RICE UNIVERSITY

Robot Manipulation Planning Under Linear Temporal Logic Specifications

by

Keliang He

A Thesis Submitted
in Partial Fulfillment of the
Requirements for the Degree

Master of Science

Approved, Thesis Committee:

Dr. Lydia E. Kavraki, Chair
Noah Harding Professor
Department of Computer Science

Dr. Moshe Y. Vardi
Karen Ostrum George Professor
Department of Computer Science

Dr. Swarat Chaudhuri
Associate Professor
Department of Computer Science

Houston, Texas

August, 2015

ABSTRACT

Robot Manipulation Planning Under Linear Temporal Logic Specifications

by

Keliang He

Automated planning for high-level manipulation tasks is highly desirable, because it enables robot manipulators to be used by non-robotics experts. This thesis presents one approach to solving manipulation planning for tasks expressed in linear temporal logic (LTL). This approach is based on earlier work on the synergistic framework for motion planning from LTL specifications, which provides probabilistic completeness guarantees. Even though the synergistic framework was shown to work well for planning for LTL tasks in the navigation domain, it lacks an abstraction that can capture the high dimensionality of manipulation. This thesis enables manipulation planning using the synergistic framework by introducing a manipulation abstraction and modifying the interaction between task and motion planning in the framework. The modified framework is shown to be effective in case studies in both simulation and physical systems. The case studies also show that the synergistic framework solves manipulation problems more effectively using the manipulation abstraction in comparison with a naive abstraction.

Acknowledgement

I would like to first thank Dr. Lydia Kavraki and Dr. Moshe Vardi for their teaching, guidance, and support from the day I started the program. Their advice and encouragement allowed me to overcome the various challenges I have faced. This thesis and the projects leading up to it could not have been possible without them.

I would also like to thank Dr. Swarat Chaudhuri for his teaching and suggestions that helped me better understand the topic.

I would like to especially thank Dr. Morteza Lahijanian who worked with me closely for the past two years. I would also like to thank everyone at the KavrakiLab, especially Dr. Devin Grady, Ryan Luna, Sailesh Prabhu, Yue Wang, Stephen Butler, Dr. Neil Dantam, and Zak Kingston for helping me through the various difficulties that occurred in my projects.

This work was supported by NSF 1139011.

Contents

Chapter 1

Introduction

Traditionally, robot manipulators have found applications in repeating a single task in highly structured environments, such as factories and warehouses. For example, industrial manipulators such as the Kuka arm are used to move parts between conveyor belts on an assembly line. The assumptions made about the environment enable engineers to pre-program a fixed path for the task.

As robotic systems developed through the decades, an increasing number of robot manipulators have entered environments that are less structured, while tasks for robots become more variant. For example, the Baxter robot by Rethink Robotics was designed for factories where the robot manipulator would be working alongside human workers. Robochef announced by Moley robotics is designed to prepare meals in people's kitchens.

In these scenarios, pre-computing paths is no longer a feasible approach. To start, it is computationally infeasible to enumerate all possible tasks the robot needs to perform. Moreover, any pre-computed path may become infeasible due to the changes in the environment caused by other agents in the environment. This makes automated planning for manipulation systems highly desirable. In automated planning, the robot is given a task that must be achieved, and the planner reasons over a model of the

world with the task given, and automatically finds continuous trajectories and/or controllers to execute to achieve the task. The automated planning system must answer the following two questions.

1. *What sequence of actions needs to be performed in order to achieve the task?*

The answer to this first question involves reasoning about discrete actions. For example, if Robochef is asked to make pasta, should the robot first add water to the pot or get the spaghetti box? Often times this information is not spelled out by the user, and it is up to the planner to pick one of many potential sequences of actions to implement. The strategy for solving these problems is called task planning.

2. *How to operate the robot to achieve each desired action?*

The answer to the second question involves reasoning about continuous motion. For example, what is the sequence of joint angles for the arm to reach the frying pan? The strategy for solving these problems is called motion planning.

One natural approach to solving the automated planning question, then, is by integrating task and motion planning (TMP). Various works [1–18] have been developed to address different cases of the TMP problem, including manipulation planning [1, 5–18]. One of the very successful approaches in the navigation domain is the synergistic framework presented in [4], and later improved in [3]. The framework in [3] approaches the TMP problem by first decomposing the robot configuration space using an *abstraction* to separate the problem into a high-level discrete task planning problem and many low-level continuous motion planning problems. Then the over-

all problem is solved through a synergistic layer that allows for effective information passing between task and motion planning. This framework has guarantees for probabilistic completeness, and has been shown to be effective in navigation domains [4].

However, when the synergistic framework is directly applied to the manipulation domain, a few challenges arise. First, the configuration space of a manipulation problem is the joint configuration space of the robot as well as all the objects. This configuration space is very high dimensional. The poor scalability of the abstraction thus makes directly applying this framework to manipulation problems impractical. More importantly, a naive decomposition of the configuration space does not capture the interaction between the object and the robot. For example, two states that share the same robot state but different object states may seem close to each other in the joint configuration space, as it only requires the object to move. In actual execution, however, these states could be far apart, for it requires moving the robot arm to handle the object, as objects cannot move themselves.

1.1 Contributions

The contributions of this thesis are:

- a manipulation abstraction that captures the manipulation domain,
- modifications to the synergistic framework that work with the manipulation abstraction to perform manipulation planning,

- experimental results on applying this modified framework to plan for the PR2 robot and the Baxter robot.

1.2 Organization

The remainder of this thesis will be structured as follows. Chapter 2 will define the manipulation planning problem, along with our formulation of a manipulation task. Chapter 3 will discuss the background information on motion planning, task planning, and the synergistic framework. Chapter 4 will present the manipulation abstraction and the modifications made to the synergistic framework to accommodate the new abstraction. Chapter 5 will show the experimental results from the modified framework. And finally Chapter 6 will discuss several key observations, and point to the directions for our future work.

Chapter 2

Problem Formulation

This chapter will formulate the central problem discussed in this thesis: manipulation planning for tasks specified using linear temporal logic. Section 2.1 will discuss the formulation of manipulation tasks using linear temporal logic. Section 2.2 will formalize the manipulation planning problem for these manipulation tasks.

2.1 The Manipulation Task

In order to discuss automated planning for manipulation tasks, we must first formalize what is a manipulation task. Unlike other robotics domains such as navigation or mapping, the focal point of a manipulation task is not on the robot itself, but rather on the objects being manipulated. For example, if a Baxter industrial robot is tasked with assembling a part, the task can be expressed by only specifying the resulting configuration of the components, without even mentioning the Baxter. Thus, a manipulation task is not expressed in terms of the behavior of the robot, but in terms of the objects being manipulated. Formally, we define the *atomic propositions* of a manipulation task as follows.

Definition 2.1

Given a scenario with a finite set of objects obj*, a finite set of labels* $\mathfrak{L}$*, and a finite*

set of locations $\mathcal{L}$*, with a function* $\mathcal{L} \to \mathfrak{L}$ *that maps each location with its labels, the* ***atomic propositions*** *of this scenario are the elements of* $\mathsf{obj} \times \mathfrak{L}$*. The element* (o, l) *can be interpreted as "Object* o *is in a location with label* l*."*

Note that we only consider a finite set of possible locations to place the objects. The location to place an object is described by the full configuration of the object when placed at that location. By doing this, we have discretized the set of possible ways to place objects in the environment. This assumption is often made in manipulation planning to simplify the problem [5–10, 12, 13, 15–18], and the study of discretizing space into suitable locations for objects is not the focus of this thesis.

The atomic propositions are the building blocks of a manipulation task. To formulate a manipulation task, it is reasonable to expect the following to be expressible.

1. an atomic proposition is true,
2. a boolean combination of atomic propositions is true,
3. a property will eventually be true,
4. a property will hold (true for every time step) until another property becomes true.

With such properties, the user could specify tasks such as "All (conjunction) of the objects should eventually be in the box, and the box should remain in the packing area until the objects are packed."

There are many different ways to compose the atomic propositions to express the tasks above [19,20], many of which could be translated into one another. For many of the equivalent representations, the choice of which method to use largely depends on the succinctness of the language for the tasks desired, how well the language meshes with the framework, as well as personal preference. The specification language used in this thesis is called *Linear Temporal Logic* [19].

2.1.1 Linear Temporal Logic

Though first introduced to specify program properties in formal verification, linear temporal logic (LTL) has recently found many applications as the specification language for robotic tasks [21–27].

Definition 2.2

Given a set of atomic propositions Π*, the syntax for linear temporal logic formulas is recursively defined as*

1. p*, where* $p \in \Pi$*,*
2. $\neg\varphi$ *(negation),* $\varphi \wedge \psi$ *(conjunction), where* φ *and* ψ *are also* LTL *formulas,*
3. $\mathcal{X}\varphi$ *(next),* $\varphi\,\mathcal{U}\psi$ *(until), where* φ *and* ψ *are also* LTL *formulas.*

Points 1 and 2 represent the syntax for boolean logic. We may include additional boolean operators such as $True$, $False$, $\varphi \vee \psi$ (disjunction), $\varphi \rightarrow \psi$ (implication), $\varphi \leftrightarrow \psi$ (equivalence), and additional temporal operators such as $\mathcal{F}\varphi$ (eventually) and $\mathcal{G}\varphi$ (globally).

The semantics of LTL is defined over *words*. A word $w = (l_0, l_1, \ldots)$ is an infinite sequence over *letters* from the set 2^{Π}. Each letter represents the set of atomic propositions that are valid for the current state of the world. A word, then, describes how the state of the world evolves through time. We use w_i to denote the suffix of the word w beginning from time step i, i.e. $w_i = (l_i, l_{i+1}, \ldots)$. Naturally, $w = w_0$. We say that a word w *satisfies* an LTL formula φ, denoted by $w \models \varphi$ if one of the following is true.

1. φ is an atomic proposition, and $\varphi \in l_0$.
2. $\varphi = \neg\psi$, and w does not satisfy ψ.
3. $\varphi = \psi_1 \wedge \psi_2$, $w \models \psi_1$, and $w \models \psi_2$.
4. $\varphi = \mathcal{X}\psi$, and $w_1 \models \psi$.
5. $\varphi = \psi_1 \mathcal{U} \psi_2$, and there exists k such that for all $0 \leq i < k$, $w_i \models \psi_1$, and $l_k \models \psi_2$.

For the additional operators, each of the following pairs are equivalent.

1. $True$ and $p \wedge (\neg q)$, for any atomic proposition p.
2. $False$ and $\neg True$.
3. $\varphi \vee \psi$ and $\neg((\neg\varphi) \vee (\neg\psi))$.
4. $\varphi \rightarrow \psi$ and $(\neg\varphi) \vee \psi$.
5. $\varphi \leftrightarrow \psi$ and $(\varphi \wedge \psi) \vee ((\neg\varphi) \wedge (\neg\psi))$.

6. $\mathcal{F}\varphi$ and $True\,\mathcal{U}\varphi$.

7. $\mathcal{G}\varphi$ and $\neg(\mathcal{F}(\neg\varphi))$.

For example, imagine a scenario where a robot is serving customers at a bar. The formula $\mathcal{F}(o_{\text{drink}}, l_{\text{cus 2}})$ expresses the task of "deliver the drink to customer 2," while the formula $(o_{\text{snack}}, l_{\text{cus 3}})\,\mathcal{U}(o_{\text{tip}}, l_{\text{cus 3}})$ expresses the task "keep the snacks in front of customer 3 until the tip jar is presented to her."

In practice, many interesting tasks that we wish for the robot to achieve must be achieved in a finite amount of time. Thus we only consider LTL formulas that can be satisfied by considering a finite prefix of the word. This fragment of LTL is called cosafe LTL [28]. Syntactically, cosafe LTL is equivalent to the fragment of LTL where the negation operator is only allowed over the boolean sub-formulas, but not over the temporal formulas. This also disallows the globally ($\mathcal{G}$) operator, as it is defined to be $\neg(\mathcal{F}(\neg\varphi))$. The semantics of co-safe LTL remains the same as LTL.

Using cosafe LTL prevents us from expressing tasks that execute infinitely, such as surveillance tasks. For these tasks, full LTL is used. If the task involves the robot reacting to environment changes, often GR(1) [21] is used. All the example tasks we have discussed so far must be accomplished within a finite time bound, thus the cosafe fragment of LTL is sufficient.

2.2 Problem Definition

As discussed previously, the atomic propositions Π of the LTL formulas are of the form "Object o is at a location with label l." For each moment in the robot's execution of the manipulation task, we can retrieve the atomic propositions that hold by observing the location of the objects, and the labelling of those locations. Thus from each moment in the execution, a letter from 2^{Π} can be generated. We call this mapping from the states of the world to the letters *the labelling function.*

By writing down the letter each time the output of the labelling function changes, we can generate a word that corresponds to the execution. Then for a given task expressed in cosafe LTL, the goal of the manipulation planning problem is to find a finite execution that will generate a word that satisfies the LTL task.

Definition 2.3

Given a robot arm with configuration space $\mathcal{C}_R$, a set of objects O with configuration space $\mathcal{C}_O$, a set of atomic propositions Π, a labelling function $L : \mathcal{C}_O \to 2^{\Pi}$, and a cosafe LTL *formula φ over Π, find a path $P : [0, 1] \to \mathcal{C}_R \times \mathcal{C}_O$ such that the word generated by P through L satisfies φ, and physical constraints are respected.*

Note that though it is certainly the case that many robotic systems have more than one arm to manipulate objects, we consider only using a single arm in this thesis. Thus in this case, the location of the end effector of the robot can be regarded as the location of the robot, as the robot must act upon objects through its only end effector. The topic of using multiple arms to cooperate in order to achieve tasks is a

topic for future work.

There may be many different physical constraints depending on the system being considered. One constraint is that the robot must not be in collision with the objects not being manipulated. Another is that the objects must remain stationary unless acted upon by the robot. For example, in the task previously discussed, "deliver the drink to customer 2," ($\mathcal{F}(o_{\text{drink}}, l_{\text{cus 2}})$) the planner must find continuous trajectories to grasp the drink, and move it to a location labelled customer 2, while avoiding collision with any obstacles. If a collision with another object is unavoidable, then the planner must also find continuous trajectories to remove them before serving the drink.

Chapter 3

Background and Related Work

This chapter will provide the background necessary for this thesis. Section 3.1 and 3.2 will discuss the problem of task planning and motion planning, respectively. Section 3.3 will introduce the original synergistic framework [3,4] designed for task and motion planning, along with a brief comparison with other approaches for solving similar problems.

3.1 Discrete Task Planning

To find a solution to the manipulation planning problem, we must find a sequence of actions to achieve the task. Task planning solves this problem by considering a discrete model of the world and the task. Formally, the planner is given a finite discrete set of state S, and a set of actions A. The planner is also given a transition function $\gamma : S \times A \rightarrow 2^S$, which tells us given a state and the action performed, which states are possible successors. In our formulation, we will consider the actions to be deterministic, so γ is a function from $S \times A$ to S. The task planning problem is then formulated as follows [29].

Definition 3.1

Given S, A, γ, a starting state s_0, and an evaluation function T that tells us whether

a sequence of states $(s_0, s_1, \ldots)$ *satisfy the goal, find a sequence of actions* $(a_0, a_1, \ldots)$ *such that* $\gamma(s_i, a_i) = s_{i+1}$*, and* $T(s_0, s_1, \ldots)$ *returns true.*

For a given continuous problem, there are many different ways the problem can be discretized. For example, in manipulation planning, one could define each state to describe the location of all the objects that are considered for manipulation. Other formulations may include the location and/or behavior of the robot. Each of these formulation would incurs a different task planning formulation. The choice of this discretization is an important one, and will be discussed in the following chapter.

In artificial intelligence, one popular way to describe a task planning problem is through the planning domain description language (PDDL). PDDL was initially designed for the international planning competition (IPC), and therefore is general enough to accommodate many different planning domains and instances. It has since become the standard format for describing describing planning domains.

PDDL [30] uses *domain file* and *instance files*. The domain file describes the predicates of the system. The set of predicates that hold would determine the state of the system. By not representing the states explicitly, PDDL domains are more compact than an explicit representation of the problem. The domain file also contains the actions of the system. Each action contains the precondition that must be satisfied for the action to be applied, and the effect of the action. The instance file describes the objects available in the environment, as well as the goal of the task.

For manipulation domains, the problem can often be express using predicates for locations of the objects and the manipulator, and a few actions that represent picking, placing, and arm moving. In manipulation planning literature that uses PDDL, much work has focused on goals that only specify a target set of states to reach. However, the goal for these works can be extended to incorporate some LTL tasks by state trajectory constraints introduced in PDDL3.0 [31]. Generally, a common strategy used for planning for tasks with these state trajectory constraints is by converting them into a deterministic finite automaton, and incorporating the automaton state as a predicate in the domain [32].

In the robotics domain, the planning problem is often represented using an abstraction to describe the domain along with a logical formula to describe the task. The abstractions may involve partitions of the space according to areas of interest [3, 21, 22], or decompositions of the workspace [23, 25–27]. The task description ranges from full LTLto its fragments such as GR(1) [21] and co-safe LTL [3]. In some cases, these abstractions can be equivalently represented using PDDL domain specifications, and in many cases, the logical formulas can also translated into PDDL goals. As discussed in Chapter 2, the choice of representation depends on many factors including the particulars of the framework being used. For this thesis, LTL formulas will be used to represent the problem.

Two of the most popular approaches for solving task planning problems currently are SAT-based planning and search-based planning. These two approaches have ex-

perienced much success in the planning competitions [31], and have been receiving much attention for their competitive runtime. Both approaches are applicable to manipulation task planning. The following subsections will highlight the key ideas of each approach, and their benefits.

3.1.1 SAT-Based Task Planning

SAT-based planners convert the domain and the task into a boolean satisfiability (SAT) formula, and use existing SAT solvers to find solutions [33–35]. For the interested reader, [29] provides a more detailed discussion on this topic. First, the planner uses a combination of different predicates to describe all the states S. Then the combination of all the actions in the domain is translated into a boolean formula relating the predicates of the previous time step to the predicates of the next time step. The planner begins planning by checking if the predicates at the initial state satisfy the goal. If not, the planner asserts the predicates at the initial state, and the transition between the initial (zeroth) time step and the first time step, and that the goal is reached in the first step. If these assertions can hold at the same time, then a solution is found. Otherwise, the planner continues to deepen this search, until a solution is found, or some limit on the number of steps is reached.

SAT-based planners are successful because instead of considering one state at a time, a boolean formula captures a large set of states, and this set of states can be explored at the same time. SAT-based planners can also take advantage of the

improvement in SAT solving [29], improving its performance simply by using a newer and better SAT solver.

3.1.2 Search-Based Task Planning

Search-based planners [36–38] view the domain as a graph over all states in S. The edges of this graph are instantiations of the actions A allowed in the domain. The task is converted into goal states on the graph. Then the planner uses graph search techniques such as Dijkstra's or A* to find a solution path to the goal states.

One of the main advantages of search-based planners is that additional information could be incorporated via heuristics, most notably the FastForward [37] and FastDownward [38] planners. There are domain independent heuristics that can speed up the search in general, as well as domain specific ones if more information is known about the particular domain. Because the underlying structure reasoned over is a graph, weights over the nodes and edges can be incorporated in these techniques to further guide the search in a desired direction. This is useful in task and motion planning, as the weights can be adjusted according to the results from motion planning. In this thesis, task planning is performed using search-based planning. The study of task planning strategy for task and motion planning is not a focus of this thesis, but is an important topic for future research.

3.2 Continuous Motion Planning

The other component of manipulation planning is motion planning. Given the state of the environment, motion planning considers the question of finding a continuous robot trajectory from an initial configuration to a desired target, without colliding with the environment. For manipulation, both moving the arm toward a grasp and transferring an object (by considering the transferred object as a part of the arm) can be considered as instances of this problem.

More precisely, we define a configuration to contain the information necessary to fully describe the state of the robot. We call the space of all such configurations the *configuration space* C. Let the portion of C that is collision free with the environment C_{free}, then we formally define the motion planning problem as follows [39].

Definition 3.2

Given configuration space C, the free space C_{free}, a start configuration $q_0 \in C$ and goal set $G \subset C$, find a path $P : [0, 1] \rightarrow C_{free}$ such that $P(0) = q_0$, and $P(q) \in G$.

3.2.1 Sampling-Based Motion Planning

One of the most popular approaches used for motion planning for manipulators is the sampling-based approach [40–45]. Many other popular approaches, such as space decomposition-based and optimization-based approaches, do not scale well into the manipulation domain, where the manipulators generally have at least 6 degrees of freedom.

In their simplest form, sampling-based motion planners maintain a graph of configurations called the *roadmap* that initially contains q_0. Then the planner samples a random configuration q in C, and adds q to the graph or uses q to find other configurations to add to the graph. This process is repeated until a point g in G is added to the graph. Then, a graph search algorithm such as Dijkstra's algorithm or A* is used to find a path from start to q_0 to q.

For manipulation problems, especially cases where only a single plan is needed, the graph is often a tree rooted at q_0 [41–45]. In these cases, inverse traversal of the tree can be used instead of graph search, speeding up the search process.

Most sampling-based motion planners guarantee probabilistic completeness [46]. This means that if a solution from q_0 to G exists in C_{free}, the probability that the planner will find a solution in time t goes to 1 as t approaches infinity. On the other hand, if no solution is available, a sampling-based planner will continue to add configurations to the graph, and never confirm that no solution exists.

3.3 Synergistic Framework for Task and Motion Planning

The synergistic framework, proposed in [4], is a planning framework designed for motion planning. It has been shown to be very successful in the navigation domain, where the dynamics of the system are complex. This framework uses discrete reasoning to guide exploration of the continuous space. The framework was expanded in [3] to plan for navigation tasks under LTL specifications. This thesis will build on the

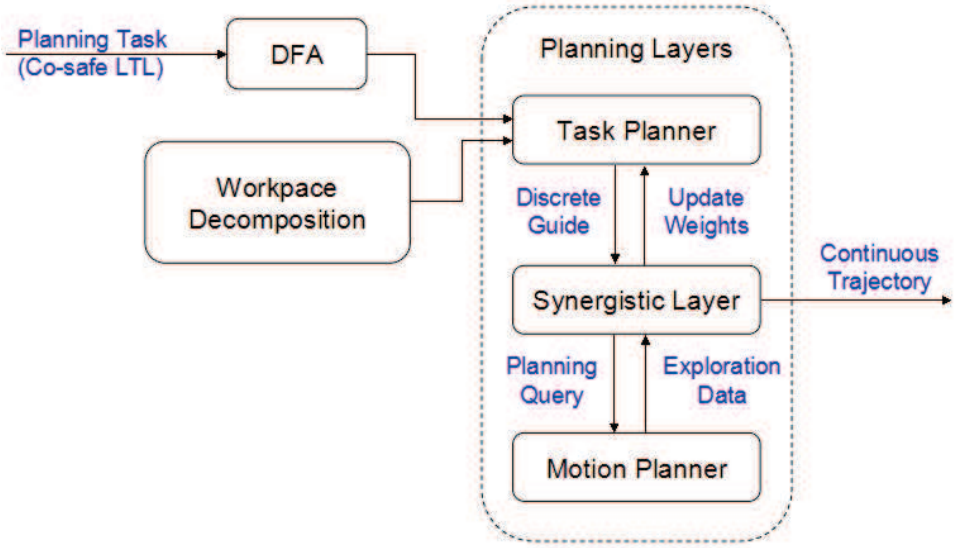


Figure 3.1 : An overview of the synergistic framework. The communication between task and motion planning is performed by a synergistic layer.

work in [3, 4] for a task and motion planning framework for manipulation problems.

Figure 3.1 shows the general structure of the synergistic framework in [3] when applied to the TMP problem in navigation. The framework first converts the LTL task into a deterministic finite automaton (DFA), and uses triangulation to decompose the robot workspace. The framework then takes a product of the DFA with the decomposition, and uses search-based task planning techniques to find a discrete plan called the *guide*. The framework then uses a sampling-based motion planner to explore along this guide for some time. If motion plans are found for all segments of the guide, then the plan is returned. Otherwise, information regarding the exploration done by the motion planner is used to generate weights for the discrete task planning problem. Task planning is performed again with the updated weighted to guide the

search toward under-explored regions. Then the new guide produced is explored using the motion planner. Through this interleaving of task and motion planning, the synergistic framework eventually finds a motion plan, if one exists.

3.3.1 Converting LTL to DFA

The original synergistic framework considered only navigation domains [3, 4], where the atomic propositions only specify if the robot is in certain propositional regions. LTL is used to compose the atomic propositions. LTL formulas can be converted to an automaton which can be more easily considered. For a formula with atomic propositions Π, an automaton can be constructed such that the labels on the edges are sets of letters from $2^{2^{\Pi}}$, and any accepting word of the automaton would be an accepting word of the LTL formula. Thus the automaton would be equivalent to the LTL specification [47]. Because only finite tasks written in cosafe LTL are used for the robot tasks, the automaton generated is always a finite automaton. The synergistic framework uses spot [48] to perform this conversion from LTL to DFA.

3.3.2 Workspace Decomposition

The synergistic framework also introduced a workspace decomposition technique in the navigation domain to guide the search. The decomposition is achieved by triangulating the workspace of the robot, while ensuring that each decomposition cell lies in the same region. This decomposition is then recorded as a graph, where the nodes are the cells of the decomposition, and neighboring cells have edges between

them. The cells contained in propositional regions are labelled with the corresponding labels. The workspace decomposition serves two purposes. First, it captures which propositional regions each physical location lies. Second, the decomposition captures the connectivity of the space, guiding the motion planner to search in more relevant areas.

3.3.3 Planning Layers

The DFA and the workspace decomposition are then passed to the three planning layers, shown in Figure 3.1. The task planning layer finds a discrete sequence of transitions between decomposition regions, called a guide, to satisfy the task. The motion planning layer explores the continuous space to find motion to achieve these transitions. The synergy layer coordinates the task and motion planning layers by feeding the motion planner with segments of the guide, and adjusting weights to steer the task planner.

The task planner first takes a product between the DFA and the workspace decomposition, combining them into a product graph. Each node in the product graph is a pair (v, z) where v is the decomposition cell and z is the automaton state. An edge from (v, z) to (v', z') exists if v and v' are connected in the decomposition, and the propositional label of v' is a letter in the edge from z to z'. Intuitively, the transition from a product node to the next is possible, if the cells are adjacent, and the label of the new cell enables the corresponding transition of the DFA. Dijkstra's algorithm

is used to find a path from (v_0, z_0), where v_0 is the cell of the initial configuration of the robot and z_0 is the starting state of the DFA, to any node whose DFA state is accepting. If Dijkstra's algorithm terminates without returning a path, then the task is unachievable. Otherwise, this path, called the guide, is passed to the synergistic layer.

The synergistic layer then divides the guide into motion planning queries that are passed to a sampling-based motion planner. Initially, only the initial state of the robot is available to the motion planner, so search starts from v_0 to the next decomposition cell in the guide. In time, more cells in the guide are reached, so the synergistic layer would also invoke the motion planner to explore those cells. The selection of a cell to explore in the guide is weighted so that later part of the guide is encouraged. Doing this helps the framework to work on regions at the frontier of exploration that are close to finding a complete solution. After a given amount of time, if no solution is found, the synergistic layer gathers information on the amount of exploration done in each of the cells. For cells that have been heavily explored with no success, their weights are increased in the decomposition graph to indicate the difficulty in searching through these cells. Thus in future searches of the product graph, these cells are discouraged, while cells that are under-explored are more likely to be involved in a guide. This process continues until a solution is found in the continuous space from the initial state to achieve the LTL task.

3.3.4 Probabilistic Completeness

The synergistic framework as described in [4] is probabilistically complete. The intuition behind the proof is that if a solution in the continuous space exists, then it must pass through some sequence of decomposition cells. This induces a valid guide. If the synergistic framework does not find a solution, it will continue to try each guide, and increase the weights. Thus if given enough time, the framework will consider each possible guide infinitely often, and each cell will be considered by the motion planner infinitely often. Because the motion planner is probabilistically complete, this makes the framework probabilistically complete for finding continuous executions to achieve the LTL task. Later chapters of the thesis will show that this result can be extended to the manipulation domain.

3.3.5 Comparison with Other TMP Frameworks

There are a number of other approaches [1, 2, 5–18] for solving various cases of the TMP problem. One approach is decoupled planning [14, 15]. In decoupled planners, all motion plans that might potentially be needed are computed first, by providing each planning query a set amount of time. Then the task planner is called to select motions from the queries that returned with success. Another approach is hierarchical planning [5–13, 17, 18], where the task planner is first called to find a sequence of desirable actions. Then the motion planner is given some time to find a continuous implementation for each action. If an action is not found, then the task planner is

called again with that instance of the action removed.

In both of these approaches, each motion planning query is only considered once. If the underlying motion planner is not complete, as is the case with sampling-based motion planners, then these frameworks may rule out an achievable action due to not giving the motion planner enough time, causing the framework to miss a solution. Some frameworks [17] can guarantee probabilistic completeness in the case where no dead-end states exist and the actions follow uniformity requirements. However, in the general case, these frameworks cannot provide the probabilistic completeness guarantee proven for the synergistic framework.

Traditional manipulation planning frameworks such as aSyMov [1, 11] solve the manipulation problem (using possibly multiple robots) by first constructing a roadmap in the continuous space for each robot as well as the objects. Then the planner composes these roadmaps, and looks for a path on this composite roadmap to reach the goal. If no path to goal is found, the planner picks a roadmap to extend upon, and search is performed again. These frameworks are probabilistically complete. However, when the number of objects and robots increase, composing the roadmaps becomes expensive. This prevents the framework from expanding to larger problems.

Chapter 4

Manipulation Planning with the Synergistic Framework

Though the synergistic framework had much success in the navigation domain, it cannot be directly applied to manipulation problems. The biggest problem is that the decomposition employed is no longer suitable for manipulation problems. The workspace decomposition worked well for navigation problems because it captured where the propositional regions are through the labelling of the decomposition cells, and the connectivity of the space through the decomposition. In manipulation, however, using such workspace decomposition presents a problem. The atomic propositions are no longer over the location of the robot, so decomposing the workspace does not reveal the location of the atomic propositions.

One may suggest that instead of decomposing the robot workspace, decomposing the joint configuration space of the robot and the objects could be considered. By including the objects, this decomposition would capture the atomic propositions. However, such decompositions would still fail to capture the connectivity of the space. Even if two decomposition cells are adjacent, we could not say that the two cells are connected. For example, if two cells correspond to the the same robot states, but slightly different object configurations, transitioning between these two cells could in

fact be very difficult, as complex motion for the manipulator could be required to move the object.

Thus to use the synergistic framework for manipulation problems, we need a new abstraction, one that could capture the atomic propositions of manipulation problems, as well as capture the connectivity of manipulation domains. This chapter will detail a novel manipulation abstraction to handle this problem (Section 4.1), as well as modifications to the synergistic framework to accommodate this new abstraction (Section 4.2). This work was presented at ICRA 2015 [49], and is our first effort to task and motion planning for manipulation problems. Potential improvements and future work will be discussed in Chapter 6.

4.1 The Manipulation Abstraction

Similar to the workspace decomposition, the manipulation abstraction is also a graph, with nodes labelled with the atomic proposition true that those nodes. However, each manipulation abstraction node contains more information than the workspace decomposition. A manipulation node is a tuple $(\mathsf{obj}_1\mathsf{Loc}, \mathsf{obj}_2\mathsf{Loc}, \ldots, \mathsf{eeLoc}, \mathsf{grpObj}, \mathsf{action})$.

1. $\mathsf{obj}_i\mathsf{Loc}$, location of the ith object. The location of each object in a node can be either one of the locations of interest $\mathcal{L}$, or inside the gripper.

2. eeLoc, location of robot end effector. As discussed in Chapter 2, we will only be considering a single manipulator arm, so only a single variable is used for the end effector location. The robot end effector can be one of any nodes in the

location graph, including the locations of interest $\mathcal{L}$. The location graph used in our case studies is shown in Figure 4.1. In this location graph, the locations of interests are connected by a single intermediate area. Note that because objects could only be placed in the locations of interest, the intermediate area never contains a label. In general, the location graph could be expanded to better capture the locality of the space.

3. grpObj, object in the end effector. The abstraction node also records the object in the robot end effector. Though this information could be inferred from the locations of all objects, in practice it saves time to record this information rather than search through the object to find which, if any, object is in the gripper.

4. action, the action being performed. The abstraction node also maintains the action being performed. In the current framework, there are four actions. The GRASP and PLACE actions represent picking up an object from a location and dropping an object down at a location. The HOLD and MOVE actions represent moving the robot arm with and without an object in the gripper, respectively.

The four actions form an action graph, as seen in Figure 4.2. The action graph follows the general intuition of manipulation that if a robot has just picked up an object with a GRASP, then the robot can HOLD the object to another place, and afterwards can continue to HOLD , or PLACE it down at another location, etc. From the action graph, we find the edges of the manipulation abstraction. Each edge (v, v') in the abstraction must satisfy that $(v.\mathsf{action}, v'.\mathsf{action})$ is an action in the

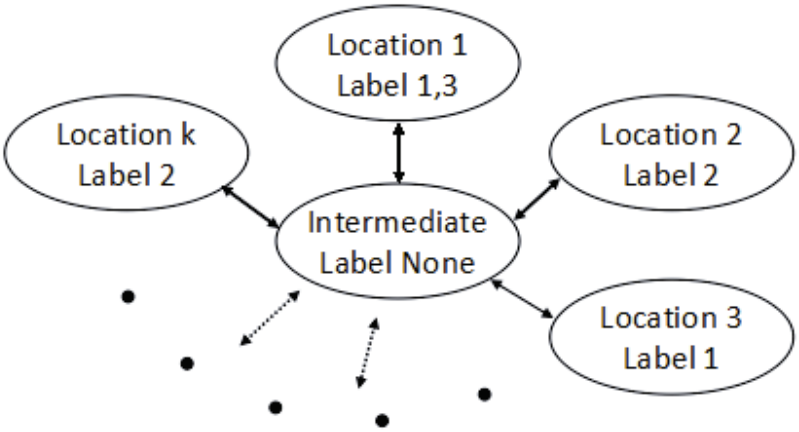


Figure 4.1 : An example of the location graph used in this thesis. The locations are connected through a single intermediate area. As mentioned in Section 2.1, each location has a set of labels associated with it.

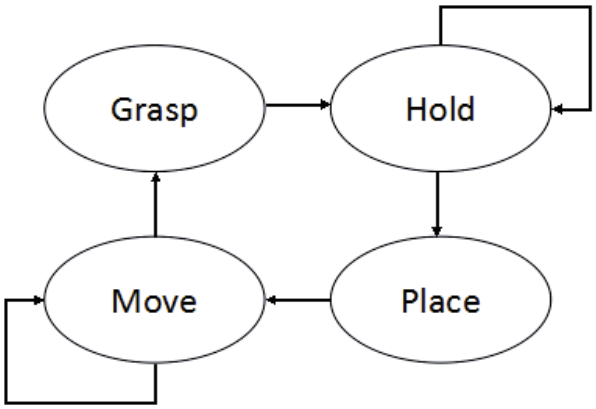


Figure 4.2 : The action graph used in this thesis. Each case for transition in the abstraction relates to an edge on the action graph.

action graph. In addition, for the edge in the action graph, the induced edge in the abstraction must satisfy the following, where unmentioned variables remain the same.

1. $v.\mathsf{action} = \textsc{grasp}$, $v'.\mathsf{action} = \textsc{hold}$. In this case, there must be some i such that $v.\mathsf{obj}_i\mathsf{Loc} = v.\mathsf{eeLoc}$. This would result in $v'.\mathsf{grpObj} = i$ and $v'.\mathsf{obj}_i\mathsf{Loc}$ is the gripper.

2. $v.\mathsf{action} = \textsc{hold}$, $v'.\mathsf{action} = \textsc{hold}$. In this case, all variables remain the same, except for $v'.\mathsf{eeLoc}$, which could take any value adjacent to that of $v.\mathsf{eeLoc}$ in the location graph.

3. $v.\mathsf{action} = \textsc{hold}$, $v'.\mathsf{action} = \textsc{place}$. This edge exists as long as $v.\mathsf{eeLoc}$ is in one of the locations of interest.

4. $v.\mathsf{action} = \textsc{place}$, $v'.\mathsf{action} = \textsc{move}$. In this case, $v'.\mathsf{grpObj}$ will be empty. Let i be the object such that $v.\mathsf{grpObj} = i$, then $v.\mathsf{obj}_j\mathsf{Loc}$ must not be $v.\mathsf{eeLoc}$ for any $j \neq i$. Finally, $v'.\mathsf{obj}_i\mathsf{Loc} = v.\mathsf{eeLoc}$.

5. $v.\mathsf{action} = \textsc{move}$, $v'.\mathsf{action} = \textsc{move}$. In this case, all variables remain the same, except for $v'.\mathsf{eeLoc}$, which could take any value adjacent to that of $v.\mathsf{eeLoc}$ in the location graph.

6. $v.\mathsf{action} = \textsc{move}$, $v'.\mathsf{action} = \textsc{grasp}$, This edge exists as long as $v.\mathsf{eeLoc}$ is in one of the locations of interest.

Figure 4.3 shows a fragment of a possible manipulation abstraction, where only one object is being considered. We see that from the GRASP node, because the robot

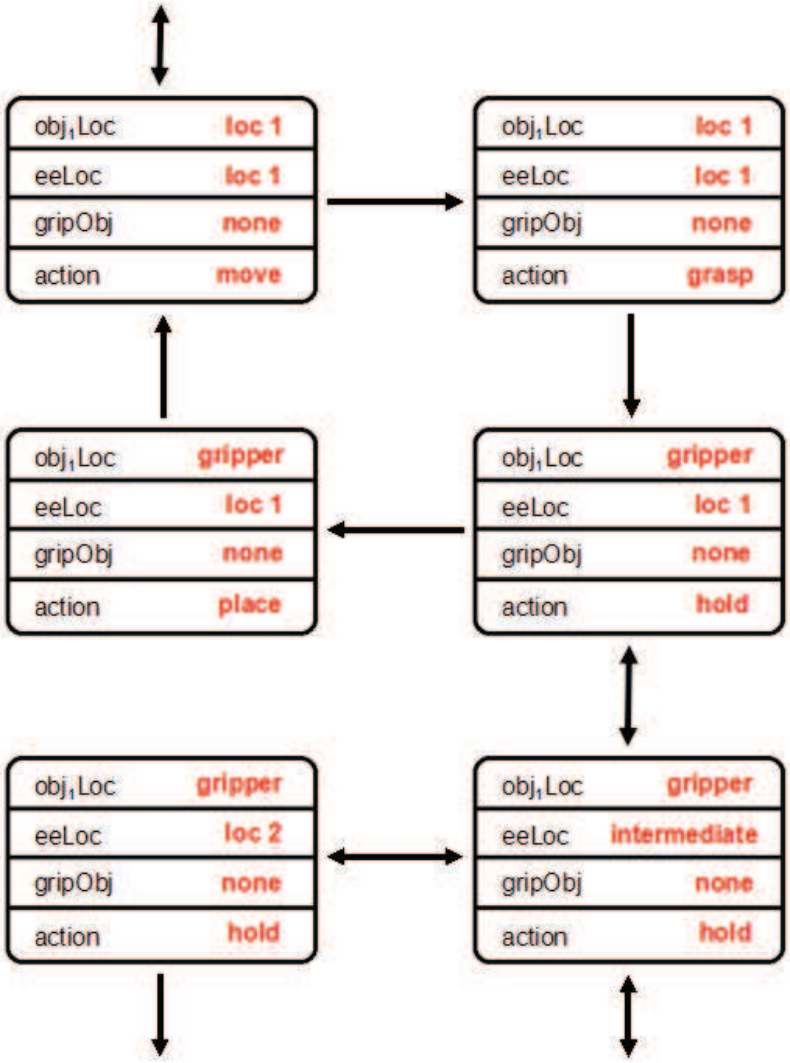


Figure 4.3 : A portion of an example manipulation abstraction. There is only one object in the scene. In practice, this abstraction is stored as an implicit graph, and the task planner generates the abstraction nodes as needed.

end effector is at the same place with the object, we can transition into a HOLD node, where the object is in the gripper. From that point, we could go into a PLACE node to drop the object back down, or transfer the object elsewhere by continuing with other HOLD nodes.

Note that the abstraction graph itself does not guarantee that each node is consistent. One could potentially construct an abstraction node that violates physical constraints. For example, if $v.\mathsf{obj}_i\mathsf{Loc}$ is not the gripper, but $v.\mathsf{grpObj} = i$, this causes a conflict in the physical space, as the object must either be in the gripper or not in the gripper. However, if one starts with a physically consistent node, by following the edges, a physically inconsistent node will never be reached. By maintaining the abstraction as an implicit graph, as long as the initial node is consistent, we ensure that all the nodes that we consider are also consistent.

By using this new abstraction, we can capture the atomic propositions of the manipulation problem. Recall that the atomic propositions reason over whether a given object is at a location with a given label. Using the abstraction graph, the atomic propositions that hold in any given node can be found by examining the labels of the location of each object. We also capture the connectivity of the space. The manipulation abstraction as a graph encodes all the possible ways the robot can manipulate the objects, so that a path on the abstraction corresponds to a potential execution in the physical space.

4.2 Modifications to the Synergistic Framework

By replacing the workspace decomposition with the manipulation abstraction in the synergistic framework, we can now attempt to plan for manipulation problems. However, the synergistic framework was designed for navigation problems that used the workspace decomposition. Recall from Section 3.3 that the task planner produced a guide, which is a sequence of decomposition cell-DFA state pairs (v, z), such that if the robot traverses the cells v, the task would be accomplished. With the manipulation abstraction, the guide is now a sequence of abstraction node-DFA state pairs. Thus continuous motion plans must be found for achieving the actions encoded by the sequence of abstraction nodes. In order to do this, we must modify the planning layers to find continuous motion plans for the abstraction nodes.

4.2.1 Precomputed Grasp and Place

One assumption we make in the manipulation domain relates to the actions GRASP and PLACE. These actions raise two problems in continuous execution. First, grasping and placing are very difficult to plan continuous trajectories for, as they require the system to reach a goal state that is very close to collision. In the case of GRASP, the robot end effector must be able to close on the object without previously colliding with the object, and in the case of PLACE, the same must be true between the object and the supporting surface. Additionally, even if a plan is found, the uncertainties in execution often cause the trajectory not to be executed perfectly, resulting in collision

before GRASP and PLACE could be fully performed.

In robotic systems, this problem is often dealt with by using designated controllers for grasping and placing of objects. These controllers may involve encoder, visual, or force feedback to be robust enough to handle uncertainties in the physical space. These controllers generally do not consider a model of the global environment, so they could not be invoked locally when the end effector is close to the desired location. However, if the end effector is within some proximity of the object, generally known as the *pre-grasp*, or *pre-image*, then the controller can reliably perform the desired motion, and end in a configuration known as the *post-image*.

In general, finding such controllers is a manageable task. This thesis used the Baxter robotic system for experiments, and visual servoing was used to perform the GRASP and PLACE actions near the goal position. Chapter 5 will demonstrate the efficacy of these controllers through experiments performed on the physical Baxter. For the PR2 robot in simulation, we assume the objects attach to the gripper perfectly.

4.2.2 Packaging Motion Planning Queries

If controllers already exist for the GRASP and PLACE nodes, all that remains to be found are continuous executions for the HOLD and MOVE actions. After receiving the guide from the task planning layer, the synergistic layer first extracts the segments that consist of only HOLD and MOVE nodes. For each such segment, the synergistic layer creates a motion planning query from the post-image of the previous action, to

the pre-image of the next action. This is reasonable as in the location graph, all the locations are connected through an intermediate area. Thus, any motion through the space respects the abstraction, so long as the start and goal configuration are in the designated regions.

In the original synergistic framework, even if a cell has already been explored, and solutions have already been found to transition to the next cell, this transition may be reconsidered in further explorations. This is because navigation problems involve complex dynamics, therefore finding new ways to move between cells provides diversity in possible future moves. In the manipulation setting, however, because the GRASP and PLACE controllers would cause the robot to always end in the same post-image, finding multiple solutions for a sequence of HOLD or MOVE does not affect the starting state of the next segment. Therefore, in the manipulation setting, we query the motion planner with the HOLD and MOVE segments in their order in the guide. When a solution is found, the segment is stored, so that if the segment reappears in a future guide, the plan can be recalled rather than replanned for.

4.2.3 A Different Weighing Scheme

As explained in the previous section, in the manipulation setting, it is no longer beneficial to reconsider segments that are already solved. Therefore, the weighing system in the synergistic framework needs to be reconsidered. Initially, we set the transitions involving GRASP and PLACE nodes to be 0, and all other segments to be 1.

If a segment has been queried, but the motion planner timed out on finding a solution, then the segment is likely to be very difficult to plan for, or even infeasible. In this case, we increase the weight (cost) in the abstraction for all transitions involved in that segment, proportional to the amount of time spent on the segment by the motion planner. On the other hand, if a motion plan is successfully found, the weights for all transitions in that segment are reduced to 0.

Intuitively, by setting the weight of GRASP and PLACE to 0 and reducing the weight of abstraction transitions already found, the task planner considers taking these actions to be free in future iterations of planning. By doing this, the task planner is encouraged to take advantage of these already found plans if possible, to quickly find a solution. One could consider the total weight of a path to be an estimation on the amount of effort remaining that is required to find a continuous implementation for the entire guide.

4.2.4 Probabilistic Completeness

Similar to the original synergistic framework, the new manipulation planning framework still considers and invokes the motion planner for all the transitions in the abstraction graph infinitely often, if a solution has not been found. For completeness, however, because the GRASP and PLACE actions are precomputed, we assume that the GRASP and PLACE actions capture all possible ways to grasp and place the objects.

The assumption on GRASP and PLACE is an important one. To assume GRASP and

PLACE capture all possible ways to perform the corresponding actions, two properties must be ensured. The first property is that if the grasping of an object could be performed from any grasp configuration, it could be performed using the precomputed GRASP action. This property could potentially be violated if an object is oddly shaped, and requires being manipulated from a certain direction that the precomputed action does not capture. The second property is that all possible configurations of the robot after the placement of an object are connected in the configuration space. This ensures that the precomputed PLACE action does not cause a situation where the placement of the object restricts the set of possible locations for the robot to reach in the following action.

Given this assumption, we guarantee that as long as a solution has not been found, the framework will reconsider every possible MOVE and HOLD action. Because the motion planners are probabilistically complete, and the GRASP and PLACE actions capture all the ways to grasp and place the objects, the new framework for manipulation remains probabilistically complete.

Chapter 5

Experimental Results

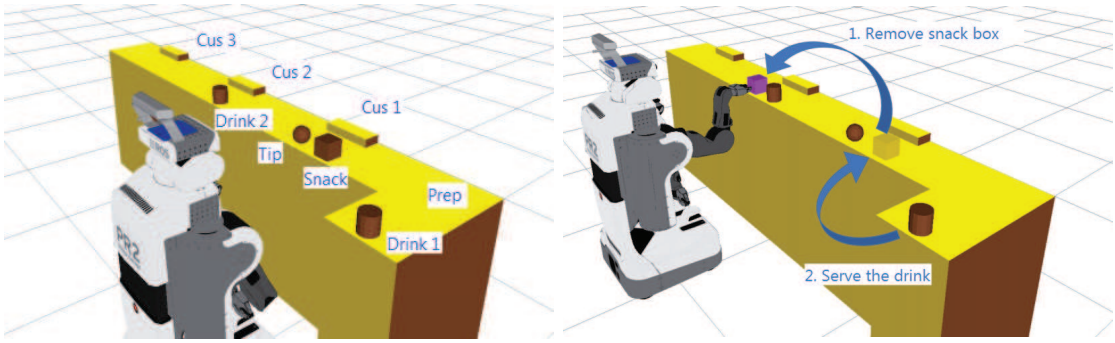
5.1 Implementation

The modified synergistic framework was implemented as a part of the open motion planning library (OMPL) [50]. The conversion from the LTL task to DFA was performed using spot [48]. Motion planning was performed using the KPIECE planner [45] implemented in OMPL, through the ROS MoveIt! package [51]. We constructed two scenarios to test the modified framework.

5.2 Case Study: PR2 in Simulation

The first scenario studied was a PR2 robot in simulation, presented in [49]. In this scenario, the robot is tasked with serving customers at a bar. The scene involved locations and labels for a preparation area, as well as for an adjustable number of customers. The scene contains a number of objects: cups for drinks, a snack box, and a tip jar.

Figure 5.1 shows one example in this case study. In this experiment, three customer regions are present, marked by the flat boxes at the edge of the bar. The robot is allowed to use only its right arm. Because the bar scene is very long, the robot



(a) (b)

Figure 5.1 : The PR2 case study. (a) Experimental setup. The customer locations are marked with a horizontal bar. Initially, both serving locations for customer 1 are occupied. (b) The robot intelligently chooses to remove the snack box to make way for the drink.

must move its base to reach different portions of the workspace. In order to quickly plan for these motions, we first use sampling based motion planning to move the base into the vicinity of the location the end effector needs to reach, and then plan for the arm to grasp or place the object. Since this scenario is in simulation, the GRASP and PLACE actions are performed simply by making calls to the planning environment to attach and detach objects to the manipulator. The tasks examined were

Serve customer 1 with drink number 1.

$$\varphi_1 = \mathcal{F}(o_{\text{drink1}}, l_{\text{cus 1}})$$

Though this is a simple task, we complicated the problem by setting the initial location of the other objects to occupy all serving locations labelled with customer 1, as seen in Figure 5.1(a). The robot execution is shown in Figure 5.1(b). Even

though no extra information was given to instruct the planner of the need to do so, the planning framework is able to, through task and motion planning, recognize that the locations for customer 1 are occupied, and chooses to first remove an object in order to achieve the task.

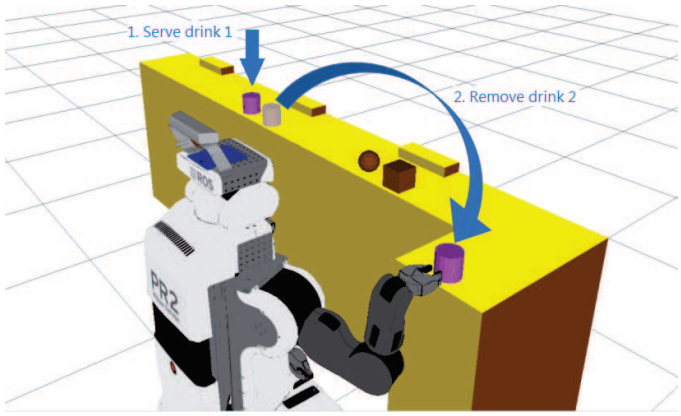
Note that often times the robot has a choice in the order to achieve the task. In this following example, the robot is asked to perform the following task.

Serve customer 2 with drink 1, and remove the empty drink 2. Then, offer snacks to each customer, and ask for tip from those served.

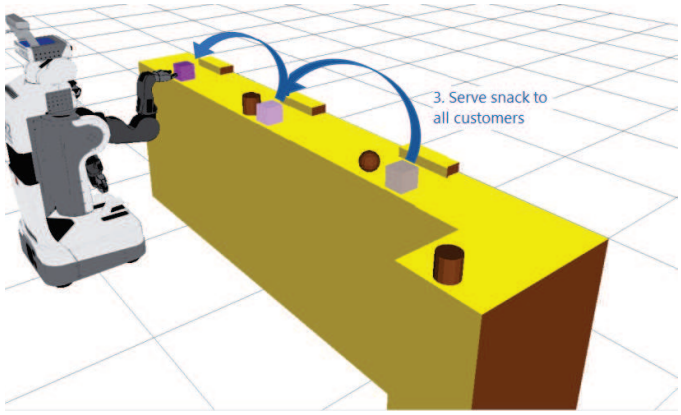
$$\varphi_2 = \mathcal{F}((o_{\text{drink1}}, l_{\text{cus 2}}) \wedge (o_{\text{drink2}}, l_{\text{prep}}) \bigwedge_{i=1}^{k} \mathcal{F}\big((o_{\text{snack}}, l_{\text{cus i}}) \wedge \mathcal{F}(o_{\text{tipjar}}, l_{\text{cus i}})\big))$$

For this task, the robot has a choice as to the order the objective is completed. After serving the drinks, the robot could offer snacks to each customer, and after all customers get snacks, the robot will get the tip. Alternatively, the robot could offer snacks to one customer, and immediately ask for tip, then go on to the next customer. Through task and motion planning, the framework realizes that the latter is difficult, as the room in front of customer 2 is limited due to the presence of drink 2. We can see in Figure 5.2, that the robot picks the former order to achieve the task.

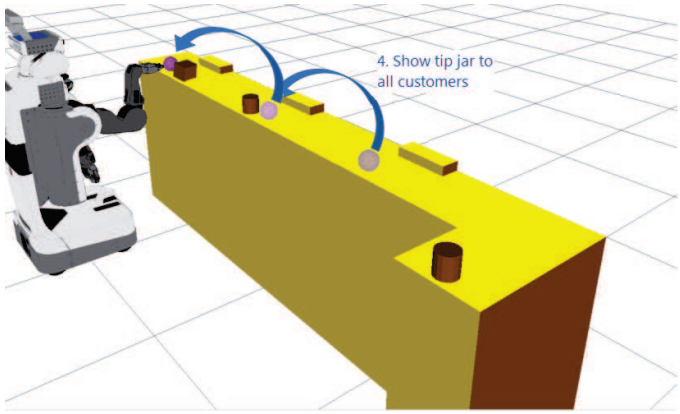
Finally, we ask the planner to plan for the following task.



(a)



(b)



(c)

Figure 5.2 : The second PR2 task φ_2. The starting configuration is identical to Figure 5.1(a). (a) The robot first serves drink 1 and retrieves drink 2 back to the preparation area. (b) The robot decides to first serve snacks to all customers. (c) The robot finishes the task by presenting the tip jar to all customers.

Serve snacks to all customers.

$$\varphi_3 = \bigwedge_{i=1}^{k} \mathcal{F}\big(o_{\text{snack}}, l_{\text{cus i}}\big)$$

For this task, we vary the number of customers. Each time a new customer is added, two new locations and an extra label are added for that customer. The added number of locations causes an increase in the size of the abstraction, while the added number of labels causes in increase in the size of the specification. Note that the geometries of this scene are constructed differently from the previous examples, to facilitate the addition of new locations. The runtime results of this example will be shown in Section 5.4.

5.3 Case Study: Baxter Robot

The second scenario studied is the Baxter robotic system. This environment involved 6 locations with 3 different labels: two labels for two customers and one for a trash area, as seen in Figure 5.3. The task performed is

Trash the empty can, then serve the snack to both customers, and ask for tip from those served.

$$\varphi_4 = \mathcal{F}((o_{\text{can}}, l_{\text{trash}}) \wedge \mathcal{F}\big((o_{\text{snack}}, l_{\text{cus 1}}) \wedge \mathcal{F}(o_{\text{tipjar}}, l_{\text{cus 1}})\big)$$
$$\wedge \mathcal{F}\big((o_{\text{snack}}, l_{\text{cus 2}}) \wedge \mathcal{F}(o_{\text{tipjar}}, l_{\text{cus 2}})\big))$$

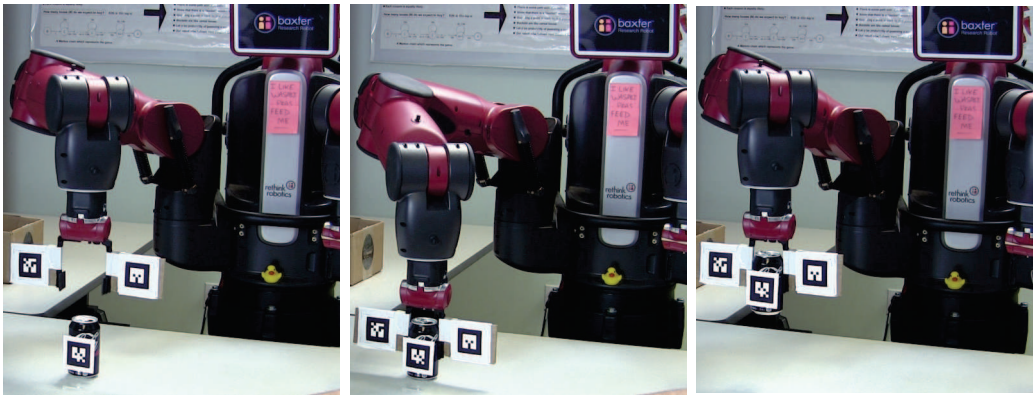
(a) (b)

(c) (d)

(e) (f)

(g) (h)

Figure 5.3 : Baxter performing task φ_4. (a) Experimental setup. The Black can in front is the empty can to be trashed. Green jar on the left is the snack box. Shorter orange jar in the back is the tip jar. Area A is for customer 1. Area B is for customer 2. Box labelled with C is the trash box. (b) The robot starts by reaching for the can. (c) Robot drops the can in the trash area. (d,e) Snacks are being brought to customer 2, then customer 1. (f) Robot grabs the tip jar. (g,h) Tip jar shown to customer 1, then 2, completing the task.



(a) Pre-image (b) Grasping (c) Post-image

Figure 5.4 : Baxter performing a GRASP action to pick up a can. The tags on the robot gripper and object are used for visual feedback control.

which is similar to the second PR2 task. Figure 5.3 shows screenshots from an actual execution. Similar to the PR2, the robot is restricted to using its right arm only. Because the Baxter does not have a mobile base, motion planning is directly performed on the manipulator chain from the shoulder of the robot to the gripper. The robot successfully achieves the given task, by first trashing the can, then offering snacks to both customers, and finally presenting the tip jar to both customers.

Figure 5.4 shows an instance of the the robot using the assumed controller for a GRASP action. Recall that the framework assumes that for GRASP and PLACE actions, these controllers could secure the grasping and placing of the objects, given that the end effector is within the pre-image of the controller. Figure 5.4(a) shows the pre-image of the GRASP instance. From that point, the tags on the object and the end effector are used for visual feedback control to secure the grasp, and finally the object is grasped, and the end effector moves to a known post-image. Because

Scenario	Task	$\lvert\mathsf{obj}\rvert$	$\lvert\mathcal{L}\rvert$	$\lvert A_\varphi\rvert$	$\lvert\mathcal{P}\rvert$	$T_{\text{task}}(s)$	$T_{\text{motion}}(s)$
PR2-1	φ_1	4	8	2	44,100	2.76	12.32
PR2-1	φ_2	4	8	27	75,511	33.12	31.15
PR2-2	$\varphi_3, k=2$	4	6	4	30,888	1.56	0.65
PR2-2	$\varphi_3, k=3$	4	8	8	273,564	17.06	0.87
PR2-2	$\varphi_3, k=4$	4	10	16	1,692,432	119.46	1.05
Baxter	φ_4	3	6	27	19,370	0.94	0.70

Table 5.1 : Runtime results for the manipulation synergistic framework. The k value for φ_3 denotes the number of customers assumed in the scene. $\lvert\mathsf{obj}\rvert$ and $\lvert\mathcal{L}\rvert$ are the number of objects and locations, respectively. $\lvert A_\varphi\rvert$ is the number of states in the DFA, used as an indicator of complexity of task. $\lvert\mathcal{P}\rvert$ is the number of product graph nodes generated during task planning. T_{task} and T_{motion} are the amount of time the framework used in task and motion planning, respectively. Times are recorded with two timers run simutaneously to monitor both components. Times are averaged over 50 runs.

the controllers use visual feedback, they are robust enough to account for variations in the position of the object, even though humans interact with the objects as well. The MOVE and HOLD actions are performed by following the trajectories generated by the motion planner. The robot successfully avoids collision with the environment by following these trajectories.

5.4 Framework Runtime

Table 5.1 shows the runtime of the algorithm as well as the number of product graph nodes generated for the set of scenarios created. Even though task planning and motion planning are interleaved, we measure both times to demonstrate the scalability of the framework. The number of objects, locations indicate the complexity of the manipulation domain, while the number of states in the DFA indicates the complexity

of the task.

The total amount of motion planning time is determined by many factors, including the number of actions that require planning, and the complexity of the physical space, which relates to how frequently motion planning times out. We do see from Table 5.1 that for the same scenario, when the task is more complex, motion planning in general takes longer. This is largely due to a higher depth of the solution, i.e., a larger number of actions required for the solution, which causes the number of motion planning queries to increase overall. However, comparing across the difference scenarios, it appears the motion planning time is highly dependent on the physical space the robot is working within, as a more difficult environment causes the motion planning queries to take longer to solve, or timeout and return failure.

Task planning, on the other hand, is performed by running Dijkstra's algorithm through the product between the abstraction and the DFA generated from the task. When more objects and locations are added, the number of reachable nodes in the abstraction and the degree of the abstraction increase significantly. In general, the number of reachable nodes in the abstraction is

$$2|L|P_{|L|+1}^{|O|}$$

where $|O|$, $|L|$ are the number of objects and number of locations, respectively, and P_n^k denotes the number of k-permutations of n elements. This permutation number is a very fast growing function. As can be seen from the three test cases with φ_3, the

number of objects remains the same, while the number of locations and size of the formula grow linearly. This causes an exponential growth in the number of automaton states, as well as the size of the reachable portion of the abstraction.

The construction of the product graph is performed on-the-fly, i.e., a product graph node is constructed only when it is visited. Thus even though the number of automaton states and the size of the automaton grow very quickly, the size of the product graph could potentially grow less significantly. In this thesis, Dijkstra's algorithm is used to search this product graph, which proceeds in a breadth-first manner. In the end, as shown in Table 5.1, this causes the product graph to grow significantly, causing a drastic increase the task planning time. In experiments where 5 customers were used for the φ_3 example, the planner failed within a 10-minute timeout, without completing a single iteration of task planning. Therefore, searching the product graph is currently the bottleneck of scaling the framework to larger problems. Improving the speed of this search is a key point of future work.

5.5 Manipulation Abstraction vs Naive Abstraction

As runtime is affected by the abstraction, it is important to consider the choice of abstraction constructed for this problem. In addition to the manipulation abstraction introduced in Section 3.1, a naive abstraction was also implemented for purposes of comparison. This naive abstraction was inspired by domain specification languages such as PDDL. In this abstraction, the nodes only contained information regarding the

		Manip Abstraction		Naive Abstraction		
$\|\mathsf{obj}\|$	$\|\mathcal{L}\|$	Time (s)	$\|\mathcal{P}\|$	Time (s)	$\|\mathcal{P}\|$	% Speed Up
3	6	.883	19,370	1.12	8,941	26.8
3	8	2.28	66,164	4.84	31,137	71.6
3	10	7.51	168,966	15.5	80,461	106.4
3	12	16.2	360,800	39.8	173,185	145.7
4	6	4.03	77,246	5.34	35,653	32.5
5	6	12.6	228,540	16.8	105,481	26.8

Table 5.2 : Runtime for a single run of task planner for manipulation and naive abstractions. The task used is the Baxter task. Average runtime computed over 50 runs. $|\mathsf{obj}|$, $|\mathcal{L}|$, and $|\mathcal{P}|$ denotes the number objects, number of locations, and number of nodes explored in the product graph, respectively.

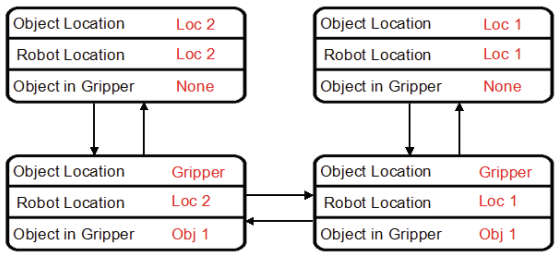


Figure 5.5 : A portion of a possible the naive abstraction with only one object. This abstraction is more compact, at the cost of a higher average degree for this graph.

locations of the objects and the robot, but not the action being performed. Also, this abstraction does not use a location graph, and thus does not have an intermediate area. Instead, this abstraction assumes transitions of the robot between any two locations of interest. A fragment of a possible naive abstraction is shown in Figure 5.5.

Intuitively, this naive abstraction captures the state of the robot and the objects like the manipulation abstraction, but does not capture the robot action being performed or any locality of the space. Because the abstraction no longer provides information on the action being performed, the guide also does not contain this information. Thus the work of extracting the correct action from the guide is given to the synergistic layer.

Our experiments show that this abstraction could also be used to plan for manipulation problems. Additionally, by examining the guide generated by the first iteration of task planning, we found that, when weights are at initial values, we found that the guide found by the task planning using the two different abstractions corresponded to the same actions. Thus we focus our comparison on the runtime of the task planner when the two different planners are used. Table 5.2 shows the number of product graph nodes generated as well as the average time used by the task planner to find one guide. The experiments were performed using the task from the Baxter scenario, where the baseline is 3 objects in 6 locations. Adding locations to the baseline gave the framework more choices as to intermediate locations to drop objects, which affects

the size and average degree of the abstraction. Adding objects irrelevant to the task made the problem more difficult as objects may need to be moved out of the way for a task, causing the search to be deeper. As we can see, in all cases, the manipulation abstraction generates more nodes in the product graph, but the search for a guide is faster.

The manipulation abstraction, by containing more information on the nodes, has several effects on task planning. First, including more information on the nodes causes the number of possible nodes to increase, thus the task planner generates more product graph nodes when using the manipulation abstraction. On the other hand, by incorporating information regarding the action, it is faster to find the successors of a given node. Also, because nodes with the same location for the objects and the gripper but different action are differentiated, and an intermediate region is used, the average degree of the abstraction graph is less than that of the naive abstraction.

The runtime of Dijkstra's algorithm depends on the number of nodes expanded, the number of edges considered, and the cost of evaluating an edge. The combination of faster computation for successors and less edges considered due to a lower degree overcomes the higher number of nodes explored, resulting in a faster search for the manipulation abstraction than a naive abstraction, at the expense of using more space. This trade off is magnified when the degree of the abstraction is larger, as shown in Table 5.2, as the speed gain in using the manipulation abstraction is higher for problems with more locations.

As Table 5.2 shows, task planning runtime scales poorly with the number of objects and locations for both abstractions. To improve the scalability of the framework, the search technique used and the abstraction need to be reconsidered. This is an important aspect of future work.

Chapter 6

Conclusion and Future Work

In this thesis, we present a series of modifications to the synergistic framework to enable task and motion planning using this framework. The modifications include the manipulation abstraction that captures the manipulation problem, as well as modifications to the weighing system of the synergistic framework. We show through two case studies that this framework can effectively plan for real manipulation problems. We also show that the manipulation abstraction outperforms a naive abstraction for all the cases studied.

A component that could be investigated in the future is the location graph discussed in Section 4.1. Currently, the location graph only consists of the locations of interest and a single intermediate area. It remains to be investigated if using additional areas to partition the workspace will help guide motion planning and reduce overall planning time. A more complex location graph could also enable a new weighing scheme that can generalize the difficulty of a particular instance of an action to other similar actions, by recording the difficulty of motion planning in the location graph.

One area to investigate is the methods used to search the product graph for task planning. As discussed in Chapter 4, task planning is currently performed by

running Dijkstra's algorithm on the product graph. We see in Section 5.4 that this causes the task planner to generate a large number of product graph nodes, and is currently the bottleneck for scaling to solve larger problems. Due to the breadth-first behavior of Dijkstra's algorithm, increases in number of objects, number of locations, and size of specification all directly translate to an exponential increase in runtime. By employing more advanced techniques discussed in Section 3.1, we hope to avoid searching in parts of the product graph that are less likely to generate leads. This could mitigate the growth in the product graph, leading to faster task planning.

In this thesis, the weighing scheme used in the product graph of the synergistic framework was only studied to the extend of encoding the successfulness of motion planning queries. The weighing scheme could be improved by encoding more detailed information regarding the exploration data from the motion planner, such as the coverage of the workspace. This has been implemented for navigation problems described in [3, 4].

One assumption being made in this thesis is that the robot is considered to be a single manipulator. In many settings, such the the PR2 robot and Baxter robot presented here, the robot has more than one manipulator arm. It is desirable to utilize all possible ways for the robot to manipulate objects. Actions could be performed in parallel using multiple manipulators, or multiple manipulators could collaborate to achieve a single action that is previously unachievable. However, it is challenging to plan for such behaviors, due to the need to coordinate between the different manip-

ulators. We would like to investigate methods to tackle these challenges and utilize the full physical capabilities of the robotic platforms.

Even though the planning queries requested by the synergistic layer are different depending on the action being performed, the queries are still related. The manipulator and the nonmanipulable obstacles are the same for each query. Thus motion planning data from one query could potentially be used for other queries. One possibility is to reuse the samples from previous planning queries, and lazily evaluate the validity of the samples and the edges between them. This could potentially speed up motion planning, especially for scenarios where motion planning is difficult.

Bibliography

[1] S. Cambon, R. Alami, and F. Gravot, "A hybrid approach to intricate motion, manipulation and task planning," *The International Journal of Robotics Research*, vol. 28, no. 1, pp. 104–126, 2009.

[2] K. Hauser and J. Latombe, "Integrating task and prm motion planning: Dealing with many infeasible motion planning queries," *International Conference on Automated Planning and Scheduling*, no. 1, pp. 34–41, 2009.

[3] A. Bhatia, M. Maly, L. E. Kavraki, and M. Y. Vardi, "Motion planning with complex goals," *IEEE Robotics & Automation Magazine*, vol. 18, pp. 55–64, Sep. 2011.

[4] E. Plaku, L. E. Kavraki, and M. Y. Vardi, "Motion planning with dynamics by a synergistic combination of layers of planning," *IEEE Transactions on Robotics*, vol. 26, no. 3, pp. 469–482, 2010.

[5] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 1470–1477, 2011.

[6] L. P. Kaelbling and T. Lozano-Pérez, "Integrated task and motion planning in belief space," *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1194–1227, 2013.

[7] T. Lozano-Pérez and L. P. Kaelbling, "A constraint-based method for solving sequential manipulation planning problems," in *Int. Conf. on Intelligent Robots and Systems*, pp. 3684–3691, 2014.

[8] C. Dornhege, P. Eyerich, T. Keller, S. Trüg, M. Brenner, and B. Nebel, "Semantic attachments for domain-independent planning systems," *Springer Tracts in Advanced Robotics*, vol. 76, no. STAR, pp. 99–115, 2012.

[9] E. Erdem, K. Haspalamutgil, C. Palaz, V. Patoglu, and T. Uras, "Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 4575–4581, 2011.

[10] C. R. Garrett, L. P. Kaelbling, and T. Lozano-Pérez, "FFRob : An efficient heuristic for task and motion planning," *WAFR*, pp. 179–195, 2014.

[11]

[12] W. N. N. Hung, X. Song, J. Tan, X. Li, J. Zhang, R. Wang, and P. Gao, "Motion Planning with Satisfiability Modulo Theories," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 113–118, 2014.

[13] L. Karlsson, J. Bidot, F. Lagriffoul, A. Saffiotti, U. Hillenbrand, and F. Schmidt, "Combining Task and Path Planning for a Humanoid Two-arm Robotic System," *Proceedings of TAMPRA: Combining Task and Motion Planning for Real-World Applications (ICAPS workshop)*, pp. 13–20, 2012.

[14] G. Konidaris, L. P. Kaelbling, and T. Lozano-Pérez, "Constructing Symbolic Representations for High-Level Planning," *AAAI*, pp. 1932–1940.

[15] S. Nedunuri, S. Prabhu, M. Moll, S. Chaudhuri, and L. E. Kavraki, "SMT-Based Synthesis of Integrated Task and Motion Plans from Plan Outlines," *International Conference on Robotics & Automation (ICRA)*, pp. 655–662, 2014.

[16] S. Srivastava, L. Riano, S. Russell, and P. Abbeel, "Using Classical Planners for Tasks with Continuous Operators in Robotics," *ICAPS Workshop on Planning and Robotics*, 2013.

[17] 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," *Proceedings - IEEE International Conference on Robotics and Automation*, no. M, pp. 0–7, 2014.

[18] J. Wolfe, B. Marthi, and S. Russell, "Combined task and motion planning for mobile manipulation," *ICAPS*, pp. 254–258, 2010.

[19] E. M. Clarke, O. Grumberg, and D. Peled, *Model checking*. MIT Press, 1999.

[20] A. E. Gerevini and D. Long, "Plan Constraints and Preferences in PDDL3," *Technical Report*, pp. 1–12, 2005.

[21] H. Kress-Gazit, G. Fainekos, and G. J. Pappas, "Where's waldo? sensor-based temporal logic motion planning," in *Int. Conf. on Robotics and Automation*, (Rome, Italy), pp. 3116–3121, IEEE, 2007.

[22] H. Kress-Gazit, T. Wongpiromsarn, and U. Topcu, "Correct, reactive robot control from abstraction and temporal logic specifications," *IEEE Robotics and Automation Magazine*, vol. 18, no. 3, pp. 65–74, 2011.

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

[24] C. Vasile and C. Belta, "Sampling-based temporal logic path planning," in *Int. Conf. on Intelligent Robots and Systems*, pp. 4817–4822, IEEE, Nov 2013.

[25] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon control for temporal logic specifications," in *Int. Conf. on Hybrid Systems: Computation and Control*, pp. 101–110, 2010.

[26] S. C. Livingston, R. M. Murray, and J. W. Burdick, "Backtracking temporal logic synthesis for uncertain environments," in *Int. Conf. Robotics and Automation*, pp. 5163–5170, IEEE, 2012.

[27] E. M. Wolff, U. Topcu, and R. M. Murray, "Automaton-guided controller synthesis for nonlinear systems with temporal logic," in *Int. Conf. on Intelligent Robots and Systems*, pp. 4332–4339, IEEE, 2013.

[28] O. Kupferman and M. Y. Vardi, "Model checking of safety properties," *Formal Methods in System Design*, vol. 19, pp. 291–314, 2001.

[29] M. Ghallab, D. Nau, and P. Traverso, *Automated planning: theory & practice*. Elsevier, 2004.

[30] M. Ghallab, C. Knoblock, D. Wilkins, A. Barrett, D. Christianson, M. Friedman, C. Kwok, K. Golden, S. Penberthy, D. E. Smith, *et al.*, "Pddl-the planning domain definition language," 1998.

[31] A. E. Gerevini, P. Haslum, D. Long, A. Saetti, and Y. Dimopoulos, "Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners," *Artificial Intelligence*, vol. 173, no. 5-6, pp. 619–668, 2009.

[32] S. Edelkamp, "Cost-optimal symbolic planning with state trajectory and preference constraints," *Frontiers in Artificial Intelligence and Applications*, vol. 141, pp. 841–842, 2006.

[33] H. A. Kautz, B. Selman, *et al.*, "Planning as satisfiability.," in *ECAI*, vol. 92, pp. 359–363, 1992.

[34] H. Kautz, D. McAllester, and B. Selman, "Encoding plans in propositional logic," *KR*, vol. 96, pp. 374–384, 1996.

[35] J. Rintanen, K. Heljanko, and I. Niemelä, "Planning as satisfiability: parallel plans and algorithms for plan search," *Artificial Intelligence*, vol. 170, no. 12, pp. 1031–1080, 2006.

[36] A. L. Blum and M. L. Furst, "Fast Planning Through Planning Graph Analysis," *Artificial Intelligence*, pp. 90:281–300, 1997.

[37] J. Hoffmann, "FF: The fast-forward planning system," *AI magazine*, vol. 22, pp. 57–62, 2001.

[38] M. Helmert, "The fast downward planning system," *Journal of Artificial Intelligence Research*, vol. 26, pp. 191–246, 2006.

[39] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion Theory, Algorithms, and Implementation.* Cambridge: MIT Press, 2005.

[40] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, pp. 566–580, Aug 1996.

[41] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," Tech. Rep. 98-11, Department of Computer Science, Iowa State University, Ames, IA, 1998.

[42] D. Hsu, J.-C. Latombe, and R. Motwani, "Path planning in expansive configuration spaces," *Intl. J. of Computational Geometry and Applications*, vol. 9, no. 4-5, pp. 495–512, 1999.

[43] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Int. Conf. on Robotics and Automation*, vol. 2, pp. 995–1001, IEEE, 2000.

[44] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *International Journal of Robotics and Research*, vol. 20, no. 5, pp. 378–400, 2001.

[45] I. A. Şucan and L. E. Kavraki, "A sampling-based tree planner for systems with complex dynamics," *IEEE Transactions on Robotics*, vol. 28, no. 1, pp. 116–131, 2012.

[46] L. E. Kavraki, M. N. Kolountzakis, and J. C. Latombe, "Analysis of probabilistic roadmaps for path planning," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 1, pp. 166–171, 1998.

[47] M. Y. Vardi and P. Wolper, "An automata-theoretic approach to automatic program verification," in *Proceedings of the Symposium on Logic in Computer Science (LICS)*, pp. 332–344, 1986.

[48] A. Duret-Lutz and D. Poitrenaud, "Spot: An extensible model checking library using transition-based generalized buchi automata," in *Int. Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*, pp. 76–83, IEEE, 2004.

[49] K. He, M. Lahijanian, L. E. Kavraki, and M. Y. Vardi, "Towards manipulation planning with temporal logic specifications," in *IEEE Conference on Robotics and Automation*, pp. 346–352, May 2015.

[50] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, pp. 72–82, December 2012.

[51] S. Chitta, I. Sucan, and S. Cousins, "Moveit!," *IEEE Robotics Automation Magazine*, vol. 19, no. 1, pp. 18–19, 2012.