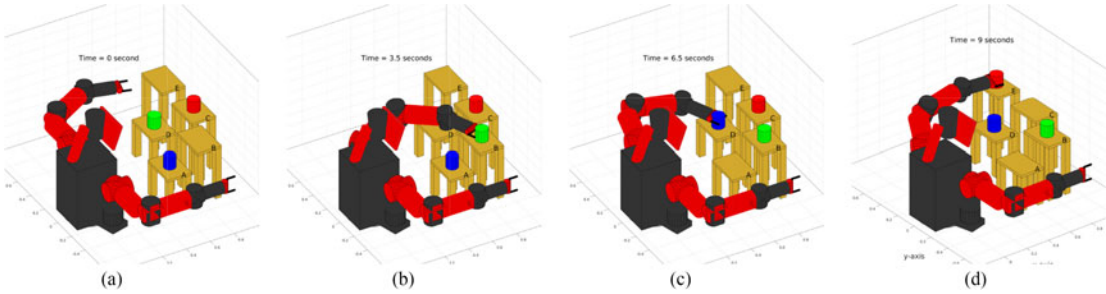

Fig. 4. Motion planning for specification ϕ_o^1. (a) Initial configuration. (b) Arm moves o_g first. (c) o_b is moved next. (d) Finally arm moves o_r.

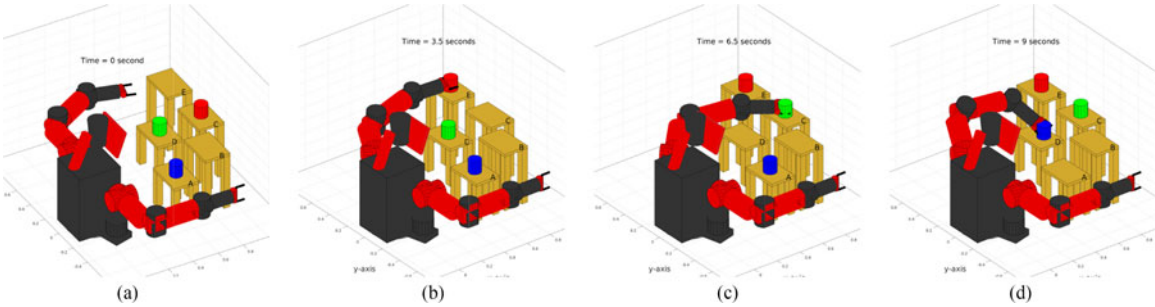

Fig. 5. Motion planning for specification ϕ_o^2. (a) Initial configuration. (b) Arm moves o_r first. (c) o_g is moved next. (d) Finally arm moves o_b.

industrial robot, Baxter[1][2]. The task specification is, however, only defined in terms of the left arm of the Baxter. Fig. 4 shows the workspace environment of the Baxter, involving 5 tables (object locations) of different heights, where 3 different objects can be placed. The tables act as obstacles since the arm can not pass through the tables and should avoid them. We consider two MTL task specifications:

$$\phi_o^1 = \Diamond_{[0,9]}(\lambda_{o_b} = D) \wedge \Diamond_{[0,9]}(\lambda_{o_r} = E),$$

$$\phi_o^2 = \Diamond_{[0,9]}(\lambda_{o_b} = D) \wedge \Diamond_{[0,9]}(\lambda_{o_r} = E) \wedge \neg(\lambda_{o_b} = D)\mathcal{U}(\lambda_{o_r} = E),$$

where o_r, o_g, o_b is used to denote the objects colored *red, green* and *blue* respectively and the five tables are labeled *A,B,C,D* and *E*. The first specification ϕ_o^1 states that within 9 seconds the manipulator arm must *eventually* place the objects o_b and o_r on the tables *D* and *E* respectively. ϕ_o^2 extends this specification by also requiring that o_b can not be placed on table *D* before object o_r is placed on table E. The total trajectory duration for both cases was chosen to be 9 seconds, since that is the minimum time required to evaluate satisfaction of the specifications.

We present the results of the motion planning problem for ϕ_o^1 and ϕ_o^2 in Figs. 4 and 5 respectively. The accompanying video submission, also shows different viewing angles of the environment. Observe that for satisfaction of ϕ_o^1, to place o_b on table *D*, the object o_g needs to be moved from D to vacant D. Once the end-effector picks up o_g there is a choice of placing it either on the table *B* or *E* as the other ones are occupied. However, since ϕ_o requires moving o_r to *E*, the MILP problem from Section III-A correctly chooses table *B* to place o_g. As otherwise, the end-effector would have to move o_g from *E* to make space for placing o_r increasing the number of times the objects are moved around. Thus, the MILP problem is also capable of finding a sequence of movements that do not require *backtracking* in terms of placing objects at locations. In this case the algorithm automatically moves o_b before moving o_r to satisfy ϕ_o^1. On the other hand, the MILP algorithm chooses the correct sequence of moving o_r to *E*, o_g to *C* and o_b to *D*, as shown in Fig. 5 to satisfy ϕ_o^2. Clearly, these sequences of end-effector motion are not explicitly stated in the task specifications but the hierarchical framework combining the MILP problem with the gradient-descent optimization algorithm automatically selects the sequence. The work presented in [12] is closely related, in the sense that the method presented also handles such task specifications with temporal constraints, which the other TMP methods can not handle. However, unlike [12] our proposed method can produce optimal motion plan to complete the desired task. Moreover, [12] does not handle specific timing restrictions on manipulating objects, which our algorithm does take into account as can be seen from both Figs. 4 and 5, presenting that indeed the objects o_b and o_r are moved respectively to the tables *D* and *E* within 9 seconds, satisfying the MTL specifications.

TABLE I
TIME TAKEN TO SOLVE THE MOTION PLANNING PROBLEM

MTL spec	Time (s) (min, 25%, median, 75%, max)				
ϕ_g^1	23.93	56.32	68.75	86.62	507.92
ϕ_o^2	71.45	138.70	167.23	188.79	325.24

Table I presents the time required for solving the motion planning problem for 50 independent runs with randomly generated initial configuration for the arm, in a 64-bit Linux machine with Intel Core i7 3.4-GHz CPU and 16 GB RAM. For each initial configuration of the arm, we executed the proposed motion planning algorithm for different random initial trajectories until

[1]https://www.rethinkrobotics.com/products/baxter
[2]https://github.com/rpiRobotics/baxter-on-wheels-sim

a trajectory satisfying the specification was found, to broaden the search space of the optimization algorithm. In all of those 50 trials the algorithm successfully found trajectories for the manipulator arm to satisfy the specification. The results showcase, that even though the algorithm searches locally to initial trajectory of the arm, executing the search process in neighborhood of different initial trajectories embeds the property of *probabilistical completeness* in the motion planner by broadening the search space of the optimization algorithm.

V. Conclusion

In this letter, we proposed an optimization approach for generating control input signals for manipulator robot arm so that the closed loop execution trajectory of the arm satisfies desired task specification expressed in terms of MTL formulae. Our approach is based on utilizing MILP to determine the sequence of movements for the end-effector, followed by computing descent (ascent) vectors using the functional gradient of the objective and the constraint functions for minimization (maximization) of the objective value, utilizing the method of calculus of variations. We utilize the optimization framework to determine trajectories and their durations between different object locations and update the MILP problem accordingly to aid determining the sequence of movements. Even though the gradient method only guarantees locally optimal solutions based on the initial trajectory of the manipulator arm, executing the method from different initial seeds broadens the search space of the optimization algorithm thereby also making the method *probabilistically complete*. In addition to solving the TMP problem by following the hierarchical method, we also can update the information about the workspace of the manipulator arm. Moreover, the low-level optimization framework proposed here is not only limited to manipulator arms. The framework is applicable to generate control input signals for satisfying correct system behaviors defined in terms of MTL specifications for any general (non)linear dynamical system. On the other hand, the MILP problem is limited to the specific workspace domain considered in this letter. This is similar to the concept of *plan outline* in [11] that helps the SMT solver to ignore implausible actions (such as trying to place an object before picking it up). One might need to add more constraints or relax some, in order to completely represent a different workspace.

References

[1] R. M. Murray, Z. Li, and S. S. Sastry, *A Mathematical Introduction to Robotic Manipulation*. Boca Raton, FL, USA: CRC Press, 1994.

[2] J. C. Latombe, *Robot Motion Planning*. Norwell, MA, USA: Kluwer, 1991.

[3] O. Takahashi and R. J. Schilling, "Motion planning in a plane using generalized voronoi diagrams," *IEEE Trans. Robot. Autom.*, vol. 5, no. 2, pp. 143–150, Apr. 1989.

[4] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Autom.*, vol. 12, no. 4, pp. 566–580, Aug. 1996.

[5] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, vol. 2, 2000, pp. 995–1001.

[6] M. Zucker *et al.*, "Chomp: Covariant hamiltonian optimization for motion planning," *Int. J. Robot. Res.*, vol. 32, nos. 9–10, pp. 1164–1193, 2013.

[7] J. Hoffmann and B. Nebel, "The FF planning system: Fast plan generation through heuristic search," *J. Artif. Intell. Res.*, vol. 14, pp. 253–302, 2001.

[8] J. Rintanen, "Planning as satisfiability: Heuristics," *Artif. Intell.*, vol. 193, pp. 45–86, 2012.

[9] S. Cambon, R. Alami, and F. Gravot, "A hybrid approach to intricate motion, manipulation and task planning," *Int. J. Robot. Res.*, vol. 28, no. 1, pp. 104–126, 2009.

[10] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer," in *Proc. 2014 IEEE Int. Conf. Robot. Autom.*, 2014, pp. 639–646.

[11] S. Nedunuri, S. Prabhu, M. Moll, S. Chaudhuri, and L. E. Kavraki, "SMT-based synthesis of integrated task and motion plans from plan outlines," in *Proc. 2014 IEEE Int. Conf. Robot. Autom.*, 2014, pp. 655–662.

[12] K. He, M. Lahijanian, L. E. Kavraki, and M. Y. Vardi, "Towards manipulation planning with temporal logic specifications," in *Proc. 2015 IEEE Int. Conf. Robot. Autom.*, 2015, pp. 346–352.

[13] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, "Incremental task and motion planning: A constraint-based approach," in *Proc. Robot. Sci. Syst.*, 2016, pp. 1–6.

[14] F. Lagriffoul and B. Andres, "Combining task and motion planning: A culprit detection problem," *Int. J. Robot. Res.*, vol. 35, no. 8, pp. 890–927, 2016.

[15] A. Pnueli, "The temporal logic of programs," in *Proc. 1977 18th Annu. Symp. Found. Comput. Sci.*, 1977, pp. 46–57.

[16] M. Kloetzer and C. Belta, "A fully automated framework for control of linear systems from temporal logic specifications," *IEEE Trans. Autom. Control*, vol. 53, no. 1, pp. 287–297, Feb. 2008.

[17] G. E. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for dynamic robots," *Automatica*, vol. 45, no. 2, pp. 343–352, 2009.

[18] A. Pnueli and R. Rosner, "On the synthesis of a reactive module," in *Proc. 16th ACM SIGPLAN-SIGACT Symp. Principles Program. Lang.*, 1989, pp. 179–190.

[19] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning with deterministic μ-calculus specifications," in *Proc. 2012 Amer. Control Conf.*, Montreal, QC, 2012, pp. 735–742.

[20] S. C. Livingston, E. M. Wolff, and R. M. Murray, "Cross-entropy temporal logic motion planning," in *Proc. 18th Int. Conf. Hybrid Syst. Comput. Control*, Seattle, WA, 2015, pp. 269–278.

[21] G. E. Fainekos and G. J. Pappas, "Robustness of temporal logic specifications for continuous-time signals," *Theoretical Comput. Sci.*, vol. 410, no. 42, pp. 4262–4291, 2009.

[22] A. K. Winn and A. A. Julius, "Optimization of human generated trajectories for safety controller synthesis," in *Proc. 2013 Amer. Control Conf.*, Washington, DC, 2013, pp. 4374–4379.

[23] H. Abbas, A. Winn, G. Fainekos, and A. A. Julius, "Functional gradient descent method for metric temporal logic specifications," in *Proc. Amer. Control Conf.*, Portland, OR, 2014, pp. 2312–2317.

[24] S. Chinchali, S. C. Livingston, U. Topcu, J. W. Burdick, and R. M. Murray, "Towards formal synthesis of reactive controllers for dexterous robotic manipulation," in *Proc. 2012 IEEE Int. Conf. Robot. Autom.*, 2012, pp. 5183–5189.

[25] A. Menon, B. Cohen, and M. Likhachev, "Motion planning for smooth pickup of moving objects," in *Proc. 2014 IEEE Int. Conf. Robot. Autom.*, 2014, pp. 453–460.

[26] S. Saha and A. A. Julius, "An MILP approach for real-time optimal controller synthesis with metric temporal logic specifications," in *Proc. 2016 Amer. Control Conf.*, 2016, pp. 1105–1110.

[27] Y. Annpureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan, *S-Taliro: A Tool for Temporal Logic Falsification for Hybrid Systems*. New York, NY, USA: Springer, 2011.

[28] J. Löfberg, "YALMIP: A toolbox for modeling and optimization in MATLAB," in *Proc. CACSD Conf.*, 2004. [Online]. Available: http://control.ee.ethz.ch/\,joloef/yalmip.php

[29] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space," *IEEE J. Robot. Autom.*, vol. 4, no. 2, pp. 193–203, Apr. 1988.

[30] J. Pan, S. Chitta, and D. Manocha, "FCL: A general purpose library for collision and proximity queries," in *Proc. 2012 IEEE Int. Conf. Robot. Autom.*, 2012, pp. 3859–3866.

[31] R. H. Byrd, J. C. Gilbert, and J. Nocedal, "A trust region method based on interior point techniques for nonlinear programming," *Math. Program.*, vol. 89, no. 1, pp. 149–185, 2000.

[32] H. K. Khalil, *Nonlinear Systems*, 3rd ed. Englewood Cliffs, NJ, USA: Prentice-Hall, 2002.