Combining High-Level Causal Reasoning with Low-Level Geometric Reasoning and Motion Planning for Robotic Manipulation

by

Can Palaz

Submitted to the Graduate School of Sabancı University
in partial fulfillment of the requirements for the degree of
Master of Science

Sabancı University

August, 2011

Combining High-Level Causal Reasoning with
Low-Level Geometric Reasoning and Motion Planning
for Robotic Manipulation

APPROVED BY:

Assist. Prof. Dr. Volkan Patoğlu (Thesis Co-advisor)	[illegible]
Assist. Prof. Dr. Esra Erdem (Thesis Co-advisor)	[illegible]
Prof. Dr. Kemal İnan	[illegible]
Prof. Dr. Ali Rana Atılgan	[illegible]
Assist. Prof. Dr. Müjdat Çetin	[illegible]

DATE OF APPROVAL: 09.08.2011

© Can Palaz, 2011

All Rights Reserved

Combining High-Level Causal Reasoning with Low-Level Geometric Reasoning and Motion Planning for Robotic Manipulation

Can Palaz
ME, Master of Science, 2011
Thesis co-advisors: Assist. Prof. Dr. Volkan Patoğlu,
Assist. Prof. Dr. Esra Erdem

Keywords: Manipulation planning, task planning, motion planning, reasoning, artificial intelligence.

Abstract

We present a modular planning framework for manipulation tasks that combines high-level representation and causality-based reasoning with low-level geometric reasoning and motion planning. This framework features bilateral interaction between task and motion planning, and embeds geometric reasoning in causal reasoning. The causal reasoner guides the motion planner by finding an optimal task-plan; if there is no feasible kinematic solution for that task-plan then the motion planner guides the causal reasoner by modifying the planning problem with new temporal constraints. The geometric reasoner guides the causal reasoner to find feasible kinematic solutions by means of external predicates/functions. We show the applicability of this method on two sample problems: extended towers of Hanoi and multiple robot manipulation inside a maze.

We focus on two main problems in this planning framework: i) a systemic analysis of various levels of integration between high-level representation and causality-based reasoning with low-level geometric reasoning and motion planning and ii) generalization of the planning framework to continuous domains. For the former, we consider various levels of integration in the two domains mentioned above, to check which level of integration achieves better performance. For the latter, we abstract configurations at the representation level by continuous regions instead of discrete positions, and introduce an incremental sampling-based method coupled to a goal region-based probabilistic path planner for extracting specific goal configurations required for generating valid plans for execution. This way, we tightly integrate high-level reasoning and region-based motion planning and provide a general framework for addressing a wide spectrum of manipulation problems.

Üst-Seviye Nedensel Akıl-Yürütmenin Alt-Seviye Geometrik Akıl-Yürütme ve Hareket Planlama ile Robotik Manipülasyon için Kaynaştırılması

Can Palaz
ME, Yüksek Lisans Tezi, 2011
Tez Eşdanışmanları: Yrd. Doç. Prof. Dr. Volkan Patoğlu,
Yrd. Doç. Prof. Dr. Esra Erdem

Anahtar kelimeler: Manipülasyon planlama, görev planlama, hareket planlama, akıl-yürütme, yapay zeka.

Özetçe

Üst-seviye gösterim ve nedensellik tabanlı akıl-yürütme ile alt-seviye geometrik akıl-yürütme ve hareket planlamayı kaynaştıran modüler bir mimari sunulmuştur. Bahsi geçen mimaride görev planlama ve hareket planlama arasında iki-yönlü etkileşim bulunmakta ve geometrik akıl-yürütme, nedensel akıl-yürütmenin içine yerleştirilebilmektedir. Nedensel akıl-yürütücü, hareket planlayıcıyı eniyilenmiş bir görev planı bularak yönlendirmekte; şayet görev planını sağlayan uygulanabilir bir kinematik çözüm bulunamazsa, hareket planlayıcı, planlama problemini zamana bağlı kısıtlar ekleyerek değiştirmek suretiyle yönlendirmektedir. Geometrik akıl-yürütme ise nedensel akıl-yürütücüyü uygulanabilir çözümler bulması için dış fonksiyonlar yardımıyla yönlendirmektedir. Bu yöntemin uygulanabilirliği iki örnek problem üzerinde gösterilmiştir: Genişletilmiş Hanoi kuleleri bulmacası ve iki robotun bir labirent içindeki manipülasyonu.

Bu mimari kapsamında iki ana problem üzerinde yoğunlaşılmıştır: i) Üst-seviye gösterim ve nedensellik tabanlı akıl-yürütücü ile alt-seviye geometrik akıl-yürütme ve hareket planlamanın bütünleştirilmesinin sistematik bir çözümlemesi; ii) planlama mimarisinin sürekli alanlar için genellenmesi. İlk problem için, yukarıda bahsi geçen örnekler üzerinde, çeşitli seviyelerdeki bütünleştirmelerin hesaplama verimliliğini araştırılmıştır. İkinci problem için, gösterim seviyesinde, kesikli konumlar yerine sürekli alanlar kullanılarak önerilen mimari genellenmiştir. Ayrıca öne sürülen örnekleme tabanlı bir metot ile birlikte alan tabanlı bir hareket planlayıcı kullanılmıştır. Bu yolla, üst-seviye akıl-yürütme ile alan tabanlı hareket planlama bütünleştirilmiş ve geniş kapsamlı manipülasyon problemlerini ele alacak genel bir sistem sağlanmıştır.

Acknowledgements

It is a great pleasure to extend my gratitude to my thesis advisors Assist. Prof. Dr. Volkan Patoğlu and Assist. Prof. Dr. Esra Erdem for their precious guidance and support. I am greatly indebted to them for their supervision and excellent advises throughout my Master study. I would gratefully thank Prof. Dr. Kemal İnan, Prof. Dr. Ali Rana Atılgan, and Assist. Prof. Dr. Müjdat Çetin for their feedbacks and spending their valuable time to serve as my jurors.

I would like to acknowledge the financial support provided by Sabancı University through my Master education under Internally-funded Research Projects scholarship.

Many thanks to Kadir Haspalamutgil, Tansel Uras, Ozan Tokatlı, Alper Ergin, Ahmetcan Erdoğan, and other fellow graduate students for making the laboratory enjoyable and memorable.

Finally, I would like to thank my family for all their love and support throughout my life.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Manipulation planning asks the question of how to autonomously generate robot motion sequences that manipulate movable objects, possibly in an environment with obstacles, in order to perform a specified task. These problems involve objects that can only move when picked up by robots and the order of pick-and-place operations for manipulation may matter to obtain a feasible kinematic solution [1]. Therefore, geometric reasoning and motion planning alone are not sufficient to solve these problems. On the other hand, every pick-and-place operation physically changes the state of the environment continuously. The abstraction and discretization planning of actions may fail to capture these changes and result in infeasible plans. For these reasons; planning of actions such as the pick-and-place operations need to be integrated with the motion planning problem.

Motivated by this challenge, we introduce a formal hybrid planning framework that combines high-level representation and causality-based reasoning with low-level geometric reasoning and motion planning to obtain plans to be executed by robotic manipulators.

Our hybrid planning framework features a general interface between high-level causal reasoning and low-level geometric reasoning and motion planning, utilizing external predicates/functions of high-level representation formalism. External predicates/functions allow externally defined procedures/functions (in some programming language, like C++) to be embedded into the logical formalism of the high-level representation language. These external predicates/functions may implement, for instance, collision detection. Then, the idea is to embed such a function into the formulation of a robotic system, to be able to generate feasible plans. With the use of external predicates/functions, we are able to decide the amount of integration between high-level reasoning and geometric reasoning. For instance, all geometric constraints can be implemented as a C++ program, e.g., for collision detection as in the example above, and then they can be used to specify the executability of actions in the high-level formulation of a dynamic system. In this way, using such a formulation, feasible plans can be found to be executed later on. In that sense, the geometric reasoner guides the causal reasoner to find feasible kinematic solutions.

Alternatively; some of the geometric constraints can be implemented as an external predicate/function, since checking all constraints may be time consuming. Then, since some constraints are not considered in the definition of the external predicate/function, the plan computed using the formulation of the dynamic system may not be always feasible. In such cases, in order to guarantee a physically correct plan, we introduce another sort of guidance between task and motion planning: When a motion plan fails due to geometric constraints not captured by the external predicates/functions during task planning, the planning problem is modified considering domain-specific information about the possible causes of the failure. In this way, the causal reasoner finds a plan to a more “relevant” planning problem. Note that in a classical 3-layer robot control architecture, such failures are detected at the execution level. However, in our approach, they are detected at the representation level, thanks to the tight bilateral interaction between high-level representation and low-level geometric reasoning, before executing the plan.

We show the applicability of our hybrid planning framework on two domains: extended towers of Hanoi and multiple robot manipulation inside a maze. In each case, we represent the action domain in the logic-based formalism $\mathscr{C}+$, implement the external functions in C++, use the causal reasoner CCalc to compute task plans. We use our Rapidly-exploring Random Trees-based (RRT) [2] motion planner for geometric reasoning.

Furthermore, we study two important questions: i) a systemic analysis of various levels of integration between high-level representation and causality-based reasoning with low-level geometric reasoning and motion planning and ii) and generalization of the planning framework into continuous domains. For the first problem, we consider various levels of integration in the two domains mentioned above, to check which level of integration achieves better performance. Note that our planning framework allows such a systematic analysis due to flexible use of external predicates/functions as part of high-level representation and reasoning. For the second problem, we introduce a new method to apply probabilistic motion planning methods to operate with goal regions, by incrementally determining goal positions and utilizing the Inverse Kinematics Bidirectional RRT (IKBiRRT) [3] algorithm.

1.1 Contributions

- We introduce a hybrid planning framework that combines high-level representation and causality-based reasoning with low-level geometric reasoning and motion planning. Novel features of this framework can be listed as follows:

- Geometric reasoning guides task planning at the representation level.
    - External predicates/functions enable embedding geometric reasoning and motion planning into task planning. In this way, feasibility checks are handled at the representation level.
    - Delegating some feasibility checks to external predicates/functions allows control over the level of integration between high-level reasoning and geometric reasoning.
    - Since the integration is formulated at the representation level, our hybrid planning framework enables various search engines to be utilized for high level reasoning, such as (parallel) SAT solvers, without modifying their algorithms. Thus, the framework inherits advantages of these underlying solvers.
    - Since external predicates/functions can be implemented in any language, geometric reasoning can be modularly added to high-level representation of the domain. Thus, existing functions for geometric reasoning can be utilized in our hybrid planning framework.
- Task planning guides motion planning by computing a task plan.
    - Task planner calculates an optimal plan in terms of plan length. This plan contains all of the necessary primitive actions and their order of execution, including pick-and-place actions which are beyond the scope of motion planning.
    - Motion planners lack the ability to determine if there exists a fea-

sible path (i.e., a continuous trajectory corresponding to a discrete action in the given task plan). Usually, they are asked to determine if there is a feasible trajectory with respect to a given time threshold. However, in the worst case, infeasible motion planning queries take long time since the planners are required to reach the given threshold before they can return an answer. A task plan limits the search space for the motion planner and does so by mostly eliminating infeasible regions of the configuration space, which significantly increases computational efficiency.

  - Motion planning guides task planning by means of temporal constraints.
    - When all the feasibility checks are not delegated to external predicates/functions, motion planning may fail due to a geometric/ kinematic details not captured by the task planner. In such a case, the description of the planning problem is modified taking into account some domain specific information from motion-level (e.g., the causes of infeasibilities), and the causal reasoner is asked to solve a more "relevant" planning problem.
    - Instead of guiding the task planner at the search level by manipulating its search algorithm directly, the motion planner guides the task planner at the representation level by presenting to it the "right" planning problem.

- We show the applicability of our hybrid planning framework by physically implementing it on two example domains: Extended Towers of Hanoi and multiple robot manipulation inside a maze.
- We systematically analyze the influence of the level of integration between

high-level task planning and low-level geometric reasoning and path planning.

  - External predicates/functions allow task planning to capture physical constraints in more detail. However, their presence in the formalism brings a computational burden on the task planner. On the other hand, embedding only a partial set of the physical constraints at the representation level implies that task planner can suggest infeasible plans and replanning with the guidance motion planning may be more frequently required. Therefore, there exists an inherent trade-off on the level of integration and computational efficiency/tractability. We systematically analyze this trade-off with some experiments on the two domains mentioned above.

- We generalize our hybrid planning framework to continuous domains in robotic manipulation problems

  - Instead of viewing a configuration of a robotic manipulation domain as a set of discrete objects (e.g., viewing a maze as a grid of discrete points), we view them in a more abstract way as continuous regions. We introduce an incremental sampling-based method coupled to a goal-region-based probabilistic path planner for extracting specific goal configurations required for generating valid plans for execution.

- We show the effectiveness of the generalized hybrid planning framework by implementing a dynamic simulation of a mobile manipulation task with continuous goal regions.

1.2 Outline

Following is the structure of the thesis: In Chapter 2, we describe the action description language $\mathcal{C}+$and the causal reasoner CCalc; and give a brief description of RRT-based motion planning algorithms. Then in Chapter 3, we describe the overall architecture of our hybrid planning framework. In Chapter 4, we discuss the impact of the level of integration of low-level into the high-level through empirical data, and show the applicability of the framework on physical setups. Then in Chapter 5, we propose a method for generalizing the framework into domains where continuous regions are assumed. In Chapter 6, we conclude.

Chapter 2

Preliminaries

Before delving into the details of our hybrid planning framework, its components shall be described for a better understanding. Our approach of integrating high-level causal reasoning and low-level geometric reasoning consists two components. High-level representation formalism ($\mathscr{C}+$ [4]) and causal reasoner (CCALC [5]) are utilized for causal reasoning; whereas variants of Rapidly exploring Random Trees (RRTs) [2] algorithm are used for geometric reasoning and motion planning.

Now, we describe these components in detail.

2.1 Causal Reasoning with $\mathscr{C}+$ and CCALC

By describing a domain in $\mathscr{C}+$ (by a set of rules called "causal laws") and using the reasoning mechanism CCALC, high-level task plans with several properties (such as being of minimum number of steps, obeying constraints) can be calculated. In the following, syntax and the semantics of the action description language $\mathscr{C}+$ (similar to [4]) are described, and the reasons behind the selection are explained.

2.1.1 Syntax of Causal Laws in $\mathcal{C}+$

We start with a *(multi-valued propositional) signature* that consists of a set σ of *constants* of two sorts, along with a nonempty finite set $Dom(c)$ of *value names*, disjoint from σ, assigned to each constant c. An *atom* of σ is an expression of the form $c = v$ ("the value of c is v") where $c \in \sigma$ and $v \in Dom(c)$. A *formula* of σ is a propositional combination of atoms. If c is a Boolean constant, we will use c (resp. $\neg c$) as shorthand for the atom $c = True$ (resp. $c = False$).

A signature consists of two sorts of constants: *fluent constants* and *action constants*. Intuitively, fluent constants denote "fluents" characterizing a state; action constants denote "actions" characterizing an event leading from one state to another.

A *fluent formula* is a formula such that all constants occurring in it are fluent constants. An *action formula* is a formula that contains at least one action constant and no fluent constants.

An *action description* is a set of *causal laws* of three sorts. *Static laws* are of the form

$$\textbf{caused } F \textbf{ if } G \tag{2.1}$$

where F and G are fluent formulas. *Action dynamic laws* are of the form (2.1) where F is an action formula and G is a formula. *Fluent dynamic laws* are of the form

$$\textbf{caused } F \textbf{ if } G \textbf{ after } H \tag{2.2}$$

where F and G are as above, and H is a fluent formula. In (2.1) and (2.2) the part **if** G can be dropped if G is *True*; the part F is called the *head*.

Abbreviations for causal laws

While describing action domains, we can use some abbreviations. For instance, we can describe the (conditional) direct effects of actions using expressions of the form

$$c \textbf{ causes } F \textbf{ if } G$$

which abbreviates the fluent dynamic law

$$\textbf{caused } F \textbf{ if } \mathit{True} \textbf{ after } c \wedge G.$$

This abbreviation expresses that "executing c at a state where G holds, causes F."

We can formalize that F is a precondition of executing c by the following expression

$$\textbf{nonexecutable } c \textbf{ if } F$$

which stands for the fluent dynamic law

$$\textbf{caused } \mathit{False} \textbf{ if } \mathit{True} \textbf{ after } c \wedge F.$$

Similarly, we can prevent the execution of two actions c and c' by the expression

$$\textbf{nonexecutable } c \wedge c'.$$

We can represent the "commonsense law of inertia" also by using abbreviations. For instance, we can describe that "the value of a fluent F remains to be true unless it is caused to be false" by the expression

$$\textbf{inertial } F$$

that stands for the fluent dynamic causal law

$$\textbf{caused } F \textbf{ if } F \textbf{ after } F.$$

Note that classical task planners make such an assumption but at the search level (i.e., they cannot represent it explicitly as a part of planning domain description). Furthermore, expressions

$$\textbf{inertial } F_1, \ldots, \textbf{inertial } F_n$$

can be abbreviated as

$$\textbf{inertial } F_1, \ldots, F_n.$$

In almost all the action domains, we express that there is no cause for the occurrence of an action A by the expression

$$\textbf{exogenous } A$$

that abbreviates the following action dynamic laws:

$$\textbf{caused } A \textbf{ if } A$$
$$\textbf{caused } \neg A \textbf{ if } \neg A.$$

Example – Suitcase domain

Consider, for instance, the suitcase domain introduced in [6]: there is a suitcase with two latches $l1$ and $l2$; when these two latches are up then the suitcase automatically opens. There are three propositional fluents: $up(L)$, where L is $l1$ or $l2$, Notation: L ranges over $\{l1, l2\}$ and a ranges over action constants.

Action constants:	Domains:
$toggle(L)$	Boolean

Fluent constants:	Domains:
$up(L)$	Boolean
$open$	Boolean

Causal laws:

$toggle(L)$ **causes** $up(L)$ **if** $\neg up(L)$
$toggle(L)$ **causes** $\neg up(L)$ **if** $up(L)$

caused $open$ **if** $up(l1) \wedge up(l2)$

inertial $open, \neg open$
inertial $up(L), \neg up(L)$

exogenous a

Figure 2.1.1: The suitcase domain described in C+.

and *open*; $up(L)$ holds iff latch L is up, *open* holds iff the suitcase is open. There is an action of toggling a latch L denoted by $toggle(L)$. If a latch is down (resp. up) then it becomes up (resp. down) after toggling it. We can describe this domain in the action description language $\mathscr{C}+$ by the causal laws presented in Figure 2.1.1. The first two lines of causal laws describe the direct effects of toggling the latches. The third line describes the indirect effects of toggling. The commonsense law of inertia is expressed by the next two lines. The last line expresses that actions are exogenous.

2.1.2 Semantics for Action Descriptions

The meaning of an action description can be represented by a "transition system". A transition diagram can be thought of as a labeled directed graph. Every state is represented by a vertex labeled with the function from fluent constants to their values. Every transition $\langle s, A, s' \rangle$ is represented by an edge leading from s to s' and labeled A.

For instance, the transition system of the suitcase domain description in Figure 2.1.1 has 7 possible states:

$\{up(l1), up(l2), open\}$	(S_1)
$\{up(l1), \neg up(l2), open\}$	(S_2)
$\{up(l1), \neg up(l2), \neg open\}$	(S_3)
$\{\neg up(l1), up(l2), open\}$	(S_4)
$\{\neg up(l1), up(l2), \neg open\}$	(S_5)
$\{\neg up(l1), \neg up(l2), open\}$	(S_6)
$\{\neg up(l1), \neg up(l2), \neg open\}$	(S_7)

Note that $\{up(l1), up(l2), \neg open\}$ is not a possible state; due to the static law

$$\textbf{caused } open \textbf{ if } up(l1) \wedge up(l2)$$

at every state s, $up(l1) \wedge up(l2) \supset open$.

At each possible state there are 4 applicable actions:

$\{toggle(l1), toggle(l2)\}$	(A_1)	"toggle $l1$ and $l2''$
$\{\neg toggle(l1), \neg toggle(l2)\}$	(A_2)	"do nothing"
$\{toggle(l1), \neg toggle(l2)\}$	(A_3)	"toggle $l1$"
$\{\neg toggle(l1), toggle(l2)\}$	(A_4)	"toggle $l2$"

so that each state has 4 outgoing edges. For instance, the edges outgoing from S_1 are

$$\langle S_1, A_1, S_6\rangle, \langle S_1, A_2, S_1\rangle, \langle S_1, A_3, S_4\rangle, \langle S_1, A_4, S_2\rangle.$$

The transition system that corresponds to the suitcase domain has 28 edges.

Consider the transition $\langle S_7, A_1, S_1 \rangle$. The causal laws that are applicable to this transition are

$$\begin{array}{l}
\mathit{toggle}(l1)\ \textbf{causes}\ \mathit{up}(l1)\ \textbf{if}\ \neg \mathit{up}(l1)\\
\mathit{toggle}(l2)\ \textbf{causes}\ \mathit{up}(l2)\ \textbf{if}\ \neg \mathit{up}(l2)\\
\textbf{caused}\ \mathit{open}\ \textbf{if}\ \mathit{up}(l1) \wedge \mathit{up}(l2)\\
\textbf{exogenous}\ \mathit{toggle}(l1)\\
\textbf{exogenous}\ \mathit{toggle}(l2).
\end{array}$$

Here $\{\mathit{up}(l1), \mathit{up}(l2), \mathit{open}\}$ is the only interpretation that satisfies the heads of the first three causal laws; $\{\mathit{toggle}(l1), \mathit{toggle}(l2)\}$ is the only interpretation that satisfies the heads of the last two causal laws.

Now consider the triple $\langle S_7, A_3, S_1 \rangle$. The causal laws that are applicable to this transition are

$$\begin{array}{l}
\mathit{toggle}(l1)\ \textbf{causes}\ \mathit{up}(l1)\ \textbf{if}\ \neg \mathit{up}(l1)\\
\textbf{caused}\ \mathit{open}\ \textbf{if}\ \mathit{up}(l1) \wedge \mathit{up}(l2)\\
\textbf{exogenous}\ \mathit{toggle}(l1)\\
\textbf{exogenous}\ \mathit{toggle}(l2).
\end{array}$$

Here there are two interpretations that satisfy the heads of the first three causal laws; $\{\mathit{up}(l1), \mathit{up}(l2), \mathit{open}\}$ and $\{\mathit{up}(l1), \neg \mathit{up}(l2), \mathit{open}\}$. In other words, no causal law provides a causal explanation for $\mathit{up}(l2)$. Therefore, this triple is not a transition.

2.1.3 Queries

Given an action domain description represented in a fragment of $\mathscr{C}+$ as described above, we can perform various reasoning tasks over it, such as planning, prediction and postdiction. Such reasoning problems are represented using queries in an "action query language" as described in [7].

We consider a variation of the action query language $\mathcal{Q}$ introduced in [7]. In this language, an *atomic query* is one of the two forms, F **holds at** t or A **holds at** t, where F is a fluent formula, A is an action formula, and t is a time step. A *query* is a propositional combination of atomic queries.

Let D be an action description and $T(D) = \langle S, V, R\rangle$ denote the transition system described by D, with a set S of states, a value function V mapping, at each state s, every fluent P to a truth value, and a set R of transitions. A *history* of D of length n is a sequence

$$s_0, A_1, s_1, \dots, s_{n-1}, A_n, s_n \tag{2.3}$$

where each $\langle s_i, A_{i+1}, s_{i+1}\rangle$ $(0 \leq i < n)$ is in R. We say that a query Q of the form F **holds at** t (resp. A **occurs at** t) is *satisfied* by a history (2.3) if s_t satisfies F (resp. if A_t satisfies A). For nonatomic queries, *satisfaction* is defined by truth tables of propositional logic. We say that a query Q is *satisfied* by an action description D, if there is a history H of D that satisfies Q.

Suppose that F and G are fluent formulas denoting an initial state and goal conditions respectively. We can describe the problem of finding a plan of length n, with a query of the form

$$F \textbf{ holds at } 0 \wedge G \textbf{ holds at } n. \tag{2.4}$$

Consider, for instance, the following planning problem over the suitcase domain defined above: initially, both latches are down and the suitcase is not open; the goal is to open the suitcase in *maxstep* steps. This problem can be expressed as a query as follows:

$$\begin{aligned} &\neg up(l1) \wedge \neg up(l2) \wedge \neg open \textbf{ holds at } 0 \wedge \\ &open \textbf{ holds at } maxstep. \end{aligned} \tag{2.5}$$

We can also solve variations of these problems, where some intermediate states are specified or where the specified actions are not executed consecutively. This allows us to enforce, for instance, further temporal constraints in a planning problem. Consider the planning problem above described by (2.5), but with a constraint that “at every time step t, if latch $l2$ is down then latch $l1$ cannot be toggled.” We can describe this new problem by modifying (2.5) as follows

$$\begin{aligned} &\neg up(l1) \wedge \neg up(l2) \wedge \neg open \textbf{ holds at } 0 \wedge \\ &open \textbf{ holds at } maxstep \wedge \\ &\textstyle\bigwedge_{0 \leq t < maxstep} \neg \big(up(l2) \textbf{ holds at } t \supset \\ &\qquad \neg toggle(l1) \textbf{ occurs at } t\big). \end{aligned} \tag{2.6}$$

We can also ensure some order of actions if desired. For instance, we can ensure that “at some time t, latch $l1$ is toggled before latch $l2$” by modifying (2.5) as follows:

$$\begin{aligned} &\neg up(l1) \wedge \neg up(l2) \wedge \neg open \textbf{ holds at } 0 \wedge \\ &open \textbf{ holds at } maxstep \wedge \\ &\textstyle\bigvee_{0 \leq t_1, t_2 < maxstep, t_1 < t_2} \big(toggle(l1) \textbf{ occurs at } t_1 \wedge \\ &\qquad toggle(l1) \textbf{ occurs at } t_2\big). \end{aligned} \tag{2.7}$$

2.1.4 CCALC

The Causal Calculator (CCALC) [5] is a reasoning system, that performs reasoning tasks over an action domain description represented in a fragment of $\mathcal{C}+$ described above. To present formulas to CCALC, conjunctions $\wedge$, disjunctions $\vee$, implications $\supset$, negations $\neg$ are replaced with the symbols `&` (or `&&`)), `++`, `->>`, and `-` respectively. In most of the action descriptions, fluents are inertial and actions are exogenous; therefore, CCALC allows us to include this information at the very beginning of the action description while declaring fluent constants and action constants. For instance, the suitcase domain represented in $\mathcal{C}+$ in Figure 2.1.1 is presented to CCALC as in Figure 2.1.2.

When we present causal laws to CCALC, we can call "external predicates/ functions" in them. These predicates/functions are not part of the signature of the domain description (i.e., they are not declared as fluents or actions). They are implemented as functions in some programming language of the user's choice, such as C++. External predicates take as input not only some parameters from the action domain description (e.g., the locations of robots) but also detailed information that is not a part of the action domain description (e.g., geometric models). They are used to check externally some conditions under which the causal laws apply, or compute externally some value of a variable/fluent/action. For instance, suppose that the external predicate `collision(X,Y,X1,Y1)` (implemented in C++) checks whether the path between `(X,Y)` and `(X1,Y1)` collides with an obstacle. Then we can express that there is no state at which the endpoints of a payload are located at `(X,Y)` and `(X1,Y1)` where `collision(X,Y,X1,Y1)` holds:

```
caused false
   if xpos(r1)=X1 & ypos(r1)=Y1 &
      xpos(r2)=X2 & ypos(r2)=Y2
```

```
    where collision(X1,Y1,X2,Y2).
```

In addition, an external predicate can accomplish some other tasks as "side - effects." For instance, while checking whether a robot located at `(X,Y)` collides with another robot at `(X1,Y1)`, the external predicate `collision(X ,Y, X1, Y1)` can form a database keeping which locations lead to a collision and which locations do not. Then this database can be reused in the future.

Another useful feature of CCALC is its ability to represent "attributes of actions" that allows us to talk about various special cases of actions. Consider, for instance, the action of "a robot `R` picking a payload". We can enforce that a robot `R` cannot pick a payload while moving by a causal law like

```
nonexecutable move(R) & pick(R).
```

However, when we want to specify some effects of these actions, we need to consider special cases of them. For instance, to express the effect of picking a payload, it may be useful to consider where the robot is picking the payload at. To express the effect of moving, it may be useful to consider in which direction and by what number of steps the robot is moving the payload. To denote these special cases of actions, we declare their attributes. For instance, for `pick`, we introduce an attribute `pickpoint` (as a function that returns which endpoint) after we declare `pick` as an exogenous action:

```
pick(robot)      :: exogenousAction;
pickpoint(robot) ::
    attribute(endpoint) of pick(robot);
```

and describe its effect ("robot `R` is holding a payload at its endpoint `P`"):

```
:- sorts
latch.

:- objects
l1, l2 :: latch.

:- variables
L :: latch.

:- constants
up(latch), open :: inertialFluent;
toggle(latch)   :: exogenousAction.

% effects of toggling
toggle(L) causes up(L) if -up(L).
toggle(L) causes -up(L) if up(L).

% suitcase is open if
%    both of the latches are open
caused open if up(l1) & up(l2).
```

Figure 2.1.2: The suitcase domain described in the language of CCALC.

```
pick(R) causes holding(R,P)
   if pickpoint(R)=P.
```

By this way, additional special cases of an action can be defined without having to modify the definitions of more general actions. Note that such a representation of actions by special cases is a form of "hierarchical" representation of actions.

Once such an action domain description is given, we can perform various reasoning tasks via queries in an action query language, like the variation of the action query language $\mathcal{Q}$ described above.

Given a domain description and a query, CCALC checks whether the query is satisfied by the domain description (in the sense of satisfiability planning of [8]) as follows:

1. it transforms the causal laws into a propositional theory Γ_D, via "causal logic" [4],
2. it transforms the query into a propositional theory Γ_P,
3. it checks whether $\Gamma_D \cup \Gamma_P$ is satisfiable;
4. if $\Gamma_D \cup \Gamma_P$ is unsatisfiable, it returns No;
5. otherwise, it returns Yes and presents an example extracted from a satisfying interpretation for $\Gamma_D \cup \Gamma_P$.

The transformations in the first two steps are different: the one in 1) is based on literal completion, whereas the one in 2) is based on a simpler procedure (see [4] for a detailed description). Such a difference allows one to check the satisfiability of other queries (for instance, for replanning) without executing the first step again. Step 3) is done automatically by a state-of-the-art SAT solver, such as MINISAT [9] or its parallel variant MANYSAT [10].

Let us give now some examples for the kind of reasoning tasks that CCALC can do. For instance, we can present query (2.5) to CCALC as follows:

```
:- query
maxstep :: 2;
0: -up(l1), -up(l2), -open;
maxstep: open.
```

Note that conjunctions $\wedge$ in a formula are denoted by commas in queries. A query of the form F **holds at** t (resp. A **occurs at** t) is represented as `t:    F` (resp. `t:`

A). The third line in the query above describes the initial state at time step 0, and the last line describes the goal condition at time step `maxstep=2`.

To find a shortest plan, we modify this query (let us label the modified query as ‘Query 1’):

```
:- query
label :: 1;  % Query 1
maxstep :: 0..infinity;
0: -up(l1), -up(l2), -open;
maxstep: open.
```

With this query, CCALC successively tries to find a plan of length `maxstep=0, 1 ,..., infinity`. For Query 1, CCALC finds a shortest plan for `maxstep = 1` where both latches are toggled at time step 0:

```
0:
ACTIONS:  toggle(l1)  toggle(l2)
1:
```

CCALC can also show the complete history of this plan, including state information, if prompted:

```
0:  -up(l1)  -up(l2)  -open
ACTIONS:  toggle(l1)  toggle(l2)
1:  up(l1)  up(l2)  open
```

We can add some constraints to a planning problem specified as a query, as in (2.6):

```
:- query
label :: 2;  % Query 2
maxstep :: 0..infinity;
0: -up(l1), -up(l2), -open;
maxstep: open;
[/\T | T<maxstep ->>
   (-(T: up(l2)) ->> -(T: toggle(l1)))].
```

With this modification, CCALC finds the following shortest plan instead:

```
0:  -up(l1)  -up(l2)  -open
ACTIONS:  toggle(l2)
1:  up(l2)  -up(l1)  -open
ACTIONS:  toggle(l1)
2:  up(l1)  up(l2)  open
```

2.1.5 Why $\mathscr{C}+$ and CCALC?

We have decided to use the action description language $\mathscr{C}+$ to describe action domains due to its expressivity: we can formalize not only effects and preconditions of actions, but also state/transition constraints and changes that do not directly involve actions; we can represent not only deterministic effects but also nondeterministic effects of actions. Also $\mathscr{C}+$ allows concurrency, unless specified otherwise via nonexecutability constraints.

We envision agents (robots) in a framework that has the capability of solving not only planning problems but also other reasoning tasks; since we aim endowing agents (robots) with various kinds of high-level reasoning mechanisms (such as prediction, postdiction, diagnosis, reasoning about shared resources, etc.) in the sense of cognitive robotics [11]. The action description language $\mathscr{C}+$, with the query language defined above, provides a common language for all these reasoning tasks, and thus allows us to setup such a framework.

CCALC can answer queries about a domain description represented in $\mathscr{C}+$ with respect to a reasoning task described in the query language above. Therefore, it allows us to solve different sorts of reasoning problems mentioned above (possibly with temporal constraints). Being able to add domain-specific temporal constraints as part of queries, for instance, allows us to do intelligent replanning to find different and "better" plans, as explained in the Chapters 3 and 4.

Due to its modular structure and generic implementation, CCALC allows us to use various kinds of search engines to answer queries: CCALC supports SAT solvers such as MINISAT and the parallel SAT solvers like MANYSAT; the user can choose which search engine to use for answering which query. Due to well-studied relations between action languages and Answer Set Programming (ASP) [12], as in [13], we can also use efficient ASP solvers instead of SAT solvers, as in [14].

CCALC also supports external predicates/functions that can be implemented in some programming language of the user's choice. These predicates/functions are important, for instance, in embedding low-level geometric reasoning in high-level reasoning, as explained in Chapter 4.

CCALC has other useful features/utilities as well: it supports additive fluents (to talk about the total effect of concurrently executing actions on numeric-valued fluents that denote shared resources), macros (to define complex notions succinctly, in some ways similar to "derived predicates"), attributes (to talk about special cases of actions). For more information about CCALC, we refer the reader to [4].

Algorithm 1 BiRRT

```
Require: Initial state $S$ and goal state $G$
  {$C$, $t$, $d$ denote the configuration space, a specified timeout and a specified distance}
  $s \leftarrow$ Find initial configuration in $C$ that corresponds to $S$;
  $g \leftarrow$ Find goal configuration in $C$ that corresponds to $G$;
  $V := \{\langle s,1\rangle, \langle g,2\rangle\}$; {The roots of Tree 1 and Tree 2}
  $E := \emptyset$; {Empty set of undirected edges in these trees}
  $connected := false$;
  while $\neg connected$ and the timeout $t$ is not exceeded do
    $p \leftarrow$ Sample a random point in $C$;
    if $p$ is collision-free then
      $p_1 \leftarrow$ Find the closest point to $p$ in $V$ with label 1;
      $p_2 \leftarrow$ Find the closest point to $p$ in $V$ with label 2;
      for $i = 1,2$ do
        if the path connecting $p_i$ ad $p$ is collision-free then
          $V := V \cup \{\langle p,i\rangle\}$; {Expand Tree $i$ with $p$}
          $E := E \cup \{\{p_i, p\}\}$;
        end if
      end for
      if both the path connecting $p$ and $p1$, and the path connecting $p$ and $p2$ are collision-free
      then
        $connected := true$;
      end if
    end if
  end while
  if $connected$ then
    $\pi \leftarrow$ Extract the trajectory from $\langle V,E\rangle$;
    return $true, \pi$;
  else
    return $false$; {No trajectory}
  end if
```

2.2 Motion Planning with RRT

Motion planning, and specifically path planning, is an area that has been well studied. Path planning problem, also known as piano mover's problem, asks for the path that leads an initial configuration to a goal configuration (for a review of motion planning, see [15]).

The most suitable branch of motion planning for this work is sampling based path planning. Algorithms of sampling based path planning can be divided into two categories in terms of usability of their samplings. Multi-query path planning algorithms, such as variants of probabilistic roadmaps (PRM [16]), sample the configurations space in advance to generate a graph, called a roadmap; path planning is carried out afterwards. Since the roadmaps are generated without regarding the path planning query, they can be used multiple times. However, the overhead time of generating a roadmap negatively affects the overall computation time, which proves to be inefficient in situations where the environment changes rapidly.

Single-query algorithms, such as RRT [2], only try to connect the initial configuration to a goal configuration, without regarding reusability. Therefore, the computation of a single path is significantly reduced compared to PRM.

Bidirectional RRT, which is commonly referred to as RRT, tries to expand and connect two trees whose roots are initial and goal configurations. In one of the examples of Chapter 4, as well as in our other studies, a variant of this algorithm (Algorithm 1) is employed. The trees are expanded in the following manner: A point p is sampled from the configuration space. If p is collision-free, then from the tree T_i, the closest point p_i to p is calculated. If the path connecting p_i to p is collision-free, then point p and edge (p_i, p) are added to T_i.

Chapter 3

A Hybrid Planning Framework for Robotic Manipulations

While manipulating objects, the order of pick-and-place actions may affect the feasibility of a plan. Consider, for instance, a robot is to place an object to a given location. Further, the goal location may be occupied by another object. In such a case, motion planners alone are not able to solve the problem. On the other hand, as the burden on task planning is increased by adding the details of low-level, the problem becomes harder to solve for the task planner. As a result, some amount of abstraction is unavoidable. However, this may lead to task plans whose execution in the real world is not possible. Therefore, similar to motion planning, task planning is not sufficient by itself for capturing all geometric constraints and details.

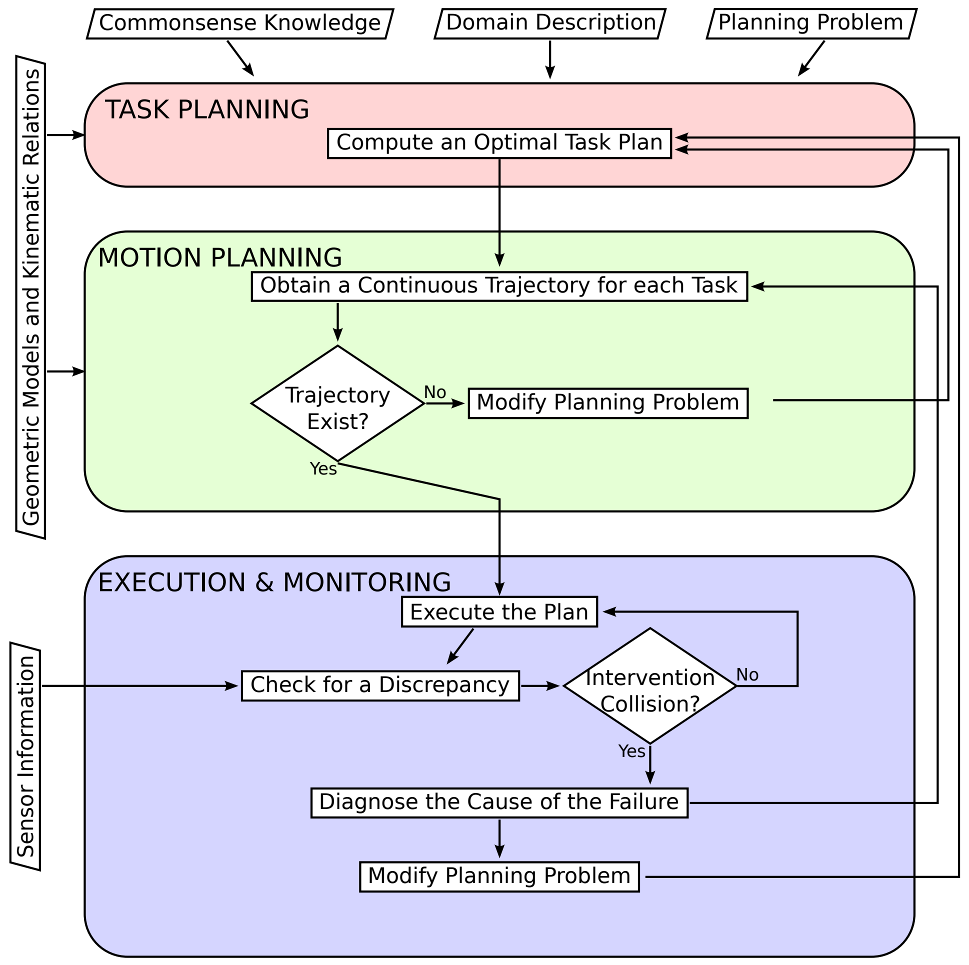

Figure 3.1.1: Overall system architecture

3.1 Overall System Architecture

The overall architecture of our formal framework that combines high-level representation and causality-based reasoning with low-level geometric reasoning and motion planning to generate feasible continuous trajectories and a modular execution monitoring framework is illustrated in Figure 3.1.1.

In this framework, we start with an action domain description and planning problem description in the input language of CCALC, geometric models in some modeling system (e.g., Wavefront or VRML), and functions (e.g., kinematics, look-up tables, etc.) whose truth values are to be calculated for task planning. These inputs can be described as follows:

- Geometric models include links of the robots, payloads that shall be manipulated, and obstacles in the environment.
- Functions, such as kinematic relations, are to be used in the planning process. Their values are calculated for availabilities of states and transitions. These functions may make use of the geometric models provided.
- An action domain description is a set of causal laws that express direct effects and preconditions of actions of robots, causal relations that do not involve these actions directly (e.g., ramifications), and state and transition constraints. These causal laws may include external predicates, that use the provided functions, expressing conditions that involve geometric reasoning (as shown in Section 2.1) so that geometric models are also taken into account while a task plan is computed.
- A planning problem description is a set of formulas that express an initial state, goal conditions, and temporal constraints.

Given an action domain description and a planning problem, and using CCALC; first we compute a plan of sequence of actions $\langle A_0, \ldots, A_n \rangle$, that is optimal in plan length, and its complete history $\langle S_0, A_0, S_1, \ldots, S_n, A_n, S_{n+1} \rangle$, that includes intermediate states. The computed plan may involve concurrent execution of actions by multiple robots; so each A_i is a set of primitive actions.

Next, using plan and history, and considering kinematic relations; we calculate a continuous trajectory for each transition. Considering the state before the transition and effects of the transition, a continuous trajectory for each robot is calculated by a motion planning algorithm (Section 2.2).

If the motion planner fails to find a continuous trajectory for a transition, we identify the cause of that failure. There may be various kinds of failure with different causes. For instance, state caused by a transition may not be valid in reality. Similarly, although previous and next states of a transition may be valid, motion planner may fail to find a trajectory due to obstacles. Depending on the cause of the failure, we modify the planning problem (to state such a transition or state is invalid) by adding domain-specific temporal constraints to avoid similar sorts of failure, as shown in Section 2.1. Afterwards, the modified planning problem is solved by CCALC, generating a different optimal task plan.

In the following, we describe, in detail, how task planning guides motion planning and how motion planning guides task planning, as illustrated in the manipulation planning framework shown in Figure 3.1.2.

3.2 Task Planning Guides Motion Planning

The aim of planning is ultimately to obtain robotic motion sequences to perform a given task. Applicability of a plan may be effected by the order of involved primitive actions, such as pick and place; and motion planners are not able to determine such an order. Similarly, motion planners cannot reason on the effects

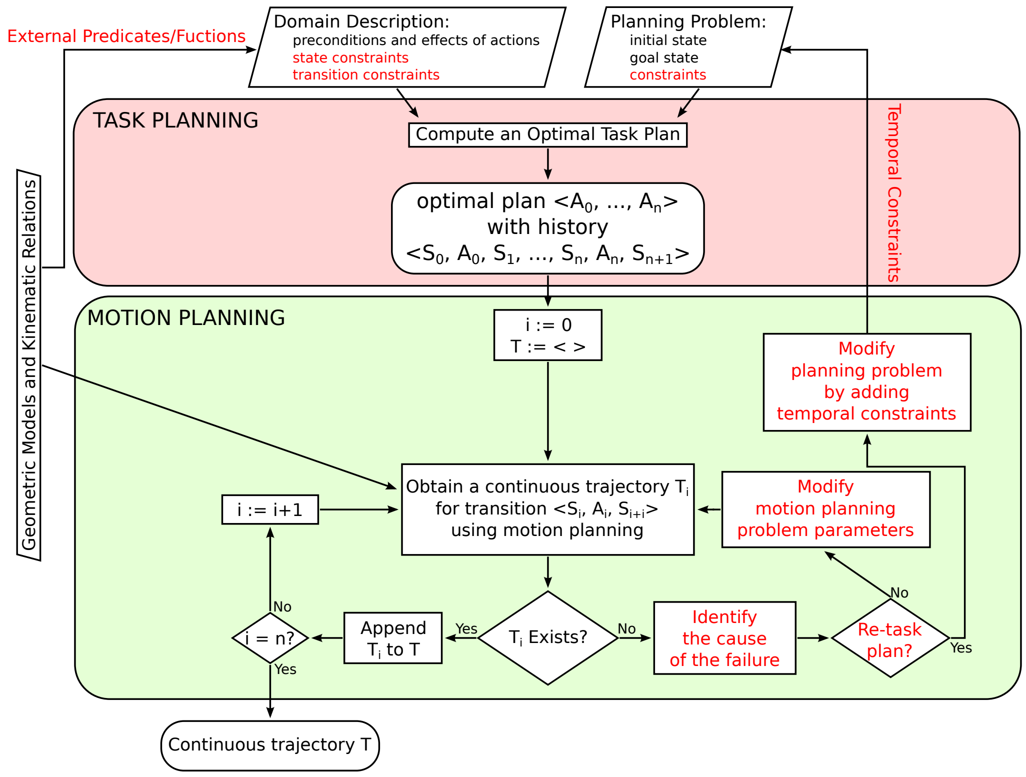
External Predicates/Fuctions
Domain Description:
preconditions and effects of actions
state constraints
transition constraints
Planning Problem:
initial state
goal state
constraints
TASK PLANNING
Compute an Optimal Task Plan
optimal plan <A0, ..., An> with history <S0, A0, S1, ..., Sn, An, Sn+1>
Temporal Constraints
Geometric Models and Kinematic Relations
MOTION PLANNING
i := 0
T := < >
Modify planning problem by adding temporal constraints
Obtain a continuous trajectory Ti for transition <Si, Ai, Si+i> using motion planning
i := i+1
Modify motion planning problem parameters
No
Yes
Append Ti to T
Ti Exists?
Identify the cause of the failure
Re-task plan?
i = n?
Continuous trajectory T

Figure 3.1.2: Manipulation planning framework

of geometry changing actions.

By the help of a task plan and state history $\langle S_0, A_0, S_1, \ldots, S_n, A_n, S_{n+1} \rangle$, motion planning is used for planning a single transition $\langle S_i, A_i, S_i \rangle$ at a time, which is what motion planning algorithms are essentially designed for. If a motion plan for such a transition is calculated, it is appended to an incrementally created overall motion plan. Otherwise, this information is used for the guiding the task planner, as described in the next section.

Algorithm 2 TASK&MOTION_PLAN

```
Require: Action domain description $\mathscr{D}$, and planning problem $\mathscr{P}$ with the initial state(s) $S$ and
  the goal $G$
  while true do
    $plan, P \leftarrow$ Compute a shortest task plan $P$ of length $n$ (within a history $H = \langle S_0, A_0, S_1, ..., S_n, A_n, S_{n+1} \rangle$, where $S_0 = S$ and $S_{n+1} = G$) using CCALC with $\mathscr{D}$ and $\mathscr{P}$ (if there is such a plan);
    if $\neg plan$ then
      return false;
    end if
    $T := \langle\rangle$; {Initially the trajectory is empty}
    $trajectoryFound := true$;
    $i := 0$;
    while trajectoryFound do
      $\langle S_i, A_i, S_{i+1} \rangle \leftarrow$ Extract from $H$ the next transition;
      {Compute a trajectory $\pi$ for $\langle S_i, A_i, S_{i+1} \rangle$, if one exists}
      $trajectoryFound, \pi \leftarrow$ MOTION_PLAN($S_i$,$S_{i+1}$);
      if $\neg trajectoryFound$ then
        $\mathscr{P} \leftarrow$ Identify the cause of the failure and modify the planning problem $\mathscr{P}$ accordingly;
      else
        $T \leftarrow$ Append $\pi$ to $T$;
        if $A_i$ is the last action then
          return $true, P, H, T$;
        end if
      end if
      i++;
    end while
  end while
```

3.3 Motion Planning Guides Task Planning

If a motion plan cannot be calculated for an instance of transition, by using the information, the cause of the failure is tried to be determined. There are three elements of a transition $\langle S_i, A_i, S_i + 1 \rangle$: Current state, set of action to be executed at the current state, and next state. As mentioned in the previous section, the motion plans are generated incrementally, therefore any failure caused by the initial state would have been captured in the previous iteration. Remaining possibilities are that the goal state may not be available (e.g., a robot may be colliding), and such a transition given the initial state cannot be executed (e.g., a collision-free path may not be computed). Former can be immediately captured. In such a case, the planning problem is modified by the addition of a temporal constraint stating that state $S_i + 1$ cannot occur.

When a motion planner cannot find a feasible trajectory due to some impossible state or transition, the task planning problem is modified accordingly. In this way, the motion planner guides the task planner to find plans that are more likely to be feasible.

By the addition of temporal constraints, the motion planner is used to guide the task planner by asking to find more "relevant" plans.

3.4 Related Work

In earlier studies, the state-of-the-art motion planning systems have been extended to handle some manipulation planning problems [17, 18] based on the idea of viewing the configuration space as consisting of regions connected by lower-dimensional submanifolds, such as transit and transfer manifolds. However, such manipulation planners are domain-specific and cannot handle action planning in a generic manner: each one addresses a specific sort of manipulation planning problems without making use of task planning. As a result, there has been a growing interest in hybrid manipulation planning approaches that utilize both task planning and motion planning. The traditional approaches to hybrid manipulation planning have been top-down, separating high-level task planning from lower-level motion planning. In these approaches, task planning is simplified by ignoring low-level (geometric) details; however, due to such an abstraction, the resulting plans may be inefficient/infeasible.

Recently, more elaborate approaches have been proposed to integrate task and motion planning more tightly *at the search level.* For instance, [19–24] take advantage of a forward-search task planner to incrementally build a task plan, while checking its kinematic/geometric feasibility at each step by a motion planner; all these approaches use different methods to utilize the information from the task-level to guide and narrow the search in the configuration space. By this way, the task planner helps focus the search process during motion planning.

In particular, Cambon *et al.* [19, 20] describe the manipulation problem in a hybrid symbolic and geometric domain, but attack the problem in a decoupled manner: they first solve the relaxed task planning problem (by ignoring the geometric constraints) using a heuristic forward planner and then call a sampling-based motion planner to geometrically validate the highest priority action of this task plan; the roadmap of validated actions are added to a global roadmap, while invalidated actions are given a lower priority. Hauser and Latombe [21] propose a probabilistically complete incremental task/motion planner based on heuristic search. Acknowledging the non-expansive nature of the feasible space due to varying dimensionality of subspaces, the motion planner samples each subspace individually and composes them to construct a roadmap [21]. The incremental task planner capitalizes on the fact that transition spaces are easier to sample and use an estimate of non-emptiness of the transition spaces as a geometrically motivated metric to focus the search on the task space. Plaku and Hager [22] propose a heuristic search approach similar to that of [21] to combine sampling based motion planning with symbolic task planning. In particular, a tree-based search is utilized for the motion planner to obtain dynamically feasible trajectories and this search is guided by the task planner relative to a utility heuristic, whose estimates are updated by a sampling-based motion planner. Wolfe *et al.* [24] represent the manipulation problem by a vertically integrated Hierarchical Task Network (HTN) [25] where transition models are defined for primitive actions at the bottom level. The transition models are procedural and calls for external geometric planners, such as Rapidly exploring Random Trees (RRTs) [2], to solve for feasibility and action cost queries when these primitive actions are invoked. Given such an HTN, a hierarchically optimal plan (optimal plan subject to the hierarchy constraints introduced during HTN modeling) is found through an exhaustive search that uses subtask relevance as a heuristic to speed up the search. [23] follows a different approach to hierarchical planning: their planner is based on goal-regression, so the search starts from the goal towards the initial state; because of this approach, instead of checking preconditions of tasks, the algorithm checks the configuration "in the now". Also sometimes the successor states in the search tree are decided by making use of geometric planning. For instance, geometric reasoning can be used to find out from which location the object shall be moved so that the object is at its current region. Note that each one of these approaches presents a specialized combination of task and motion planning at the search level, and do not consider a general interface between task and motion planning.

With this motivation, [26] and [27] introduce an alternative approach to integrating task and motion planning by considering a general interface between them, using "external predicates/functions" whose values for ground instances are computed by an external mechanism, e.g., by a C++ program. The concept of external predicates/functions is not new; they have existed as undocumented features of the planner TLPlan [28] and the Causal Calculator (CCALC) [5]. The idea in [26,27] is to use external predicates/functions in an action domain description, for checking the feasibility of a primitive action by a motion planner. [26] applies this approach in the action description language $\mathscr{C}+$ [4] using CCALC, while [27] extends the planning domain description language PDDL [29] to support external predicates/functions (called semantic attachments) and modifies the planner FF [30] accordingly.

3.5 Summary of Contributions

In our studies, we use the action description language $\mathcal{C}+$ as the high-level representation and causal reasoning formalism, and CCALC as the automated causal reasoner. Our contribution is an alternative approach for integrating task and motion planners that combines various advantages of some of the related approaches discussed above with some other advantages inherited from the high-level causality-based representation and reasoning formal framework we use in our studies. As in [26, 27], we also consider a general interface between high-level causal reasoning and low-level geometric reasoning and motion planning, using external predicates/functions, but in a more flexible way. The first novelty of our approach is the flexible use of external predicates/functions for feasibility checks: Instead of delegating all sorts of feasibility checks to external predicates as in [26, 27], we can decide which sorts of feasibility check should be done by external predicates as part of high-level reasoning, and which sorts of feasibility checks should be done by motion planning later on. For instance, external predicates can check only collisions of robots with each other and with other objects (so they do not check, for instance, collisions of objects with each other), and they can be used in action domain description to specify conditional effects of these robots' actions. By this way, geometric reasoning is "embedded" in high-level representation: while computing a task-plan, the causal reasoner takes into account geometric models and kinematic relations by means of external predicates implemented for geometric reasoning. In that sense, the geometric reasoner guides the causal reasoner to find feasible kinematic solutions. Such a flexibility may be useful if gathering all sorts of feasibility checks in one external predicate/function is computationally disadvantageous, or if checking feasibility in a modular way (using different mechanisms/algorithms/solvers) is preferred.

Since we do not delegate all the feasibility checks to external predicates/ functions, in addition to the bilateral interactions between task and motion planing as discussed above (i.e., task plans guide search in motion planning, motion planners check feasibility of task plans and thus affect search of task planners), we need to find tighter interactions between task and motion planning to handle all feasibility checks for a guaranteed-feasible kinematic solution. This requirement brings out the second novelty of our approach — another interaction between causal reasoning and motion planning: when a motion planning failure occurs due to some infeasibility not captured by geometric reasoning handled by external predicates/functions, the description of the planning problem is modified taking into account some domain-specific information from motion-level (e.g., the causes of infeasibilities), and the causal reasoner is asked to solve a more "relevant" planning problem. Therefore, instead of guiding the task planner at the *search level* by manipulating its search algorithm directly, the motion planner guides the task planner at the *representation level* by presenting to it the "right" planning problem.

The algorithms used in the causal reasoner CCALC are sound and complete [4]. Therefore, our hybrid task and motion planning algorithm is sound and complete to the extent that the motion planning algorithm is sound and complete. For instance, the RRT algorithm that we use for motion planning is resolution-sound and probabilistically complete [2], so is our hybrid planning algorithm.

Compared with the recent efforts for integration of task and motion planning at the search level, bringing (a certain amount of) geometric/kinematic details to the representation level allows one to utilize different formulations (declarative or procedural) and reasoning systems. Other novelties of our approach are of this sort: they are inherited from the high-level causality-based representation and reasoning formalism (action language $\mathcal{C}+$) and its automated reasoner (CCALC) we use in our studies. Due to the expressiveness of the formalism, we can handle concurrent actions by multiple agents, nondeterministic effects, multi-valued actions/fluents, additive fluents for reasoning about shared resources, state/transition constraints, and changes that do not involve actions (e.g., ramifications of actions), defaults, etc. Due to the input language of the causal reasoner, we can use external predicates/functions implemented in some programming language of the users' choice, and we can solve modified planning problems that involve temporal constraints. Due to the search mechanism of the causal reasoner, we can compute optimal plans (e.g., in terms of its length or total cost of actions). Due to the modular structure of the causal reasoner, we can experiment with various search engines, such as (parallel) SAT solvers, without modifying their algorithms, and thus inherit advantages of these underlying solvers. Due to the capabilities of the causal reasoner, our approach substantially extends the classes of manipulation problems that can be solved. In particular, we can solve not only planning but also prediction/postdiction and diagnosis problems.

We illustrate the usefulness of this approach with a physical implementation of two problems that involves two robots working for a common goal, in Chapter 4. In each problem, we show how the robots find and execute a plan using our hybrid planning framework. Also, we present snapshots from the process of execution of these plans.

Chapter 4

Different Levels of Integration between High-Level Task Planning and Geometric Reasoning

When we present causal laws to CCALC, we can call external predicates/ functions in them. These predicates/functions are not part of the signature of the domain description (i.e., they are not declared as fluents or actions). They are implemented as functions in some programming language of the user's choice, such as C++. External predicates take as input not only some parameters from the action domain description (e.g., the locations of robots) but also detailed information that is not a part of not the action domain description (e.g., geometric models). This feature enables the integration between geometric reasoning and task planning, as in the example of collision detection. While describing the domain, we can choose to embed all of the geometric constraints into the formulation; or similarly decide to implement some of the constraints. For instance, in a domain where two robots and a payload exist, some of the possible choices would be to check collisions for robots at a state, check collisions for robots during a transition, check collisions for the payload at a state, or check collisions for the payload during a transition. Therefore, there exists a selection of different levels of integration between task planning and geometric reasoning.

Selection of the level of integration is a design problem that effects the performance of the framework, as established in the remainder of this chapter. However, the decision is not trivial as there is a trade-off between the level of integration and the size of the problem. As the integration becomes more profound, in other words, as the number of fluents that is required for an external predicate increases, the time required for transformation of the causal laws into a propositional theory increases dramatically. On the other hand, more integration is ensued by a more physically accurate plan; therefore the expectancy of failure during the motion planning stage is lower.

In this chapter, we systematically investigate the influence of different levels of integration over computational efficiency with two domains: Extended Towers of Hanoi and Maze Problem. In each domain, we give domain descriptions in the language of CCALC with interpretations of causal laws, explain external predicates/functions and their physical meanings, give implementation details on system parameters, specifically SAT solver, motion planner, analyze the influence of different levels of integration, and present a sample problem and show its applicability on physical setups.

4.1 Extended Towers of Hanoi

Towers of Hanoi consists of a number of different-sized rings and three identical pegs which the rings can slide onto; the pegs are attached to and perpendicular to a platform. Initially all the rings are stacked on a single peg in decreasing order so that the largest ring is the base of the tower and the smallest ring is the tip of the tower. The aim is to move all the rings to another peg with respect to the following two constraints: Only the topmost rings can be moved, and one at a time; and no ring can be stacked on top of a smaller one. For n rings, a solution can be found in $2^n - 1$ steps.

We extend this puzzle by adding the possibility of rotating rings about the peg axis with respect to the following constraints: Only the topmost rings can be rotated, in place or while being moved from one peg to another, and one at a time; and a ring can be rotated in multiples of 90° about the peg axis. Therefore, a configuration of rings include also their orientations. We suppose that there is a mark (e.g., an arrow) put on each ring, and that the arrow can be directed in four different directions with 90° intervals at least one direction being along the line connecting the pegs together; then there are four possible orientations of a ring. Also the initial and the goal configurations can be any legitimate configuration of rings. The goal is to move and/or rotate the rings in such a way as to transform the initial configuration to the goal configuration.

We consider Cartesian mechanisms with magnetic end-effectors to carry out the transitions. Since a robot can only move in three dimensional space without rotations, we require two robots in order to rotate the rings. Geometric representation of the domain is given in Figure 4.1.1.

4.1.1 Action Domain Description

In this problem, pegs are denoted by unique constants `p1, p2, p3` and rings are identified by unique numbers `1, 2, 3, 4, ..., n` such that Ring i is smaller than Ring j if $i < j$. We suppose that there is a mark (e.g., an arrow) on

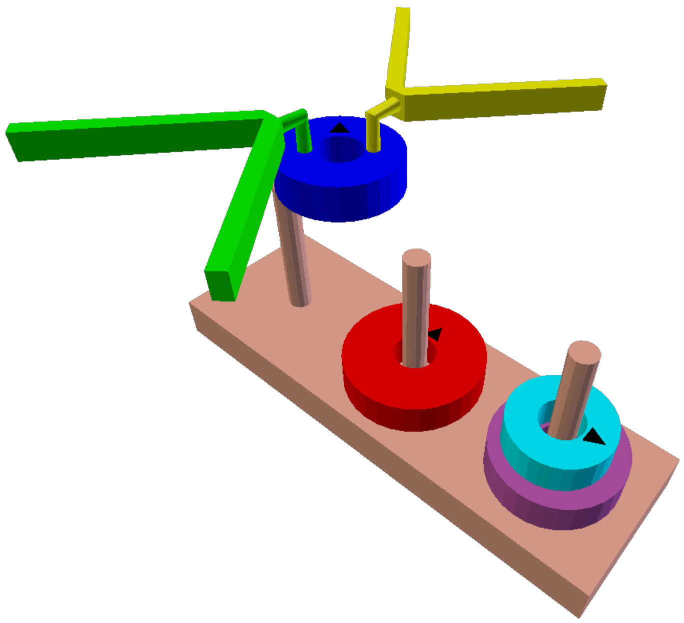

Figure 4.1.1: 3D representation of Extended Towers of Hanoi

each ring, and that the arrow can be directed in four different directions with 90° intervals at least one direction being along the line connecting the pegs together; these four possible orientations are uniquely denoted by `1, 2, 3, 4`. Each ring `D` has a location and an orientation, specified by the functional fluents `loc(D)` and `ort(D)`: a ring `D` is located either on top of another ring `D1` in which case `loc(D) = D1`, or on the base of a peg `P` in which case `loc(D) = P`.

We also consider two auxiliary functions: `base(D) = P` ("ring `D` is somewhere on peg `P`") and `clear(L)` ("ring/peg `L` has no other ring on top of it"). The former is declared as a functional fluent, and defined recursively in terms of `loc(D)`.

We consider two kinds of actions: `move(P1,P2)` (move the ring at the top of the stack on peg `P1` to the top of peg `P2`) and `rotate(P,O)` (rotate the ring at the top of the stack on peg `P` so that its orientation becomes `O`). Notice that we do not introduce an action for rotating a disk while moving it; this will be handled by concurrency.

Direct effects of actions

Moving the topmost ring `D` of the stack at peg `P1`, to the topmost location `L` of the stack at peg `P2` changes the location of `D`:

```
move(P1,P2) causes loc(D)=L
    if base(D)=P1 & clear(D) & base(L)=P2 & clear(L).
```

Here `base(D)=P1 & clear(D)` expresses that ring `D` is on top of the stack at peg `P1`, whereas `base(L)=P2 & clear(L)` expresses that `L` is the topmost location of peg `P2` (i.e., `L` is either the base of `P2` or the ring at the top of the stack on `P2`).

Similarly, the effect of rotating the topmost ring `D` at peg `P` to the orientation `O` can be described by the causal law

```
rotate(P,O) causes ort(D)=O if base(D)=P & clear(D).
```

Preconditions of actions

It is not possible to move a ring from an empty peg:

```
nonexecutable move(P,P1) if clear(P).
```

Only one ring can be moved at each step:

```
nonexecutable move(P,P1) & move(P2,P3) if P@<P2.
```

Table 4.1.1: Problem size of Extended Hanoi Towers (Grounding)

Level	Atoms	Rules	Clauses	New Atoms
No	151	939	4969	468
Full	151	1011	5041	468

Only one ring can be rotated at each step:

```
nonexecutable rotate(P,O) & rotate(P1,O1) if P@<P1.
```

A ring cannot be rotated while moving another ring:

```
nonexecutable move(P,P1) & rotate(P2,O) if P2\=P.
```

Constraints

The following constraint does not allow two rings to occupy the same location:

```
constraint loc(D)\=loc(D1) where D<D1.
```

This constraint prevents moving a ring to two different pegs at the same time. In that sense, it is a qualification constraint.

A ring cannot be on top of a smaller one:

```
constraint loc(D)\=D1 where D>=D1.
```

Note that this is a qualification constraint as well: it prohibits moving a ring on top of a smaller one.

Two Levels of Integration

Due to the simplicity of the domain, two levels are available for investigation. First is level of no integration, which uses the domain description provided above.

Therefore, during the task planning, geometric reasoning will not be considered for feasibility evaluation.

For the second level, full integration is regarded, and following causal law is added to the description.

```
caused false
   if base(D)=P & ort(D)=O after base(D)=P1 & ort(D)=O1
   where notrajectory(P1, O1, P, O).
```

This causal law states that a disk is not allowed to change its base and orientation when no trajectory exists for the transition. The function requires four arguments from the domain. By the use of this external predicate/function, we embed all of the physical changes that can occur during a transition.

4.1.2 Physical Implementation

In our studies, we have physically implemented the setup and used the framework for solving queries. The controllers of the robots are implemented on a PC-based architecture, that compromises of a PCI I/O card and workstation, simultaneously running RTX real-time operating system and Windows XP SP2. For robust trajectory tracking of the end-effectors, feedback controllers are implemented in real-time. The robots are planar parallel mechanisms (pantographs) with two degrees-of-freedom. To enable pick and drop actions, the end-effectors of the pantograph robots are equipped with linear servo motors with magnetic tips acting out of plane. These servo motors are controlled via their dedicated amplifiers driven by the analog outputs of the control card.

Below is one of the queries we studied:

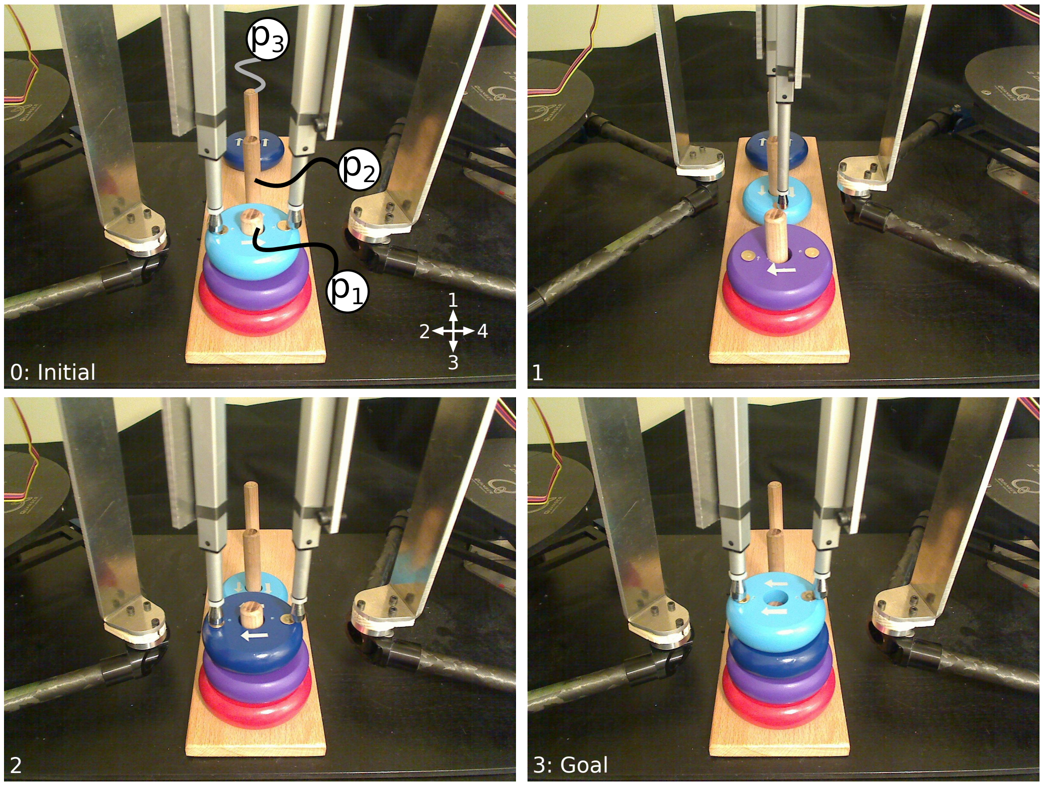

Figure 4.1.2: Snapshots of Extended Towers of Hanoi implementation

```
:- query
label:: 0 ;
maxstep:: 0..infinity;
0: loc(1) = 3, ort(1)=4,
    loc(2) = p3, ort(2)=1,
    loc(3) = 4, ort(3)=2,
    loc(4) = p1, ort(4)=2;
maxstep: loc(1) = 2, ort(1)=2,
    loc(2) = 3, ort(2)=2,
    loc(3) = 4, ort(3)=2,
    loc(4) = p1, ort(4)=2;
```

Figure 4.1.2 is the snapshots taken during the execution of the plan on the physical system. The goal is to collect all of the rings on `p1` and facing direction `2`. However, the smallest ring `1` is already at `p1`, therefore it needs to be removed before `2` can be placed. Moreover, `1` is facing `4`. Due to physical constraints, it cannot be rotated to its desired orientation in one step. According to the plan, the first set of actions are to move the topmost ring from `p1` to `p2`, `move(p1, p2)`, and rotate it to direction `3`, `rotate(p1, 3)`. Next, `move(p3, p1)` and `rotate(p3, 2)` move the topmost ring at `p3`, `2`, to `p1` and face it to its desired orientation, `2`. Finally, `move(p2, p1)` and `rotate(p2, 2)` move the last ring to `p1` and rotate it to `2`.

4.1.3 Experimental Results

Framework is asked for a solution; using above description and the query below as the inputs.

```
:- query
label :: 0 ;
maxstep :: 15 ..infinity;
0: loc(1) = 2, loc(2) = 3, loc(3) = 4, loc(4) = p1,
   ort(1)=2, ort(2)=2, ort(3)=2, ort(4)=2;
maxstep: loc(1) = 2, loc(2) = 3, loc(3) = 4, loc(4) = p2,
   ort(1)=2, ort(2)=4, ort(3)=2, ort(4)=4.
```

Table 4.1.2: Plan informations of Extended Hanoi Towers

Level	Query size		maxstep	Total plans	Total time
	Atoms	Clauses			(s)
No	7699	71749	16	87	374
	7699	71749	16	89	393
	7699	71749	16	95	466
	7699	71749	16	91	410
	7699	71749	16	94	416
	7699	71749	16	93	418
	7699	71749	16	91	400
	7699	71749	16	87	379
	7699	71749	16	88	380
	7699	71749	16	88	383
Average	7699	71749	16	90.3	401.9
Std. Dev.	0	0	0	2.3	27.4
Full	7699	72901	16	1	9
	7699	72901	16	1	13
	7699	72901	16	1	10
	7699	72901	16	1	11
	7699	72901	16	1	11
	7699	72901	16	1	11
	7699	72901	16	1	11
	7699	72901	16	1	11
	7699	72901	16	1	10
	7699	72901	16	1	11
Average	7699	72901	16	1	10.8
Std. Dev.	0	0	0	0	0.9

which states a problem where, in the beginning, all of the disks are located at `p1` and facing direction `2`; and we want the disks to be at `p2` with orientations are `2`, `4`, `2` and `4`, in respective order.

For this implementation, we have employed MANYSAT as the SAT solver, which is a parallel SAT solver that is designed for multiple core computations. As a result, the task plan, that is calculated for one query, is not deterministic; although it is still optimal in plan length. In addition, we have used RRT (Algorithm 1) as the motion planner, which is a probabilistic motion planning algorithm. Consequently, the number of total task plans and computation time required for obtaining a full plan are not deterministic either. Therefore, more than multiple runs are required for a sound analysis.

By asking the same query, Table 4.1.1 and Table 4.1.2 are obtained as the result of ten runs. By observing the data, we conclude that the integration between geometric reasoning and task planning has increased the efficiency. In the non-integrated case, the task planning was not able to capture invalid transitions, therefore guidance of motion planning was required in order to obtain a feasible solution.

4.2 Maze Problem

Consider two robots, and a payload (a long metal stick) on a platform. Suppose that each robot has a magnet at its end-effector so that it can hold the payload only at one end. None of the robots can carry the payload alone; they have to hold the payload at both ends to be able to carry it. The goal is to place the payload at a specified goal position on the platform. Figure 4.2.1 is geometric representation of a sample domain.

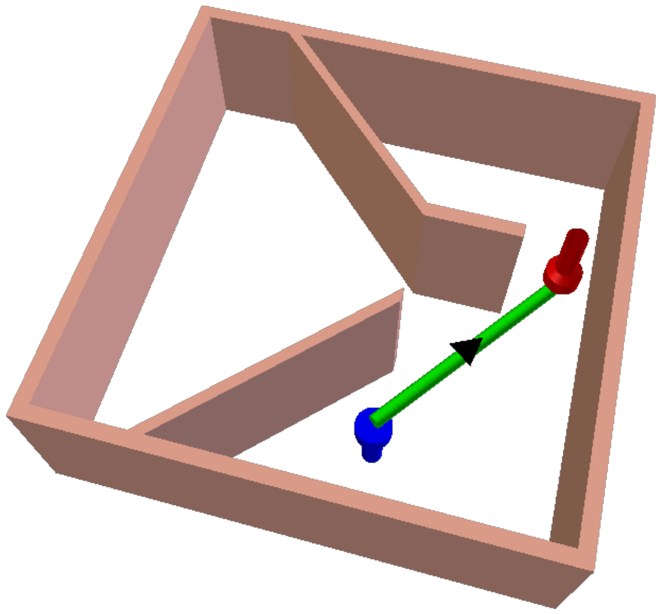

Figure 4.2.1: 3D representation of Maze Problem

4.2.1 Action Domain Description

We view the platform as a maze. We represent the robots by the constants `r1` and `r2`. We describe the payload by its end points, and denote them by the constants `pl1` and `pl2`.

We characterize each robot by its end-effector, and describe its position by a grid point on the maze. The location `(X,Y)` of a robot `R` is specified by two functional fluents, `xpos(R)=X` and `ypos(R)=Y`. Similarly, the location `(X,Y)` of an end point `P1` of the payload is specified by two fluents, `xpay(P1)=X` and `ypay(P1)=Y`. Movements of a robot `R` in some direction `D` are described by actions of the form `move(R,D)`. Each such action has an attribute that specifies the number of steps to be taken by the robot. The robots are to move in a linear fashion for each transition.

In the following, suppose that `R` denotes a robot, `P1` and `P2` denote the end points of the payload, `N` and `N1` range over nonnegative integers `1`, ..., `maxN`, and `D` and `D1` range over all directions, `up`, `down`, `right`, `left`. Also suppose that `X1`, `X2`, `Y1`, `Y2` range over nonnegative integers `0`, ..., `maxXY`.

We present the causal laws in the language of CCALC.

Direct effects of actions

We describe the effect of a robot's moving right, by the causal laws

```
move(R,right) causes xpos(R)=X2
    if steps(R,right)=N & xpos(R)=X1
    where X2=X1+N & X2 =< maxN.
```

Similarly, we describe the effects of moving in other directions.

Ramifications

If a robot `R` is at the same location as an end point `P1` of the payload, the end-effector of that robot attracts that end point:

```
caused on(R,P1) if xpos(R)=xpay(P1) & ypos(R)=ypay(P1).
```

Then the location of the payload is determined by the locations of the robots:

```
caused xpay(P1)=X1 if on(R,P1) & xpos(R)=X1.
caused ypay(P1)=Y1 if on(R,P1) & ypos(R)=Y1.
```

Preconditions of actions

We describe that a robot cannot move in opposite directions by the causal laws

```
nonexecutable move(R,up) & move(R,down).
nonexecutable move(R,left) & move(R,right).
```

Note that we do not prohibit a robot to move in vertical directions concurrently: For instance, a robot can move up and right during the same transition. However, this transition must be as a linear motion.

We describe each robot's range of motion, taking into account the Pythagorean Theorem, by the causal laws

```
nonexecutable move(R,D) & move(R,D1)
    if D @< D1 & steps(R,D)=N & steps(R,D1)=N1
    where N*N+N1*N1 > maxN*maxN.
```

The robots can carry the payload only if both of them hold the payload at its end points.

```
nonexecutable move(R,D) if -canCarry & on(R,P1).
```

The conditions under which two robots can carry the payload are described by `canCarry`:

```
caused canCarry if on(r1,P1) & on(r2,P2) & P1\=P2
    after on(r1,P1) & on(r2,P2) & P1\=P2.
```

Note that it is required by the causal laws above that the robots wait for one step immediately after they hold the payload at both ends.

Constraints

We make sure that a payload cannot move places unless it is carried by the causal laws

```
caused false if xpay(P1)=X1 & X1\=X2
    after -canCarry & xpay(P1)=X2.
caused false if ypay(P1)=Y1 & Y1\=Y2
    after -canCarry & ypay(P1)=Y2
```

Since CCALC can only deal with integers, we cannot keep track of the exact locations of the payload. (Consider, for instance, moving one end of the horizontally-situated payload up by 2 steps.) Therefore, we allow the payload's length change with a small tolerance for a more flexible motion. Suppose that `linklengthsq` denotes the square of the length of the payload; and `tolerance` denotes the maximum change allowed in the payload's length. The following laws ensure that the payload's length cannot increase/decrease more than `tolerance`:

```
caused false
    if xpay(pl1)=X1 & xpay(pl2)=X2
        & ypay(pl1)=Y1 & ypay(pl2)=Y2
    where
        (X2-X1)*(X2-X1)+(Y2-Y1)*(Y2-Y1)
            <(linklengthsq-tolerance) ++
        (X2-X1)*(X2-X1)+(Y2-Y1)*(Y2-Y1)
            >(linklengthsq+tolerance).
```

Three Levels of Integration

For this domain, we investigate three levels of integration for comparison purposes. In the first level, no integration is considered: No geometric reasoning is embedded into causal-reasoning. Therefore, the domain description, for this case, is as described above.

In the second level, which is considered to be partial integration, following two causal laws are added to the domain description:

```
caused false if xpos(R)=X1 & ypos(R)=Y1
    after xpos(R)=X2 & ypos(R)=Y2
    where collision_robot_transition(X1,Y1,X2,Y2).
caused false if xpay(pl1)=X1 & ypay(pl1)=Y1
            & xpay(pl2)=X2 & ypay(pl2)=Y2 & canCarry
    where collision_payload_state(X1,Y1,X2,Y2).
```

Former causal law above states that the transition of a robot from `(X1,Y1)` to `(X1, Y2)` is not allowed to cause collision. The external predicate function `collision_robot_transition`, which is implemented in C++ and Python, returns the collision situation of such a transition, and requires four arguments. Note that, since the robots are described to move only linearly, there is no need to use a motion planner, but only geometric reasoning for collision detection. By the addition of this causal law, the task planner also fulfills the duty of a motion planner, but only for the motion of the robots.

Latter causal law states that a payload whose end points are located at `(X1, Y1)` and `(X1, Y2)` is not allowed to cause collision. Similar to the other, external predicate function `collision_payload_state` is implemented in the same manner, and requires four parameters as well.

Third level, which is considered as full integration in this domain, requires addition of the following causal law:

```
caused false if xpay(pl1)=X1 & xpay(pl2)=X2
        & ypay(pl1)=Y1 & ypay(pl2)=Y2
    after xpay(pl1)=X3 & xpay(pl2)=X4
        & ypay(pl1)=Y3 & ypay(pl2)=Y4
    where collision_payload_transition(X1, Y1, X2, Y2,
                                            X3, Y3, X4, Y4).
```

By introducing this causal law, transition of a payload is also handled during task planning, which is the only geometric event that is not included in the partial integration.

4.2.2 Physical Implementation

We have physically implemented the setup with simple robots, using LEGO MINDSTORMS NXT. The controllers of the robots are implemented on NXT, an embedded controller, with an ARM7 microprocessor. Feedback controllers are implemented in one of the compatibles languages of NXT, called NXC. The robots are wheeled mechanisms with a one degree-of-freedom arm, resulting in two degrees-of-freedom. To manipulate the payload, the end-effectors of the robots are equipped with magnetic tips acting out of plane.

Following query was solved using the framework.

```
:- query
label :: 0 ;
maxstep :: 0..infinity;
```

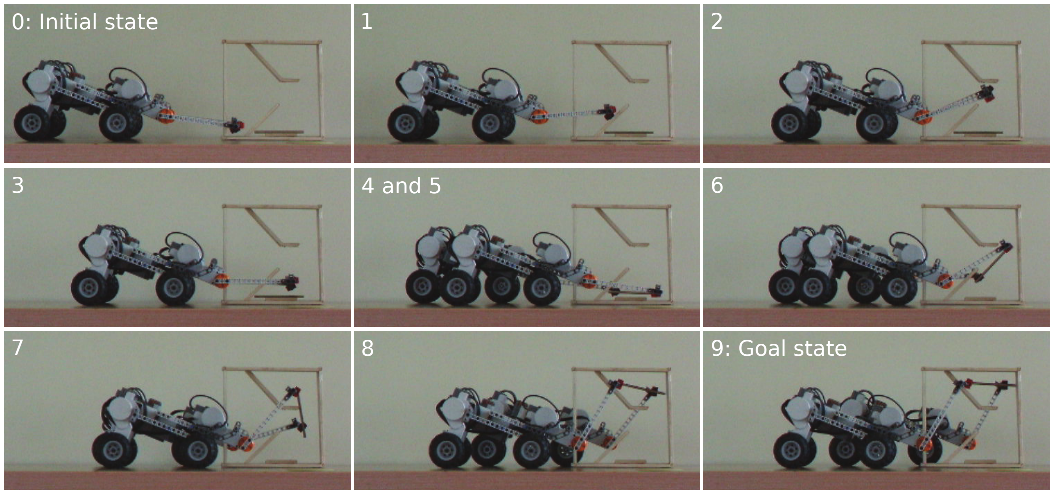

Figure 4.2.2: Snapshots of Maze Problem implementation

```
% At initial step
0:
    % r1 is at (1,1)
    xpos(r1)=1, ypos(r1)=1,
    % r2 is at (1,1)
    xpos(r2)=1, ypos(r2)=1,
    % pl1 is at (4,1)
    xpay(pl1)=4, ypay(pl1)=1,
    % pl2 is at (9,1)
    xpay(pl2)=9, ypay(pl2)=1,
    % Robots are not holding the payload
    -canCarry;
% At goal step
```

```
maxstep:
    % pl1 is at (9,9)
    xpay(pl1)=9, ypay(pl1)=9,
    % pl2 is at (4,9)
    xpay(pl2)=4, ypay(pl2)=9.
```

In Figure 4.2.2 are snapshots taken during the execution of the plan. First, the robots move towards end-points of the payload (steps 1-4). Then they wait for one step to ensure the magnetic tips hold the end-points (step 5). Afterwards, the robots carry the payload to its goal configuration (steps 6-9).

4.2.3 Experimental Results

For this implementation of the system, no probabilistic motion planner is employed. In addition, SAT solver of choice is MINISAT, which is a single processor SAT solver that gives the same solution to each identical trial. Therefore, this implementation is deterministic, unlike the one in previous section. Thus, we consider 10 problem instances with different mazes (Figure 4.2.3). The problem instances are given in Table 4.2.1.

Table 4.2.2 shows the grounding sizes of the problems for each of the mazes. As the gridsize and level of integration increases, the problem becomes larger also, therefore more computational effort is needed. Especially for mazes (2) and (3), grounding stage could not be completed successfully due to limits of physical memory. Consequently, no plan can be computed. Table 4.2.3 shows the planning results. In general, the number of total plans decreases as the level of integration increases. However, in most cases, full integration proves to be computationally expensive. When no and partial integrations are compared, partial integration proves more efficient in more complicated problems.

Table 4.2.1: Queries of the Maze Problem

Query	Maze	Initial State				Goal State	
		r1	r2	pl1	pl2	pl1	pl2
1	1	(0, 0)	(0, 0)	(0, 1)	(5, 1)	(5, 5)	(0, 5)
2	2	(0, 0)	(0, 0)	(0, 1)	(5, 1)	(0, 1)	(5, 1)
3	3	(0, 0)	(0, 0)	(2, 0)	(2, 5)	(3, 5)	(3, 0)
4	3	(0, 0)	(0, 0)	(2, 0)	(2, 5)	(3, 0)	(3, 5)
5	4	(0, 0)	(0, 0)	(3, 0)	(0, 4)	(7, 4)	(4, 0)
6	4	(0, 0)	(0, 0)	(3, 0)	(0, 4)	(4, 0)	(7, 4)
7	5	(0, 0)	(0, 0)	(0, 1)	(0, 6)	(7, 1)	(7, 6)
8	5	(0, 0)	(0, 0)	(0, 1)	(0, 6)	(7, 6)	(7, 1)
9	6	(1, 1)	(1, 1)	(4, 1)	(9, 1)	(9, 9)	(4, 9)
10	6	(1, 1)	(1, 1)	(4, 1)	(9, 1)	(1, 4)	(1, 9)

Table 4.2.2: Problem size of Maze (Grounding)

Maze	Level	Atoms	Rules	Clauses	New Atoms	Success
(1)	No	154	3242	5510	510	☑
	Partial	154	3242	5510	510	☑
	Full	154	3242	5510	510	☑
(2)	No	154	3242	5510	510	☑
	Partial	154	4978	7246	510	☑
	Full	154	1559012	1561280	510	☒
(3)	No	154	3242	5510	510	☑
	Partial	154	4682	6950	510	☑
	Full	154	1439932	1442200	510	☒
(4)	No	186	8450	12222	830	☑
	Partial	186	15898	19670	830	☑
(5)	No	186	8450	12222	830	☑
	Partial	186	20898	24670	830	☑
(6)	No	234	28292	34920	1430	☑
	Partial	234	63416	70044	1430	☑

Table 4.2.3: Plan informations of Maze

Query	Maze	Level	Query size		maxstep	Total plans	Total time
			Atoms	Clauses			(s)
1	(1)	no	2819	22000	6	1	5
		partial	2819	22000	6	1	34
		full	2819	22000	6	1	79220
2	(2)	no	2819	22000	6	2	7
		partial	2819	29236	6	1	35
		full	N/A	N/A	N/A	N/A	N/A
3	(3)	no	2388	18702	5	2	10
		partial	2388	23646	5	1	35
		full	N/A	N/A	N/A	N/A	N/A
4	(3)	no	2819	22000	6	381	1256
		partial	2819	27820	6	4	44
		full	N/A	N/A	N/A	N/A	N/A
5	(4)	no	2933	32828	4	3	16
		partial	2933	56312	4	4	238
6	(4)	no	4850	53514	7	1019	6763
		partial	4850	92954	7	320	1529
7	(5)	no	6128	67278	9	519	2019
		partial	6128	153390	9	1	228
8	(5)	no	6128	67278	9	557	2416
		partial	6128	153390	9	4	223
9	(6)	no	N/A	N/A	N/A	3409+	N/A
		partial	10763	435777	10	9	951
10	(6)	partial	11774	652722	11	1	877

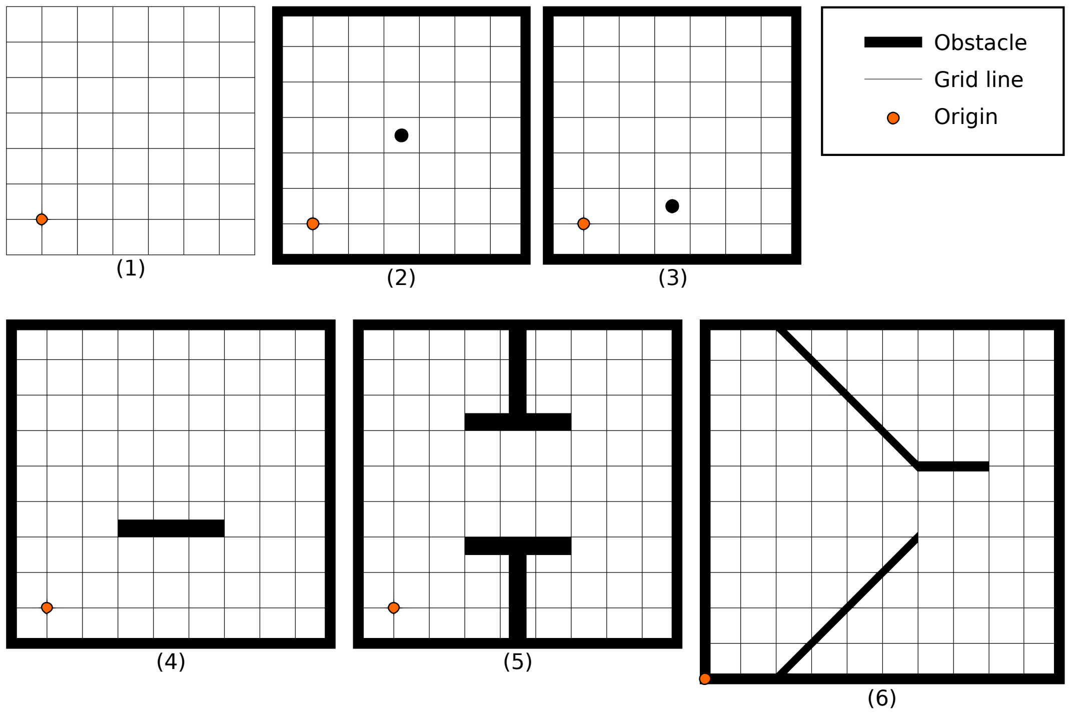

Figure 4.2.3: Domains for the Maze Problem

4.3 Discussion

We observe from the experimental results presented in Tables 4.1.1, 4.1.2, 4.2.2, and 4.2.3 that the level of integration of low-level geometric reasoning into high-level causality-based reasoning dramatically effects the computational efficiency.

1. With more detailed external predicates/functions, it gets harder to compute a task plan.

2. Having less number of physical constraints as causal laws in the domain description, in general, decreases the applicability of a plan and increases

the number of plannings that are required to obtain a feasible task plan. Therefore computation can require considerably long time.

In short, using external predicates/functions that necessitate a reasonable number of domain variables (which, in many domains, corresponds to partial integration) proves to be the best policy.

On very simple domains or queries, level of no integration may seem to have a good performance, however, such cases are very limited. Even in a very simple domain as extended towers of Hanoi, addition of a single constraint with geometric reasoning greatly increases computational efficiency. In a more geometrically relevant problem (maze (6) of Maze Problem), although the calculation of a task plan demands a short time, a feasible task plan could not be calculated in level of no integration; due to allocated memory limitations. This level should be avoided, unless domain and task in consideration are considerably simple.

In this chapter, we introduced two non-trivial example domains with the details of their descriptions and external predicates/functions. We investigated the effects of the amount of integration of low-level geometric reasoning into high-level causality based reasoning in a systematic manner and presented a comparison with provided data. Moreover, we concluded on a guideline for choosing the level of integration.

Chapter 5

Generalization to Continuous Domains

One of the common characteristics of the previous examples is that reasoning included exact positions of object that were manipulated. However, this need not be the case in a manipulation problem. One can argue that, in the human thought process, rather than considering exact coordinates, qualitative notions are taken into account. For instance, saying "this box needs to be located on top of the table" is more meaningful in a thought process than saying "the coordinates of the box needs to be this".

Motivated by this, we introduce a new approach where *locations* are considered instead of *exact positions*. Locations can be regarded as a set of positions in space, which has the same characteristics for a given purpose. For instance, if a box needs to be located on a table, any position on the table satisfies the requirement. On the other hand, an item may need to be located at a specific position. Still, this approach does not lose expressiveness as a location can correspond to a single position in space.

Algorithm 3 IKBiRRT

```
Require: Initial state $q_s$, WGR $W$
  $T_a$.Init($q_s$); $T_b$.Init(NULL);
  while TimeRemaining() do
    $T_{goal}$ = GetBackwardTree($T_a$, $T_b$);
    if $T_{goal}$.size = 0 or rand(0, 1) < $P_{sample}$ then
      AddIKSolutions($T_{goal}$, $W$);
    else
      $q_{rand} \leftarrow$ RandConfig();
      $q^a_{near} \leftarrow$ NearesetNeighbor($T_a$, $q_{rand}$);
      $q^a_{reached} \leftarrow$ Extend($T_a$, $q^a_{near}$, $q_{rand}$);
      $q^b_{near} \leftarrow$ NearesetNeighbor($T_b$, $q^a_{reached}$);
      $q^b_{reached} \leftarrow$ Extend($T_b$, $q^b_{near}$, $q^a_{reached}$);
      if $q^a_{reached} = q^b_{reached}$ then
        $P \leftarrow$ ExtractPath($T_a$, $q^a_{reached}$, $T_b$, $q^b_{reached}$);
        return SmoothPath($P$);
      else
        Swap($T_a$, $T_b$);
      end if
    end if
  end while
  return NULL
```

Describing a space by locations rather than by a grid is expected to ease the burden on the high-level: Depending on the resolution, the number of locations would be less than the number of grid points. However, this implies that geometric reasoning must be handled differently, as goal configurations are no longer single points. In the following section, a variant of RRT path planning algorithm is introduced. Then, in Section 5.2, the extension of the manipulation planning system for handling continuous regions is explained.

5.1 Inverse Kinematics Bidirectional RRT

In order to have more flexibility, in this work, we utilize a more recent variant of RRT, which was one of the two algorithms introduced in [3]. This approach introduces workspace goal regions (WGR) as a means of representing the desired goal in the task space, rather than to reach a single point in configuration space. Furthermore, it can handle multiple disconnected regions.

We adopted inverse kinematics bidirectional RRT (IKBiRRT, Algorithm 3). Unlike RRT, this algorithm uses both configuration and task spaces. It probabilistically generates points in WGRs, and solves the inverse kinematics problem to obtain configuration space points. Then, depending on the collision of these points, they are added to the goal tree. Remainder of the algorithm is similar to that of RRT.

Since we require the inverse kinematics, let us solve forward and inverse kinematics of the robot KUKA youBot, a robotic manipulator with a holonomic base that is designed for research applications.

5.1.1 Kinematics of KUKA youBot

KUKA youBot (Figure 5.1.1) is combination of a 5 DoF robotic arm and a holonomic platform; therefore, it has 8 degrees of freedom which renders it as a redundant manipulator. Its generalized coordinates are given below.

- x: Projection of $\mathbf{p^{oa}}$ over $\mathbf{i_N}$
- y: Projection of $\mathbf{p^{oa}}$ over $\mathbf{j_N}$
- q_0: Rotation angle of frame A with respect to frame N about $\mathbf{k_N}$
- q_1: Rotation angle of frame B with respect to frame A about $-\mathbf{j_A}$
- q_2: Rotation angle of frame C with respect to frame B about $-\mathbf{j_B}$
- q_3: Rotation angle of frame D with respect to frame C about $-\mathbf{j_C}$
- q_4: Rotation angle of frame E with respect to frame D about $-\mathbf{j_D}$

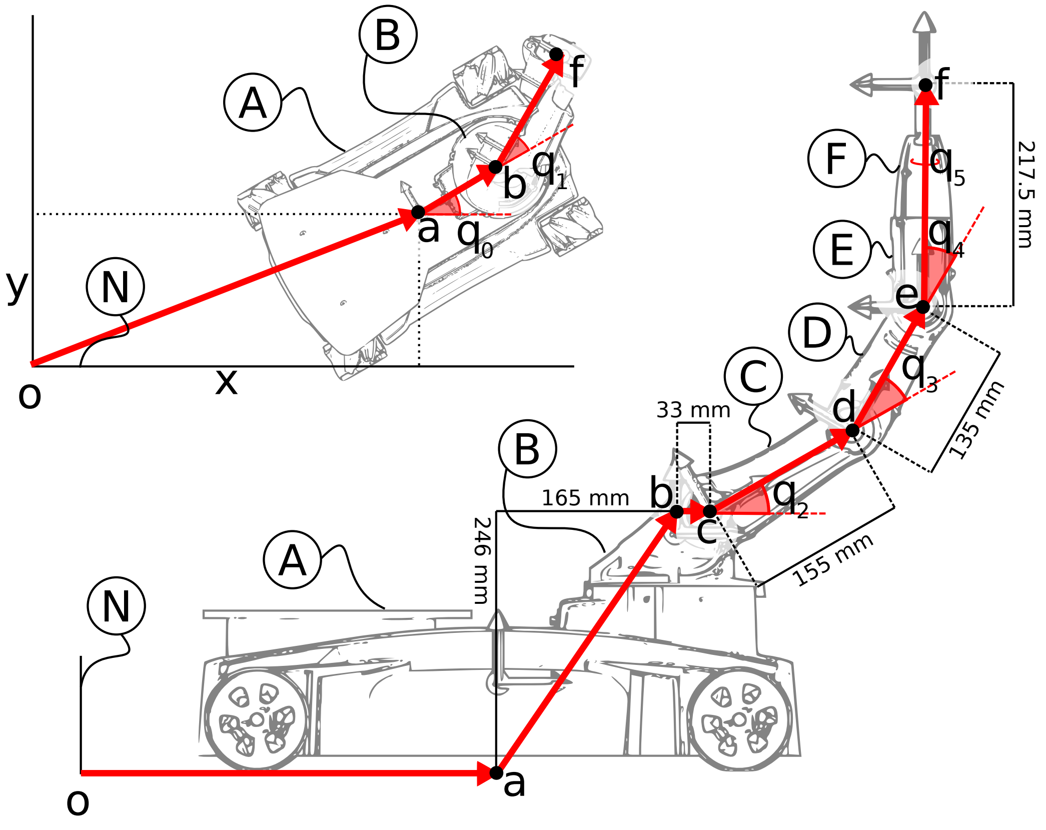

Figure 5.1.1: KUKA Youbot

- q_5: Rotation angle of frame F with respect to frame E about $\mathbf{i_E}$

where $\mathbf{i_X}, \mathbf{j_X}, \mathbf{k_X}$ denote $\mathbf{i}, \mathbf{j}, \mathbf{k}$ basis vectors of frame X, minding a right-handed frame; and $\mathbf{p^{ab}}$ denotes the position vector from point a to point b.

Due to being an open kinematic chain, its forwards kinematics analysis is trivial. Position of the end-effector $\mathbf{p^{of}}$ is given by the following equation.

$$\mathbf{p^{oa}} + \mathbf{p^{ab}} + \mathbf{p^{bc}} + \mathbf{p^{cd}} + \mathbf{p^{de}} + \mathbf{p^{ef}} = \mathbf{p^{of}} \tag{5.1}$$

If the above expression is decomposed in to its components on each axis, below expressions are obtained for the coordinates $[x_f \; y_f \; z_f]^T$ of point f in the Newtonian frame N:

$$\begin{aligned} x_f = x &+ 165\,cos(q_0) + 33\,cos(q_1+q_0) + 155\,sin(q_2)cos(q_1+q_0) \\ &+ 135\,sin(q_2+q_3)cos(q_1+q_0) + 217.5\,cos(q_1+q_0)sin(q_2+q_3+q_4) \end{aligned} \tag{5.2}$$

$$\begin{aligned} y_f = y &+ 165\,sin(q_0) + 33\,sin(q_1+q_0) + 155\,sin(q_2)sin(q_1+q_0) \\ &+ 135\,sin(q_1+q_0)sin(q_2+q_3) + 217.5\,sin(q_1+q_0)sin(q_2+q_3+q_4) \end{aligned} \tag{5.3}$$

$$z_f = 246 + 155\,cos(q_2) + 135\,cos(q_2+q_3) + 217.5\,cos(q_2+q_3+q_4) \tag{5.4}$$

For orientation representation, $body321$, i.e. Yaw(ψ)-Pitch(θ)-Roll(ϕ) Euler angles, notation is adopted during implementation:

$$\psi = q_0 + q_1 \tag{5.5}$$

$$\theta = q_2 + q_3 + q_4 \tag{5.6}$$

$$\phi = q_5 \tag{5.7}$$

Inverse kinematics problem, on the other hand, is more cumbersome. Since the robot has two redundant degrees of freedom, in addition to the end-effector pose, two additional pieces of information are needed. In this analysis, orientation of the robot base q_0 is assumed to be given. Also, an auxiliary variable s (Figure 5.1.2) is defined and assumed to be given. Depending on the joint limits, different solutions exist for each pair of (q_0, s).

To solve the inverse kinematics problem, let us decompose the problem into orientation and position problems, and start with orientation problem. Since the robot does not have an obvious spherical wrist, three suitable degrees of freedom should be dedicated for orientation purposes. We choose q_1 for ψ and q_4 for θ; for q_5 is the only choice for ϕ.

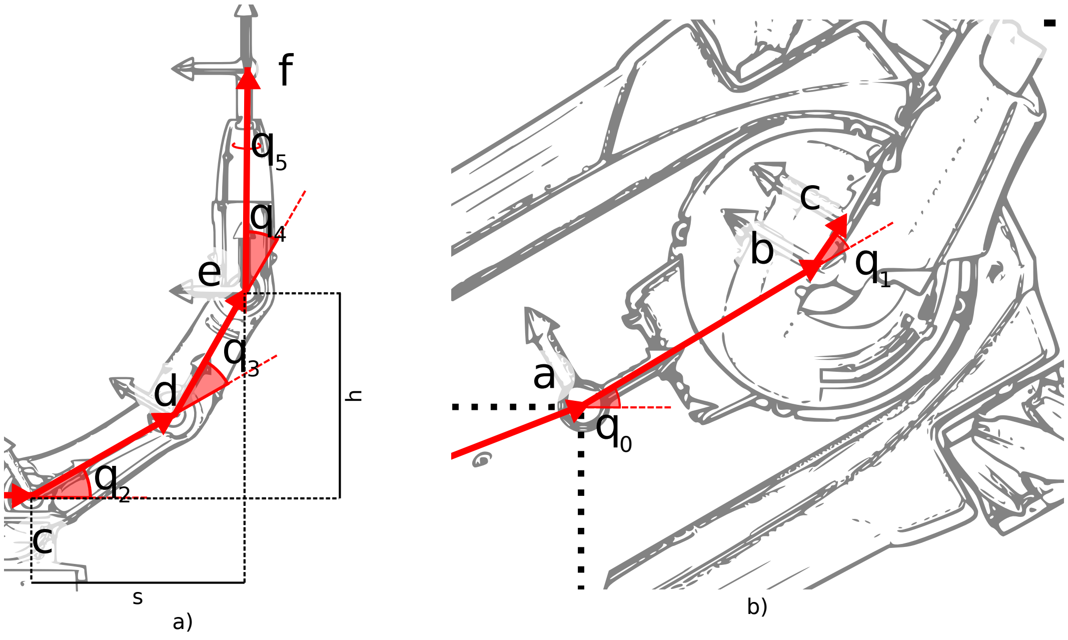

Figure 5.1.2: Youbot position inverse kinematics component

Let us, also, decompose position problem into two parts: Height (Figure 5.1.2.a) and planar position (Figure 5.1.2.b). Two joint variables determine the height of the end-effector: q_2 and q_3. These angles can be extracted by using trigonometric equalities for given s and h. h is calculated by the following equation:

$$h = z_f - 246.0 - 217.5\, sin(\theta) \tag{5.8}$$

q_2 and q_3 are calculated as:

$$q_3 = \pm 2\, atan\left(\sqrt{\frac{(155.0+135.0)^2 - (s^2+h^2)}{(s^2+h^2) - (155.0-135.0)^2}}\right) \tag{5.9}$$

$$\begin{aligned} q_2 &= atan2(h,s) \\ &\quad - atan2(135.0\, sin(q_3), 155.0 + 135.0\, cos(q_3)) \end{aligned} \tag{5.10}$$

where, elbow-up and elbow-down configurations depend on the sign of q_3. Using Equation 5.6, q_4 is given as:

$$q_4 = \theta - q_2 - q_3 \tag{5.11}$$

At this point, position of point b can be calculated. Next is the planar component of the position, in other words base pose. For a given q_0, Equation 5.5 gives q_1.

$$q_1 = \psi - q_0 \tag{5.12}$$

x and y can trivially be obtained.

$$x = x_f - (217.5\ cos(\theta) + s + 33.0)\ cos(\psi) - 165.0\ cos(q_0) \tag{5.13}$$

$$y = y_f - (217.5\ cos(\theta) + s + 33.0)\ sin(\psi) - 165.0\ sin(q_0) \tag{5.14}$$

5.2 System Architecture

Our generalized manipulation planning architecture is illustrated in Figure 5.2.1. The idea is to abstract configurations at the representation level in the domain description to consider locations, instead of exact position. Although the task planning mechanism remains as it was, abstraction needs to be compensated as exact positions are requisite for manipulation planning and execution.

The system is provided with domain description, planning problem, and geometric information (models, positions, kinematics, etc.); and calculates task and motion plans to satisfy the goals.

First, the system performs a preprocess. By using the geometric information, this preprocess parcels off the locations into regions, which is the medium that guides the geometric reasoning; determines the regions and locations that objects are located at; and prepares the domain description and planning problem accordingly.

Once the domain description and planning problem are ready, they are passed on to CCALC for the calculation of the task plan. Note that if a task plan can be calculated, it will be at region level. However, for execution on real robots, exact positions are still required. In an attempt to overcome this; we introduce a method that extracts position information from regions. At each step, the system samples each region to position an object (Algorithm 4). Using the task plan, the manipulated item at each step is positioned in space, by the means of sampling. If

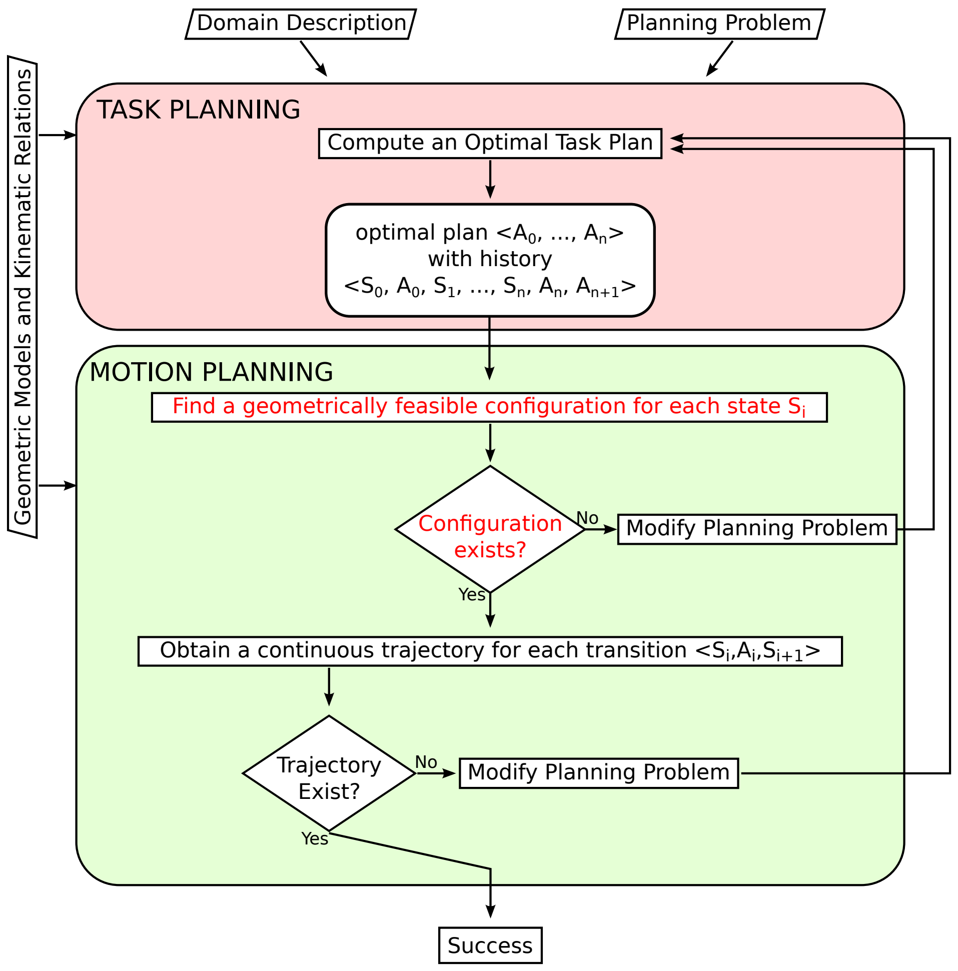
Domain Description
Planning Problem
Geometric Models and Kinematic Relations
TASK PLANNING
Compute an Optimal Task Plan
optimal plan <A0, ..., An> with history <S0, A0, S1, ..., Sn, An, An+1>
MOTION PLANNING
Find a geometrically feasible configuration for each state Si
Configuration exists?
No
Modify Planning Problem
Yes
Obtain a continuous trajectory for each transition <Si,Ai,Si+1>
Trajectory Exist?
No
Modify Planning Problem
Yes
Success

Figure 5.2.1: New planning architecture

such a position cannot be calculated, this may be due to two reasons.

- There are obstacles in the region that prevent the item from being positioned.
- Other manipulable objects in the same region occupy the space.

In the former, nothing can be done to overcome, therefore, a temporal constraint that states the state is invalid needs to be added. In the latter case, however, information of the objects whose position has been changed in the task plan is utilized. In Algorithm 4, first, rearranging the objects in the same region is tried. If this fails also, a similar idea is tried for the objects in the same location, without changing their regions. If the outcome is failure, only then the temporal constraint that renders the state invalid is introduced.

After successfully placing the objects at each state, the initial and goal configurations are decided; therefore the motion planner can be called. The motion planning algorithm chosen for the new framework is IKBiRRT (Algorithm 3). The motivation behind this selection is the following: After an object is positioned at a point, there may be multiple robot configurations that reaches the same point, especially in the case of using redundant manipulators. Thus, by the use of this algorithm, the system can consider multiple goal configurations in the configuration space of the robot.

If path planning is also successful, then the resulting plan is passed to execution and monitoring systems. In a case where a failure shall occur, the planning problem is modified accordingly, and the system is asked for a new plan (loops in Figure 5.2.1).

The applicability of the architecture is shown in the following example.

Algorithm 4 DETERMINE_STATES

```
Require: Task plan $T$, History $H$
  $planLength \leftarrow$ getPlanLength($T$);
  $stepno \leftarrow$ 0;
  while $stepno < planLength$ do
    $A \leftarrow$ getActionName($T$, $stepno$);
    $F \leftarrow$ getFluents($T$, $stepno$);
    if $A =$ pickup then
      $item \leftarrow$ getPickUpItem($T$, $stepno$);
      setAllPosesAfterStep($item$, $robotCarrying$, $stepno + 1$);
    end if
    if $A =$ putdown then
      $item \leftarrow$ getHeldItem($F$, $stepno$);
      if itemPositionedAtStep($item$) $< stepno$ then
        $region \leftarrow$ getTargetRegion($A$, $stepno$);
        $position \leftarrow$ positionItemsAtRegion($item$, $region$);
        if exists($position$) then
          setAllPosesAfterStep($item$, $position$, $stepno + 1$);
          itemPositionedAtStep($item$) $\leftarrow stepno$;
        else
          $items \leftarrow$ getItemsAtRegion($S$, $stepno$, $region$);
          $positions \leftarrow$ positionItemsAtRegion($items$, $region$);
          if exists($positions$) then
            setAllPosesAfterStep($items$, $positions$, $stepno + 1$);
            itemsPositionedAtStep($items$) $\leftarrow stepno$;
            $stepno \leftarrow -1$;
          else
            $location \leftarrow$ getLocationOfRegion($region$);
            $items, regions \leftarrow$ getItemsAndRegionsAtLocation($S$, $stepno$, $location$);
            $positions \leftarrow$ positionItemsAtRegions($items$, $regions$);
            if exists($positions$) then
              setAllPosesAfterStep($items$, $positions$, $stepno + 1$);
              itemsPositionedAtStep($items$) $\leftarrow stepno$;
              $stepno \leftarrow -1$;
            else
              addTemporalConstraint();
            end if
          end if
        end if
      end if
      $stepno \leftarrow stepno + 1$;
    end if
  end while
  return NULL
```

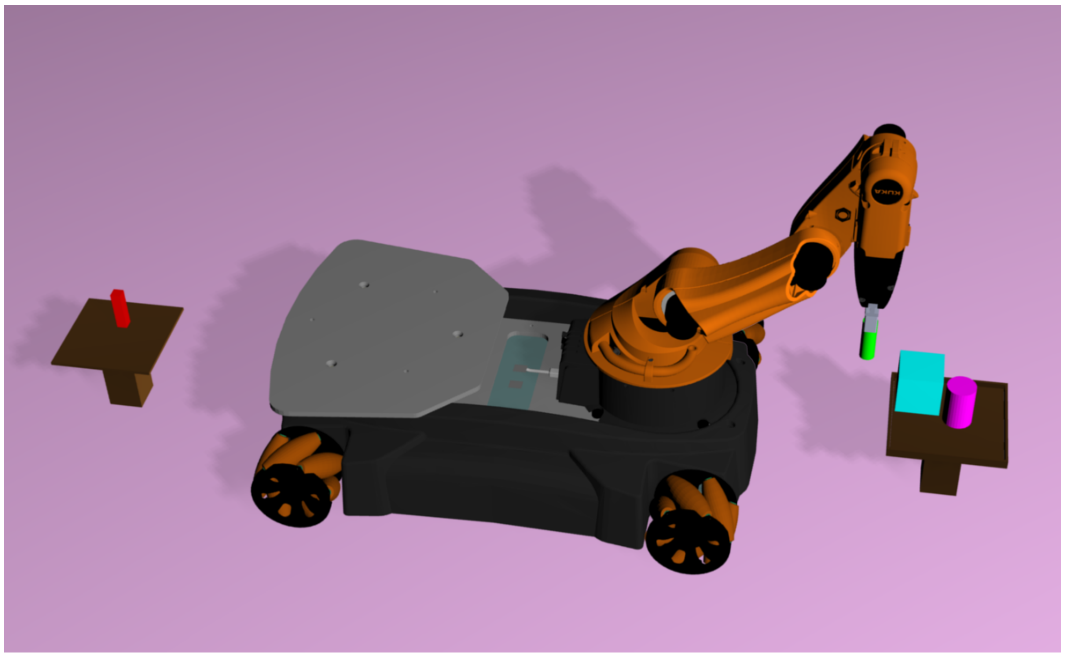

Figure 5.3.1: 3D representation of Waiter Robot problem

5.3 Case Study: Waiter Robot

Consider a KUKA youBot, several heterogeneous items, i.e., items that are not identical, and locations. The robot has a two fingered gripper as its end-effector and can carry the items with it. The items can be positioned on the locations. The goal is to position the items at their goal locations.

5.3.1 Action Domain Description

In this problem, we consider a single robot, denoted by the constant `z`. Items, locations and regions are described by positive integers up to `itemNo`, `locationNo` and `regionNo`, respectively. Each item `I`, as well as the robot, has a location that is specified by the fluents `locationOf(I)` and `locationOf(z)`. In addition, items are associated with regions, specified by the fluent `regionOf(I)`.

The robot has few actions: `move`, `pickup` and `putdown`. Each of the actions are associated with attributes that specify the details of the action.

Description of the domain in the language of CCALC is given below.

Direct effects of actions

Moving of the robot is described by the causal law

```
move causes locationOf(Z)=L if moveLocation=L.
```

Action of the robot picking up an item is given as follows

```
pickup causes holding(I) if pickupItem=I.
```

Finally, robot putting down the item it is holding is describes below

```
putdown causes -holding(I) if holding(I).
putdown causes regionOf(I)=R
    if holding(I) & putdownRegion=R.
```

Ramifications

If the robot picked up an item, then the item is no more associated with a region.

```
caused regionOf(I)=none if holding(I).
```

Robot holding an item implies that the item is moved at least once.

```
caused wasMoved(I) if holding(I).
```

The item that is being held by the robot has the same location as the robot.

```
caused locationOf(I)=L
    if holding(I) & locationOf(Z)=L.
```

Preconditions of actions

Robot cannot execute actions that conflict with each other.

```
nonexecutable move & pickup.
nonexecutable move & putdown.
nonexecutable pickup & putdown.
```

Moving of robot to the same location it is located at is forbidden.

```
nonexecutable move
    if moveLocation=L & locationOf(Z)=L.
```

Picking up an item is forbidden if the robot is at a different location.

```
nonexecutable pickup
    if pickupItem=I & locationOf(I)=L
        & locationOf(Z)=L1 & L\=L1.
```

Similarlar, picking up an item while holding another is forbidden.

```
caused false if holding(I) & holding(I1) & I\=I1.
```

Robot cannot execute put down action unless it is holding an item.

```
nonexecutable putdown if [/\I | -holding(I)].
```

Constraints

An item cannot be at a region that is not associated with the location it is currently at.

```
caused false if locationOf(I)=L & regionOf(I)=R
    where -regionOfLocation(L, R).
```

where `regionOfLocation(L, R)` is an external predicate/function that returns `true` if `R` is a region of `L`.

Integration

To guide the task planning at the representation level, following causal law is added to the formulation.

```
nonexecutable putdown
    if holding(I) & putdownRegion = R
        & regionOf(I1) = R & locationOf(I1) = L
        & emptyRegion(R1) & R\=R1
    where -fit(R, I, I1) & regionOfLocation(L, R1).
```

where `fit` is an external predicate/function that determines whether the region `R` is filled up when items `I` and `I1` are placed on. This causal law states that, the robot cannot put the item down to a region, if there is another item on that region large enough so that the two items fill it up, in the presence of other regions in the same location with no items.

5.3.2 Example Query

Figure 5.3.2 illustrates the sample problem. There are two tables defined as the locations and six items in the problem. Items are initially positioned on top of the tables as depicted in Figure 5.3.2. We are interested in finding a plan to move all of the items on top of the second table.

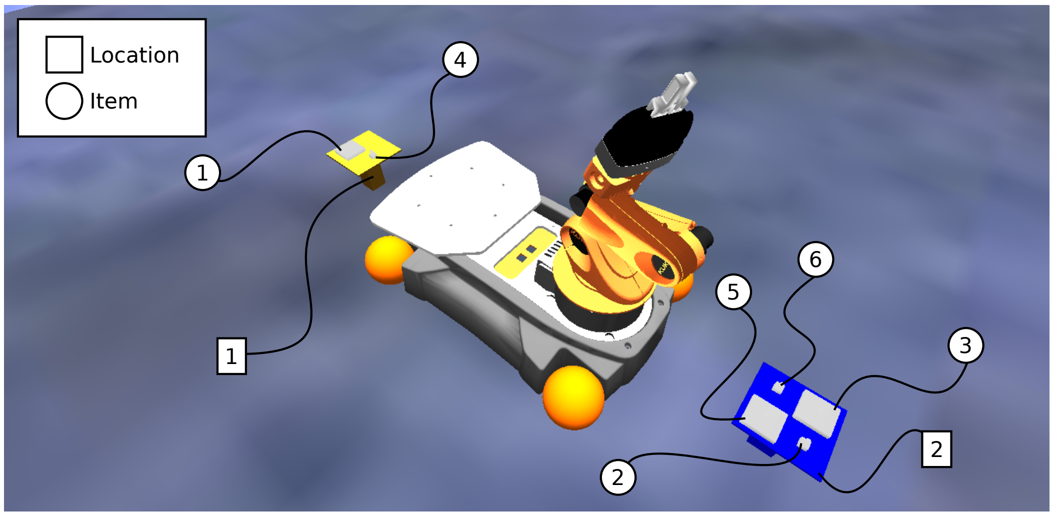

Figure 5.3.2: Query of Waiter Robot Problem

The locations are parceled off to regions according to the size of the larger items, which, in this case, are `1`, `3`, and `5`. After preprocessing, each location is divided into four regions. Initial state of the world is also generated in this process. The query reads as follows:

```
:- query
label :: 0;
maxstep :: 0..infinity;
0:  [/\I | -holding(I)], [/\I | -wasMoved(I)],
    locationOf(z)=2,
    locationOf(1)=1, regionOf(1)=3,
    locationOf(2)=2, regionOf(2)=8,
    locationOf(3)=2, regionOf(3)=6,
    locationOf(4)=1, regionOf(4)=2,
    locationOf(5)=2, regionOf(5)=7,
    locationOf(6)=2, regionOf(6)=5;
maxstep: [/\I | -holding(I)], clear(1).
```

where `clear(x)` is a macro that states that no item is located on `x`.

CCALC calculates the following task plan:

```
0: move, moveLocation=1
1: pickup, pickupItem=4
2: move, moveLocation=2
3: putdown, putdownRegion=5
4: pickup, pickupItem=6
5: putdown, putdownRegion=8
6: move, moveLocation=1
7: pickup, pickupItem=1
8: move, moveLocation=2
9: putdown, putdownRegion=5
```

5.3.3 Dynamic Simulation of the Waiter Robot Problem

The problem is implemented in Robot Operating System (ROS) and simulated in a dynamic simulation environment for robotic applications, Gazebo, which utilizes a rigid body simulator, Open Dynamics Engine. Velocity control of the youBot base and joint position control of the youBot arm are implemented in C++, while position control of the youBot base and all other algorithms are written in Python.

Figure 5.3.3 presents snapshots from the execution of a plan on the dynamic simulator Gazebo. Initially, the robot and the items are positioned as shown in Figure 5.3.3.a. First, the robot moves to a position in location `1`. Then, it picks up

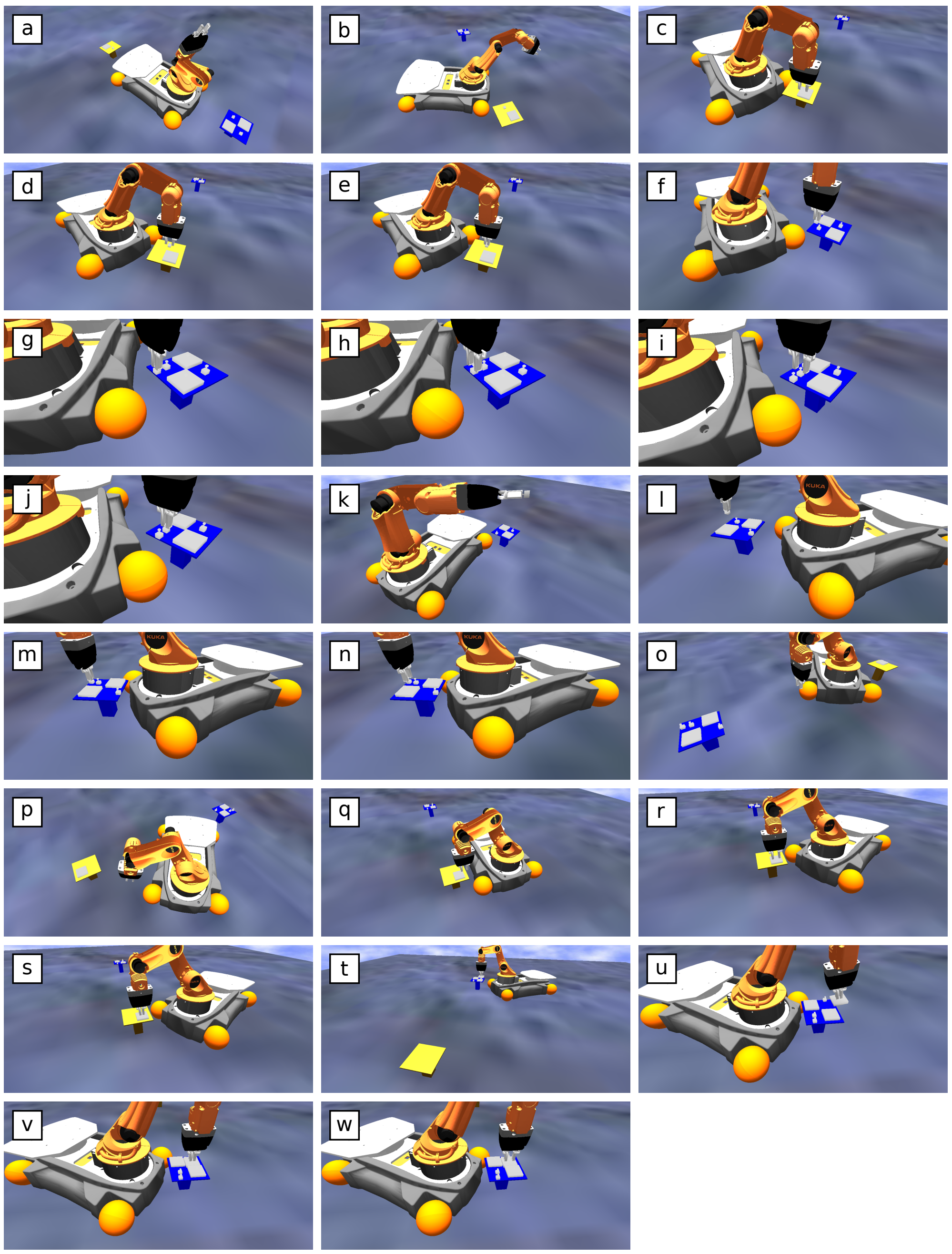

Figure 5.3.3: Snapshots from execution of a plan for Waiter Robot problem

Table 5.4.1: Problem size of Waiter Robot for the feasible query

Atoms	Clauses
1922	12874

item `4`. While holding the item, it moves to location `2`, and puts down item to region `5`. After positioning item `4`, no room enough for item `1` remains. Therefore, to make room, the robot picks up item `6`, and puts it down to region `8`. Then, it moves to location `1`, picks up item `1`, and moves to location `2`. Finally, the robot puts item `1` down to region `5` to conclude the execution.

5.4 Discussion

We have proposed a method for generalizing our framework to handling continuous domains. The first difference of the new approach is that position representation is abstracted from exact point representation into locations and regions. We argue that this is a more intuitive way of specifying tasks for a higher-level representation.

Although we consider a more complex domain in Section 5.3 compared to domains introduced in Chapter 4, Table 5.4.1 indicates that problem size in grounding level is considerably smaller; therefore, computation of a task plan requires less time.

In addition, to fulfil the need of exact positions for the execution, we introduced a method where exact positions are generated incrementally for each step through sampling regions for objects. Due to the presence of an external predicate/function that can reason about geometries, calculated task plans respect geometric constraint, instead of performing a blind search. Consequently, the performance of the position generation method is shown to be satisfactory.

Chapter 6

Conclusion

We have presented a novel approach to combine high-level representation and causality-based reasoning with low-level geometric reasoning and motion planning for robotic manipulation. Our hybrid planning framework extends the class of problems that can be solved by task planning and motion planning. In the framework, we exploit a tight integration between high-level causality-based reasoning, and geometric reasoning and path planning. Components of the framework guide one another at the representation and the search level, in order to obtain a physically correct plan.

We have showed the applicability of our framework in two example domains that were investigated for various levels of integration. In both of the problems, our planning framework finds a plan for problems where two robots are involved and their concurrent actions are required for successful completion of execution. We have presented instances from the execution of the plans.

We have presented a systematic analysis of the influence of the level of integration between high-level task planning and low-level geometric reasoning and path planning. External predicates allow externally defined procedures/functions to be embedded into the logical formalism. They can be utilized in task planning to capture physical constraints. However, they bring a computational burden on the task planner, especially at the grounding level. On the other hand, partially embedding physical constraints at the representation level implies that task planner may compute infeasible plans and replanning with the guidance of motion planning may be more frequently required. Therefore, there exists an inherent trade-off on the level of integration and computational efficiency/tractability. In our studies, we have concluded that having partial integration, by the means of external predicates/functions that require a low number of domain variables, is computationally more efficient.

Additionally, we have generalized our hybrid planning framework to consider continuous domains instead of domains with exact positions. In order to do so, we introduce the notions of location and region that abstract the position of an object in the representation level. However, exact positions of objects are essential for manipulation planning and execution, therefore they need to be extracted from the available region information. We introduce a method that samples the regions, and determines the positions of objects incrementally according to the task plan. Once the positions are obtained in the three-dimensional space, we utilize IKBiRRT path planning algorithm, one of whose features is to consider task space and configuration space in its planning mechanism, for generation of robot trajectories. We, then, illustrate the generalized hybrid planning framework in an example domain.

Feature works include addition of perception and learning components into the proposed hybrid planning framework. Additionally, further integration levels can be considered and systematic analysis can be extended to other domains.

Bibliography

[1] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic, Dordrecht, 1991.

[2] Steven M. Lavalle. Rapidly-exploring random trees: A new tool for path planning. Technical report, 1998.

[3] Dmitry Berenson, Siddhartha Srinivasa, David Ferguson, Alvaro Collet Romea, and James Kuffner. Manipulation planning with workspace goal regions. In *IEEE International Conference on Robotics and Automation (ICRA '09)*, May 2009.

[4] Enrico Giunchiglia, Joohyung Lee, Vladimir Lifschitz, Norman McCain, and Hudson Turner. Nonmonotonic causal theories. *Artificial Intelligence*, 153:49–104, 2004.

[5] Norman McCain and Hudson Turner. Causal theories of action and change. In *Proc. of AAAI/IAAI*, pages 460–465, 1997.

[6] Fangzhen Lin. Embracing causality in specifying the indirect effects of actions. In *Proc. of IJCAI*, pages 1985–1991, 1995.

[7] Michael Gelfond and Vladimir Lifschitz. Action languages. *Electronic Transactions on Artificial Intelligence*, 2:193–210, 1998.

[8] Henry Kautz and Bart Selman. Planning as satisfiability. In *Proc. of ECAI*, pages 359–363, 1992.

[9] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *Proc. of SAT*, pages 502–518, 2003.

[10] Youssef Hamadi, Saïd Jabbour, and Lakhdar Sais. Control-based clause sharing in parallel sat solving. In *Proc. of IJCAI*, pages 499–504, 2009.

[11] Hector Levesque and Gerhard Lakemeyer. Cognitive robotics. In *Handbook of Knowledge Representation*. Elsevier, 2007.

[12] Vladimir Lifschitz. What is answer set programming? In *Proc. of AAAI*, 2008.

[13] Vladimir Lifschitz and Hudson Turner. Representing transition systems by logic programs. In *Logic Programming and Non-monotonic Reasoning: Proc. Fifth Int'l Conf. (Lecture Notes in Artificial Intelligence 1730)*, pages 92–106, 1999.

[14] Vladimir Lifschitz. Action languages, answer sets and planning. In *The Logic Programming Paradigm: a 25-Year Perspective*, pages 357–373. Springer Verlag, 1999.

[15] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

[16] Lydia Kavraki, Petr Svestka, Jean-Claude Latombe, and Mark Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[17] R. Alami, J.P. Laumond, and T. Simeon. Two manipulation planning algorithms. In *Workshop on Algorithmic Foundations of Robotics*, pages 945–952, 1994.

[18] Y. Koga and J. C. Latombe. On multi arm manipulation planning. In *Proc. of ICRA*, pages 945–952, 1994.

[19] Fabien Gravot, Stephane Cambon, and Rachid Alami. *Robotics Research The Eleventh International Symposium*, volume 15 of *Springer Tracts in Advanced Robotics*, chapter aSyMov:A Planner That Deals with Intricate Symbolic and Geometric Problems, pages 100–110. Springer, 2005.

[20] Stephane Cambon, Rachid Alami, and Fabien Gravot. A hybrid approach to intricate motion, manipulation and task planning. 28(1):104–126, 2009.

[21] Kris Hauser and Jean-Claude Latombe. Integrating task and PRM motion planning: Dealing with many infeasible motion planning queries. In *Workshop on Bridging the Gap between Task and Motion Planning at ICAPS*, 2009.

[22] Erion Plaku and Gregory D. Hager. Sampling-based motion and symbolic action planning with geometric and differential constraints. In *Proc. of ICRA*, pages 5002–5008, 2010.

[23] Leslie Pack Kaelbling and Tomas Lozano-Perez. Hierarchical planning in the now. In *Proc. of ICRA Workshop on Mobile Manipulation*, 2010.

[24] Jason Wolfe, Bhaskara Marthi, and Stuart Russell. Combined task and motion planning for mobile manipulation. In *Proc. of ICAPS*, pages 254–258, 2010.

[25] Earl D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5(2):115–135, 1974.

[26] Ozan Caldiran, Kadir Haspalamutgil, Abdullah Ok, Can Palaz, Esra Erdem, and Volkan Patoglu. Bridging the gap between high-level reasoning and low-level control. In *Proc. of LPNMR*, 2009.

[27] Patrick Eyerich, Thomas Keller, and Bernhard Nebel. Combining action and motion planning via semantic attachments. In *Workshop on Combining Action and Motion Planning at ICAPS*, 2010.

[28] Fahiem Bacchus and Froduald Kabanza. Using temporal logic to express search control knowledge for planning. *Artificial Intelligence*, 116(1–2):123–191, 2000.

[29] Maria Fox and Derek Long. Pddl2.1: An extension to pddl for expressing temporal planning domains. *Journal Artificial Intelligence Research*, 20:61–124, 2003.

[30] Jörg Hoffmann and Bernhard Nebel. The ff planning system: Fast plan generation through heuristic search. *Journal Artificial Intelligence Research*, 14:253–302, 2001.