mαcrϕ@ufmg

Universidade Federal de Minas Gerais
Programa de Pós-Graduação em Engenharia Elétrica
MACRO Research Group - Mecatronics, Control and Robotics

MANIPULATION TASK PLANNING AND MOTION CONTROL USING TASK RELAXATIONS

Marcos da Silva Pereira

Belo Horizonte, Brazil
2020

Marcos da Silva Pereira

MANIPULATION TASK PLANNING AND MOTION CONTROL USING TASK RELAXATIONS

Dissertation submitted to the Programa de Pós-Graduação em Engenharia Elétrica of Escola de Engenharia at the Universidade Federal de Minas Gerais, in partial fulfillment of the requirements for the degree of Master in Electrical Engineering.

Advisor: Bruno Vilhena Adorno

Belo Horizonte, Brazil
2020

To my parents Dácia and José Poli.

Acknowledgements

My first contact with robotics of manipulators was during my year abroad at the Technical University of Dresden (TU Dresden) in Germany with grantee from CAPES and the Science without Borders Program. Among other courses, I attended to a course about fundamentals of robot manipulators. I must thank CAPES for the financial support. It has significantly changed the course of my studies. After one year at TU Dresden, I did an internship at Bosch Rexroth in Germany where I had my first strong experience with software development to which I am grateful. It has certainly contributed to my technical development in other fields. Next, I returned to the University of Brasília (UnB) to finish my undergrad studies.

Back at UnB, I decided to continue in the field of robot manipulators. I had some colleagues that worked at the robotics and automation laboratory (LARA). They introduced me to Prof. Mariana Bernardes and Dr. Luis Figueredo (at the time, pursuing his PhD). Prof. Mariana accepted to guide me in the field of robot manipulators and Dr. Luis helped me a lot with his applied and theoretical background. Both introduced me to the field of robotics using dual quaternions (DQs) through the works of Prof. Bruno Adorno. I got very curious about DQs and decided someday I would like to do research in that field. Hence, I must thank Prof. Mariana and Dr. Luis for their guidance during my last undergrad semesters.

Before finishing my studies, I did my mandatory internship again at Bosch Rexroth in Germany, during which I had my first contact with programmable logic controllers. Once again I am thankful to Bosch Rexroth for the opportunity. After concluding my undergrad studies, I spent some more time at Rexroth, but decided to come back to Brazil. In Brazil I decided to do a Master and applied at the Federal University of Minas Gerais (UFMG). After being accepted, I talked with Prof. Mariana and Dr. Luis and both recommended me to Prof. Bruno.

As soon as I became a student from Prof. Bruno, I was astonished by how professional and organized he was. In less than a week, he gave me all the information needed for me to feel welcomed at UFMG and introduced me to the members of the MACRO research group. I received a warm welcome and was well explained how everything worked. Here I must thank Rafael, Mariana, Juan (Juancho), Stella, and Frederico. All of them helped

me both with their friendship and with technical advice. Later on, Ana Christina joined the group and became a great work colleague. We were both starting our master studies. We shared valuable technical ideas through absolutely insightful discussions to solve a lot of different issues and bugs. Here I must also thank Gabriel Pacheco, who also started his master at the same time, with whom I also debated about a variety of curious problems. Thales Silva also joined our discussions and contributed with interesting ideas. Because of all of these colleagues, I have had a joyful and enjoyable master's period.

The knowledge curve required during a master escalates quickly. I am very grateful to Prof. Bruno who provided me all the knowledge background needed through lectures, books, papers, and the amazing DQ Robotics library. Here I must also acknowledge Dr. Murilo Marinho for his help with fixing issues of the DQ Robotics. Additionally to providing technical and theoretical tools, Prof. Bruno and I had productive meetings to discuss about my master's project during which interesting ideas emerged. Moreover, our weekly group meetings with lots of questions and answer sections deeply contributed to my learning process. Lastly, all the discussions and interactions with the students and professors of the MACRO group helped me to shape and build this master's thesis. Concerning financial support, I must thank CAPES for the scholarship. However, life is not only doing research, working, and writing papers, people also make friends and have other activities. Besides all my technical hobbies, I like running.

After joining a running group in Belo Horizonte (BH), I met a lot of funny and interesting local people. I am thankful to all the friends I made there. Together, we trained weekly, participated of the BH Half Marathon and of the International Pampulha Race, but most importantly we had our monthly *resenhas* (pub meetings) and our parties. I have to mention Jaqueline, John, Mariana, and Ivan. They have made my weeks and weekends in BH lighter and full of laughs.

Along with new friends, I also carry friendships and family ties who have backed me all the way until now. I must specially thank my life long friend Tiago, my sisters Joana and Cecília, my nephew Jonas, my newborn nephew Martin, my closest cousins Pedro, Mateus, Tobias, and Mariana, and all the rest of my family. They make my life warmer and brighter.

Finally, I am most deeply thankful to my parents Dacia and José Poli who have been inspiring, encouraging and supporting me along all my life.

Agradecimentos

Meu primeiro contato com robótica de manipuladores foi durante meu intercâmbio na Univeridade Técnica de Dresden na Alemanha (TU Dresden) com bolsa da CAPES através do programa Ciência Sem Fronteiras. Dentre outras disciplinas, eu fiz uma disciplina sobre fundamentos de robótica de manipuladores. Eu agradeço a CAPES pelo apoio financeiro. Essa oportunidade alterou significativamente o curso dos meus estudos. Após um ano na TU Dresden, eu estagiei na Bosch Rexroth na Alemanha e tive meu primeiro contato forte com desenvolvimento de software. Sou grato a eles por isso. Certamente contribuiu para o meu aprendizado técnico em outras áreas. Após o estágio, eu retornei a Universidade de Brasília (UnB) para finalizar minha graduação.

De volta na UnB, eu decidi continuar na área de robótica de manipuladores. Eu tinha alguns colegas que trabalhavam no Laboratório de Automação e Robótica (LARA). Eles me apresentaram à Profa. Mariana Bernardes e ao Dr. Luis Figueredo (na época fazendo seu doutorado). A Profa. Mariana aceitou me orientar na área de robótica de manipuladores e o Dr. Luis me ajudou bastante com sua forte base teórica e aplicada. Ambos me introduziram à robótica utilizando quatérnios duais (QD) através dos trabalhos do Prof. Bruno Adorno. Eu fiquei bastante curioso sobre QDs e decidi que algum dia gostaria de fazer pesquisa na área. Portanto, eu preciso agradecer a Profa. Mariana e ao Dr. Luis pela orientação durante meus últimos semestres da graduação.

Antes de finalizar minha graduação, eu fiz meu estágio obrigatório novamente na Bosch Rexroth na Alemanha onde eu tive meu primeiro contato com controladores lógico programáveis. Novamente, eu agradeço a Bosch Rexroth pela oportunidade. Após me formar, eu passei mais algum tempo na Rexroth, mas decidi retornar ao Brasil. Decidi fazer o mestrado e apliquei na Universidade Federal de Minas Gerais (UFMG). Depois de ser aceito, conversei com a Profa. Mariana e com o Dr. Luis e ambos me recomendaram para o Prof. Bruno.

Assim que conheci o Prof. Bruno e iniciei as atividades com ele, eu achei incrível o quão profissional e organizado ele é. Em menos de uma semana ele me explicou tudo que era necessário para eu me sentir em casa na UFMG e me apresentou aos membros do grupo de pesquisa MACRO. Fui muito bem recebido e rapidamente me explicaram como o grupo funciona. Agradeço ao Rafael, à Mariana, ao Juan (Juancho), à Stella

e ao Frederico. Todos me ajudaram tanto com amizade quanto com questões técnicas. Algum tempo depois, a Ana Christina, que começou o mestrado junto comigo, escolheu a orientação do Prof. Bruno e se tornou uma grande colega de trabalho. Compartilhamos diversas ideias técnicas úteis através de discussões muito interessantes para resolver um monte de problemas e erros. Também preciso agradecer ao Gabriel Pacheco, que também iniciou o mestrado na mesma época. Também discutimos uma variedade de problemas curiosos. O Thales Silva também costumava participar das discussões e contribuir com ideias interessantes. Todos esses colegas tornaram o período do meu mestrado divertido e proveitoso.

A curva de conhecimento necessário durante o mestrado cresce rapidamente. Sou muito grato ao Prof. Bruno por me fornecer toda base de conhecimento necessária através de aulas, livros, artigos e da incrível biblioteca DQ Robotics. Aqui eu preciso agradecer também ao Dr. Murilo Marinho pela ajuda com a solução de problemas com a DQ Robotics. Além de fornecer as ferramentas teóricas e técnicas, eu e o Prof. Bruno tivemos reuniões bem produtivas sobre o meu projeto de mestrado durante as quais interessantes ideias surgiram. Além disso, nossos encontros semanais do grupo de pesquisa com muitas seções de perguntas e respostas contribuiram fortemente para o meu aprendizado. Por fim, todas as discussões e interações que tive com os estudantes e professores do MACRO me ajudaram a dar forma e construir essa dissertação. Em relação a suporte financeiro, eu preciso agradecer a CAPES pela bolsa de mestrado durante boa parte do período. No entanto, a vida não é apenas fazer pesquisa, trabalhar e escrever artigos, nós também fazemos amizades e temos outras atividades. Além dos meus hobbies técnicos, eu gosto de corrida.

Eu conheci várias pessoas interessantes e divertidas de Belo Horizonte (BH) depois de entrar para um grupo de corrida local. Eu sou grato a todas amizades que eu fiz lá. Nós treinamos juntos, participamos da Meia Maratona de BH e da Volta Internacional da Pampulha, mas mais importante participava das resenhas mensais e das festas do grupo. Eu preciso agradecer com carinho a Jaqueline, o John, a Mariana e o Ivan. Eles tornaram minhas semanas e finais de semana mais leves e cheios de gargalhadas.

Além dos novos amigos, eu carrego minhas amizades e familiares que tem me apoiado durante toda minha jornada até agora. Eu preciso agradecer especialmente ao meu amigo de longa data Tiago, às minhas irmãs Joana e Cecília, o meu sobrinho Jonas, meu mais novo sobrinho Martin, os meus primos mais próximos Pedro, Mateus, Tobias e Mariana e todo o resto da minha família. Eles tornam minha vida mais viva e clara.

Finalmente, sou profundamente grato aos meus pais Dácia e José Poli que vem me inspirando, encorajando e me dando suporte ao longo de toda minha vida.

Resumo

Esta dissertação de mestrado trata da integração de planejamento de tarefas e controle de movimento em robótica de manipulação para tarefas que podem ser relaxadas. O objetivo é gerar automaticamente sequências factíveis de manipulação para serem executadas por um controlador que considera restrições geométricas impostas pela tarefa. Para lidar com a alta dimensionalidade do problema de manipulação e a complexidade de especificar tarefas, foi usado um arcabouço multicamadas para planejamento de tarefa e movimento adaptado da literatura. O arcabouço adaptado consiste de um planejador de alto nível, que gera planos de tarefa para especificações em lógica temporal linear, e um controlador de movimento de baixo nível baseado em otimização com restrições, que permite definir regiões de interesse ao invés de localidades exatas e é reativo a mudanças no espaço de trabalho. Logo, não há adição de tempo de planejamento de movimento ao tempo total de planejamento. Além disso, como não há fase de replanejamento devido a falhas em um planejador de movimento, as ações para o robô são geradas apenas uma vez para cada tarefa, portanto, a busca por plano de tarefas ocorre em um grafo estático. Com relação ao relaxamento de tarefas, a ação do plano de tarefas de segurar um objeto até uma região alvo é relaxada fazendo-se controle de distância ao plano alvo ao invés de controle de pose. Desse modo, ao invés de utilizar-se seis graus de liberdade para controlar a pose, apenas um grau de liberdade é utilizado para controlar a distância ao plano. Para manter o efetuador dentro de uma região de interesse e fora de uma região proibida enquanto ele se desloca para o plano, restrições são adicionadas. Com as restrições, o efetuador se move para a região alvo no plano, que é delimitada pela combinação de primitivas geométricas. Utilizando-se restrições por planos, foi definida uma região alvo quadrada que resulta em uma região de interesse na forma de um tronco de pirâmide invertido. Além disso, foi proposta uma nova interpretação de restrição cônica chamada restrição ponto-cone que permite definir regiões alvos circulares e que resulta em um tronco de cone invertido. Essa abordagem foi testada com tarefas de *pick-and-place* com complexidade similar à das tarefas realizadas no arcabouço original. O número de nós de planejamento gerados foi reduzido, o que implica em menor tempo total de planejamento. Por fim, foi mostrado que o efetuador permanece dentro da região de interesse e se move em direção ao plano alvo tanto no caso da região quadrada quanto no caso da região circular.

Abstract

This master thesis addresses the integration of task planning and motion control in robotic manipulation for tasks that can be relaxed and the generation of feasible manipulation sequences that are executed by a controller that explicitly accounts for the task geometric constraints. To cope with the high dimensionality of the manipulation problem and the complexity of specifying the tasks, we use a multi-layered framework for task and motion planning adapted from the literature. The adapted framework consists of a high-level planner, which generates task plans for linear temporal logic specifications, and a low-level motion controller, based on constrained optimization, that allows defining regions of interest instead of exact locations while being reactive to changes in the workspace. Thus, there is no low-level motion planning time added to the total planning time. Moreover, since there is no replanning phase due to motion planner failures, the robot actions are generated only once for each task because the search for a plan occurs on a static graph. Concerning task relaxation, the task plan action of holding an object toward a target region is relaxed by controlling the end-effector distance to a target plane instead of requiring pose control. This way, instead of requiring six degrees of freedom to control the pose, only one degree of freedom is used to control the distance to a plane. We add constraints that keep the end-effector inside a region of interest and outside a restricted region while it moves toward the plane. Thus, it moves toward the target region that is constrained by the combination of geometric primitives. By using plane constraints, we define a rectangular target region that results in an inverted pyramid trunk region of interest. Additionally, we propose a new interpretation of conic constraints called point-cone constraint that allows defining circular target regions resulting in an inverted cone trunk. We evaluated the adapted framework with pick-and-place tasks with similar complexity to the original framework and showed that the number of plan nodes generated is smaller than the one in the original framework, which implies less total planning time. Lastly, it is shown that the end-effector remains within the regions of interest and moves toward the target region for both the rectangular and circular target regions.

Contents

List of Figures

List of Tables

Acronyms

ITMP	Integration of task and motion planning.
LTL	Linear temporal logic.
STL	Signal temporal logic.
DFA	Deterministic finite automaton.
AI	Artificial intelligence.
STRIPS	Stanford Research Institute Problem Solver.
ADL	Action Description Language.
PDDL	Planning Domain Description Language.
MTM	Manipulation Task Model.
BDD	Binary decision diagram.
SMAP	Sampling based motion and symbolic action planner.
VFI	Vector field inequality.
DOF	Degrees of freedom.
OMPL	Open Motion Planning Library
ROS	Robot Operating System.

Notation

$\mathbb{A}$	Set of atomic propositions.
$a_0, a_1, \ldots$	Atomic propositions from the set $\mathbb{A}$.
$2^{\mathbb{A}}$	Alphabet given by the power set of $\mathbb{A}$.
$A_0, A_1, \ldots$	Letters formed by atomic propositions.
σ	Words formed by letters.
φ	Linear temporal logic (LTL) formula.
$\mathcal{U}$	LTL operator "until".
$\mathcal{X}$	LTL operator "next".
$\mathcal{E}$	LTL operator "eventually".
$\models$	Satisfying relation $\sigma \models \varphi$: a word σ satisfies a formula φ.
$\not\models$	Does not satisfy relation.
O	Set of objects.
$o_0, o_1, \ldots$	Objects from the set O.
Γ	Set of labels.
$l_0, l_1, \ldots$	Labels from the set Γ.
$p_0, p_1, \ldots$	Atomic propositions of the form $(o_i, l_j) \in O \times \Gamma$, which means that "object o_i is in location with the label l_j".
$\mathcal{L}$	Set of locations.
$\mathrm{loc}_0, \mathrm{loc}_1, \ldots$	Locations of interest.
$\mathscr{L}$	Location graph.

$O_{\mathcal{L}}$	Set of objects locations.
$\bar{o}_{i,\text{loc}}$	Location of te ith object.
$\mathfrak{L}$	Function that maps locations to labels.
$\mathcal{R}$	Abstraction graph.
$\mathcal{M}$	Set of motion primitives.
$\mathscr{M}$	Motion graph.
V	Set of abstraction $\mathcal{R}$ nodes.
v	Node of the set V.
α	Motion primitive action $\alpha \in \mathcal{M}$.
$\mathfrak{o}_{\mathcal{L}}$	Tuple $\mathfrak{o}_{\mathcal{L}} \in O_{\mathcal{L}}$ that indicates the location of each object.
$\mathcal{A}_{\varphi}$	Deterministic finite automaton (DFA) obtained from the LTL formula φ.
Z	Finite set of states from the DFA.
z	State from the set Z.
Σ	Set of events that causes transitions in the DFA.
δ	Transition function from the DFA.
F	Set of final states from the DFA.
$\mathcal{P}$	Product graph.
$V_{\mathcal{P}}$	Product graph nodes.
$\mathfrak{p}$	Node from the product graph.
$\mathit{\Pi}$	Generates the letters that causes the transition in the automaton.
L	Labeling function.
$\mathfrak{P}$	Function that determines if an object label in a proposition matches the label of the corresponding object in the tuple $\mathfrak{o}_{\mathcal{L}}$.
T, F	True and false values.
$\mathfrak{F}$	Function that returns the object location in $\mathfrak{o}_{\mathcal{L}}$ corresponding the proposition object.

$\varnothing$	Represents the empty set.
$\boldsymbol{q}$	Manipulator joints vector.
$\boldsymbol{x}$	Task vector.
$\tilde{\boldsymbol{x}}$	Task error.
$\boldsymbol{u}$	Control input vector.
λ	Controller damping factor.
η	Controller gain.
$\boldsymbol{J}$	Task Jacobian.
η_x	Constraint gain of geometric primitive x.
$d_{\#1,\#2}$	Distance between geometric entities #1 and #2.
$\tilde{d}$	Signed distance between geometric entities.
$d_{x,\text{safe}}$	Safe distance to geometric entity x.
$\mathfrak{W}(x)$	Constraints with regard to geometric entity x.
$\underline{\boldsymbol{\pi}}$	Plane.
c	Infinite cylinder c.
$\underline{\boldsymbol{l}}$	Line.
$D_{\#1,\#2}$	Squared distance between geometric entities #1 and #2.
$\tilde{D}$	Signed squared distance between geometric entities.
$l_{\text{cone}}^{\text{point}}$	Point-cone constraint center line.
$l_{z,\text{cone}}$	Line-cone constraint line passing through the end-effector z-axis.
P_n^k	The k-permutation from n elements.
$\mathbb{H}$	Set of quaternions.
$\hat{\imath}, \hat{\jmath}, \hat{k}$	Quaternion units.
$\boldsymbol{h}, \boldsymbol{x}, \boldsymbol{y}$	Quaternions.
$\boldsymbol{h}^*$	Quaternion conjugate.

$\mathbb{H}_p$	Set of pure quaternions.
$\mathbb{S}^3$	Set of unit quaternions with unit norm.
$\langle \boldsymbol{a}, \boldsymbol{b} \rangle$	Inner product between pure quaternions $\boldsymbol{a}$ and $\boldsymbol{b}$.
$\boldsymbol{a} \times \boldsymbol{b}$	Cross product between pure quaternions $\boldsymbol{a}$ and $\boldsymbol{b}$.
$\boldsymbol{r}$	Unit quaternion representing rotation.
$\boldsymbol{n}$	Pure quaternion representing the rotation axis.
$\mathcal{H}$	Set of dual quaternions.
ε	Dual unit.
$\underline{\boldsymbol{h}}, \underline{\boldsymbol{x}}, \underline{\boldsymbol{y}}$	Dual quaternions.
$\underline{\boldsymbol{h}}^*$	Dual quaternion conjugate.
$\mathcal{H}_p$	Set of pure dual quaternions.
$\langle \underline{\boldsymbol{a}}, \underline{\boldsymbol{b}} \rangle$	Inner product between pure dual quaternions $\underline{\boldsymbol{a}}$ and $\underline{\boldsymbol{b}}$.
$\underline{\boldsymbol{a}} \times \underline{\boldsymbol{b}}$	Cross product between pure dual quaternions $\underline{\boldsymbol{a}}$ and $\underline{\boldsymbol{b}}$.
$\underline{\boldsymbol{\mathcal{S}}}$	Set of unit dual quaternions.
vec_4	Operator that maps quaternion to a four-dimensional column real vector.
vec_8	Operator that maps dual quaternion to an eight-dimensional column real vector.
$\mathcal{I}$	Region of interest in $\mathbb{H}_p$.
P_π	Set of all points on a plane.
$\mathcal{A}_\pi, \mathcal{B}_\pi$	Half-spaces above and below a plane $\underline{\boldsymbol{\pi}}$.
$c(\underline{\boldsymbol{l}}, R)$	Infinite cylinder c with centerline $\underline{\boldsymbol{l}}$ and radius R.
$\zeta_{\#1,\#2}$	Line segment between points #1 and #2.
$C(\boldsymbol{p}_\pi, R)$	Circle with radius R and centerpoint $\boldsymbol{p}$ on plane π.
$\mathfrak{C}_{\boldsymbol{p}_A}^{C(\boldsymbol{p}_\pi, R)}$	Cone with appex $\boldsymbol{p}_A$ and base circle $C(\boldsymbol{p}_\pi, R)$.

$\mathcal{I}_{\mathcal{Q}}$	Inverted pyramid trunk region of interest.
$\mathcal{I}_{\mathfrak{C}}$	Inverted cone trunk region of interest.

1 Introduction

With the development of robots, humans became even more capable of making machines autonomously transform the world. Robots were placed in factories to do the simple repetitive task of transferring materials between locations (W. Spong et al., 2005). Robots had to be manually programmed for every new task. Moreover, robots would only achieve the task in a highly structured environment. As sensing capabilities were added, they were programmed to weld, paint, assemble, etc. These robots were coined as robot manipulators, which are robotic arms equipped with an end-effector to modify the environment. Their success in the industry grew the attention of other fields. They were adapted to execute domestic tasks (Cha et al., 2015), to explore space and underwater environments (Laryssa et al., 2002; Khatib et al., 2016), to provide physical assistance to medical surgeons (Hager et al., 2008), and also to do nuclear material manipulation similar to the old teleoperation devices (Kuban and Martin, 1984). Although manipulators were modified to work in a wide range of environments, there is still a need to specify and generate task plans for them to operate autonomously. This is called manipulation task planning.

The planning of manipulation tasks can be applied in different environment. For instance, the mobile manipulators that will work in restaurants in the future must be capable of serving clients in a given order (He et al., 2015). Mobile manipulators that will work washing dishes or other kind of objects must also follow a specific order of manipulation (Kaelbling and Lozano-Perez, 2011). Task planning also appears in the field of mobile robots. Mobile robots working in hospitals monitoring patients need to visit them in the correct sequence (Kress-Gazit et al., 2007). Additionally, mobile robots

working in search and rescue missions must coordinate their actions to find injured people (Kress-Gazit et al., 2007). Mobile robots operating under energy and battery constraints also need to consider when they must stop at a recharge station, but still be capable of achieving the task (Kundu and Saha, 2019). Among the vast applications of task planning in robotics, we focus on an assistive robotics application scenario.

This work focuses on the application scenario of a manipulator that must execute tasks for a seated person with limited or no lower limbs mobility. There are arm support systems (passive or active) that can help people with weak arm and hand functions. However, people with motor disabilities face difficulties to manipulate objects in a large environment due to their reachability limitations (Iskandar et al., 2019). In this context, manipulators appear as suitable helpers to do pick-and-place tasks for people with motor disabilities. This way, people regain part of their independence back since the manipulator can reach farther objects in the environment.

We have created a simulation scene with the Jaco robot from Kinova Robotics, cuboid objects, and regions where the objects must be placed. The robot can assist the person by manipulating objects between regions of interest in a given sequence specified by the person. For instance, while the person reads a book, the robot may heat meat and cool salad and, afterward, serve both to the person. See Figure 1.1. To solve this kind of task autonomously, we make use of manipulation task planning.

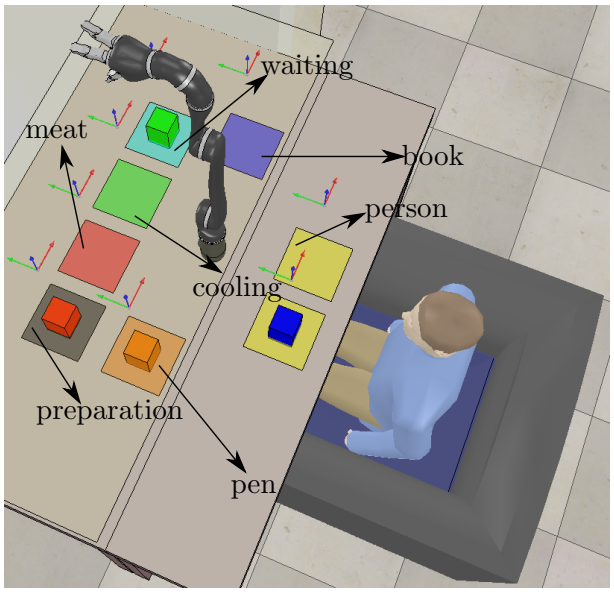


Figure 1.1: Assistive robotics scenario where the manipulator must pick and place objects for a person with motor disabilities. The *red*, *green*, *blue*, and *orange* objects represent meat, salad, book, and pen, respectively.

The goal of manipulation task planning is to automatically generate feasible movement sequences to manipulate objects to place them in their correct locations in the workspace (Erdem et al., 2011). To accomplish this, the manipulation order must be regarded (Erdem

et al., 2011). Therefore, the geometrical description of the problem and motion planning alone are not enough to solve the problem. As a result, there is a need to do integration of task and motion planning (ITMP) (Erdem et al., 2011).

Motion planning and task planning have complementary advantages. Task planners decide what the robot must do to achieve a task, but they do not consider the detailed arrangement of the environment (Kaelbling and Lozano-Perez, 2011). More specifically, they generate a sequence of actions to satisfy a given specification. Meanwhile, a motion planner must have a map of the environment to generate a trajectory to manipulate the objects based on the task plan (Kaelbling and Lozano-Perez, 2011). Lastly, the trajectories are sent to a motion controller that executes them. The question that remains is how to solve each of these steps and integrate them (see Figure 1.2).

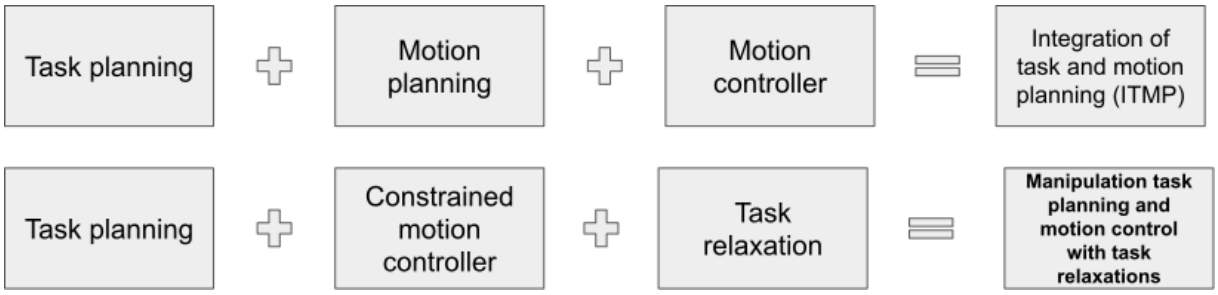


Figure 1.2: Necessary steps to do ITMP (top row) and manipulation task planning and motion control with task relaxations (bottom row). In this work we have adopted the latter.

The task planner requires a way to specify the tasks. A commonly used option in robotics is linear temporal logic (LTL) (Baier and Katoen, 2008, ch. 5) since it allows describing rich specifications with high expressivity by associating actions to locations and applying temporal operators to define the order of execution. Next, model-checking techniques solve the task planning problem by reasoning over an abstract model of the robot and formal specifications (Bhatia et al., 2011). LTL has been frequently used in mobile robotics (Kloetzer and Belta, 2008; Kress-Gazit et al., 2007; Bhatia et al., 2011; McMahon and Plaku, 2014; Kundu and Saha, 2019; Kloetzer and Mahulea, 2015). However, the techniques for mobile robotics depend on the discrete representation of the state-space, which makes them intractable to manipulation planning due to the large dimensionality of the manipulation problem. To solve this problem, He et al. (2015) propose a framework that allows the planning of high-level manipulation tasks specified in LTL and also handles the ITMP problem.

The ITMP problem is hard because the high-level task descriptions ignore the geometrical preconditions to physically realize the task (Srivastava et al., 2014). Even simple high-level tasks such as pick and place of objects have continuous parameters, geometrical preconditions, and effects. As a result, the method of generating a sequence of tasks with the task planner and executing the motion planner for each task may fail. This

happens because there is a gap between the task planner representation and the physical environment (Lozano-Perez et al., 1987). One way to solve this is to use an interface layer between the high-level and low-level that allows the task planner to plan in an abstract state-space. In this state-space, the geometry of the problem is ignored (Srivastava et al., 2014). Since the generated task plan may not be realizable by the motion planner (Latombe and Jean-Claude, 1991), the geometrical constraints are detected by the low-level planner in a continuous state-space and are sent back to the task planner through the interface layer (Srivastava et al., 2014). In this sense, ITMP works started using multi-layered frameworks that use a task planner to generate a motion sequence for the motion planner, which checks the viability of executing the task (Erdem et al., 2011; Kaelbling and Lozano-Perez, 2011; Srivastava et al., 2014; Lozano-Perez and Kaelbling, 2014; He et al., 2015).

In addition to using LTL, He et al. (2015) also propose a multi-layered framework that enhances the state of the art of ITMP by allowing more expressivity, making it possible to specify temporal constraints at the task level reducing the complexity of the low-level planning. The goal of the framework is to determine the tasks of pre-grasping and placing of objects between locations of interest in order to realize a task specified in LTL. The framework has a coordinating layer between the task planner and the motion planner that assigns weights to plans proportionally to the difficulty of executing a task plan based on the motion planning result. In this sense, there is a replanning phase every time the motion planning fails. Furthermore, a new task plan must be found. Hence, a considerable amount of time is spent on task and motion planning.

From the task point of view, the framework proposed by He et al. (2015) only allows placing the object in exact locations of interest in the workspace. Nonetheless, some tasks may benefit from using a region of interest instead of an exact location, that is, the task may be relaxed (see Figure (1.3)). A common solution is to discretize the workspace in multiple locations to define different possibilities for placing the manipulated objects. However, this approach increases the number of locations in the planning process. As a result, the number of possibilities that the planner must consider quickly grows and the total planning time increases. In contrast to the discretization approach, , the constrained motion controller (Marinho et al., 2019) solves the motion problem and allows to keep the robot outside a restricted region or inside a safe region by using constraints.

(a) Manipulator grasping items: the boxes area can be represented by regions of interest. (Source: RightHand Robotics)

(b) Manipulator placing object within a region of interest. (Source: RightHand Robotics)

(c) Manipulator moving items: the placing task can be relaxed to allow the item to be placed anywhere within the drawer region. (Source: Nvidia)

Figure 1.3: Pick-and-place robotic environments that can benefit from task relaxation to regions of interest.

By using suitable constraints in the constrained motion controller (Marinho et al., 2019), we define regions of interest, which greatly reduces the computational burden of the overall task relaxation method, with the additional advantages that there is no need of motion planning and the system is reactive. Considering the aforementioned techniques, the main focus of this work lies on solving the ITMP problem based on the manipulation task planning framework of He et al. (2015), in which the tasks are specified in LTL, and on the constrained motion controller of Marinho et al. (2019) that makes the system reactive and allows to relax tasks without incurring in high computational complexity. Figure 1.2 shows our approach.

1.1 Objective and Contributions

The goal of this work is to solve the ITMP problem for manipulation tasks combining the manipulation task planning framework proposed by He et al. (2015) and the constrained motion controller of Marinho et al. (2019). As a result, it is possible to plan and relax manipulation tasks in a computationally efficient manner. In this sense, the specific objectives are:

- Implement the planning framework of He et al. (2015) without the need of the interface and the motion planning layers;
- Adapt the planning framework to use the constrained motion controller of Marinho et al. (2019) instead of the motion planning layer.

The first step of implementing the planning framework proposed by He et al. (2015) consists in specifying a task using LTL. Then, the LTL task specification is converted into a deterministic finite automaton (DFA) that specifies all the ways the robot can execute

the task. Also, an abstraction, which captures all the ways the robot can manipulate objects, is defined and then combined to the DFA into a product graph that represents how the robot can move objects to achieve the specified task. Next, the Dijkstra's algorithm is used to search for a path on the graph from an initial node to an accepting one. Finally, the high-level plan is executed by the constrained motion controller of Marinho et al. (2019), which is based on an optimization problem that minimizes the joint velocities in the ℓ_2-norm sense while respecting hard constraints, such as obstacles in the workspace, joint limits, etc. On the one hand, the system is reactive to changes in the environment. On the other hand, the system may suffer from local minima.

While the ITMP problem has been addressed in the framework of He et al. (2015) and the constrained motion controller of Marinho et al. (2019) solves the execution of task plans allowing task relaxation, task relaxation requires the definition of regions of interest.

In this context, the main contributions of this work are:

- The ITMP problem is solved in a computationally efficient manner;
- The ITMP framework is adapted to efficiently accommodate relaxed tasks;
- To reduce the total planning time of the framework, it is proposed the use of a constrained motion controller (Marinho et al., 2019) that allows doing obstacle avoidance and to specify joint limits. Hence, it is possible to directly execute the actions from the task plan satisfying the task specification;
- Well-known constraints are combined to define regions of interest for relaxed tasks;
- In addition to well-known constraints, a new approach to define conic constraints is presented. As a result, it becomes possible to define a circular region of interest in a simpler way that requires fewer calculations during the control loop.

1.2 Structure of the Text

In addtion to this introduction chapter, this thesis organization is outlined as follows:

Chapter 2 presents a review of the state of the art regarding task planning, ITMP, and task relaxation. Next, in Chapter 3, the framework of He et al. (2015) is explained. In Chapter 4, the constrained motion controller of Marinho et al. (2019) and the constraints used in this work are presented. It alsoincludes the new approach to the point-cone constraint. Afterward, in Chapter 5, the task relaxation and the definition of the regions of interest are presented together with the advantages of the approach to the overall planning framework. In Chapter 6, the adapted ITMP framework for relaxed tasks is evaluated in a simulation environment on a Jaco robot from Kinova Robotics that has to perform tasks for a seated person with limited or no lower limbs mobility. Lastly, Chapter 7 summarizes the results obtained and presents future works.

2

State of the Art

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

Motion planning and task planning are two problems in robotics studied from different points of view. Task planners generate plans by processing a high number of action states, that is, states that describe what the robot must do regardless of where objects are located in the environment (Kaelbling and Lozano-Perez, 2011). This means that they can decide to cross a room without taking into account the objects in the middle of the way (Kaelbling and Lozano-Perez, 2011). On the other hand, a motion planner treats the geometry of the problem, that is, it must know the location of each object to plan how to manipulate them (Kaelbling and Lozano-Perez, 2011). It generates trajectories to move the robot from a starting point to a goal region while avoiding obstacles (Choset et al., 2005). For instance, in the case of manipulators, it can specify how to get a cake tray out of an oven but it does not predict if the oven door is open or closed.

Hierarchically speaking, bottom-up motion planning techniques focus on gathering information from the environment through sensors and creating control references for

closed-loop controllers by using detailed robot models to move the robot between different configurations (Kress-Gazit et al., 2007). On the other hand, top-down task planning techniques usually focus on finding a discrete action plan for the robot to execute a specified task (Kress-Gazit et al., 2007). Neither one of the approaches alone is sufficient to accomplish a manipulation task. Therefore, there is a need of ITMP to solve complex manipulation tasks.

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

2.1 Task Planning

Task planning appears in many guises. For instance, consider a robot that has to pick a box from a desk drawer and place it inside a wardrobe. Initially, the drawer and the wardrobe are closed. At the end of execution, the box must be inside the wardrobe and both must be closed. The high-level controller does not define the continuous trajectory that the manipulator must execute, it only specifies the task execution constraints. The high-level task specification does not define a motion sequence to be executed. The order in which the robot will open the drawer and the wardrobe and pick up the box is not specified. The robot can first open the drawer, pick up the box, place it on the table, open the wardrobe and continue the task or it can first open the drawer and the wardrobe and place the box directly in the wardrobe and only at the end close both. There are a lot of possibilities.

Artificial intelligence (AI) has been dealing with the solution of problems as the above. Fikes and Nilsson (1971) propose a problem solver called STRIPS (Stanford Research Institute Problem Solver) that searches for a model in a space of world models to reach the desired goal by applying operators to the world models. Operators are actions that have preconditions and effects. However, STRIPS does not regard temporal specifications and only uses linear sequences of operators (Fikes and Nilsson, 1971). Furthermore, STRIPS operators do not allow their effects to be conditional. To enhance STRIPS, Gelfond and Lifschitz (1998) propose the Action Description Languages (ADLs) that operate on states of a transition system. In this context, a transition system can be understood as a labeled directed graph in which each vertex represents a system state and actions causes the transition between the states. Later on, the Planning Domain Description Language (PDDL) (Ghallab et al., 1998) was derived, among other formalisms, from ADL. PDDL is

composed of domain files that contain the predicates, actions, and effects and an instance file that contains the objects, initial state and goal description. The system state depends on the predicates that are true. Moreover, the predicates dictate the preconditions for an action to be executed and the effects represent the changes an action causes in the world. Finally, the goal descriptions are used to specify the state that the system must reach. For robotic domains, it is common to define predicates for locations of objects and the robot, and some actions representing grasping, holding, moving, and placing. Although these formalisms can describe task planning problems, they require lower layers to translate their very high-level plans to robot actions. In this sense, Lana et al. (2015) propose the Manipulation Task Model (MTM) that is dedicated to the planning of robotic manipulation tasks in a bottom-up manner.

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

An interesting alternative for the aforementioned formalisms is linear temporal logic (LTL), which allows to specify Boolean and temporal constraints. Also, LTL has correctness and completeness guarantees (Baier and Katoen, 2008, ch. 5), but the number of states of the specified task has combinatorial growth, which is also known as the state-explosion problem (Wongpiromsarn et al., 2010). Therefore, frameworks based on LTL usually build a discrete abstraction of the robotic system (Kress-Gazit et al., 2007; He et al., 2015; Kloetzer and Belta, 2008; Bhatia et al., 2011), but such construction is usually non-trivial. From the application point of view, LTL can be used to specify a wide range of robotic tasks such as coverage, sequencing, conditions, avoidance and counting (Kress-Gazit et al., 2007; McMahon and Plaku, 2014). In our application, described in Chapter 1, it is desirable to specify tasks with temporal constraints that enable enforcing execution order. Hence, LTL appears as a suitable formalism. See Table 2.1.

Table 2.1: Comparison of task planning methods.

Work	Advantages	Disadvantages
Fikes and Nilsson (1971); Gelfond and Lifschitz (1998); Ghallab et al. (1998)	Specify high-level tasks using actions with preconditions and effects for a scene and goal description.	Require lower layers to translate their very high-level plans to robot actions.
Lana et al. (2015)	Specify sequence of actions that the robot must execute. The actions contains wrenches and twists that the robot must use. This way, it is easier to use motion controllers to execute the task.	Work only for manipulation applications. For instance, navigation tasks cannot be specified. Additionally, temporal specifications are not allowed.
Linear temporal logic (LTL)	Allows to specify Boolean and temporal constraints. Allows to check correctness and completeness of systems. Allows specifying tasks such as coverage, sequencing, conditions, avoidance and counting.	Suffers from the state-explosion problem. Usually requires a discrete abstraction of the robotic system.

LTL has been used mainly in mobile robotics. Kress-Gazit et al. (2007) propose the construction of controllers for mobile robots that generate continuous trajectories satisfying LTL task specifications. The LTL task specification explicitly models sensor inputs. This way, the task descriptions consider dynamic environments and enable multi-robot search and rescue style missions. Bhatia et al. (2011) propose the use of co-safe[1] LTL formulas that allow to specify finite-horizon tasks. The LTL formula depends on the regions of interest in the workspace for the robot. This means that the specification contains the sequence of regions the robot must visit. McMahon and Plaku (2014) deal with the problem of computing a control function that considers dynamics and a co-safe LTL specification to generate a collision-free trajectory for nonlinear, high-dimensional mobile robots. Kloetzer and Mahulea (2015) present a method that controls a group of mobile robots to accomplish a task specified in LTL_x (LTL which does not use the operator Next and is interpreted over an infinite set of truth values). Differently from the other works, the LTL_x specification accounts for the tasks of the whole team of robots instead of specifying tasks for each robot. Another difference is that the locations may appear or disappear based on probability density functions. Most of the aforementioned works use LTL task specifications that contain the regions that the robot must visit. As a result, they depend on the discrete representation of the state space that is commonly represented by a discrete abstraction. Moreover, they usually convert the LTL specification formula into an automaton whose behaviors satisfy the formula. Next, they combine the discrete

[1]Co-safe LTL formulas are LTL formulas that can be interpreted using finite words and only contain the temporal operators Next, Eventually and Until. Moreover, the negation operator is not allowed over temporal formulas.

abstraction with the automaton to obtain a product graph that contains the sequence of regions the robot must visit while satisfying the task specification.

In addition to the other approaches, Kloetzer and Belta (2008) propose a framework to solve the following problem: given an LTL_x formula, find a set of initial states and a feedback control law for a system such that all the closed-loop trajectories of the system satisfy the formula and stay inside a polytope that can, for instance, represent the environment boundaries for a planar robot. Also in the context of abstraction-free temporal logic frameworks, Lindemann (2018) proposes to obtain control strategies for single- and multi-agent input-affine dynamical systems such that it satisfies a signal temporal logic (STL) task[2]. The first controller is a model predictive controller (MPC) that uses reformulated STL robust semantics that allows obtaining a computationally efficient method to check how robustly a task is satisfied. The next strategy is based on a continuous-time feedback control law robust to noise that satisfies as best as possible the STL task. Next, a control method that connects temporal properties with time-varying control barrier functions through predicate functions is proposed. The method allows a broader range of STL specifications but loses robustness to noise. Lastly, a decentralized continuous-time feedback control law is obtained for single-agents that forms together a multi-agents system in which each agent must satisfy a STL specification. More recently, Kundu and Saha (2019) propose an algorithm to generate trajectories for robots that need to meet functional and energy requirements from an LTL formula specification. The task specification depends on locations where the robot must go and on the energy level of the robot. The motion of the robot is captured using a set of motion primitives and the robot moves in an occupancy grid in discrete steps.

As can be seen, the abstraction techniques for mobile robotics commonly depend on the discrete representation of the state-space, which makes them intractable to manipulation planning due to the large dimensionality of the manipulation problem. To solve this problem, He et al. (2015) propose a framework that allows the planning of high-level manipulation tasks specified in co-safe LTL formulas. The formulas contain the order of manipulation and the location where each object must be placed in the environment. From the task planning point of view, the high-level planner converts the co-safe LTL task specification into a DFA and combines it with a suitable manipulation abstraction that describes the actions the manipulator can execute to manipulate objects in the environment. As a result, it is possible to search for a task plan that satisfies the LTL formula and contains what the robot must do. Later on, still in the context of LTL in manipulation task planning, He et al. (2017) propose a similar planning framework to solve the problem of completing a manipulation task plan in the presence of external interference from humans by also considering human movements in the abstraction. Afterward, He et al. (2019a)

[2]STL consists of predicates (maps from $\mathbb{R}^n$ to $\{\mathsf{T}, \mathsf{F}\}$) that are obtained after evaluation of a predicate function (maps from $\mathbb{R}^n$ to $\mathbb{R}$)

developed a way to automatically construct such abstraction. In another work, He et al. (2019b) propose a solution for the LTL-based reactive manipulation planning problem that uses PDDL instead of an abstraction and binary decision diagrams (BDDs) alongside the DFA. BDDs are a compact representation of Boolean functions that map a set of boolean variables to a boolean output (Bryant, 1986). The results show a faster solution for the reactive planning problem. Since the goal of this work is to solve the task planning problem for manipulation tasks without interference, the methodology proposed by He et al. (2015) is adopted. Lastly, the framework also handles the ITMP problem.

2.2 Integration of Task and Motion Planning

To solve the ITMP problem, Kambhampati et al. (1991) investigate the use of a hybrid planning model that contributes with expressiveness and reasoning power to traditional hierarchical planners through the use of a set of specialists, which adapt the plan to satisfy unexpected additional constraints. Hence, the planner and the specialists must know the constraints imposed by their decisions to avoid inconsistent tasks. To add more flexibility to ITMP, some works have started using multi-layered frameworks that use a task planner to generate a motion sequence for the motion planner, which checks the viability of executing the task (Erdem et al., 2011; Kaelbling and Lozano-Perez, 2011; Srivastava et al., 2014; Lozano-Perez and Kaelbling, 2014; He et al., 2015; Dornhege et al., 2009; Bhatia et al., 2011). Other works consider constraints imposed by the environment or by the system dynamics and construct task plans incrementally (Cambon et al., 2009; Plaku and Hager, 2010).

Dornhege et al. (2009) decompose the manipulation problem in a symbolic part which represents high-level tasks and a geometrical part which represents the manipulator kinematics and the environment description. Erdem et al. (2011) use predicates and functions to test execution viability but specifying which checks must happen at high- or low-level. In addition to using LTL, Bhatia et al. (2011) propose a multi-layered framework, which builds a discrete abstraction of the system. It generates high-level plans based on information from the low-level planner. Afterward, it uses the sampling-based low-level planner to generate trajectories for the high-level plans. The work of Srivastava et al. (2014) uses three layers: high-level, interface, and low-level. The interface layer allows the communication between task and motion planners in order to identify low-level planner failures to demand a new task from the high-level planner. The procedure repeats itself until a low-level trajectory generated for the high-level specifications can be executed successfully or until all the high-level tasks are eliminated. The interface between high-level and low-level planning is common in ITMP works since the low-level planner cannot always generate feasible plans for a high-level plan. However, there are also approaches that are implemented in a top-down manner.

The method proposed by Plaku and Hager (2010) called sampling-based motion and symbolic action planner (SMAP) generates a task plan tree from a PDDL specification that contains actions with preconditions and postconditions to transform the world. The predicates for each action depends on a finite collection of continuous variables in the world. In the initial state of the world, symbolic action planning is done to select actions whose preconditions are satisfied. Next, sampling-based motion planners that allow treating collision avoidance and differential constraints are used to extend the plan tree to continuous states that satisfy the action postconditions. This process is repeated until the PDDL specification goal is reached. Kaelbling and Lozano-Perez (2011) propose a top-down ITMP aggressively hierarchical architecture that generates plans for a task specification based on predicates, world states, goals, and operators that modify the world. Given a task specification, the initial plan state has the plan top-level goal preconditions and the operators to be applied to the state. If an operator is primitive it is directly executed, else a new plan step with its preconditions is added to the plan tree. The plan tree is built in a depth-first manner until the set of top-level goal preconditions are satisfied. By not modifying the first high-level plan step until it is satisfied, the hierarchical problem decomposition is simpler and results in lower search times. Furthermore, the technique operates in the continuous geometry domain, that is, it does not require a priori discretization of the state-space. Later on, Lozano-Perez and Kaelbling (2014) propose to use partial plans for a manipulation problem, which is solved as a constraint satisfaction problem.

In addition to using LTL, He et al. (2015) also propose a multi-layered framework to solve the ITMP problem, which uses a coordinating layer between high- and low-level to decide about the difficulty of executing the task. Furthermore, the use of LTL allows the description of complex manipulation tasks with temporal constraints, which other ITMP techniques are not capable of describing (He et al., 2015). On the other hand, the state-explosion problem also appears in LTL planners, but the multi-layer abstraction technique reduces the problem. Nonetheless, the size of the abstraction still quickly grows with an increase in the number of locations. As a result, the state-space explosion problem is still present. Another drawback is that it also uses a motion planner in the low-level layer, which requires motion planning time, in addition to the high-level planning time.

Lastly, although we have chosen to adapt the framework of He et al. (2015), there are already studies using it as a benchmark for ITMP. In this sense, we have to mention the task and motion planning framework of Dantam et al. (2018). Dantam et al. (2018) propose a method that allows using more flexible abstractions than the domain-specific manipulation abstraction used by He et al. (2015). Instead of using LTL to describe the task domain, PDDL is used. More specifically, they define a formal language that accommodates preconditions, actions, task operators, and effects. Therefore, a task plan is a string in the task language that leads to an accepting state. Moreover, constraint-based

task planning is done by using incremental satisfiability modulo theories (SMT). SMT allows adding rules to Boolean satisfiability and has a more expressive way of expressing constraints (logical assertions about motion feasibility). From the motion planning point of view, the method uses off-the-shelf RRTs sampling-based motion planners from the Open Motion Planning Library (Sucan et al., 2012). Finally, the ITMP problem is solved by using geometric information from failed motion plans to update the constraints used in SMT. This way, this novel method for SMT allows new plans to take into account previous failed plans in the planning process. Both task and motion planning horizons are increased in the case of a failed plan. The results show that, in the case of tasks with multiple objects, the method of Dantam et al. (2018) has significantly better time performance when compared to the framework of He et al. (2015). Table 2.2 summarizes the frameworks that are built on the ideas of the aforementioned works or that solve some of the issues still present in ITMP with temporal logic.

Table 2.2: Comparison of frameworks that solve the robotic task planning problem.

Work	Advantages	Disadvantages
Dantam et al. (2018)	Plan effectively for manipulation tasks with multiple objects.	Strongly couple task and motion planning. This way, it is more difficult to implement methods for task relaxation.
Lindemann (2018)	Propose an abstraction-free signal temporal logic method that does not suffer from the curse of dimensionality and that is computationally efficient for single- and multi-agent systems.	It cannot be directly applied to plan manipulation tasks. The work focused on ground and aerial vehicles and on multi-robot scenarios.
He et al. (2015)	Generate plans for manipulation tasks. Use LTL to specify the tasks. Decouple high- and low-level layers resulting in an interesting option to do task relaxation since the low–level motion planner can be substituted for a constrained motion controller that accommodates several constraints.	Require an abstraction that suffers from the curse of dimensionality in the case of using multiple objects or having a highly complex task specification.

2.3 Task Relaxation

The framework proposed by He et al. (2015) solves the ITMP problem. However, some manipulation tasks are described by using regions of interest that contain multiple locations. For instance, if it is desired that a manipulator pick an object and place it on a table, it could place it anywhere on the table. Thus, a common approach is to discretize the table

into many possible locations and sample one of them. Therefore, a possible solution for such cases is to discretize the workspace in multiple locations to define regions of interest for placing the manipulated objects. However, a discretization of the workspace does not scale well for the high-level planner in the work of He et al. (2015), since an increase in the number of locations may cause the state-space explosion problem.

In the context of workspace discretization, Garrett et al. (2018) propose the FFRob algorithm for solving task and motion planning problems. In the FFRob, a state is represented by a set of values that indicate the robot configuration, the object being held, and the pose of each object. Garrett et al. (2018) also propose the extended action specification (EAS) to specify tasks. EAS is similar to STRIPS but allows preconditions to be logical formulas that determine, for instance, when the robot can perform an action. Given a task specification, the FFRob alternates between sampling and planning phases until a solution is found. In the sampling phase, they discretize the pick and place problem by creating symbolic actions from a finite sampled set of states. Next, the planning phase searches the discretized possible options. They also define a relaxed state as a set of states formed by all combinations of possible values for a state. In other words, a relaxed state represents all the possible robot configurations, objects, and poses of objects for the state. This means that the robot simultaneously tests all actions of the set of states represented by the relaxed state. However, in the work of He et al. (2015), a similar approach would quickly increase the number of planning states resulting in longer planning time. Differently from the task relaxation in a top-down manner, it is also possible to approach the task relaxation problem in a bottom-up manner, that is, by making use of kinematic redundancy in the low-level motion controller.

Classic kinematic redundancy techniques allow task constraints in addition to the main task control law. This is done by using the degrees of freedom not being used in the main task. Hence, a secondary task that does not disturb the main task and consider constraints can be added. In this sense, Mansard and Chaumette (2009) propose the directional redundancy method that only activates the secondary task if it does not increase the error of the main task, which preserves the stability of the system. This is done by building an operator similar to the one that projects in the nullspace of the Jacobian that allows the control law error components to converge separately, which enables to add constraints to ensure that each component surely decreases. In the context of task relaxation, this method could be used to move the end-effector towards a target pose in the workspace while satisfying constraints such as joint-limits. Nonetheless, our goal is to move towards a region of interest and, hence, more constraints are required. This can be done by exploiting kinematic singularity avoidance techniques and switching between control strategies.

The kinematic singularity avoidance techniques in the low-level controller for redundant manipulators are usually based on using self-motion (i.e. movements of the links that do not disturb the end-effector location (Bedrossian, 1990)). The most used technique stems

from the optimization of a manipulability function projected on the Jacobian nullspace. Moreover, a secondary task can be satisfied through the projection in the nullspace of the Jacobian (Liégeois, 1977). In this context, Figueredo et al. (2014) propose the relaxation of task constraints and the switching to control tasks with fewer degrees of freedom to allow the use of self-motion whenever possible. The switching strategy is based on relaxing the control requirements according to offline defined geometric task objectives. For instance, consider a manipulator that has to place a box on a table. The box pose requirements may be satisfied after the box is within a bounded geometric region (i.e. region of interest) over the table and, hence, position control alone can be used to place the box on the table. The switching rule requires the definition of geometric controllable sets. They define which task primitive (i.e. distance, position, orientation, or pose) should be controlled to achieve a geometric task. Lastly, to enhance the task relaxation capabilities, additional optimization criteria with respect to the task-space variables and the task-space constraints such as obstacles and joints limits could be considered. To do this, a possible solution is to use hierarchical least-square optimization (Escande et al., 2014).

Least-square optimization can be used to fulfill as best as possible a set of constraints that may not be feasible (Escande et al., 2014). Furthermore, when the constraints are linear on the control inputs, the least-square problem is written as a quadratic program. Thus, Escande et al. (2014) propose a method to solve a hierarchical quadratic program in a computationally efficient manner that allows to solve problems with many variables fast enough to be used in real-time control. This is achieved by means of lexicographic optimization. In other words, it is not possible to optimize an objective with lower priority without increasing the objective of higher priority. As a result, the multi-objective problem turns into a single iterative process that can be solved at once. However, the work focused mostly on applying the technique to solve the inverse kinematics and adding simple obstacle-avoidance constraints. Meanwhile, Marinho et al. 2019 used similar concepts to propose a constrained motion controller.

The constrained motion controller (Marinho et al., 2019) allows the definition of constraints to keep the robot outside a restricted region or inside a safe region and the system to be reactive. By using suitable constraints, we can define regions of interest instead of discrete poses for the end-effector. The idea is to specify geometrical constraints, based on those regions of interest, to accomplish the desired task generated by the high-level task-planner without resorting to low-level motion planning. Therefore, we adapt the framework of He et al. (2015) to use the constrained motion controller. Constrained motion controllers depend on the specification of a set of constraints that must be respected by the system. They can be described by minimization problems (Laumond et al., 2015) that make use of extra available degrees of freedom of the robot to achieve the task while respecting the constraints. This enables the generation of collision-free motions by using mathematical programming. Nonetheless, in the general case, there are no analytical

solutions and numeric solvers must be used (Goncalves et al., 2016; Escande et al., 2014). Table 2.3 summarizes the task relaxation methods. Since our goal is to relax the tasks to regions of interest, we have adopted the constrained motion controller of (Marinho et al., 2019).

Table 2.3: Comparison of the task relaxation methods.

Work	Advantages	Disadvantages
Garrett et al. (2018)	Allow relaxed states that discretize a state into all the possible robot configurations, objects, and poses of objects for the state.	Does not scale well when combined with the framework of He et al. (2015) because of the discretization.
Mansard and Chaumette (2009)	Allow specifying constraints such as joints limits.	Depending on the number of robot DOF, it does not accomodate constraints such as obstacle avoidance in the case of having multiple obstacles because the number of DOF available for the secondary task is not enough.
Figueredo et al. (2014)	Switch between control objectives during task execution reducing the number of degrees of freedom required. This way, the remaining degrees of freedom can accomodate secondary tasks.	Do task relaxation in an ad hoc manner by requiring to specify when to switch between control objectives.
Marinho et al. (2019)	Allow the definition of constraints to keep the robot outside a restricted region or inside a safe region. Thus, it is possible to define regions of interest by defining suitable constraints.	May suffer from local minima.

Lastly, some of the regions of interest in the scene may be complicated to describe using only the constraints proposed in the framework of Marinho et al. (2019). For instance, to define a squared region of interest, it is needed to use at least four planes and four inequalities in the control law. However, for relaxed tasks, the form of the region of interest is not always important. In this sense, we adopt a new approach to define point-cone constraints, which requires less complex calculations during running time and only one inequality in the control law. This approach enables to define a circular region of interest by using only a cone.

2.4 Conclusion

The first part of this chapter introduces different formalisms that allow solving the task planning problem. Among other formalisms, LTL emerged as a suitable option to specify

robotic tasks because it allows to specify different tasks such as coverage, sequencing, conditions, avoidance and counting (Kress-Gazit et al., 2007; McMahon and Plaku, 2014). LTL has been mainly used in mobile robotics, but it is used to describe manipulation tasks in the planning framework of He et al. (2015). The second section of this chapter gives an overview of ITMP and cites the ITMP solution in the work of He et al. (2015) that still spends a significant amount of time with motion planning. Despite its drawbacks, we have chosen to use the LTL based planning framework of He et al. (2015) because it is specific for manipulation tasks without external interference—from a human—in the robot task and has the LTL expressivity. The final part of this chapter is concerned about manipulation tasks that can be best described using a region with multiple locations, that is, the task is relaxed to a region of interest. Although a common solution is to discretize the workspace into multiple locations, the framework of He et al. (2015) planning complexity quickly grows with an increase in the number of locations. In this context, we propose to solve the task relaxation problem with the framework proposed in Marinho et al. (2019) to mitigate the state-space explosion problem, since it allows to add constraints and keep the robot within a desired region of interest without discretization. As a result, the planning framework becomes more computationally efficient for relaxed tasks and is also reactive. Lastly, we propose a new approach to define conic constraints in the framework of Marinho et al. (2019) which requires less complex calculations during runtime and only one inequality in the control law. In contrast, the plane constraints require four inequalities to define a region of interest.

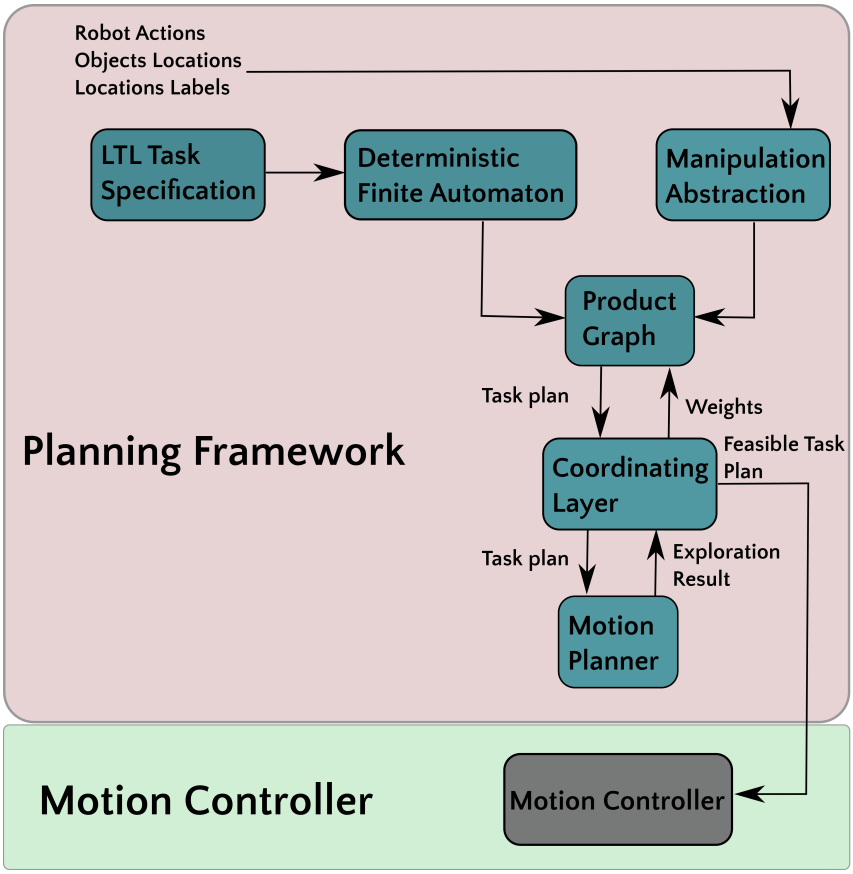
3

Manipulation Planning Framework

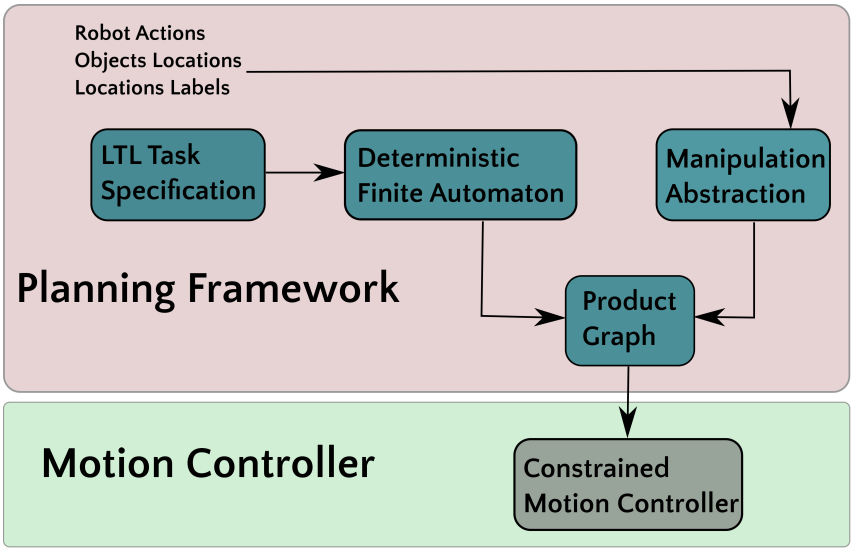
In this chapter, we cover the manipulation planning framework of He et al. (2015). First, we introduce linear temporal logic (LTL). In the sequence, we state the manipulation problem in assistive robotics. Next, we describe the planning framework. It is composed of a high-level planner, which receives task specifications in LTL, a coordinating layer, which assigns weights proportional to the difficulty of executing a high-level plan and a low-level motion planner, which generates motion plans for the high-level plans. Lastly, we introduce a new approach to the manipulation planning framework that substitutes the coordinating layer, the motion planning layer, and the motion controller for a single constrained motion controller. Although, we lose the probabilistic completeness of the original framework, the system is reactive to changes in the environment.

The high-level planner receives a task specification in LTL that is converted into a deterministic finite automaton (DFA) that specifies all the ways the robot can execute the task. Also, a manipulation abstraction, which captures all the ways the robot can manipulate the objects in the workspace, is defined and combined to the DFA into a product graph that represents all the ways the robot can execute the task. Afterward, the Dijkstra's algorithm is used to search for a path on the product graph that represents a complete task plan. Finally, the task plan is sent to a low-level motion planner that generates a motion plan and sends the result back to the coordinating layer, which then sends the plan to the motion controller. If the motion planner does not find a solution, the coordinating layer increase the weight for that plan on the product graph and requires a new task plan on the product graph. This generates a synergy between the layers resulting

in the generation of feasible continuous trajectories for the manipulator. Meanwhile, in the adapted framework, the task plan is sent to the constrained motion controller that executes it. Hence, there is no need for a motion planner and, hence, there is no replanning phase, thus saving computational time and resulting in a reactive system. Figure 3.1 depicts the original and the adapted framework.



(a) Original manipulation planning framework proposed by He et al. (2015)



(b) Adapted manipulation planning framework

Figure 3.1: Manipulation planning frameworks: the *red* area represents the planning framework and its steps in *cyan;* the *green* area represents the motion controller in *gray*. The manipulation abstraction receives the robot actions, the objects locations and locations labels as input.

3.1 Linear Temporal Logic

In LTL, a *proposition* is a statement that can be true or false, but not both, and *atomic propositions* are the ones that do not depend on the truth or falsity of any other proposition. Let $\mathbb{A} = \{a_0, a_1, ..., a_N\}$ be a set of atomic propositions. LTL semantics is defined over infinite traces, that is, words over the alphabet $2^{\mathbb{A}}$, where $2^{\mathbb{A}}$ is the power set of $\mathbb{A}$. Given letters $A_i \in 2^{\mathbb{A}}$, with $i = 0, 1, 2, \ldots$, words are finite or infinite sequences such as $\sigma = A_0 A_1 A_2 \cdots A_n$ and $\sigma = A_0 A_1 A_2 \cdots$. For example, consider $\mathbb{A} = \{a_0, a_1, a_2\}$, hence, a possible word would be $\sigma = A_0 A_1 A_0 A_1 A_0 A_1 \cdots$, with $A_0 = \{a_0, a_1\}$ and $A_1 = \{a_0\}$, which means that a_0 is always true as it belongs to all letters, a_1 alternates between true (when it appears in a letter) and false (when it does not appear in a letter), and a_2 is always false, because it does not belong to any letter in the word σ.

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

$$\varphi = a \,|\, \neg a \,|\, \varphi_1 \wedge \varphi_2 \,|\, \varphi_1 \vee \varphi_2 \,|\, \varphi_1 \mathcal{U} \varphi_2 \,|\, \mathcal{X}\varphi_1 \,|\, \mathcal{E}\varphi_1, \tag{3.1}$$

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

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

1. $\varphi = \mathsf{T}$[2] is always true;
2. $\varphi = a_0$ is true if and only if a_0 is true;
3. $\varphi = \neg a_0$ is true if and only if a_0 is false;
4. if $\varphi_0 = a_0$, $\varphi_1 = a_1$, then $\varphi = \varphi_0 \wedge \varphi_1$ is true if and only if both a_0 and a_1 are true;
5. if $\varphi_0 = a_0$, $\varphi_1 = a_1$, then $\varphi = \varphi_0 \vee \varphi_1$ is true if and only if a_0 is true or a_1 is true or both are true;
6. if $\varphi_0 = a_0$, $\varphi_1 = a_1$, then $\varphi = \varphi_0 \mathcal{U} \varphi_1$ is true if and only if a_1 becomes true after a_0 was already true;

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

[2] The notation T and F indicates the Boolean values "true" and "false".

7. if $\varphi_1 = a_1$, then $\varphi = \mathcal{X}\varphi_1$ will be definitely true if and only if a_1 becomes true in the next time step;

8. if $\varphi_1 = a_1$, then $\varphi = \mathcal{E}\varphi_1$ will be eventually true since a_1 becomes true at some point in the future.

The formulas for φ, φ_1, and φ_2 could be any formula containing the operators in (3.1) with an arbitrary number of propositions. Although in the previous example we used simple formulas for the sake of clarity, they can be arbitrarily more complex. For instance, $\varphi_2 = \mathcal{E}(a_0 \wedge \mathcal{X}\mathcal{E}(a_1))$ means that eventually a_0 will be true and, afterward, a_1 will eventually be true. More specifically, the operation $\mathcal{E}(a_0 \wedge \mathcal{X}\mathcal{E}a_1)$ implies that a_0 and $\mathcal{X}\mathcal{E}a_1$ will surely be true in an unknown time in the future. Let us assume, for the sake of example, that a_0 becomes true in time j. The operation $\mathcal{E}a_1$ implies that a_1 will surely be true in an unknown time k in the future and $\mathcal{X}\mathcal{E}a_1$ implies that $k \geq j + 1$. Therefore, eventually a_0 will be true, and, after that, eventually a_1 will be true. The formula φ_2 could be used, for instance, to specify a pick-and-place task that requires two objects to be eventually manipulated in a given order.

The semantics of syntactically co-safe LTL formulas are defined over infinite words from the set $2^{\mathbb{A}}$. Given $\sigma = A_0A_1A_2\cdots$, let $\text{suffix}(\sigma, i) = A_iA_{i+1}\cdots$ and $\text{prefix}(\sigma, i) = A_0A_1\cdots A_{i-1}$. Recall that an atomic proposition is true when it belongs to a letter and false otherwise. The satisfaction relation $\sigma \vDash \varphi$ indicates that the word σ contains letters that satisfy the formula φ. For example, if $\mathbb{A} = \{a_0, a_1, a_2\}$, $A_0 = \{a_0, a_1\}$, and $\varphi = (a_0 \wedge a_1) \vee a_2$, then $\sigma = A_0$ satisfies φ because $A_0 \vDash (a_0 \wedge a_1)$ and, hence, $\sigma \vDash (a_0 \wedge a_1) \vee a_2$. Given $\mathbb{A} = \{a_0, a_1, \ldots, a_N\}$, the LTL formulas φ_1 and φ_2, a word $\sigma = A_0A_1A_2\cdots$ satisfies a LTL formula φ, denoted by $\sigma \vDash \varphi$, if one of the following is true.

1. $\sigma \vDash \mathsf{T}$;

2. $\sigma \vDash a_0 \iff A_0 \vDash a_0$, that is, $a_0 \in A_0$;

3. $\sigma \vDash \varphi_1 \wedge \varphi_2 \iff \sigma \vDash \varphi_1 \wedge \sigma \vDash \varphi_2$;

4. $\sigma \vDash \neg\varphi \iff \sigma \nvDash \varphi$;[3]

5. $\sigma \vDash \mathcal{X}\varphi \iff \text{suffix}(\sigma, 1) = A_1A_2A_3\cdots \vDash \varphi$;

6. $\sigma \vDash \varphi_1\mathcal{U}\varphi_2 \iff$ there exists $j \geq 0$ such that $\text{suffix}(\sigma, j) = A_jA_{j+1}A_{j+2}\cdots \vDash \varphi_2$ and $\text{suffix}(\sigma, i) = A_iA_{i+1}A_{i+2}\cdots \vDash \varphi_1$ for $0 \leq i < j$;

7. $\sigma \vDash \mathcal{E}\varphi$ if and only if there exists $j \geq 0$ and $\text{suffix}(\sigma, j) \vDash \varphi$.

To exemplify each one of the properties above, let $\mathbb{A} = \{a_0, a_1, a_2, a_3\}$, $A_0 = \{a_0\}$, $A_1 = \{a_0, a_1\}$, $A_2 = \{a_0, a_1, a_2\}$. Let also $\varphi = a_1$, $\varphi_1 = a_0$, $\varphi_2 = a_0 \vee (a_1 \wedge a_2)$, $\varphi_3 = a_0 \wedge a_1 \wedge a_2$, $\varphi_4 = a_3$, and $\sigma = A_0A_1A_2$. Thus,

[3] $\sigma \nvDash \varphi$ means that σ does not satisfy φ.

1. $\sigma \vDash \mathsf{T}$ because true does not depend on any proposition;
2. $\sigma \vDash a_0$ because all letters of σ contain a_0, that is, $a_0 \in A_0$, $a_0 \in A_1$, and $a_0 \in A_2$;
3. $\sigma \vDash \varphi_1 \wedge \varphi_2$ means that $\sigma \vDash a_0 \wedge (a_0 \vee (a_1 \wedge a_2))$ because $\sigma \vDash a_0$ and $\sigma \vDash a_0 \vee (a_1 \wedge a_2)$ since $a_0 \in A_0$, $a_0 \in A_1$ and $a_0 \in A_2$;
4. $\sigma \vDash \neg\varphi_4$ means that $\sigma \vDash \neg a_3$ because $a_3 \notin A_0$, $a_3 \notin A_1$, $a_3 \notin A_2$, hence $\sigma \nvDash \varphi_4$;
5. $\sigma \vDash \mathcal{X}\varphi$ means that $\sigma \vDash \mathcal{X}a_1$ because $a_1 \in A_1$, and $a_1 \in A_2$, that is, $\text{suffix}(\sigma, 1) = A_1A_2 \vDash \varphi$;
6. $\sigma \vDash \varphi_1 \mathcal{U} \varphi_3$ means that $\sigma \vDash a_0 \mathcal{U}(a_0 \wedge a_1 \wedge a_2)$ because $a_0 \in A_0$, $a_0 \in A_1$, and $a_0, a_1, a_2 \in A_2$, that is, $\text{suffix}(\sigma, 0) = \text{A}_0\text{A}_1\text{A}_2 \vDash \varphi_1$ and $\text{suffix}(\sigma, 2) = A_2 \vDash \varphi_3$;
7. $\sigma \vDash \mathcal{E}\varphi_3$ means that $\sigma \vDash \mathcal{E}(a_0 \wedge a_1 \wedge a_2)$ because $a_0, a_1, a_2 \in A_2$, that is, $\text{suffix}(\sigma, 2) = A_2$.

In this work, manipulation tasks must be achieved over a finite time horizon. Hence, we use only co-safe LTL formulas, which are the ones that can be interpreted by considering finite words (Kupferman and Y. Vardi, 2001). To define co-safe LTL formulas we recall some definitions. First, a language is a set of finite or infinite words. A bad prefix for a language is a finite word that does not belong to the language anymore when concatenated to an infinite word (Kupferman and Y. Vardi, 2001). A safety language is such that every word not in the language has a finite bad prefix. Meanwhile, a co-safety language is such that every word in the language has a good prefix (i.e. a prefix that when concatenated to an infinite word belongs to the language) (Kupferman and Y. Vardi, 2001). In this sense, an LTL formula is safe if the set of words that satisfy the formula is a safety language and an LTL formula is co-safe if the set of words that satisfy the formula is a co-safety language (Kupferman and Y. Vardi, 2001). In other words, an LTL formula is co-safe if and only if every infinite word satisfying the formula has a good finite prefix that can be concatenated with any infinite word and still satisfy the formula. Syntactically, co-safe LTL formulas contain only the temporal operators $\mathcal{X}$, $\mathcal{E}$, $\mathcal{U}$, and the negation operator is only allowed over atomic propositions, but not over temporal formulas.

Although the manipulation tasks in this work must be completed in finite time, other fragments of LTL can describe infinite time horizon tasks. For instance, LTL_x formulas, which do not contain the operator "next", are interpreted over infinite words of the alphabet (Kloetzer and Mahulea, 2015). Tasks such as navigation and surveillance of more regions in an arbitrary or specific order can be described in LTL_x (Kloetzer and Mahulea, 2015). Another class of LTL formulas called Generalized Reactivity(1) is used to describe tasks that have a response to a given input (Kress-Gazit et al., 2007). For example, sensor-based robotic tasks usually require a response of the robot after a sensor input occurs as in the case of a robot that senses an obstacle and must stop.

See the works of Kupferman and Y. Vardi (2001); Baier and Katoen (2008) for more formal definitions of the syntax and semantics of LTL. Examples of LTL specifications for manipulation tasks used in this work are given in Section 3.3.1.

3.2 Problem Statement

We define our problem based on the requirements of the assistive robotics application scenario presented in Chapter 1 (See Figure 1.1). The scenario is composed of a manipulator that can pick-and-place objects between locations and a person with limited or no lower limbs mobility. The manipulator must execute pick-and-place tasks specified by the person. In this sense, our problem is defined as follows: given a finite set of cuboid objects O, a finite set of locations $\mathcal{L}$, and a finite set of actions $\mathcal{M}$, find a sequence of actions $\alpha_0, \alpha_1, \ldots, \alpha_m \in \mathcal{M}$ that manipulate the objects $o_0, o_1, \ldots, o_n \in O$ between locations $\text{loc}_0, \text{loc}_1, \ldots, \text{loc}_k \in \mathcal{L}$ to satisfy a linear temporal logic (LTL) specification φ. The sequence of actions is obtained by the planning framework that will be presented in Section 3.3 and the actions are executed by the constrained motion controller presented in Chapter 4.

3.3 Planning Framework

3.3.1 Linear Temporal Logic Task Specification

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

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

3.3.2 Manipulation Abstraction

The next step is to define an abstraction graph $\mathcal{R} = (V, E, L)$ that captures all the ways the robot can manipulate the objects, in which the set of nodes V consists of the Cartesian product between the set $\mathcal{M}$ of motion primitives, the set $\mathcal{L}$ of locations, the set O of objects, and a set $O_{\mathcal{L}}$ of objects locations. The set E contains the edges of $\mathcal{R}$ and L is a labeling function that is defined later in Section 3.3.4 when we introduce the product graph. The set of motion primitives contains the robot actions needed for pick-and-place tasks and is defined as $\mathcal{M} \triangleq \{\text{GRASP}, \text{PLACE}, \text{HOLD}, \text{MOVE}\}$. The motion graph $\mathscr{M} \triangleq (\mathcal{M}, E_{\mathscr{M}})$, with $E_{\mathscr{M}} \subseteq \{(m_i, m_j) \in \mathcal{M} \times \mathcal{M}\}$, which is shown in Figure 3.2, defines the allowed sequence of motions.

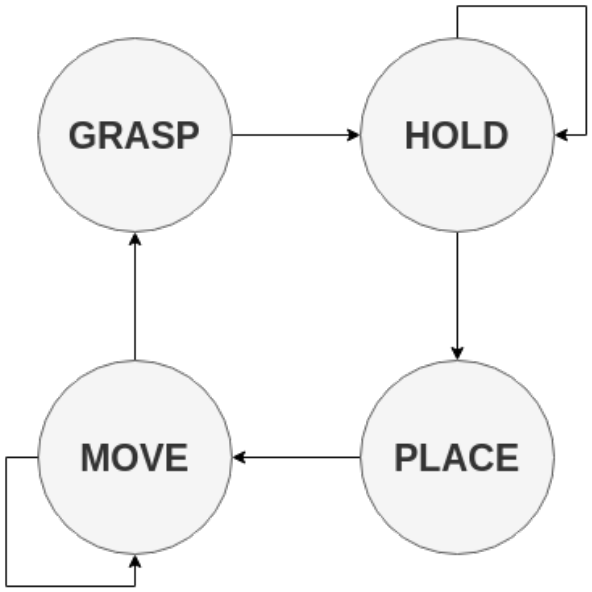


Figure 3.2: Motion graph $\mathscr{M}$

The location set $\mathcal{L} \triangleq \{\text{loc}_1, ..., \text{loc}_k, \text{loc}_{\text{inter}}\}$ represents the locations of interest in the robot workspace where objects can be placed. Also, when an object is in the gripper, it is considered to be in the intermediate location $\text{loc}_{\text{inter}}$. In other words, an object is in the $\text{loc}_{\text{inter}}$ when it is being manipulated. In the case of the manipulator end-effector location, the $\text{loc}_{\text{inter}}$ is defined as an arbitrary location in the workspace. Furthermore, the location graph $\mathscr{L} \triangleq (\mathcal{L}, E_{\mathscr{L}})$ with $E_{\mathscr{L}} \subseteq \{(\text{loc}_i, \text{loc}_j) \in \mathcal{L} \times \mathcal{L}\}$, which is shown in Figure 3.3, represents how objects can be transferred between locations.

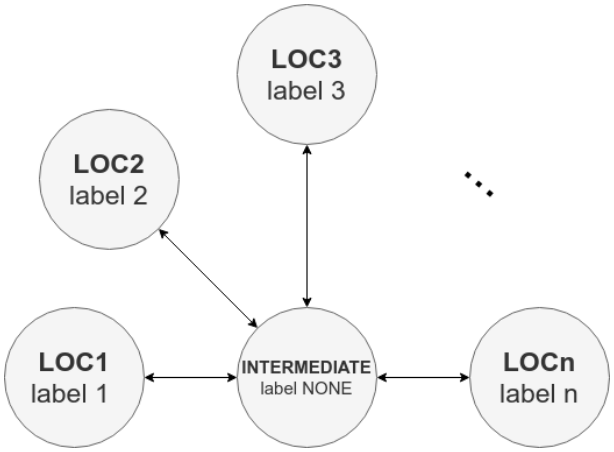


Figure 3.3: Location graph $\mathscr{L}$

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

The construction rules of the set of edges $E \subseteq \{(v_i, v_j) \in V \times V\}$ that connect the nodes V of $\mathcal{R}$ can be found in the work of He et al. (2015). Generally speaking, at least each edge $(v, v') \in E$ in the abstraction must satisfy that $(\alpha, \alpha') \in E_{\mathscr{M}}$ and the unmentioned elements remain the same, as explained next.

These are the construction rules of the set of edges $E \subseteq \{(e_i, e_j) \in V \times V\}$ that can be found in the work of He et al. (2015). They were adapted to suit the formalism adopted in this work.

1. $v = (\text{GRASP}, \text{loc}, o, \mathfrak{o}_\mathcal{L})$ and $v' = \left(\text{HOLD}, \text{loc}', o', \mathfrak{o}_\mathcal{L}\right)$. In this case, there must be some i such that $\bar{o}_{i,loc} = \text{loc}$. This results in $o' = o_i$ and $\bar{o}'_{i,loc} = \text{loc}_{\text{inter}}$. In other words, in the GRASP node, the object i being grasped must be in the end-effector location. Thus, in the HOLD node, the object in the gripper will be the object i and the location of object i will be gripper. In our case, the object is in the gripper, when it is in the $\text{loc}_{\text{inter}}$ location;

2. $v = (\text{HOLD}, \text{loc}, o, \mathfrak{o}_\mathcal{L})$ and $v' = \left(\text{HOLD}, \text{loc}', o', \mathfrak{o}_\mathcal{L}\right)$. In this case, all elements remain the same, except for loc', which can take any value adjacent to that of loc in the location graph. This means that, since an object is being held in the first node HOLD, in the second node HOLD it can be held to any other location adjacent to the first node in the location graph. For instance, if an object is being held at loc_1, it can either stay at the same location or be held to location $\text{loc}_{\text{inter}}$. See Figure 3.3;

3. $v = (\text{HOLD}, \text{loc}, o, \mathfrak{o}_\mathcal{L})$ and $v' = \left(\text{PLACE}, \text{loc}', o', \mathfrak{o}_\mathcal{L}\right)$. There will be an edge transition between them if $\text{loc} = \text{loc}' \neq \text{loc}_{\text{inter}}$, meaning that the robot will be holding the object at one of the k locations in v and placing it at that location in v';

4. $v = (\text{PLACE}, \text{loc}, o, \mathfrak{o}_\mathcal{L})$ and $v' = \left(\text{MOVE}, \text{loc}', o', \mathfrak{o}_\mathcal{L}\right)$. In this case, $o' = \varnothing$. Let i be the object such that $o = o_i$, then $\bar{o}_{j,\text{loc}}$ must not be loc for any $j \neq i$. Lastly, $\bar{o}'_{i,\text{loc}} = \text{loc}$. First, after an object is placed somewhere, the gripper will be empty. Moreover, suppose object i is in the gripper, then all objects different from i must not be at the location where the end-effector currently is. This condition must hold because the end-effector must be at a location with no object in order to place the object being held. Finally, the location of object i in the node MOVE will be equal to the location of the end-effector at the node PLACE, which means that the object i is placed where the end-effector currently is;

5. $v = (\text{MOVE}, \text{loc}, o, \mathfrak{o}_\mathcal{L})$ and $v' = \left(\text{MOVE}, \text{loc}', o', \mathfrak{o}_\mathcal{L}\right)$. In this case, all elements remain the same, except for loc', which can take any value adjacent to that of loc in the location graph. This rule is similar to rule 2. It means that the robot can move the end-effector between any location connected in the location graph. For example, if the end-effector is at loc_1, it can either stay at the same location or move to $\text{loc}_{\text{inter}}$. See Figure 3.3;

6. $v = (\text{MOVE}, \text{loc}, o, \mathfrak{o}_\mathcal{L})$ and $v' = \left(\text{GRASP}, \text{loc}', o', \mathfrak{o}_\mathcal{L}\right)$. This edge exists as long as $\text{loc} = \text{loc}' \neq \text{loc}_{\text{inter}}$. This rule is similar to rule 3. This means that the robot can only grasp an object if the end-effector is in one of the locations of interest.

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

3.3.3 Deterministic Finite Automaton

After constructing the abstraction graph $\mathcal{R}$, the formula φ is converted into a DFA $\mathcal{A}_\varphi$ that specifies all the ways the robot can execute the task while fulfilling the formula φ (He et al., 2015). First, a Büchi automaton that accepts exactly the infinite words that satisfy φ is constructed (Vardi and Wolper, 1994). If the words accepted are finite, it becomes a finite automaton (Baier and Katoen, 2008, ch. 4). Finally, if the automaton is nondeterministic, there is a DFA that accepts the same words (Vardi, 1996). See the works of Kupferman and Y. Vardi (2001); Vardi and Wolper (1994); Vardi (1996) for more details about the conversion from φ into $\mathcal{A}_\varphi$.

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

For example, consider the automaton in Figure 3.4,[4] with $\mathbb{A} = \{p_0, p_1, p_2\}$, where $p_i = (o, l_i)$ and $i \in \{0, 1, 2\}$. Therefore, $\Sigma = \{\varnothing, \{p_0\}, \{p_1\}, \{p_2\}, \{p_0, p_1\}, \{p_1, p_2\}, \{p_0, p_2\}, \{p_0, p_1, p_2\}\}$. Given $A_i = \{p_i\}$, an example of an accepted word is $\sigma = A_0 A_1 A_2$. The letter $A_0 = \{p_0\}$ means that object o will be placed at a location with label l_0, which causes the transition from state 3 to state 2 because the letter $\{p_0\}$ is equivalent to the logical condition $p_0 \wedge \neg p_1 \wedge \neg p_2$. Similarly, letters A_1 and A_2 cause the transitions to state 1 and 0, respectively. Again, the letter $\{p_1\}$ is equivalent to the logical condition $\neg p_0 \wedge p_1 \wedge \neg p_2$ and $\{p_2\}$ is equivalent to $\neg p_0 \wedge \neg p_1 \wedge p_2$. Therefore, the object o will be in locations with labels l_0, l_1, and l_2 in this sequence, but not simultaneously.

Nonetheless, there may be words or paths that do not respect the physical world. For instance, one transition may require that an object be at multiple locations at the same time, that is, $(o, l_j) \wedge (o, l_k)$. This is exemplified in Figure 3.4 by the one-letter word $\sigma = \{p_0, p_1, p_2\}$, which causes the transition from state 3 to the final state 0. This

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

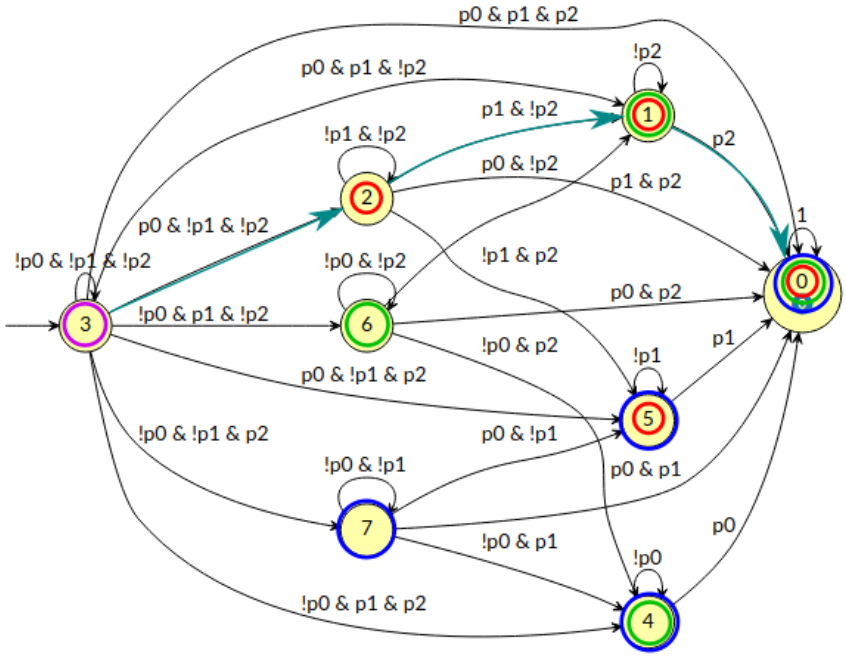


Figure 3.4: Automaton $\mathcal{A}_\varphi$ obtained from LTL specification $\varphi = \mathcal{E}p_0 \wedge \mathcal{E}p_1 \wedge \mathcal{E}p2$. State 0 is the final state. In this diagram the negation operator is given by ! and the Boolean operator "and" is given by &. States marked *red*, *green*, *blue*, and *magenta* represents the object in locations with labels l_0, l_1, and l_2 and l_{inter}, respectively. The *cyan* arrows represents an accepted path for an accepted word $\sigma = A_0A_1A_2$. States marked with multiple colors mean that the object can be at one of the locations indicated. For instance, in state 1 the object may be in location with label l_0 or l_1 depending on the path taken.

transition means that the object will be simultaneously in locations with labels l_0, l_1, and l_2.

3.3.4 Product Graph

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

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

where

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

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

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

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

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

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

3.3.5 Searching for a Path in the Product Graph

Since there may be more than one solution in $\mathcal{P}$, the Dijkstra's algorithm is used to search for the shortest accepting path on the product graph $\mathcal{P}$, that is, the shortest path from an initial node to an accepting one. The path found by Dijkstra's algorithm will have a minimum total edge weight. This path is passed to the low-level motion planner to plan the necessary motions.

3.3.6 Coordinating Layer and Low-level Motion Planner

The path found by the high-level planner is sent to the coordinating layer that breaks the path into the motion primitives. The GRASP and PLACE actions are precomputed and mean to close and open the end-effector gripper, respectively. In the original framework, the HOLD and MOVE actions initial and goal locations are mapped to configurations and are sent to the low-level motion planner, which solves the continuous motion planning query. In the work of He et al. (2015), sampling-based motion planning techniques are used to implement the paths required (Sucan and Kavraki, 2012; Kuffner and LaValle, 2000).

The coordinating layer computes the path with the help of the low-level motion planner. However, the low-level planning query may fail within the given time (He et al., 2015). This may happen due to very complex obstacles in the environment not allowing a possible solution. It may also fail because of the high dimensionality (generally 6 or higher) of the manipulation planning problem. The presence of narrow passages may also require more planning time than is given to the low-level planner, which may also be a cause for failure. In the case of failure, the coordinating layer asks the high-level planner to generate a new path. Moreover, it also updates the edge weights of the failed path on the product graph proportional to the planning time used. Hence, in the next search for a path on the product graph, the failed path will be avoided, since it has a greater total edge weight. With this approach, there is a synergy between the layers resulting in the generation of viable continuous trajectories for the robot. Last, the successful motion plan is passed to the motion controller to be executed.

3.4 A New Approach for the Planning Framework

In this work, differently from the work of He et al. (2015), the high-level plan is executed by a constrained motion controller (Marinho et al., 2019) (see Figure 3.1b). This way, we remove the need for a motion planning layer and a coordinating layer. Note that without these two layers, the edge weights in the product graph are kept constant and equal to one. Although we have used Dijkstra's algorithm to search for a path on the product graph, breadth first search can also be used since it finds optimal paths with relation to the number of edges. This approach and its advantages will be presented in the next chapter. Now, we take a closer look at the execution of the manipulation actions.

3.5 Execution of Manipulation Actions

The main objective of the constrained motion controller is to execute the task plan actions generated by the high-level planner. The activation of constraints depends on the action

to be performed. Recall from Section 3.3.2 that the robot actions are GRASP, HOLD, PLACE, and MOVE. The grasp and place actions are executed by closing and opening the gripper, respectively. The hold and move actions are executed by the motion controller. The only difference between both is that during hold, the robot is holding an object from one location to another, and during move, the robot is moving the empty end-effector from one location to another. Also, recall from Section 3.3.2 that for the robot to move between locations, it has to pass through the intermediate location (see Figure 3.3). Moreover, the task plan actions will always follow the sequence of nodes in the motion graph (see Figure 3.2). To execute the task plan, we iterate through the plan nodes. However, to correctly pick-and-place objects between locations, we break down the move, grasp, hold, and place actions into more steps.

The transferring of an object between two locations starts with a node that has an action move and end-effector location at the intermediate location. The next node will be a move node that is divided into two actions: pre-move and move. The pre-move action moves the end-effector to a location above the object on the object centerline and the move action makes the approach of the end-effector to the object location along the object centerline. This is done to always approach the object from the correct side and prevent undesirable displacements of the object since in this work the manipulable objects are cuboids. In the sequence, we have a grasp node that is split into grasp and post-grasp actions. The grasp action sends a command to close the gripper at the object's location and the post-grasp hold action makes the end-effector hold the object to a location above the target plane where the object currently is. Afterward, we have a hold node with intermediate location that makes the end-effector hold the object to the intermediate location. Next, there is a relaxed hold node that holds the object to the target plane. The relaxed hold allows placing the object anywhere within the relaxed target region. Lastly, we have a place node that is divided into place and post-place move. The place node sends a command to open the gripper and place the object on the target plane. The post-place move node moves the end-effector to a location above the object. Finally, the last node will be a move node to the intermediate location. From this point on, the process repeats itself until all the objects are on the desired locations. See Figure 3.5.

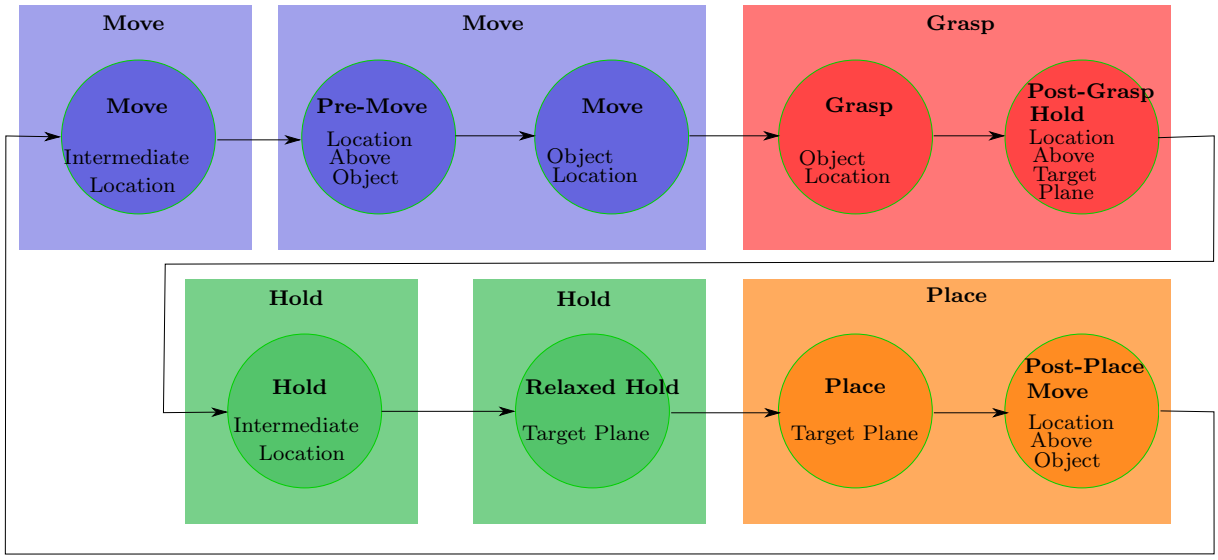


Figure 3.5: Task plan action sequence to transfer an object between two locations. The blue, red, green, and orange nodes represent the move, grasp, hold, and place nodes, respectively.

3.6 Conclusion

In this chapter, the manipulation planning framework of He et al. (2015) was presented in detail. It consists of a high-level planner, a coordinating layer, and a low-level motion planner. First, the high-level planner receives an LTL task specification, which is converted into a DFA. In the sequence, a rich abstraction is constructed, which represents all the ways the robot can manipulate objects in the workspace. Next, the abstraction and the DFA are combined into a product graph, which represents all the ways the robot can achieve the task respecting the LTL task specification. The Dijkstra's algorithm runs on the product graph and finds a path for the task plan, which is sent to a low-level sampling-based motion planner. Finally, the generated motion planner is sent to a motion controller, which executes it. In the case of a motion planning failure, the edge weights of that path in the product graph are increased, and it is searched for a new high-level plan. Because of the increased weights in the failed path, it is not chosen anymore. In this way, there is a synergy between high- and low-level planning, which generates only viable continuous trajectories for the robot. Afterward, we shortly mention a new approach for the planning framework that uses a constrained motion controller, which will be discussed on the next chapter. Last, we explain the relation between the task planner and the motion controller and how the task plan actions are executed.

4

Constrained Motion Controller

In Chapter 3, we have addressed the planning framework of He et al. (2015) and have ended the chapter mentioning that in this work we use a constrained motion controller instead of a motion planning layer. Moreover, we have given a problem statement considering the assistive robotics scenario. This chapter presents the controller and, afterward, defines suitable constraints for our assistive robotics application. In addition to well known constraints, it presents a new approach to define conic constraints. Next, we mention the available control objectives. Afterward, we summarize the constraints and control objectives used during each task plan action. Finally, we conclude the chapter with the main advantages of the constrained motion controller.

4.1 Constrained Motion Controller

The controller is based on an optimization problem that minimizes the joint velocities, $\dot{\boldsymbol{q}} \in \mathbb{R}^n$, in the ℓ_2-norm (i.e., the commonly used Euclidean norm) sense while respecting hard constraints, such as obstacles in the workspace, joints limits, etc. Given a desired task vector $\boldsymbol{x}_d \in \mathbb{R}^m$, where $\dot{\boldsymbol{x}}_d = \mathbf{0}$, $\forall\, t$, the task error $\tilde{\boldsymbol{x}} \triangleq \boldsymbol{x} - \boldsymbol{x}_d$, and a gain $\eta \in (0, \infty)$,

the control input $\boldsymbol{u}$ is obtained as (Marinho et al., 2019)

$$\begin{aligned} \boldsymbol{u} \in \underset{\dot{\boldsymbol{q}}}{\operatorname{argmin}} \quad & \left\| \boldsymbol{J}\dot{\boldsymbol{q}} + \eta\tilde{\boldsymbol{x}} \right\|_2^2 + \lambda \left\| \dot{\boldsymbol{q}} \right\|_2^2 \\ \text{subject to} \quad & \boldsymbol{W}\dot{\boldsymbol{q}} \leq \boldsymbol{w} \end{aligned} \tag{4.1}$$

where $\boldsymbol{J} \in \mathbb{R}^{m \times n}$ is the task Jacobian, $\lambda \in [0, \infty)$ is a damping factor and $\boldsymbol{W} \in \mathbb{R}^{l \times n}$ and $\boldsymbol{w} \in \mathbb{R}^l$ are used to impose linear constraints in the control inputs. In addition to describing poses, the task vector $\boldsymbol{x}_d$ can also be used to describe geometric primitives (points, lines, planes, ...) in the workspace. In this sense, the end-effector will converge to the specified desired task vector. In the case that the end-effector must follow a time varying task-vector (i.e. $\dot{\boldsymbol{x}}_d \neq \boldsymbol{0}$), a feedforward term can be added in the control law to compute the control input to track a trajectory (Adorno and Marinho, 2020)

$$\begin{aligned} \boldsymbol{u} \in \underset{\dot{\boldsymbol{q}}}{\operatorname{argmin}} \quad & \left\| \boldsymbol{J}\dot{\boldsymbol{q}} + \eta\tilde{\boldsymbol{x}} - \dot{\boldsymbol{x}}_d \right\|_2^2 + \lambda \left\| \dot{\boldsymbol{q}} \right\|_2^2 \\ \text{subject to} \quad & \boldsymbol{W}\dot{\boldsymbol{q}} \leq \boldsymbol{w} \end{aligned} \quad . \tag{4.2}$$

In order to prevent collisions with the workspace, we use the Vector Field Inequalities (VFI) framework (Marinho et al., 2019), which requires distance functions between two collidable entities and the corresponding Jacobian matrices.

The vector-field inequality for dynamic elements requires the following (Marinho et al., 2019):

1. A function $d \triangleq d(\boldsymbol{q}, t) \in \mathbb{R}$, where t is the time and $\boldsymbol{q} \in \mathbb{R}^n$ is the robot configuration vector, that encodes the (signed) distance between the two collidable entities. The robot entity is kinematically coupled to the robot, and the other entity, called the restricted zone, is part of the workspace (or part of another robot).

2. A Jacobian relating the time derivative of the distance function and the joint's velocities in the general form

$$\dot{d} = \underbrace{\frac{\partial(d(\boldsymbol{q}, t))}{\partial \boldsymbol{q}}}_{\boldsymbol{J}_d} \dot{\boldsymbol{q}} + \zeta(t), \tag{4.3}$$

   in which the residual $\zeta(t) = \dot{d} - \boldsymbol{J}_d\dot{\boldsymbol{q}}$ contains the distance dynamics unrelated to the joint's velocities. More specifically, the residual contains the components that cannot be controlled. Alternatively, $\zeta(t) = \frac{\partial(d(\boldsymbol{q},t))}{\partial t}$.

Since in this work we are not tracking or estimating the movement of the geometrical entities, the residual is $\zeta(t) = 0$ for all t.

To prevent collisions with the workspace, it is desired that the robot entity remains outside a restricted region. Hence, the signed distance $\tilde{d} \triangleq \tilde{d}(\boldsymbol{q}, t) = d - d_{\text{safe}}$ is defined, where $d_{\text{safe}} \in [0, \infty)$ is an arbitrary constant safe distance and d_{safe} represents the boundary of the restricted region (Marinho et al., 2019).

The restricted region Ω_R and safe region Ω_S are (Marinho et al., 2019):

$$\Omega_R \triangleq \{\boldsymbol{q} \in \mathbb{R}^n : \tilde{d}(\boldsymbol{q}, t) < 0, t \in [0, \infty)\}, \tag{4.4}$$

$$\Omega_S \triangleq \{\boldsymbol{q} \in \mathbb{R}^n : \tilde{d}(\boldsymbol{q}, t) \geq 0, t \in [0, \infty)\}. \tag{4.5}$$

Furthermore, the following inequalities must hold for all t (Marinho et al., 2019):

$$\dot{\tilde{d}} \geq -\eta_d \tilde{d} \iff -\boldsymbol{J}_d \dot{\boldsymbol{q}} \leq \eta_d \tilde{d}, \tag{4.6}$$

where $\eta_d \in [0, \infty)$ is used to determine the maximum approach velocity. The greater is η_d, the greater is the approach velocity.

Analogously, in order to make the robot entity remain inside a safe region, the d_{safe} is redefined to generate the signed distance $\tilde{d} = d_{\text{safe}} - d$, and the following inequalities must hold for all t (Marinho et al., 2019):

$$\boldsymbol{J}_d \dot{\boldsymbol{q}} \leq \eta_d \tilde{d}. \tag{4.7}$$

In this sense, by using the definition of restricted and safe regions, the constraints allow the definition of regions of interest.

4.2 Constraints

This section presents the constraints used in this work to prevent collisions with the workspace, define regions of interest, impose joint limits, and constrain the end-effector z-axis orientation.

4.2.1 Plane Constraints

Plane constraints can be used to prevent end-effector collisions with walls and objects in the workspace, but also to constrain the end-effector inside a region of interest. In this work, we use three plane constraints to prevent end-effector collisions with two walls and the table in the workspace (Figure 4.2a) and four planes to constrain the end-effector inside an inverted pyramid trunk region of interest as shown in Figure 4.1. Similarly to

the work of Quiroz-Omana and Adorno (2019), these seven constraints can be written as

$$\mathfrak{W}(\pi_i) : -\boldsymbol{J}_{p,n_{\underline{\boldsymbol{\pi}}_i}} \dot{\boldsymbol{q}} \leq \eta_\pi \tilde{d}_{p,n_{\underline{\boldsymbol{\pi}}_i}}, \tag{4.8}$$

where $i \in \{1, 2, \ldots, 7\}$ and $\tilde{d}_{p,n_{\underline{\boldsymbol{\pi}}_i}} = d_{p,n_{\underline{\boldsymbol{\pi}}_i}} - d_{\pi,\text{safe}}$, with $d_{\pi,\text{safe}}$ being the safe distance to each plane and $d_{p,n_{\underline{\boldsymbol{\pi}}_i}}$ and $\boldsymbol{J}_{p,n_{\underline{\boldsymbol{\pi}}_i}}$ is the Jacobian that satisfies $\dot{d}_{p,n_{\underline{\boldsymbol{\pi}}_i}} = \boldsymbol{J}_{p,n_{\underline{\boldsymbol{\pi}}_i}} \dot{\boldsymbol{q}}$ as shown in (4.3) (Marinho et al., 2019).

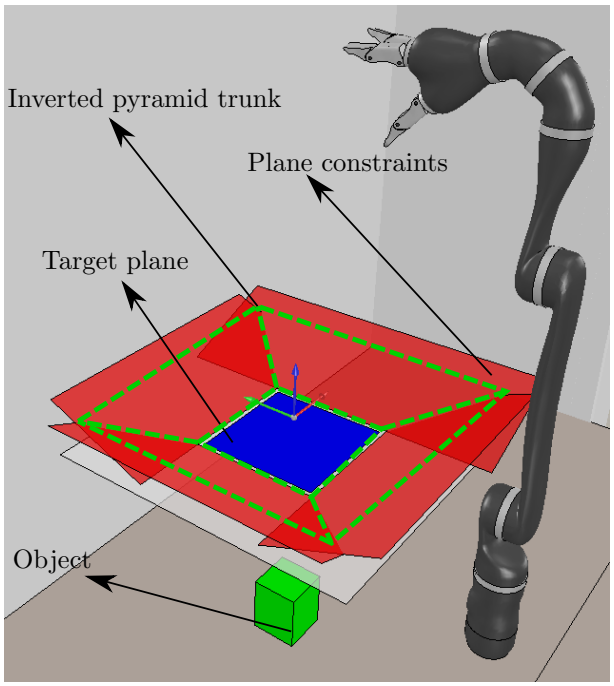


Figure 4.1: Region of interest composed of an inverted pyramid trunk. For the sake of clarity, the infinite *red* and *blue* planes that define the region of interest were truncated.

4.2.2 Cylindrical Constraints

To prevent end-effector collisions with non-manipulated objects while a given object is being manipulated, we add semi-infinite cylindrical constraints to each non-manipulated object (see Figure 4.2). Therefore, each object is constrained by a cylinder cut by a plane. Given k objects, to each one of them is associated a cylindrical and a plane constraint, which yields the following inequalities

$$\mathfrak{W}(c_j) : -\begin{bmatrix} \boldsymbol{J}_{\text{semi},j} \\ \boldsymbol{J}_{p,n_{\underline{\boldsymbol{\pi}}_j}} \end{bmatrix} \dot{\boldsymbol{q}} \leq \begin{bmatrix} \eta_l \tilde{D}_{p,\text{object}_j} \\ \eta_\pi d_{p,n_{\underline{\boldsymbol{\pi}}_j}} \end{bmatrix}, \tag{4.9}$$

where $j \in \{1, \ldots, k\}$, and $\tilde{D}_{p,\text{object}_j} = D_{p,\text{l}_j} - R_j^2$, with R_j being the radius of the cylinder around the j-th object; D_{p,l_j} and $\boldsymbol{J}_{\text{semi},j} \in \mathbb{R}^{1\times n}$, with n being the number of robot joints, are the point-static-line squared distance and its Jacobian, respectively (Marinho et al.,

2019), and

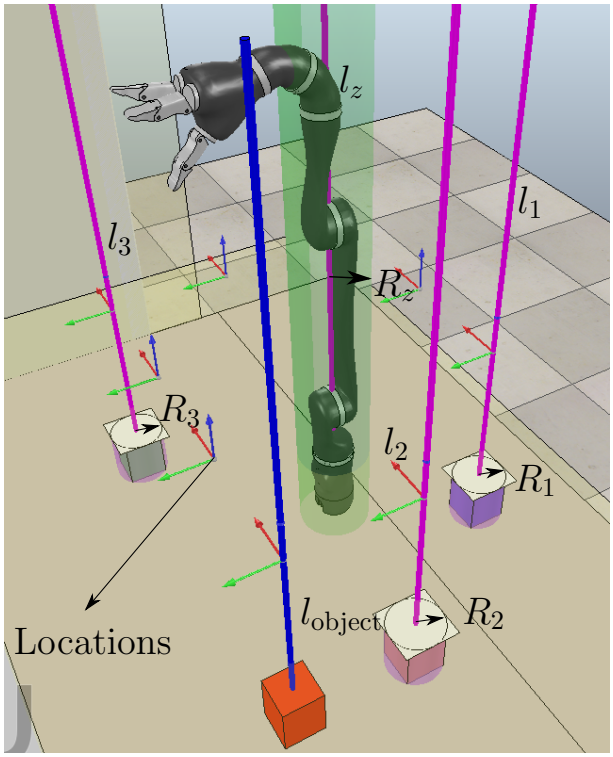
$$\boldsymbol{J}_{\text{semi},j} = \begin{cases} \boldsymbol{J}_{p,l_j} & \text{if } d_{p,n_{\underline{\pi}_j}} < 0, \\ \boldsymbol{0}^{1\times n} & \text{otherwise.} \end{cases} \tag{4.10}$$

The constraints allow any increase in the distance between the robot entity and the restricted region (Marinho et al., 2019). Nonetheless, in the case that the distance decreases, there is a decrease in the robot entity approach velocity to the restricted region. The robot entity can always approach with a slower velocity. When the signed distance becomes zero, the constraint does not allow increases in the approach velocity. Note that the constraints do not induce a repulsive motion in the robot entity (Marinho et al., 2019). In this sense, the cylindrical constraint $\mathfrak{W}(c_j)$ does not require a discontinuity in the plane constraint because the robot entity will always be able to approach the object without colliding with it. Moreover, in our work, the robot entity (i.e. the end-effector) will always approach the object from above and will not need to cross the object plane.

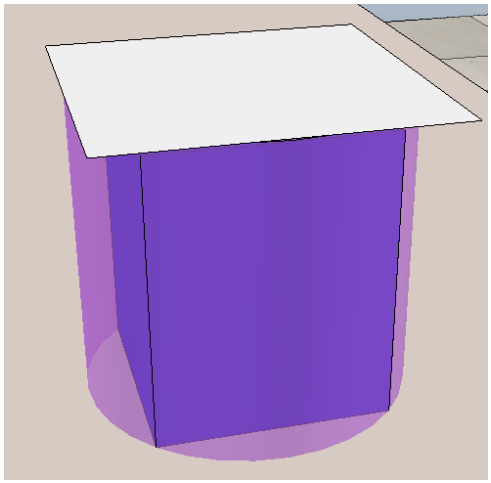
In addition to preventing end-effector collisions with non-manipulated objects, we also add an infinite cylindrical constraint collinear to the z-axis of the robot coordinate system to prevent twisted configurations caused by the end-effector passing over the robot (see Figure 4.2a). The constraint $\mathfrak{W}(l_z)$ is given by

$$\mathfrak{W}(l_z) : -\boldsymbol{J}_{p,l_z}\dot{\boldsymbol{q}} \leq \eta_l \tilde{D}_{p,l_z}, \tag{4.11}$$

where $\tilde{D}_{p,l_z} = D_{p,l_z} - R_z^2$, with R_z being the radius of the cylinder around the z-axis of the robot coordinate system; D_{p,l_z} and $\boldsymbol{J}_{p,l_z}$ are the point-static-line squared distance and its Jacobian, respectively (Marinho et al., 2019).



(a) The *red* cuboid with the *blue* centerline is the object to be manipulated, the other objects are already in their region of interest. The semi-infinite cylinders to prevent collisions with the non-manipulated objects are represented by the *light shaded purple* cylinder, with *pink* centerlines, around each object, and the *white* planes that cut each cylinder. The *light shaded yellow* planes are used to prevent collisions with the environment. The coordinate frames indicate locations in the scene. The *light shaded green* cylinder is the cylinder around the z-axis of the robot coordinate system.



(b) Cylindrical constraint combined with plane constraint. The *blue* cuboid represents the object, the *purple* cylinder indicates the cylindrical constraint, and the *white* plane cuts the cylinder.

Figure 4.2: Plane and cylindrical constraints

4.2.3 Line Constraints

The approach action of the end-effector to grasp an object is constrained to the object centerline that is parallel to the robot z-axis coordinate frame. Considering that the manipulable objects are cuboids, this is done to prevent the end-effector of approaching the object from the sideways causing unwanted collisions with the object. Given an object to be grasped, we associate a line constraint to the object, which yields the following equality constraint

$$\mathfrak{W}(l_{\text{object}}) : \boldsymbol{J}_{p,l_{\text{object}}}\dot{\boldsymbol{q}} = 0, \tag{4.12}$$

where $\boldsymbol{J}_{p,l_{\text{object}}}$ is the point-static-line Jacobian. Figure 4.2a presents the line constraint to which the end-effector is constrained while moving towards the object to be manipulated.

4.2.4 Point-Cone Constraint: A New Approach to Define Conic Constraints

In Section 4.2.1, the end-effector is constrained to an inverted pyramid trunk, and four planes are used to define the squared target region. Now, consider a manipulation task in which the end-effector must converge to a circular target region on a plane. In order to accomplish that, we can define a cone cut by a plane with its centerline perpendicular to the plane as in Figure 4.3.

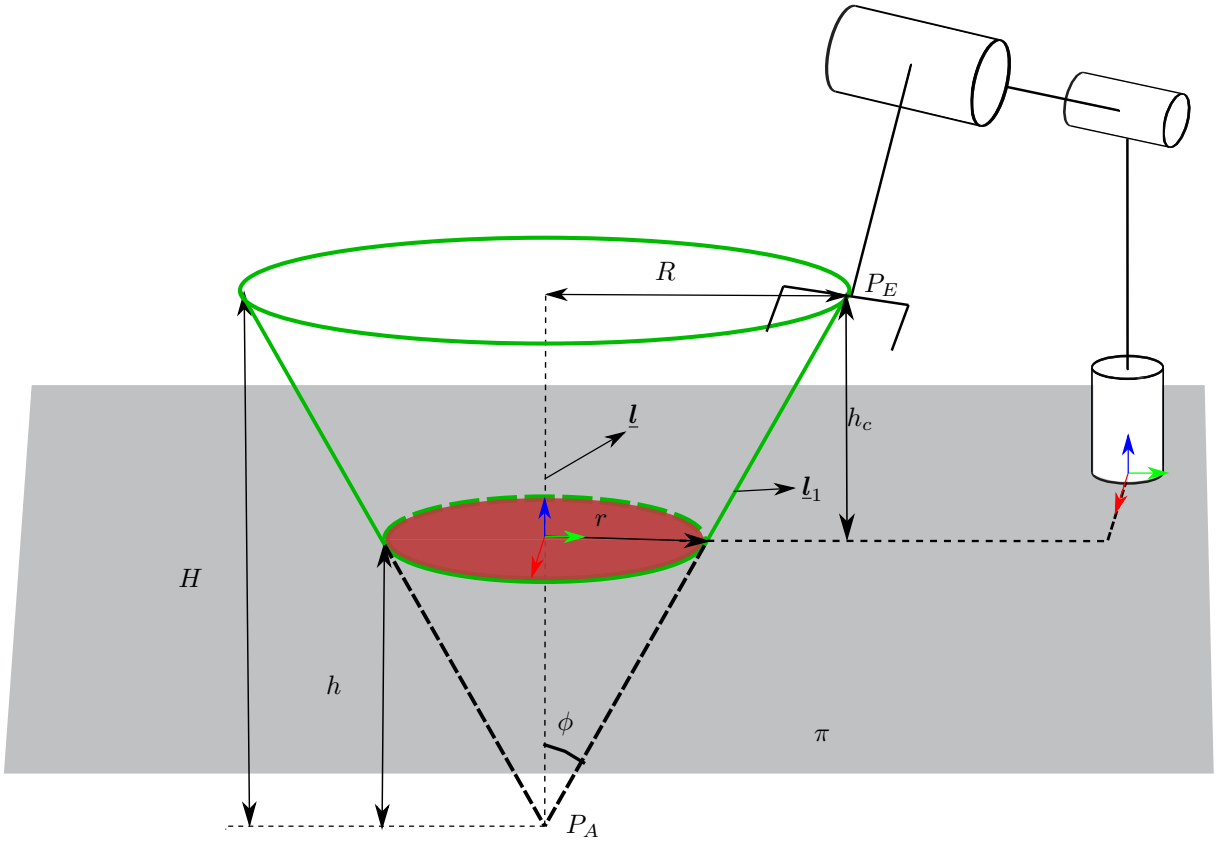


Figure 4.3: Point-cone constraint: the target circular region of interest is indicated in *red*, the *green* inverted cone trunk is the conical region of interest to which the end-effector is constrained.

First, we define the cone base radius R as the distance between the end-effector and the line $\underline{\boldsymbol{l}}$ that passes through the center of the cone. Then, the value of the target radius r of the lower circle on the plane is chosen as desired. We define H as the height of the whole cone. Next, we use the triangle relationship given by

$$\frac{R}{r} = \frac{H}{h} = \frac{h_c + h}{h} \tag{4.13}$$

to obtain the distance h between the plane π and the cone apex P_A as

$$h = \frac{h_c r}{R - r}, \tag{4.14}$$

where h_c is the distance between the end-effector and the plane π. Now, we can calculate the tangent of the angle ϕ between the cone center line $\underline{\boldsymbol{l}}$ and the line $\underline{\boldsymbol{l}}_1$ that connects the cone apex P_A to the end-effector P_E as

$$\tan\phi = \frac{R}{h_c + h}. \tag{4.15}$$

Up to now, everything is done only once.

Next, the first step is to update the value of the distance h_c between the plane and the end-effector. Afterward, the distance H' from the end-effector to the cone apex is obtained as $H' = h + h_c$. Now, the squared distance R'^2 between the end-effector and the cone is obtained as

$$R'^2 = (H'\tan\phi)^2. \tag{4.16}$$

Finally, we calculate the squared distance D^2 between the end-effector and the centerline $\underline{\boldsymbol{l}}$ to use it in the calculation of

$$\tilde{D}_l = R'^2 - D^2, \tag{4.17}$$

which is used to define the point-cone constraint given by

$$\dot{\tilde{D}}_l \geq -\eta_l \tilde{D}_l, \tag{4.18}$$

where

$$\dot{\tilde{D}}_l = \dot{R}'^2 - \dot{D}^2. \tag{4.19}$$

The derivative $\dot{D}^2$ with respect to time is given by $\dot{D}^2 = \boldsymbol{J}_{p,l}\dot{\boldsymbol{q}}$ (Marinho et al., 2019). By considering that $\dot{R}' \neq 0$, we have

$$\dot{\tilde{D}}_l = \underbrace{\frac{d}{dt}(R'^2)}_{\zeta_{R'}} - \boldsymbol{J}_{p,l}\dot{\boldsymbol{q}}. \tag{4.20}$$

Thus, we obtain

$$-\boldsymbol{J}_{p,l}\dot{\boldsymbol{q}} + \zeta_{R'} \geq -\eta_l \tilde{D}_l \Rightarrow \boldsymbol{J}_{p,l}\dot{\boldsymbol{q}} \leq \eta_l \tilde{D}_l + \zeta_{R'}. \tag{4.21}$$

To calculate $\zeta_{R'} = \frac{d}{dt}(R'^2)$, recall that $R'^2 = (H'\tan\phi)^2 = \Big((h + h_c)\tan\phi\Big)^2$. To use the same notation of Marinho et al. (2019), we define $d^{\pi}_{\boldsymbol{p},\pi} \triangleq h_c$. Hence, $\zeta_{R'}$ is given by

$$\zeta_{R'} = 2\tan^2\phi(h + d^{\pi}_{\boldsymbol{p},\pi})\dot{d}^{\pi}_{\boldsymbol{p},\pi}. \tag{4.22}$$

Since $\dot{d}^{\pi}_{\boldsymbol{p},\pi} = \boldsymbol{J}_{\boldsymbol{p},\pi}\dot{\boldsymbol{q}}$ (Marinho et al., 2019) and applying 4.22 into 4.21 we obtain the

point-cone constraint

$$\mathfrak{W}(l_{\text{cone}}^{\text{point}}) : \underbrace{\left[\boldsymbol{J}_{p,l} - 2\tan^2\phi(h + d_{\boldsymbol{p},\pi}^{\pi})\boldsymbol{J}_{\boldsymbol{p},\pi}\right]}_{\boldsymbol{J}_{\text{cone},\boldsymbol{p}}}\dot{\boldsymbol{q}} \leq \eta_l \tilde{D}_l. \tag{4.23}$$

In comparison to the plane constraints used to define a squared region of interest in Section 4.2.1, this approach of the point-cone constraint requires less complex calculations during running time. During running time, only (4.16), (4.17) and $\boldsymbol{J}_{\text{cone},\boldsymbol{p}}$ are necessary. Meanwhile, in the case of using four planes to define a squared region of interest as in Section 4.2.1, it is necessary to calculate four Jacobian matrices and four point-static-plane distances. Last, four plane constraints require four inequalities in the control law, while the point-cone constraint requires only one.

4.2.5 Additional Constraints

To prevent saturation of actuators we add joints velocities constraints $\mathfrak{W}(\dot{\boldsymbol{q}})$ (Quiroz-Omaña and Adorno, 2018, Section 4.1.2). Lastly, since we will be manipulating objects that must not suffer abrupt rotations along their axis, we add line-cone constraints $\mathfrak{W}(l_{z,\text{cone}})$ to constrain the end-effector z-axis within a cone (Quiroz-Omana and Adorno, 2019).

4.3 Control Objective

The constrained motion controller (4.1) presented in Section 4.1 allows to choose between different control objectives such as the control of end-effector pose, orientation, or position, as well as primitives attached to the end-effector, such as lines and planes, in addition to distance control from target regions such as planes and cylinders (Adorno and Marinho, 2020). By choosing a control objective, the appropriate Jacobian $\boldsymbol{J}$ and task vector $\boldsymbol{x} \in \mathbb{R}^m$ are calculated for the current joint positions $\boldsymbol{q} \in \mathbb{R}^n$. Afterward, the control signal is computed to regulate the closed-loop system to a set-point given by the task reference $\boldsymbol{x}_d \in \mathbb{R}^m$, which depends on the control objective (Adorno and Marinho, 2020).

In this work, one of the tasks that the robot must perform is to grasp objects with specific poses. Hence, the control objective is set to pose control. More specifically, the task vector $\boldsymbol{x}$ is the end-effector pose $\boldsymbol{x}_e^R$ with respect to the robot and the desired task vector $\boldsymbol{x}_d \in \mathbb{R}^m$ is the grasp pose $\boldsymbol{x}_g^R$ with respect to the robot (see Figure (4.4)) and $\boldsymbol{J} \in \mathbb{R}^{m \times n}$ is the pose Jacobian (Adorno and Marinho, 2020). The control input is given by (4.1), where $\boldsymbol{W} \in \mathbb{R}^{l \times n}$ and $\boldsymbol{w} \in \mathbb{R}^l$ are constructed by stacking the required constraint matrices for each task plan action (see Table 4.1).

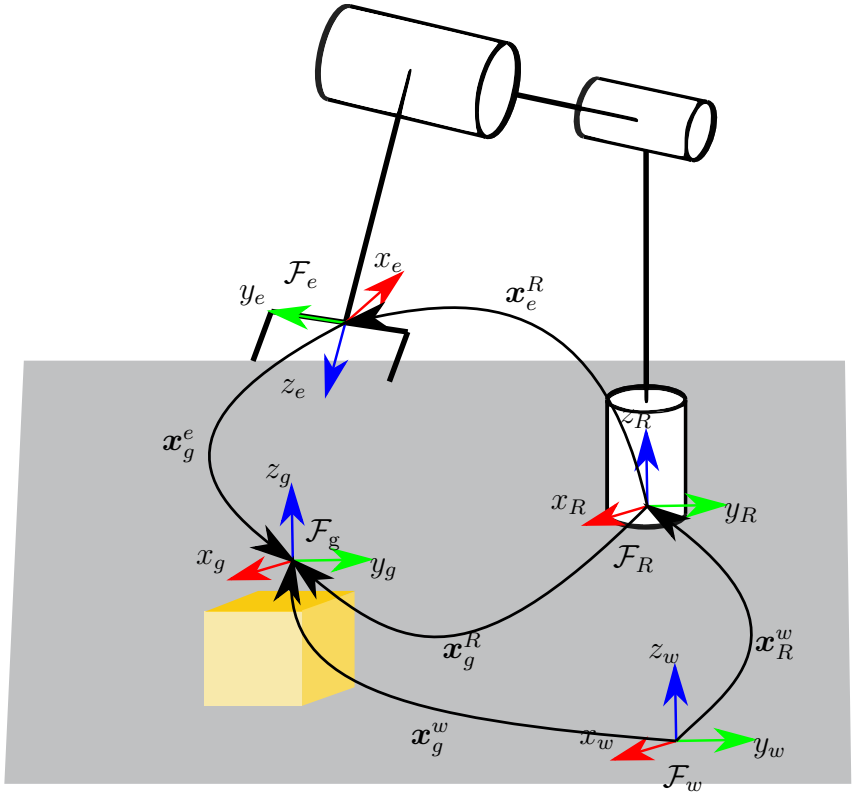


Figure 4.4: Pose control of the end-effector. The coordinate frames $\mathcal{F}_\text{g}$, $\mathcal{F}_e$, $\mathcal{F}_R$, and $\mathcal{F}_w$ indicates the frames of the grasp pose, the end-effector, the robot, and the world, respectively. The desired task vector is the grasp pose with respect to the robot given by $\boldsymbol{x}_d = \boldsymbol{x}_g^R$.

Recall from Section 2.3 that some tasks may be relaxed to a region. Instead of placing an object with a specific pose, it may be placed at a target region at a target plane. To do this, only one degree of freedom is required. Hence, there are more degrees of freedom available to satisfy additional constraints. In this context, the other task that the robot must perform is to move the end-effector towards a target plane. In this case, the control objective is set to distance to plane control. Thus, the task vector $\boldsymbol{x}$ is the distance $d_{\text{eff},n_{\underline{\boldsymbol{\pi}}_i}}$ between the end-effector and the target plane $\underline{\boldsymbol{\pi}}$ (see Figure (4.5)). Moreover, the desired task vector $\boldsymbol{x}_d \in \mathbb{R}^m$ is the desired distance d_desired between the end-effector and the target plane $\underline{\boldsymbol{\pi}}$. Thus, similarly to (4.1), the distance to plane control input $\boldsymbol{u}$ is given by

$$\begin{aligned} \boldsymbol{u} \in \text{argmin} \quad & \left\| \boldsymbol{J}\dot{\boldsymbol{q}} + \eta \tilde{d}_{\text{eff},n_{\underline{\boldsymbol{\pi}}}} \right\|^2 + \lambda \left\| \dot{\boldsymbol{q}} \right\|_2^2 \\ \text{subject to} \quad & \boldsymbol{W}\dot{\boldsymbol{q}} \leq \boldsymbol{w} \end{aligned} \quad , \tag{4.24}$$

where $\tilde{d}_{\text{eff},n_{\underline{\boldsymbol{\pi}}}} = d_{\text{eff},n_{\underline{\boldsymbol{\pi}}_i}} - d_\text{desired}$ and $\boldsymbol{J}$ is the Jacobian that satisfies $\dot{d}_{\text{eff},n_{\underline{\boldsymbol{\pi}}_i}} = \boldsymbol{J}_{\text{eff},n_{\underline{\boldsymbol{\pi}}_i}}\dot{\boldsymbol{q}}$ as shown in (4.3) (Marinho et al., 2019). Since we want the end-effector to move to the target plane, we set $d_\text{desired} = 0$. As in the case of pose control, $\boldsymbol{W} \in \mathbb{R}^{l \times n}$ and $\boldsymbol{w} \in \mathbb{R}^l$ are constructed by stacking the required constraint matrices for each task plan action (see

Table 4.1).

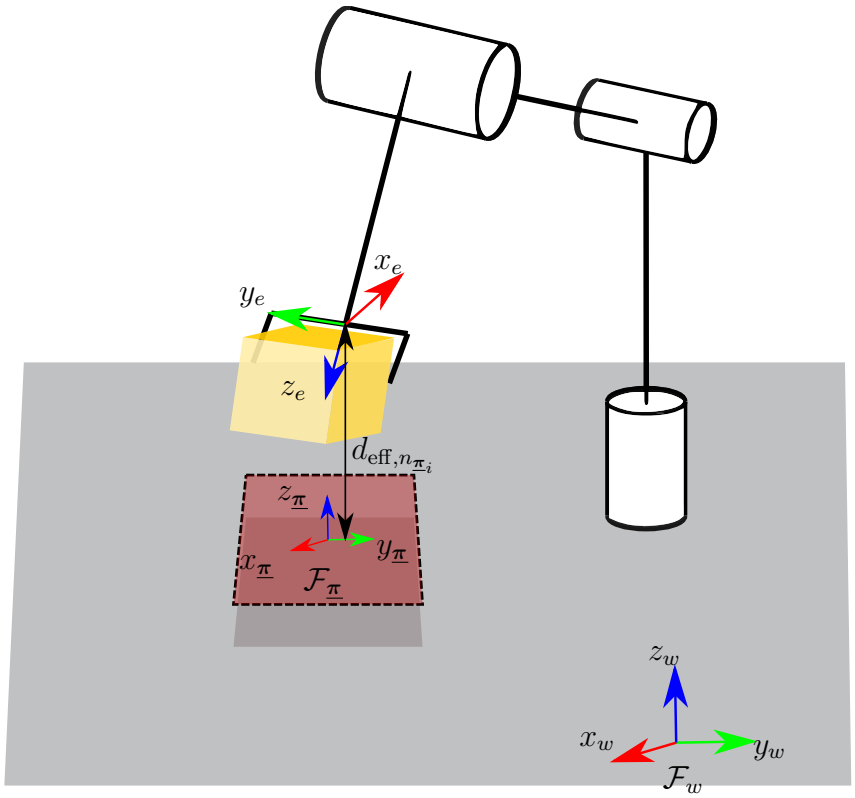


Figure 4.5: Control of distance to target plane. The distance $d_{\text{eff},n_{\underline{\pi}_i}}$ is the distance between the end-effector and the *red* target plane.

The constraints that are activated during each action of Figure (3.5) and the control objective used in this work are summarized in Table 4.1.

Action	**Constraints**	**Control Objective**
Pre-move	$\mathfrak{W}(\dot{\boldsymbol{q}})$, $\mathfrak{W}(\pi_1)$, $\mathfrak{W}(\pi_2)$, $\mathfrak{W}(\pi_3)$, $\mathfrak{W}(c_j)$ for $j \in \{1,\ldots,k\}$, $\mathfrak{W}(l_z)$, $\mathfrak{W}(l_{z,\text{cone}})$.	Pose
Move	$\mathfrak{W}(\dot{\boldsymbol{q}})$, $\mathfrak{W}(\pi_1)$, $\mathfrak{W}(\pi_2)$, $\mathfrak{W}(\pi_3)$, $\mathfrak{W}(c_j)$ for $j \in \{1,\ldots,k\}$, $\mathfrak{W}(l_{\text{object}})$,$\mathfrak{W}(l_z)$, $\mathfrak{W}(l_{z,\text{cone}})$.	Pose
Grasp	—	—
Post-grasp hold	$\mathfrak{W}(\dot{\boldsymbol{q}})$, $\mathfrak{W}(\pi_1)$, $\mathfrak{W}(\pi_2)$, $\mathfrak{W}(\pi_3)$, $\mathfrak{W}(c_j)$ for $j \in \{1,\ldots,k\}$ and $j \neq o_{\text{gripper}}$, $\mathfrak{W}(l_z)$, $\mathfrak{W}(l_{z,\text{cone}})$.	Pose
Hold	$\mathfrak{W}(\dot{\boldsymbol{q}})$, $\mathfrak{W}(\pi_1)$, $\mathfrak{W}(\pi_2)$, $\mathfrak{W}(\pi_3)$, $\mathfrak{W}(c_j)$ for $j \in \{1,\ldots,k\}$ and $j \neq o_{\text{gripper}}$, $\mathfrak{W}(l_z)$, $\mathfrak{W}(l_{z,\text{cone}})$.	Pose
Relaxed hold (Plane constraints)	$\mathfrak{W}(\dot{\boldsymbol{q}})$, $\mathfrak{W}(\pi_i)$ for $i \in \{1,\ldots,7\}$, $\mathfrak{W}(c_j)$ for $j \in \{1,\ldots,k\}$ and $j \neq o_{\text{gripper}}$, $\mathfrak{W}(l_z)$, $\mathfrak{W}(l_{z,\text{cone}})$.	Distance to target plane
Relaxed hold (Point-cone constraint)	$\mathfrak{W}(\dot{\boldsymbol{q}})$, $\mathfrak{W}(\pi_1)$, $\mathfrak{W}(\pi_2)$, $\mathfrak{W}(\pi_3)$, $\mathfrak{W}(l_{\text{cone}}^{\text{point}})$, $\mathfrak{W}(c_j)$ for $j \in \{1,...,k\}$ and $j \neq o_{\text{gripper}}$, $\mathfrak{W}(l_z)$, $\mathfrak{W}(l_{z,\text{cone}})$.	Distance to target plane
Place	—	—
Post-place move	$\mathfrak{W}(\dot{\boldsymbol{q}})$, $\mathfrak{W}(\pi_1)$, $\mathfrak{W}(\pi_2)$, $\mathfrak{W}(\pi_3)$, $\mathfrak{W}(c_j)$ for $j \in \{1,\ldots,k\}$, $\mathfrak{W}(l_z)$, $\mathfrak{W}(l_{z,\text{cone}})$.	Pose

Table 4.1: Constraints activation and control objective during each task plan action. The o_{gripper} is the number of the object currently in gripper.

4.4 Conclusion

In this chapter, we presented the constrained motion controller used to execute the high-level plans generated by the high-level planner of the framework of He et al. (2015). The controller is based on mathematical programming and uses the vector field inequalities (VFI) framework of Marinho et al. (2019) to define the constraints. As a result, the system becomes reactive, there is no need for a motion planning layer and it becomes possible to define regions of interest. In Chapter 5, this advantage will be exploited to do task relaxations and reduce the complexity of the planning framework. In Section 3.3.6, we have described the coordinating layer and its relation with the motion planning layer in the original framework. There is a need for more planning time for each motion planning query that fails. Therefore the motion planning layer increases the total planning time. In contrast, in the adapted framework, since the constrained motion controller executes the task plan and does not require planning or replanning phases, no motion planning time is added to the total planning time. With regard to the constraints used, we defined plane and cylindrical constraints and a new approach to define conic constraints. We also showed how the control inputs for pose and distance to plane control are defined and summarized which constraint and control objective is used in each task plan action. The plane and point-cone constraints defined in this chapter will be used to make task relaxations in the next chapter.

5 Task Relaxation

The planning framework of He et al. (2015) was designed for pick-and-place tasks that are described by multiple single locations. This means that each location is described by a pose in the robot workspace. However, some tasks may benefit from the definition of a region of interest—the relaxed task region—instead of a single location. First, we motivate the use of the constrained motion controller by analyzing the complexity issues related to relaxing tasks in the original planning framework of He et al. (2015). Afterward, we present the task action that is relaxed. The relaxed task constrains the end-effector to regions of interest, which are described using geometrical primitives represented by dual quaternions. Hence, we give a short introduction about dual quaternion algebra. Next, we give the definition of regions of interest and derive two regions that are used in this work.

5.1 Planning Complexity

The planning framework of He et al. (2015) depends, among other things, on the definition of the set of locations. Recall from Section (3.3.2) that the set of locations contains all the locations where objects can be in the environment. In this sense, the task space must be discretized into locations where objects can be placed. The number of locations is related to the size of the manipulation abstraction and, hence, to the computational complexity of the high-level planner. If the amount of places for objects is increased, the planning time will increase. Nonetheless, it may be interesting to define a region of interest for some tasks. One way to do this is to discretize the region into multiple locations and sample

one of the locations (see Figure 5.1). However, this would result in increasing the planning complexity, since there would be an increase in the number of locations.

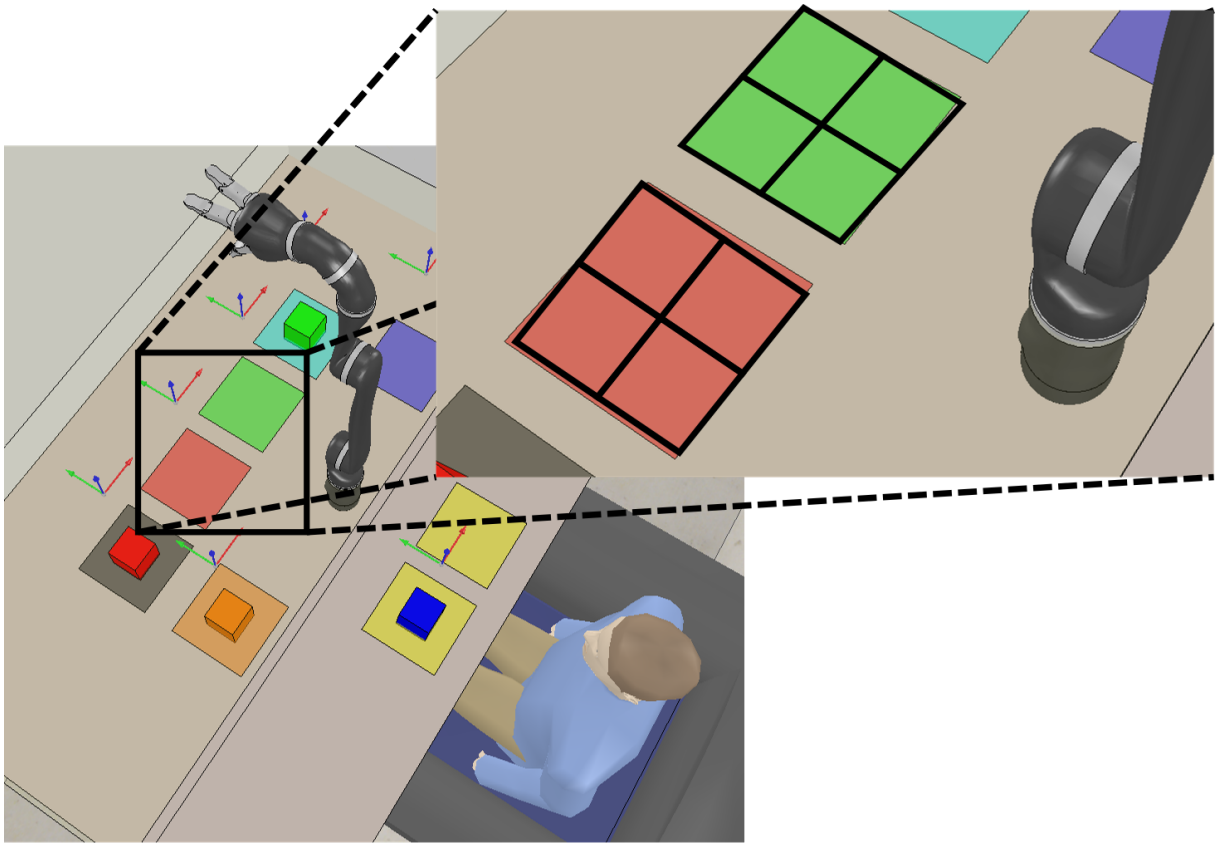


Figure 5.1: Discretization of regions in the workspace into multiple locations.

The number of valid nodes in the manipulation abstraction is given by $2(|\mathcal{L}|+1)P_{|O|}^{(|\mathcal{L}|+1)}$ (He et al., 2015), where $|\mathcal{L}|$ is the number of locations, $|O|$ is the number of objects and P_n^k is the k-permutation from n elements. Therefore, the size of the manipulation abstraction grows rapidly with the increase in the number of objects and locations. Hence, the discretization of the workspace in multiple locations does not scale well. To solve this issue, we adopt the constrained motion controller presented in Chapter 4, which allows the definition of a target region of interest without the need for discretization, and the system is reactive.

5.2 Task Relaxation

The planning framework proposed by He et al. (2015) was developed to solve the problem of manipulating objects between locations to satisfy a given specification. However, some tasks may be relaxed to a region, that is, the object may be placed anywhere within a target region on a plane. In this sense, instead of requiring pose control (i.e., position and orientation) during the whole task, it may be enough for the end-effector to move

towards a relaxed target region of a target plane while staying within a region of interest and satisfying additional constraints that prevent collisions with objects in the workspace and keep the end-effector with the correct orientation. This means that we only need to control the distance to the target plane and add constraints to define the region of interest and to satisfy additional requirements. From the control point of view, this is the same as relaxing the desired task vector to use fewer degrees of freedom (DOF). A possible solution is to switch between control objectives during runtime depending on the geometric task objective.

In the work of Figueredo et al. (2014), they propose to switch between desired task vectors and control laws according to offline defined geometric task objectives. The controller proposed by Figueredo et al. (2014) allows the use of different task vectors with different number of DOF. The goal of the switching control strategy is to enable the use of self-motion (i.e. movements of the links which do not disturb the end-effector location (Bedrossian, 1990)) to obtain better performances when specifying secondary tasks such as joint limits, obstacle avoidance, etc. To do this, complex tasks are divided into more simple tasks such that each one requires less DOF. Instead of specifying a complete pose configuration, bounded geometric regions are defined. In each region, a different task vector is used. Hence, there is more room left for self-motion during some steps of the task. This can be seen as a limitation of the task relaxation strategy of Figueredo et al. (2014). If self-motion for secondary tasks is needed during some steps of the task, the task may not be feasible. For instance, suppose that a task is divided into multiple steps such that the number of DOF decreases with each step. If the robot is not redundant and pose control is needed during any of these steps, all the available DOF will be used and there will be no room for secondary tasks such as obstacle avoidance. Hence, the task may be not physically feasible. Moreover, for feasible tasks, the switching strategy may cause error overshoots when changing from a less restrictive control to a more restrictive control. In addition to this, simple tasks such as the pick-and-place of objects in our manipulation planning framework would still require the definition of different geometric regions.

In our adapted planning framework, the transfer of an object between two locations is divided into the steps presented in Figure 3.5 and summarized in Table 4.1. In this sense, we propose the relaxation of the step that consists in holding an object from the intermediate location to a target plane. Instead of holding the object to a specific pose on the target plane, we do a relaxed holding action that holds the object to a target region on the target plane. This is done by controlling the end-effector distance to the target plane. This way, instead of using six DOF to do pose control, only one DOF is used to control the distance. The target region is obtained by using the plane or point-cone constraints presented in Section 4.2 to obtain a rectangular or circular target region, respectively. Table 5.1 recalls the actions used to pick-and-place an object between two locations, the control objective used, and the number of DOF required.

This work addressed the problem of relaxing the task of holding an object from an intermediate location to a target region on a target plane. However, by taking a closer look at each action of the pick-and-place sequence of actions, it can be seen that more steps could be relaxed. More specifically, the holding of an object toward an intermediate location could be relaxed to holding the object toward an intermediate region defined by a plane. In this case, controlling the distance to a plane control objective could be used again. Considering cuboid objects, the only action that could not be relaxed is the action of moving the end-effector toward an object location to pick the object since it is necessary to approach the object from specific directions to correctly grasp it. Table 5.1 summarizes the actions and the possibility of doing task relaxation.

Action	**Control Objective**	**Degrees of freedom**	**Can be relaxed?**
Pre-move	Pose	6	Yes
Move	Pose	6	No[1]
Grasp	—	—	—
Post-grasp hold	Pose	6	Yes
Hold	Pose	6	Yes
Relaxed hold (Plane constraints)	Distance to target plane	1	Relaxed
Relaxed hold (Point-cone constraint)	Distance to target plane	1	Relaxed
Place	—	—	—
Post-place move	Pose	6	Yes

Table 5.1: Task action, corresponding control objective, and number of degrees of freedom required.

The remainder of this chapter describes the regions of interest to which the end-effector is constrained while holding the object toward the rectangular or circular target region. To describe regions of interest, we make use of geometrical primitives that are represented by dual quaternions.

5.3 Dual Quaternion Algebra

Dual quaternions are dual numbers in which the primary and dual parts are quaternions. Dual numbers were introduced by Clifford in the nineteenth century, who proposed the dual unit ε to create a new algebra over the real numbers in which ε has the following properties: $\varepsilon \neq 0$, $\varepsilon^2 = 0$ (Adorno, 2017).

In a dual number $\underline{a} = a + \varepsilon a'$, a is the primary part and a' the dual part. Both parts are of the same type of elements. The usual operations of sum, subtraction, and multiplication

[1]This is true for cuboid objects, which is the case of this work.

consider the ε operator and are defined in the work of Adorno (2017).

Quaternions were invented by Hamilton in the nineteenth century and are an extension of the complex numbers (Adorno, 2017). They are elements of the set given by

$$\mathbb{H} \triangleq \{h_1 + \hat{\imath}h_2 + \hat{\jmath}h_3 + \hat{k}h_4 : h_1, h_2, h_3, h_4 \in \mathbb{R}\} \tag{5.1}$$

in which the imaginary units $\hat{\imath}$, $\hat{\jmath}$ and $\hat{k}$ have the following properties $\hat{\imath}^2 = \hat{\jmath}^2 = \hat{k}^2 = \hat{\imath}\hat{\jmath}\hat{k} = -1$. Given a general quaternion $\boldsymbol{h} = h_1 + \hat{\imath}h_2 + \hat{\jmath}h_3 + \hat{k}h_4$, its conjugate is defined as $\boldsymbol{h}^* \triangleq h_1 - h_2\hat{\imath} - h_3\hat{\jmath} - h_4\hat{k}$. The norm of a quaternion is $\|\boldsymbol{h}\| = \sqrt{\boldsymbol{h}\boldsymbol{h}^*}$. Moreover, the real part of $\boldsymbol{h}$ is obtained with $\mathrm{Re}\,(\boldsymbol{h})$ and contains the scalar h_1. The imaginary part $\mathrm{Im}\,(\boldsymbol{h})$ contains the imaginary components, that is, $\mathrm{Im}\,(\boldsymbol{h}) \triangleq \hat{\imath}h_2 + \hat{\jmath}h_3 + \hat{k}h_4$. As a result, $\boldsymbol{h} = \mathrm{Re}\,(\boldsymbol{h}) + \mathrm{Im}\,(\boldsymbol{h})$.

The set of pure quaternions $\mathbb{H}_p \triangleq \{\boldsymbol{h} \in \mathbb{H} : \mathrm{Re}\,(\boldsymbol{h}) = 0\}$ has a bijective relation with $\mathbb{R}^3$. Thus, the point $(x, y, z) \in \mathbb{R}^3$ can be represented by $\mathbf{p} = p_x\hat{\imath} + p_y\hat{\jmath} + p_z\hat{k}$. The set of quaternions with a unit norm is $\mathbb{S}^3 \triangleq \{\boldsymbol{h} \in \mathbb{H} : \|\boldsymbol{h}\| = 1\}$. They can be used to represent a rotation $\boldsymbol{r} \in \mathbb{S}^3$ given by $\boldsymbol{r} = \cos(\frac{\phi}{2}) + \boldsymbol{n}\sin(\frac{\phi}{2})$ of an angle $\phi \in \mathbb{R}$ around a rotation axis $\boldsymbol{n} \in \mathbb{S}^3 \cap \mathbb{H}_p$, such that $\boldsymbol{n} = n_x\hat{\imath} + n_y\hat{\jmath} + n_z\hat{k}$. Its conjugate is given by $\boldsymbol{r}^* = \cos(\frac{\phi}{2}) - \boldsymbol{n}\sin(\frac{\phi}{2})$ such that $\boldsymbol{r}\boldsymbol{r}^* = \boldsymbol{r}^*\boldsymbol{r} = 1$. Sequential rotations $\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_n$ are described by sequential quaternion multiplications $\boldsymbol{r}_1\boldsymbol{r}_2 \ldots \boldsymbol{r}_n$. Elements of the set of pure quaternions $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{H}_p$ can be used to calculate the inner product and cross product respectively (Adorno, 2017),

$$\langle \boldsymbol{a}, \boldsymbol{b} \rangle \triangleq -\frac{\boldsymbol{a}\boldsymbol{b} + \boldsymbol{b}\boldsymbol{a}}{2}, \boldsymbol{a} \times \boldsymbol{b} \triangleq \frac{\boldsymbol{a}\boldsymbol{b} - \boldsymbol{b}\boldsymbol{a}}{2}.$$

When the primary and dual parts of a dual number are quaternions, they are called dual quaternions (Adorno, 2017) and form the set

$$\mathcal{H} \triangleq \{\boldsymbol{h}_1 + \varepsilon\boldsymbol{h}_2 : \boldsymbol{h}_1, \boldsymbol{h}_2 \in \mathbb{H}, \varepsilon \neq 0, \varepsilon^2 = 0\}. \tag{5.2}$$

The conjugate of $\underline{\boldsymbol{h}} \in \mathcal{H}$ is $\underline{\boldsymbol{h}}^* = \mathrm{Re}\,(\underline{\boldsymbol{h}}) - \mathrm{Im}\,(\underline{\boldsymbol{h}})$ and its norm is defined as $\|\underline{\boldsymbol{h}}\| \triangleq \sqrt{\underline{\boldsymbol{h}}\underline{\boldsymbol{h}}^*} = \sqrt{\underline{\boldsymbol{h}}^*\underline{\boldsymbol{h}}}$. The set of pure dual quaternions is defined as $\mathcal{H}_p \triangleq \{\underline{\boldsymbol{h}} \in \mathcal{H} : \mathrm{Re}\,(\underline{\boldsymbol{h}}) = 0\}$.

A translation $\boldsymbol{p}$ can be combined with a rotation $\boldsymbol{r}$ to represent poses (position and orientation) and are elements of the set of unit dual quaternion $\underline{\boldsymbol{\mathcal{S}}} \triangleq \{\underline{\boldsymbol{h}} \in \mathcal{H} : \|\underline{\boldsymbol{h}}\| = 1\}$. A unit dual quaternion $\underline{\boldsymbol{x}} \in \underline{\boldsymbol{\mathcal{S}}}$ can always be written as $\underline{\boldsymbol{x}} = \boldsymbol{r} + \frac{1}{2}\varepsilon\boldsymbol{p}\boldsymbol{r}$ (Adorno, 2017). Last, the set $\underline{\boldsymbol{\mathcal{S}}}$ equipped with the multiplication operation forms the group $\mathrm{Spin}(3) \ltimes \mathbb{R}^3$, which double covers SE(3).

The operator $\mathrm{vec}_4 : \mathbb{H} \to \mathbb{R}^4$ is defined and maps a quaternion to a four-dimensional column vector. Moreover, a general dual quaternion is composed of eight elements $\underline{\boldsymbol{g}} = g_1 + g_2\hat{\imath} + g_3\hat{\jmath} + g_4\hat{k} + \varepsilon\,(g_5 + g_6\hat{\imath} + g_7\hat{\jmath} + g_8\hat{k})$. Analogously, the operator $\mathrm{vec}_8 : \mathcal{H} \to \mathbb{R}^8$ is used to map it into an eight-dimensional column real vector; i.e., $\mathrm{vec}_8\,\underline{\boldsymbol{g}} \triangleq [g_1\ g_2\ g_3\ g_4\ g_5\ g_6\ g_7\ g_8]^T$.

5.4 Regions of Interest

To do task relaxation, we need to define regions of interest. In this sense, this section covers the definition of a few regions of interest. In this work, regions of interest can be understood as areas or volumes in the three-dimensional euclidean space. Hence, we define them as follows:

Definition 5.1 (Region of interest in $\mathbb{R}^3$)**.** For the set of all points in $\mathbb{R}^3$, we have the following subset $I \subseteq \mathbb{R}^3$, where I is the region of interest.

Regions of interest I stem from the combination of geometrical primitives, half-planes, and half-spaces. Although regions of interest are usually represented in $\mathbb{R}^3$, we will define them using geometrical primitives represented by dual quaternions. In this sense, we will define them in the $\mathbb{H}_p$.

Definition 5.2 (Region of interest in $\mathbb{H}_p$)**.** Since there is an isomorphism $\{\mathbb{H}_p, +\} \cong \{\mathbb{R}^3, +\}$ (Figueredo C., 2016, ch. 2), the set can be used to represent vectors of $\mathbb{R}^3$ within the quaternion algebra. The isomorphism mapping between $\mathbb{H}_p$ and $\mathbb{R}^3$ is defined by $\text{vec}_3 : \mathbb{H}_p \to \mathbb{R}^3$, such that $\boldsymbol{h} = h_x\hat{\imath} + h_y\hat{\jmath} + h_z\hat{k}$ results in $\text{vec}_3\,\boldsymbol{h} = [h_x\, h_y\, h_z]^T$. The inverse mapping is defined by $\underline{\text{vec}}_3 : \mathbb{R}^3 \to \mathbb{H}_p$. Hence, for the set of all points in $\mathbb{H}_p$, we have the following subset $\mathcal{I} \subseteq \mathbb{H}_p$, where $\mathcal{I}$ is the region of interest.

5.4.1 Geometrical Primitives

Among the geometrical primitives, we shall use cylinders, planes, and half-spaces. By combining such primitives we can obtain other primitives. For instance, a point on a plane combined with a positive scalar yields a circle whereas a line combined with a positive scalar result in an infinite cylinder. A finite cylinder is obtained from the intersection of a cylinder with two planes. The intersection between planes forms polyhedra.

First, the definition of a line is given below.

Definition 5.3 (Plücker line (Adorno, 2017))**.** A Plücker line is an element of the set $\mathcal{H}_p \cap \underline{\boldsymbol{\mathcal{S}}}$ and is represented by the dual quaternion

$$\underline{\boldsymbol{l}} = \boldsymbol{l} + \varepsilon \boldsymbol{m}, \tag{5.3}$$

where $\boldsymbol{l} \in \mathbb{H}_p \cap \mathbb{S}^3$ represents the line direction and the line moment is given by $\boldsymbol{m} = \boldsymbol{p}_l \times \boldsymbol{l}$, in which $\boldsymbol{p}_l \in \mathbb{H}_p$ is an arbitrary point on the line.

A plane can be defined as follows.

Definition 5.4 (Plane (Adorno, 2017))**.** A Plane can be represented by the dual quaternion

$$\underline{\boldsymbol{\pi}} \triangleq \boldsymbol{n}_\pi + \varepsilon d_\pi, \tag{5.4}$$

where $\boldsymbol{n}_\pi \in \mathbb{H}_p \cap \mathbb{S}^3$ is the normal to the plane and $d_\pi \in \mathbb{R}$ is the signed perpendicular distance between the plane and the origin of the reference frame. Furthermore, given an arbitrary point $\boldsymbol{p}_\pi$ in the plane, the signed perpendicular distance is given by $d_\pi = \langle \boldsymbol{p}_\pi, \boldsymbol{n}_\pi \rangle$. Moreover, the set of all points on a plane is given by

$$P_\pi \triangleq \{\boldsymbol{p}_\pi \in \mathbb{H}_p : \langle \boldsymbol{p}_\pi, \boldsymbol{n}_\pi \rangle = d_\pi\}. \tag{5.5}$$

In addition to Plücker lines and planes, we shall also use the definition of half-spaces.

Definition 5.5 (Half-space)**.** The set of all points above a given plane $\underline{\boldsymbol{\pi}}$, i.e. they are on the side to which the normal is pointing, is given by

$$\mathcal{A}_\pi \triangleq \{\boldsymbol{p}_\pi \in \mathbb{H}_p : \langle \boldsymbol{p}_\pi, \boldsymbol{n}_\pi \rangle > d_\pi\} \tag{5.6}$$

and the set of all points below a given plane, i.e. they are on the opposite side to which the normal is pointing, is given by

$$\mathcal{B}_\pi \triangleq \{\boldsymbol{p}_\pi \in \mathbb{H}_p : \langle \boldsymbol{p}_\pi, \boldsymbol{n}_\pi \rangle < d_\pi\}. \tag{5.7}$$

As a result, the set of all points in space is given by

$$\mathbb{H}_p = \mathcal{A}_\pi \cup \mathcal{B}_\pi \cup P_\pi. \tag{5.8}$$

5.4.2 Distance Functions

In addition to the geometrical primitives, we also need the definition of distance functions between them to define more geometrical primitives such as cylinders, circles, and cones. The distance between two points in space is given below.

Definition 5.6 (Point-to-point distance (Marinho et al., 2019))**.** Since the quaternion norm is equivalent to the Euclidean norm, the Euclidean distance between two points $\boldsymbol{p}_1, \boldsymbol{p}_2 \in \mathbb{H}_p$ is given by

$$d_{\boldsymbol{p}_1, \boldsymbol{p}_2} = \|\boldsymbol{p}_1 - \boldsymbol{p}_2\|. \tag{5.9}$$

Now we present the squared distance between a point and a line in space.

Definition 5.7 (Point-to-line squared distance (Marinho et al., 2019))**.** Given a point $\boldsymbol{p} \in \mathbb{H}_p$ in the space, the squared distance between $\boldsymbol{p}$ and an arbitrary line $\underline{\boldsymbol{l}} = \boldsymbol{l} + \varepsilon \boldsymbol{p}_l \times \boldsymbol{l} \in \mathcal{H}_p \cap \underline{\boldsymbol{\mathcal{S}}}$ in space is given by

$$D_{\boldsymbol{p}, \underline{\boldsymbol{l}}} \triangleq \|\boldsymbol{p} \times \boldsymbol{l} - \boldsymbol{m}\|^2. \tag{5.10}$$

As a result, the point-to-line distance is given by

$$d_{\boldsymbol{p}, \underline{\boldsymbol{l}}} \triangleq \|\boldsymbol{p} \times \boldsymbol{l} - \boldsymbol{m}\|. \tag{5.11}$$

Finally, the distance between a point and a plane is given.

Definition 5.8 (Point-to-plane distance (Marinho et al., 2019))**.** Given a frame $\mathcal{F}_\pi$ attached to an arbitrary plane in space, the signed distance between a point $\boldsymbol{p} \in \mathbb{H}_p$ in the space and the plane $\underline{\boldsymbol{\pi}}$, from the point of view of the plane, is given by

$$d^{\pi}_{\boldsymbol{p},\pi} = \langle \boldsymbol{p}, \boldsymbol{n}_\pi \rangle - d_\pi. \tag{5.12}$$

5.4.3 More Geometrical Primitives

The definition of a line can be used to define infinite solid cylinders.

Definition 5.9 (Infinite cylinder)**.** An infinite cylinder is given by the set of all points within a given distance from a line

$$c(\underline{\boldsymbol{l}}, R) \triangleq \{\boldsymbol{p} \in \mathbb{H}_p : d_{\boldsymbol{p},\underline{\boldsymbol{l}}} \leq R\}, \tag{5.13}$$

where $R \in \mathbb{R}$ is the cylinder radius, $\underline{\boldsymbol{l}} \in \mathcal{H}_p \cap \underline{\boldsymbol{\mathcal{S}}}$ is an arbitrary line in space and $d_{\boldsymbol{p},\underline{\boldsymbol{l}}}$ is the point-to-line distance.

To define a cone, we will need the definition of line segments and a circle. First we define a line segment.

Definition 5.10 (Line segment)**.** A point $\boldsymbol{p}$ is on a line segment between points $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$, if the distance between $\boldsymbol{p}_1$ and $\boldsymbol{p}$ added to the distance between $\boldsymbol{p}$ and $\boldsymbol{p}_2$ is equal to the distance from $\boldsymbol{p}_1$ to $\boldsymbol{p}_2$. Therefore the set of all points on a line segment is given by

$$\zeta_{\boldsymbol{p}_1,\boldsymbol{p}_2} = \{\alpha \boldsymbol{p}_1 + (1-\alpha)\boldsymbol{p}_2 : \alpha \in [0,1]\}. \tag{5.14}$$

In addition to the definition of a line segment, we define a disk.

Definition 5.11 (Disk)**.** A disk with boundary is the set of all points on a plane within a given distance from a point

$$C(\boldsymbol{p}_\pi, R) \triangleq \{\boldsymbol{p} \in \mathbb{H}_p : \langle \boldsymbol{p}, \boldsymbol{n}_\pi \rangle = d_\pi \wedge d_{\boldsymbol{p},\boldsymbol{p}_\pi} \leq R\}, \tag{5.15}$$

where $R \in \mathbb{R}$ is the disk radius, $\boldsymbol{p}_\pi$ is the disk center, and $d_{\boldsymbol{p},\boldsymbol{p}_\pi}$ is the point-to-point distance. See Figure 5.2.

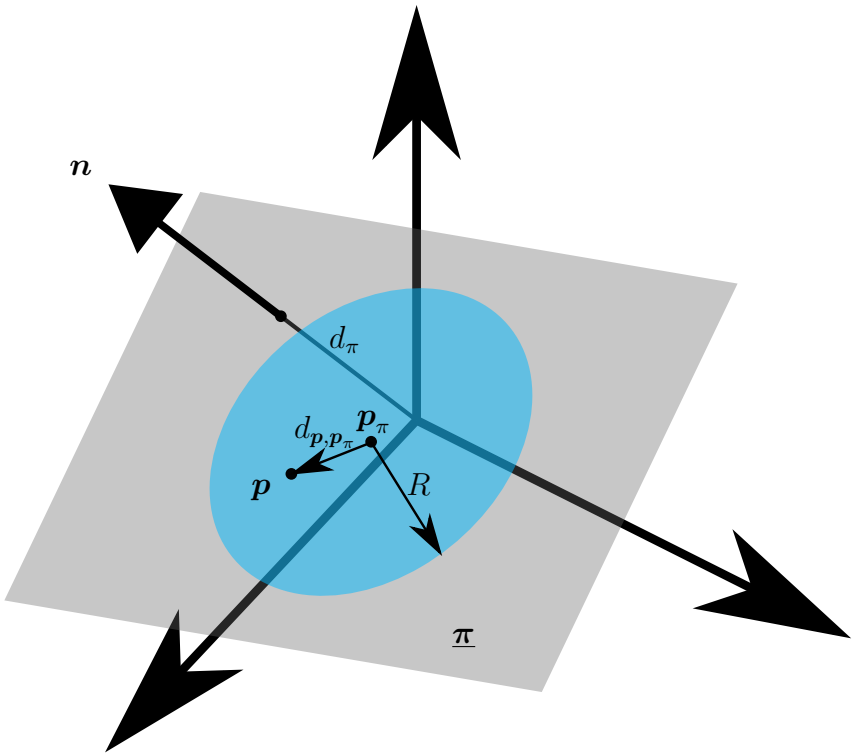


Figure 5.2: Disk $C(\boldsymbol{p}_\pi, R)$ with disk center $\boldsymbol{p}_\pi$ and radius R.

The definition of line segments and a disk can be used to define a right cone.

Definition 5.12 (Solid cone)**.** A cone is the union of all line segments connecting a common point, i.e. the apex, to all points on the cone base, i.e. a disk. Thus, the set of all points on the cone can be given by

$$\mathfrak{C}_{\boldsymbol{p}_A}^{C(\boldsymbol{p}_\pi,R)} \triangleq \bigcup_{\boldsymbol{p}_C \in C(\boldsymbol{p}_\pi,R)} \zeta_{\boldsymbol{p}_A,\boldsymbol{p}_C}, \tag{5.16}$$

where $\boldsymbol{p}_A \in \mathbb{H}_P$ is the cone apex, $C(\boldsymbol{p}_\pi, R)$ is the disk with center $\boldsymbol{p}_\pi$ and radius $R \in \mathbb{R}$, and $\boldsymbol{p}_C \in C(\boldsymbol{p}_\pi, R)$ is a point on the disk. See Figure 5.3.

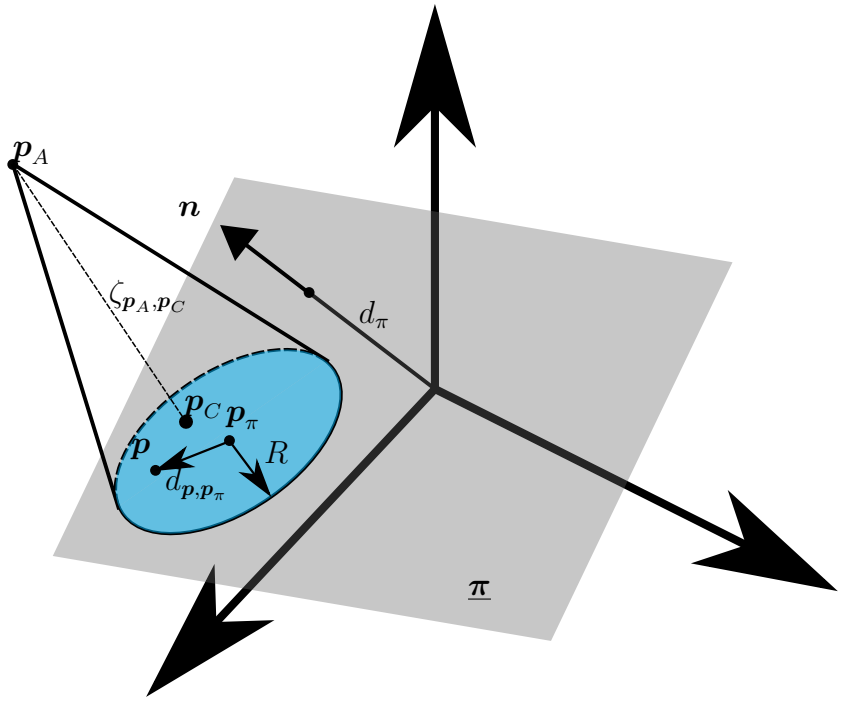


Figure 5.3: Cone $\mathfrak{C}_{\boldsymbol{p}_A}^{C(\boldsymbol{p}_\pi,R)}$ with apex $\boldsymbol{p}_A$ and cone base given by the disk $C(\boldsymbol{p}_\pi, R)$.

5.4.4 Inverted Pyramid Trunk Region of Interest

The inverted pyramid trunk region of interest $\mathcal{I}_Q$ constrained by planes in Section 4.2.1 can be defined as the intersection of the half-space above the target plane with the half-spaces that represent the points above the planes constraining the squared region. This can be done by using (5.6). Hence, we obtain

$$\mathcal{I}_Q = \mathcal{A}_{\pi_t} \cap \mathcal{A}_{\pi_1} \cap \mathcal{A}_{\pi_2} \cap \mathcal{A}_{\pi_3} \cap \mathcal{A}_{\pi_4}, \tag{5.17}$$

where $\mathcal{A}_{\pi_t}\mathcal{A}_{\pi_1}, \ldots, \mathcal{A}_{\pi_4}$ are the half-spaces formed by the planes constraining the region of interest. See Figure 5.4.

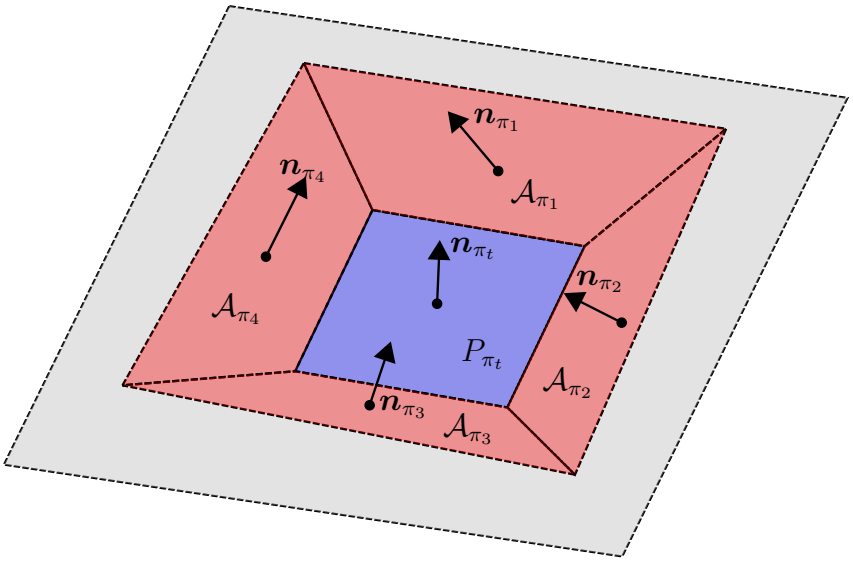


Figure 5.4: Inverted pyramid trunk region of interest $\mathcal{I}_Q$. For the sake of clarity, the infinite target plane P_{π_t} and the four half-spaces $\mathcal{A}_{\pi_1}, \ldots, \mathcal{A}_{\pi_4}$ were truncated.

5.4.5 Inverted Cone Trunk Region of Interest

The inverted cone trunk region of interest $\mathcal{I}_{\mathfrak{C}}$ constrained by the cone trunk and the target plane in Section 4.2.4 is given by the intersection of the target plane P_{π_t} parallel to the cone base plane P_{π_C} with the cone. Thus, we have

$$\mathcal{I}_{\mathfrak{C}} = \mathcal{A}_{\pi_t} \cap \mathfrak{C}_{\boldsymbol{p}_A}^{C(\boldsymbol{p}_{\pi_C}, R)}, \tag{5.18}$$

where $\mathcal{A}_{\pi_t}$ is the set of points of the half-space above the target plane to where the end-effector must move and $\mathfrak{C}_{\boldsymbol{p}_A}^{C(\boldsymbol{p}_{\pi_C}, R)}$ is the set of points within the cone. See Figure 5.5.

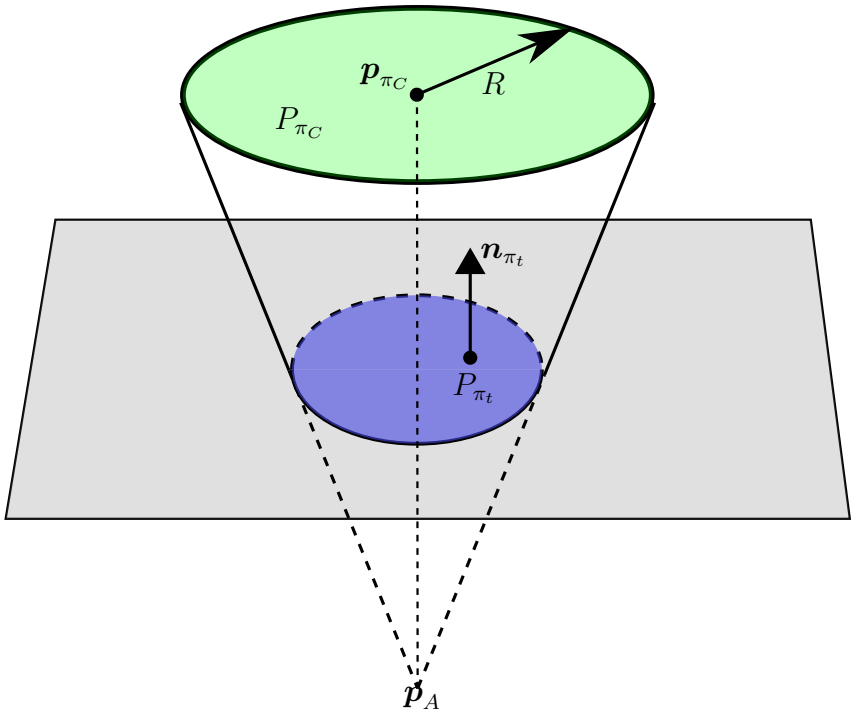


Figure 5.5: Inverted cone trunk region of interest $\mathcal{I}_{\mathfrak{C}}$. For the sake of clarity, the infinite target plane P_{π_t} was truncated.

5.5 Conclusion

This chapter has started by describing the complexity issues related to discretizing the workspace into multiple locations in the planning framework of He et al. (2015). It is shown that the discretization of the workspace into multiple locations is not suitable for the framework, since it increases the number of locations and hence the size of the planning graph. Thus, this approach does not scale well if it is desired to do task relaxation. Next, we have presented two task relaxation approaches that are based on the definition of regions of interest. The first one is based on switching between task vectors that require less DOF according to the region of interest and the second is based on choosing a control objective that can achieve the task by constraining the end-effector to a single region of interest. We adopt the latter and use dual quaternion algebra to describe the regions of interest.

6

Experiments and Results

This chapter will address the evaluation of the adapted planning framework of He et al. (2015) to use the constrained motion controller (Marinho et al., 2019). First, we present the computational tools used and evaluate the planning framework by devising four tasks and analyzing the generated task plans. The results show that by using the constrained motion controller instead of a motion planning layer, the total planning time is reduced. Afterward, in the second section of the chapter, we analyze the constraints imposed in the constrained motion controller to relax the task.

6.1 Computational Tools

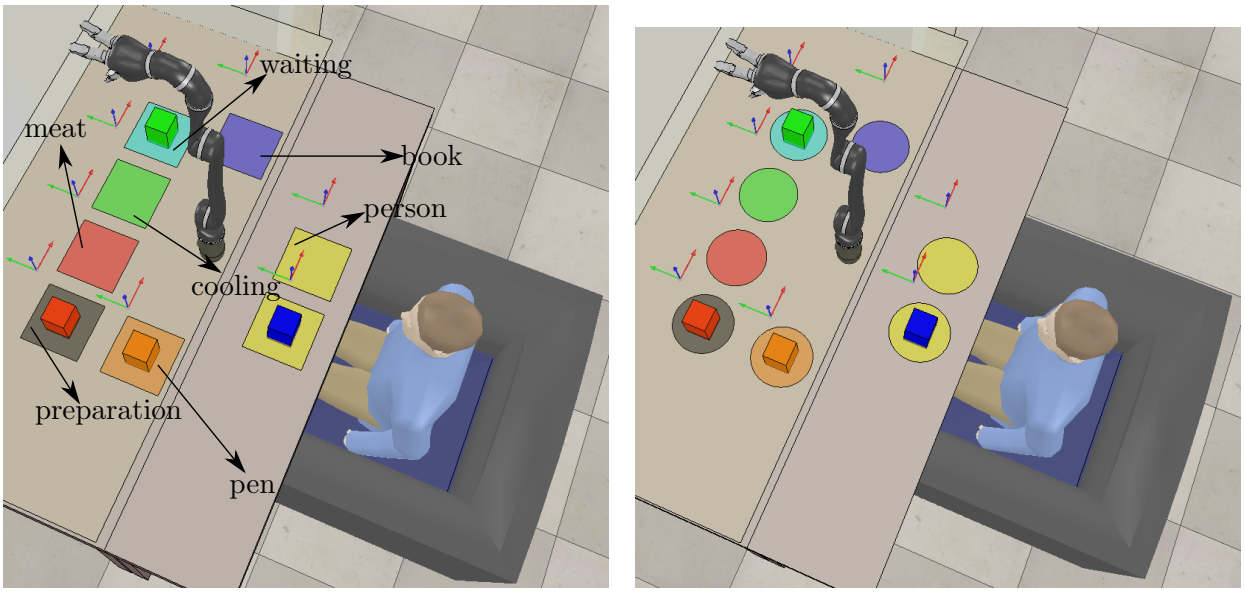
The implementation of the planning framework proposed by He et al. (2015) was done in C++ with the Boost Graph Library[1] and the automata utilities from the Open Motion Planning Library (OMPL) (Sucan et al., 2012).[2] The LTL task is processed using Spot (Duret-Lutz and Poitrenaud, 2004) and we performed simulations on CoppeliaSim[3] using ROS.[4] Furthermore, we used the DQ Robotics library (Adorno and Marinho, 2020) for robot modeling and control and to define the geometrical constraints, and constrained convex optimization was implemented using IBM ILOG CPLEX Optimization Studio.[5]

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

All the experiments were done on a single computer running Ubuntu 18.04 x64 with an Intel Core i7-8550U CPU with 16GiB of memory and a GeForce MX150.

6.2 Evaluation of the Planning Framework

To test the planning framework, we created a simulation scene on CoppeliaSim where a Kinova JACO robot must execute assistive tasks for a seated person, with limited or no lower limbs mobility, who cannot reach farther objects in the scene. The robot is placed on a table and the person is seated on a chair in front of the table. There are four colored cuboid objects o_{meat}, o_{salad} , o_{book}, o_{pen} representing, meat (red), salad (green), book (blue) and a pen (orange), respectively. In addition, eight colored regions of interest are depicted representing preparation area l_{prep} (dark gray), heating area l_{heat} (red), cooling area l_{cool} (green), waiting area l_{wait} (cyan), book area l_{book} (blue), two person-areas l_{pers} (yellow), and pen area l_{pen} (orange). Initially, the meat is in the preparation area, the salad is in the waiting area, the pen is in the pen area, and the book is in one of the person areas, as shown in Figure 6.1.



(a) Squared regions of interest.

(b) Circular regions of interest.

Figure 6.1: Scene for an LTL specification. The *red*, *green*, *blue* and *orange* cuboids represent the meat, salad, book, and pen, respectively. The regions of interest are indicated in the figure.

The planning framework generates all the graphs and searches the product graph $\mathcal{P}$ for an accepting path, that is, a task plan. Afterward, the robot iterates over the nodes of the accepting path executing the actions considering the scene locations. During HOLD, the

robot holds an object from one location to another, and during MOVE, the robot moves the empty gripper from one location to another. Let us assume the robot is currently at node k with action MOVE and end-effector location loc_i. Assume that at the node $k+1$, the end-effector location is loc_j. Therefore, the end-effector will move from loc_i to loc_j.

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

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

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

$$\varphi_1 = \mathcal{E}(p_{m,h} \wedge p_{s,p} \wedge \mathcal{X}\mathcal{E}(p_{m,p})). \tag{6.1}$$

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

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

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

$$\varphi_2 = \mathcal{E}\Big(p_{b,b} \wedge \neg p_{s,p} \wedge \neg p_{m,h} \wedge \mathcal{X}\mathcal{E}\Big(p_{b,b} \wedge \mathcal{E}(p_{s,p}) \wedge \mathcal{E}(p_{m,h}) \wedge \mathcal{X}\mathcal{E}\Big(p_{b,b} \wedge p_{s,p} \wedge \mathcal{X}\mathcal{E}(p_{s,p} \wedge p_{m,p})\Big)\Big)\Big). \quad (6.2)$$

After moving the book to the book region, task 2 gives freedom to the robot to decide if the salad will be served first or, instead, if the meat should be heated first. Next, it will serve the meat while the salad is being eaten. In comparison to task 1, task 2 specifies that the salad and the meat must be placed in front of the person. Otherwise, the robot would be free to remove the salad and only then serve the meat. The automaton generated from φ_2 has eight states and the planner explores 25098 nodes in the product graph. The total planning time average is 2.40 seconds.

Task 3: First, move pen to the person. While the person is reading the book and making notes, cool the salad and heat the meat in any order. Next, serve the meat and the salad in any order.

To write task 3, we define $p_{p,p} = (o_{\text{pen}}, l_{\text{pers}})$, $p_{s,c} = (o_{\text{salad}}, l_{\text{cool}})$, $p_{m,h} = (o_{\text{meat}}, l_{\text{heat}})$, $p_{m,p} = (o_{\text{meat}}, l_{\text{pers}})$, $p_{s,p} = (o_{\text{salad}}, l_{\text{pers}})$, $p_{b,p} = (o_{\text{book}}, l_{\text{pers}})$. Therefore,

$$\varphi_3 = \mathcal{E}\Big(p_{p,p} \wedge \neg p_{s,c} \wedge \neg p_{m,h} \wedge \neg p_{m,p} \wedge \neg p_{s,p} \wedge \mathcal{X}\mathcal{E}\Big(p_{p,p} \wedge p_{s,c} \wedge p_{m,h} \wedge \neg p_{m,p} \wedge \neg p_{s,p} \wedge p_{b,p} \wedge \mathcal{X}\mathcal{E}\Big(p_{m,p} \wedge p_{s,p}\Big)\Big)\Big). \quad (6.3)$$

As specified in task 3, first, the robot moves the pen to the person. As in task 2, the

robot has the freedom to decide what it does next: cool the salad or heat the meat. The planner generates a plan that makes the robot heat the meat and afterward cools the salad. At this point, both locations in front of the person will be occupied by the pen and the book. The planner correctly identifies that, but the task specification does not specify which location should be freed first and also does not indicate where the pen or the book should be moved. Hence, the planner again has the freedom to decide what to do. It first decides to move the pen to the preparation area. Once more, the planner can decide whether to serve the meat or the salad. It decides to serve the meat. Since both salad and meat must be served together, the planner arbitrarily moves the book to the heating area and, finally, serves the salad. The automaton generated from φ_3 has four states and the planner explores 119037 nodes in the product graph. The total planning time average is 12.94 seconds.

Task 4: First, move the pen to the person and the book to the book area in any order. Also, eventuallydo the following in any order: heat meat and eventually cool salad; serve meat and eventually serve salad; prepare salad and eventually place meat in waiting area.

The fourth task uses the following definitions $p_{p,\text{pers}} = (o_{\text{pen}}, l_{\text{pers}})$, $p_{b,b} = (o_{\text{book}}, l_{\text{book}})$, $p_{m,h} = (o_{\text{meat}}, l_{\text{heat}})$, $p_{s,c} = (o_{\text{salad}}, l_{\text{cool}})$, $p_{m,\text{pers}} = (o_{\text{meat}}, l_{\text{pers}})$, $p_{s,\text{pers}} = (o_{\text{salad}}, l_{\text{pers}})$, $p_{s,\text{prep}} = (o_{\text{salad}}, l_{\text{prep}})$, and $p_{m,w} = (o_{\text{meat}}, l_{\text{wait}})$. Hence,

$$\varphi_4 = \mathcal{E}\Big(p_{p,\text{pers}} \wedge p_{b,b} \wedge \neg p_{m,h} \wedge \neg p_{s,c} \wedge \neg p_{m,\text{pers}} \wedge \neg p_{s,\text{pers}} \wedge \neg p_{s,\text{prep}} \wedge \neg p_{m,w} \wedge \mathcal{E}\Big(p_{m,h} \wedge \mathcal{E}p_{s,c}\Big) \wedge \mathcal{E}\Big(p_{m,\text{pers}} \wedge \mathcal{E}p_{s,\text{pers}}\Big) \wedge \mathcal{E}\Big(p_{s,\text{prep}} \wedge \mathcal{E}p_{m,w}\Big)\Big). \quad (6.4)$$

Task 4 is very similar to the third task proposed by He et al. (2015) indicated as φ_7 in Table 6.1. First, the generated plan makes the robot move the book to the book area and next the pen to the person. In the sequence, the planner has a lot of options. It can execute the first part of each subtask and, afterward, the second part or it can execute both the first and second parts of each task sequentially. The planner chooses the former. Note, however, that the planner could also make a combination of both approaches. The automaton generated from φ_4 has four states and the planner explores 288166 nodes in the product graph. The total planning time average is 32.27 seconds.

Table 6.1 shows the planning data for the four tasks, which have the same number of objects (four) and locations (eight). The number of states and edges in the DFA indicates the complexity of the task. The planning time T_{planning} with the associated standard

deviation (s.d.) corresponds to an average of 50 runs. As expected, an increase in the task complexity increases the planning time because more nodes are explored in the product graph and then Dijkstra runs on a larger graph. In addition, Table 6.1 also shows the results for three tasks with similar complexities in the original framework of He et al. (2015). Last, Figure 6.2 shows the plot of the planning time for tasks 1-4 from Table 6.1. Tasks 1, 2, and 4 are similar in the adapted and original framework. Task 3 has no similar task in the original framework. The total planning time of the adapted framework is lower since there is no low-level motion planning time. Finally, to exemplify the execution of a task, a video of task 2 can be seen at `https://youtu.be/h3K-RsdAiCs`.

Table 6.1: Planning data for φ_1, φ_2, φ_3, and φ_4.

Task	$\lvert A_\varphi \rvert$	$\lvert E_{A_\varphi} \rvert$	$\lvert V_\mathcal{P} \rvert$	T_{planning}(s)	$T_{\text{low-level}}$(s)
Framework with constrained motion controller					
φ_1	3	5	16975	1.65418 (0.152028)	—
φ_2	8	20	25098	2.40551 (0.203752)	—
φ_3	4	7	119037	12.9369 (1.18004)	—
φ_4	28	218	288166	32.2673 (2.77166)	—
Original framework					
φ_5	2	3	44100	2.76	12.32
φ_6	8	27	75511	4.48	8.07
φ_7	27	290	498000	33.12	31.15

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

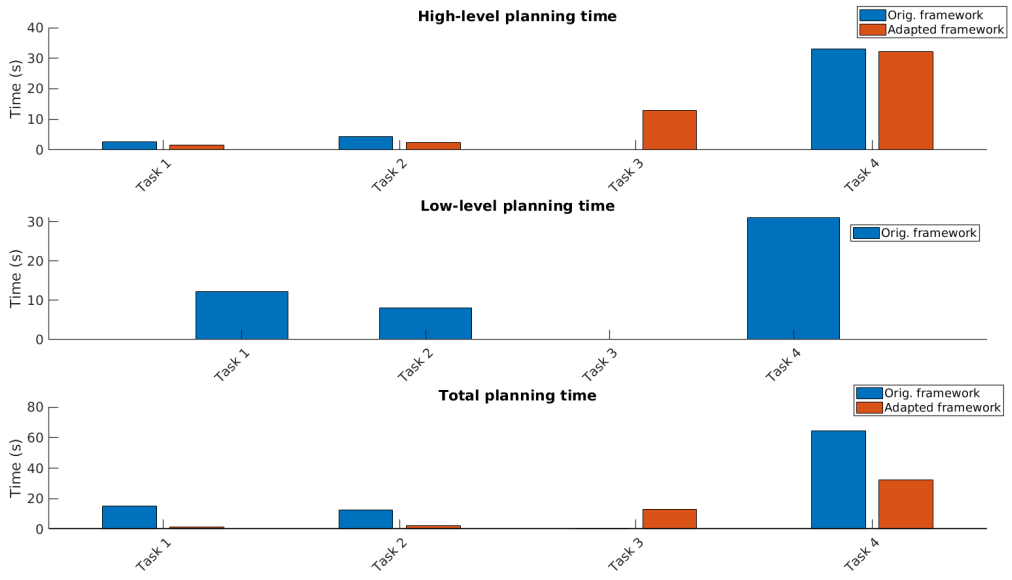


Figure 6.2: High-level and low-level planning time.

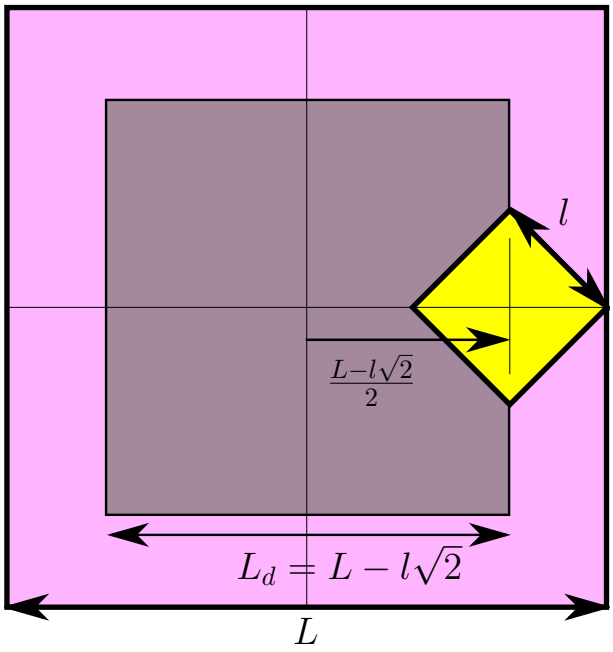
6.3 Evaluation of Relaxed Task Constraints

The main objective of task relaxation in this work is to make the end-effector place objects within a target region on a target plane to release degrees of freedom that can be used for secondary tasks such as avoiding obstacles and preventing joint limits. While the end-effector holds the object toward the target plane, it must stay within a specified region of interest. Two target regions were devised: a squared region and a circular region. The squared region is defined by using planes as presented in Section 4.2.1 and the circular region can be obtained by using a cone as discussed in Section 4.2.4. In the case of the squared target region, the end-effector must stay within an inverted pyramid trunk region of interest (Section 5.4.4). To move toward the circular target region, the end-effector must stay within a inverted cone trunk region of interest (Section 5.4.5). To define the squared and circular target regions it is necessary to determine some parameters.

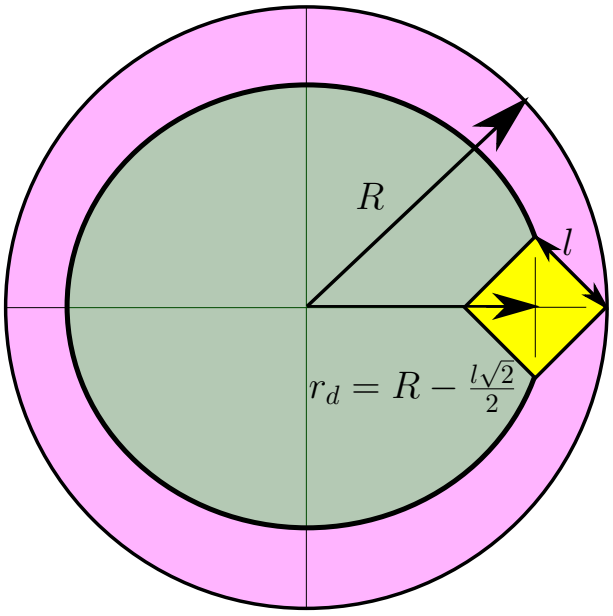
6.3.1 Determining the Parameters of the Target Regions

To define the plane constraints, it is necessary to choose the side of the squared region. The objects being manipulated in the simulation scene are cuboids (Figure 6.1) that must always fit into the target region. To always fit a cuboid with side l within a square with side L, the squared target region must be slightly smaller than the square. More specifically, the desired squared region side is $L_d = L - l\sqrt{2}$ (see Figure 6.3a).

In the case of the point-cone constraint, the target radius of the circular target region must be defined. To fit the cuboid with side l within a circular region with radius R, we take a similar approach. The desired circular target region radius r_d must be smaller than the circular region radius R and is given by $r_d = R - \frac{l\sqrt{2}}{2}$ (see Figure 6.3b).



(a) Squared region: the *gray* square with side $L - l\sqrt{2}$ indicates the squared target region, the *yellow* square indicates the squared object with side l, and the *pink* square with side L is the region where the cuboid must placed.



(b) Circular region: the *pink* circular region with radius R indicates where the cuboid must be placed, the *green* circle with radius $r_d = R - \frac{l\sqrt{2}}{2}$ is the circular target region, and the *yellow* square indicates the squared object with side l.

Figure 6.3: Target regions.

In the simulation scene used in this work (Figure 6.1), the squared regions have side $L = 0.2m$ and the circular regions have radius $R = 0.1m$. The cuboid object has side $l = 0.08m$. Now it is possible to define the side of the squared target region as $L_d = L - l\sqrt{2} = 0.086863m$ and the circular target region radius as $r_d = R - \frac{l\sqrt{2}}{2}$ obtaining $r_d = 0.0434315m$. These parameters were kept constant in all the tasks.

6.3.2 Constraints Parameters

The constraints parameters used for all simulations were kept constant. Each constraint has its own parameters. They are presented in Table 6.2.

Constraint		Gains	Parameters
Joint limits	$\mathfrak{W}(\dot{\boldsymbol{q}})$	$k = 0.1$, $\beta = 0.98$	Jaco robot joint limits
Walls and table	$\mathfrak{W}(\pi_1)$, $\mathfrak{W}(\pi_2)$, $\mathfrak{W}(\pi_3)$	$\eta_\pi = 5.0$	$d_{\pi,\text{safe}} = 0.05m$
Squared target region	$\mathfrak{W}(\pi_4)$, $\mathfrak{W}(\pi_5)$, $\mathfrak{W}(\pi_6)$, $\mathfrak{W}(\pi_7)$	$\eta_\pi = 1.0$	$d_{\pi,\text{safe}} = 0.0m$
Objects cylinders	$\mathfrak{W}(c_j)$ for $j \in \{1, \ldots, k\}$	$\eta_l = 5.0$, $\eta_\pi = 5.0$	$R = 0.0567m$, $d_{\pi,\text{safe}} = 0.15m$
Robot z-axis cylinder	$\mathfrak{W}(l_z)$	$\eta_l = 5.0$	$R_z = 0.15m$
Object centerline	$\mathfrak{W}(l_{\text{object}})$	—	—
End-effector z-axis line-cone	$\mathfrak{W}(l_{z,\text{cone}})$	$\eta_l = 50.0$	$\phi_{\text{safe}} = 0.1\text{rad}$
Circular target region	$\mathfrak{W}(l_{\text{cone}}^{\text{point}})$	$\eta_l = 50.0$	$r = r_d = 0.0434315m$

Table 6.2: Constraints parameters used during all the experiments.

The coordinates of the geometric primitives used to define the constraints are defined as follows:

- The coordinates of the walls and table planes are the coordinates of the light shaded yellow planes in the simulation scene (see Figure 4.2a). Note that the planes are slightly away from the walls and the table. This is done to increase safety and prevent collisions;
- The objects cylinders coordinates are the object's location at each time instant;
- The coordinates of the robot z-axis cylinder are the robot coordinate system coordinates;
- The coordinates of the object centerline are the object coordinate system coordinates;
- The end-effector z-axis line-cone constraint is defined by two lines at each time instant: the line collinear to the end-effector z-axis and the line parallel to the robot z-axis that passes through the end-effector position;
- The squared and circular target region coordinates are the location where the object must be placed (see Figure 4.2a).

6.3.3 Constrained Motion Controller Parameters

As the constraints parameters, the constrained motion controller parameters were also kept constant during all simulations. In addition to the controller gain η and the damping factor λ presented in Section 4.1, we also make use of three more parameters. The desired

task error norm means that the controller runs until the task error norm reaches the specified value. The stability threshold indicates the norm of the task error derivative that the controller must reach to be considered stable. Last, the number of iterations that the controller must run with the stability threshold to be considered stable is the maximal stability counter. All the controller parameters are presented below in Table 6.3.

Parameter	**Pose control**	**Distance to plane control**
Controller gain	$\eta = 100$	$\eta = 100$
Damping factor	$\lambda = 0.001$	$\lambda = 0.001$
Desired task error norm	0.009	0.009
Stability threshold	0.000001	0.0001
Maximal stability counter	100	100
Sampling time	5 ms	5 ms

Table 6.3: Constrained motion controller parameters.

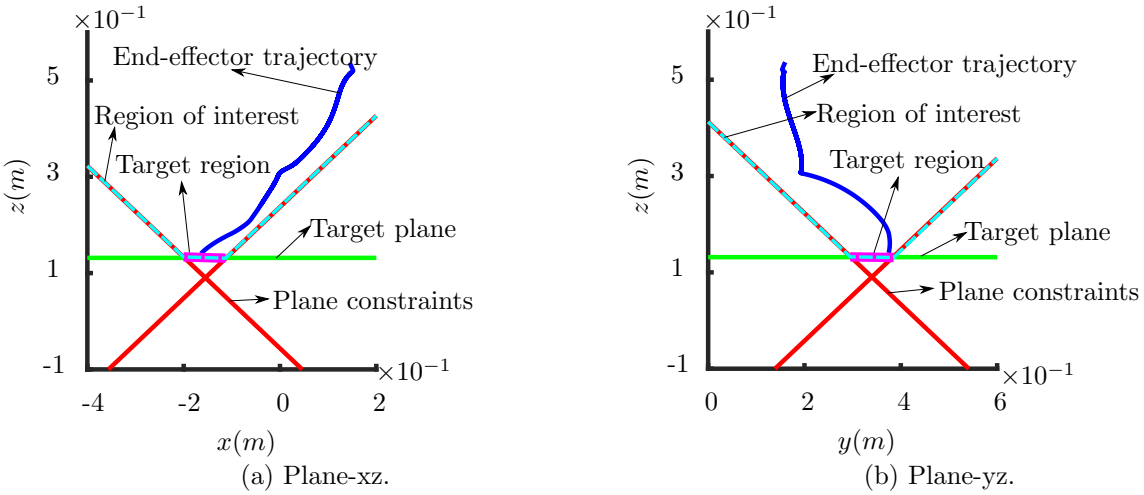
6.3.4 Constraints Evaluation

The execution of each task presents several examples where it is possible to check the satisfaction of the constraints imposed in the constrained motion controller. Regarding the constraints satisfaction and the relaxed task regions, the focus of this work is to check that the end-effector is satisfying the plane constraints and the point-cone constraint from Section 4.2.1 and Section 4.2.4. This way, we ensure that it is possible to move the end-effector from an initial configuration to a desired relaxed target region on a plane that can be squared or circular. Moreover, the end-effector stays within a specified region of interest during its trajectory towards the target region. More specifically, the end-effector stays inside of the inverted pyramid trunk (Section 5.4.4) or of the inverted cone trunk (Section 5.4.5). In this sense, we selected some examples obtained during the execution of two of the tasks to illustrate the results.

First, we present the evaluation of a snapshot of task 1 when the meat is hold toward the heating region. In addition to the plane and point-cone constraints results we also present the results of the other constraints. Next, we show a snapshot of task 2 when the book is hold toward the book region. Lastly, we present the evaluation of all constraints during the whole execution of task 2.

Task 1: Moving the meat from the intermediate location to the heating region.

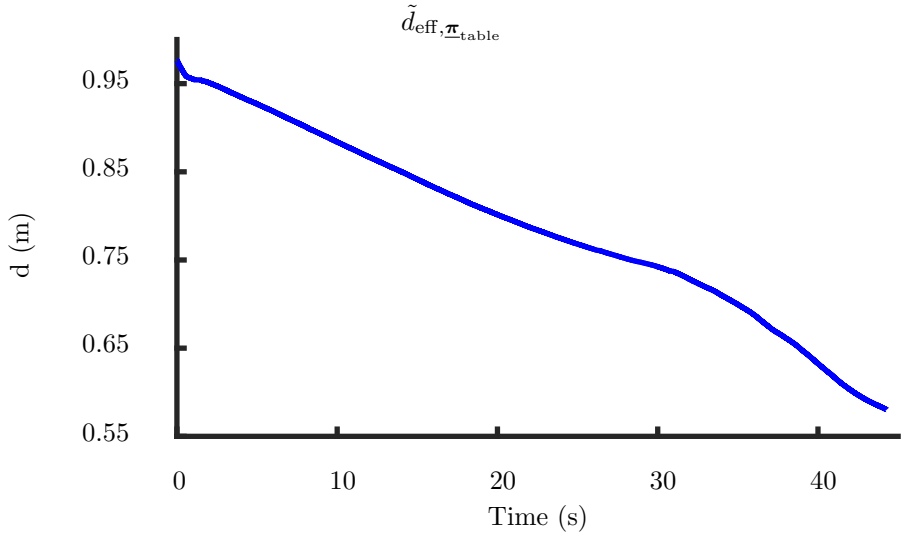
Inverted pyramid trunk region of interest The result of using the plane constraints to make the end-effector to stay inside the inverted pyramid trunk region of interest (5.17) and move toward a target squared region on the plane is shown in Figure 6.4. It can be seen that the end-effector remains within the inverted pyramid trunk region of interest and moves towards the target plane.



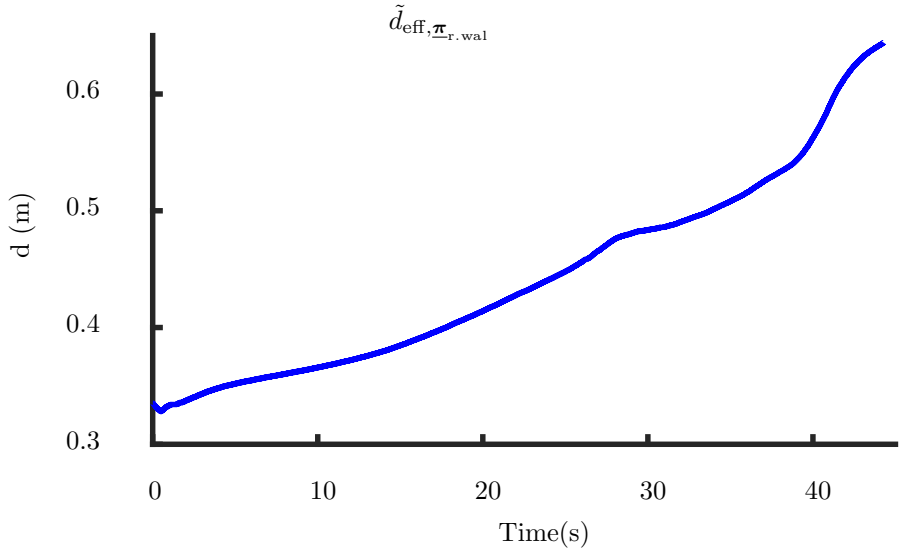
(a) Plane-xz. (b) Plane-yz.

Figure 6.4: Lateral views of the end-effector trajectory using plane constraints: moving the meat from the intermediate location to the heating location during task 1.

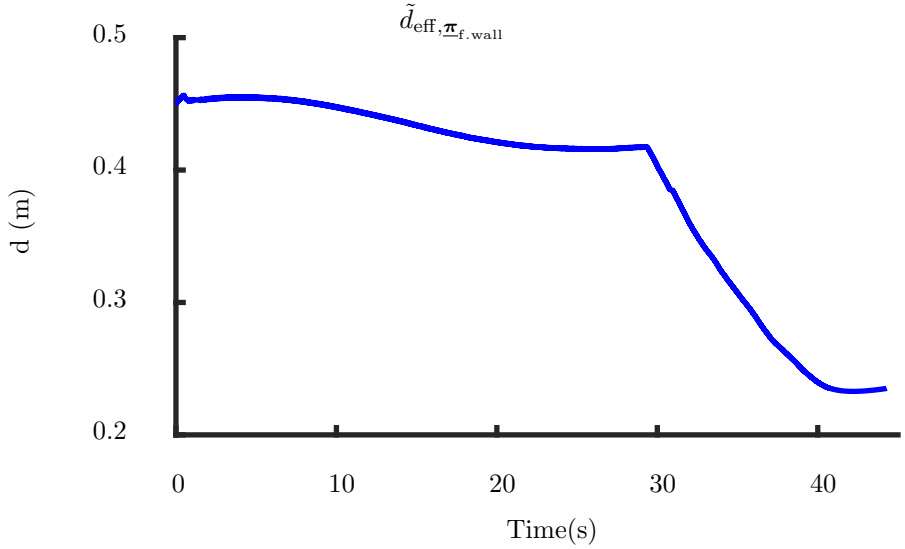
In addition to satisfying the plane constraints that constrain the end-effector to the inverted pyramid trunk, the other constraints from Table 6.2 are also satisfied. To verify if any collision happened, we analyze the signed distances $\tilde{d}$ between the constrained elements. A negative signed distance means that a collision happened. Figure 6.5 shows the signed distance between the end-effector and the planes in the environment. Figure 6.6 shows the signed distance between the end-effector and the non-manipulated objects during the task. Last, Figure 6.7 shows the signed distance between the end-effector and the infinite cylinder about the robot z-axis. These figures show that no collision occurred and, hence, that the constraints to prevent collisions with objects in the workspace were satisfied.



(a) Constraint $\mathfrak{W}(\pi_1)$: $\tilde{d}_{\text{eff},\underline{\boldsymbol{\pi}}_{\text{table}}}$.

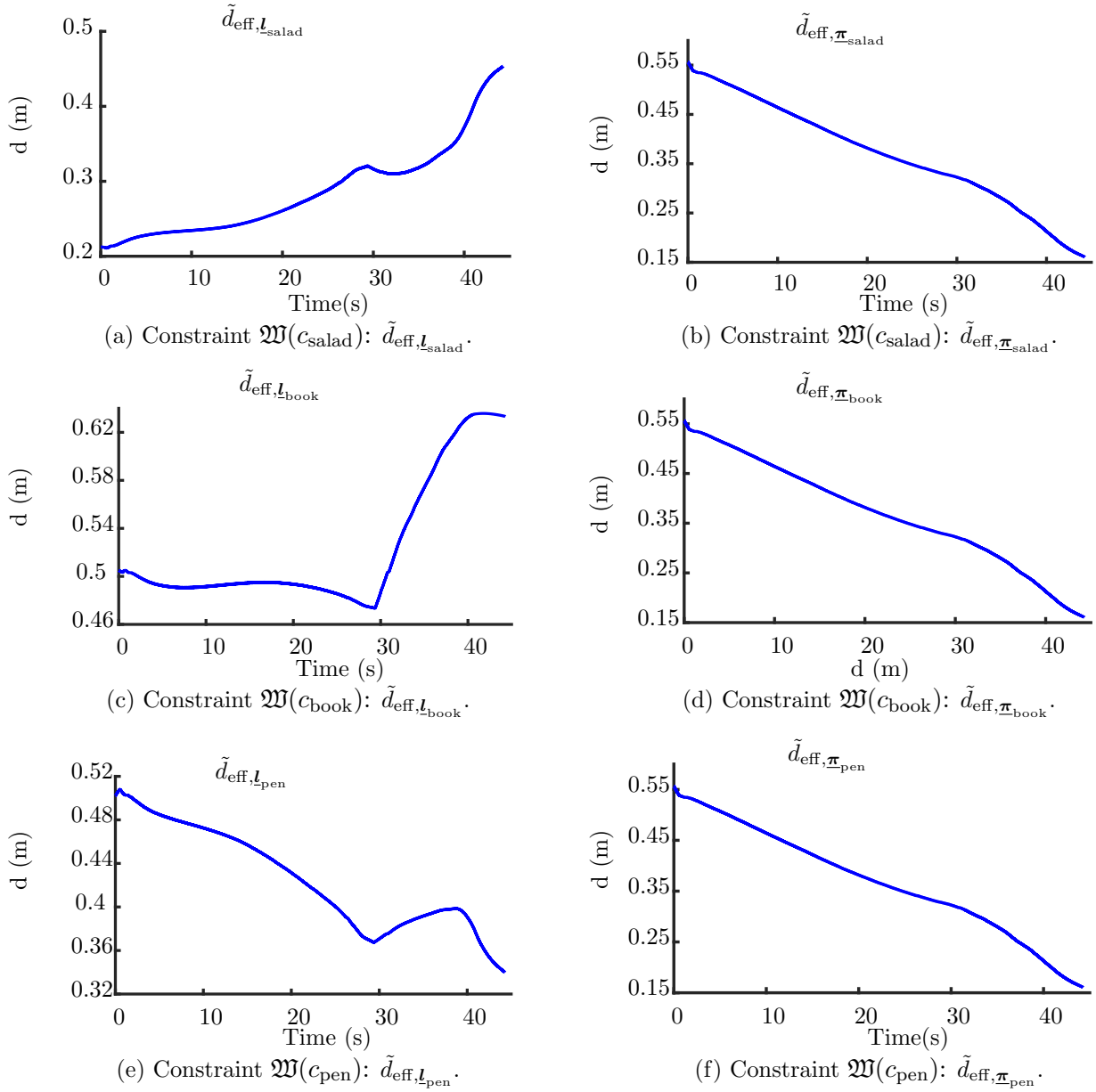


(b) Constraint $\mathfrak{W}(\pi_2)$: $\tilde{d}_{\text{eff},\underline{\boldsymbol{\pi}}_{\text{r.wal}}}$.



(c) Constraint $\mathfrak{W}(\pi_3)$: $\tilde{d}_{\text{eff},\underline{\boldsymbol{\pi}}_{\text{f.wall}}}$.

Figure 6.5: Signed distances between end-effector and environment planes: the table top, right wall (r. wall), and front wall (f. wall).



(a) Constraint $\mathfrak{W}(c_{\text{salad}})$: $\tilde{d}_{\text{eff},\underline{\boldsymbol{l}}_{\text{salad}}}$.

(b) Constraint $\mathfrak{W}(c_{\text{salad}})$: $\tilde{d}_{\text{eff},\underline{\boldsymbol{\pi}}_{\text{salad}}}$.

(c) Constraint $\mathfrak{W}(c_{\text{book}})$: $\tilde{d}_{\text{eff},\underline{\boldsymbol{l}}_{\text{book}}}$.

(d) Constraint $\mathfrak{W}(c_{\text{book}})$: $\tilde{d}_{\text{eff},\underline{\boldsymbol{\pi}}_{\text{book}}}$.

(e) Constraint $\mathfrak{W}(c_{\text{pen}})$: $\tilde{d}_{\text{eff},\underline{\boldsymbol{l}}_{\text{pen}}}$.

(f) Constraint $\mathfrak{W}(c_{\text{pen}})$: $\tilde{d}_{\text{eff},\underline{\boldsymbol{\pi}}_{\text{pen}}}$.

Figure 6.6: Signed distances between end-effector and object cylinder line and plane.

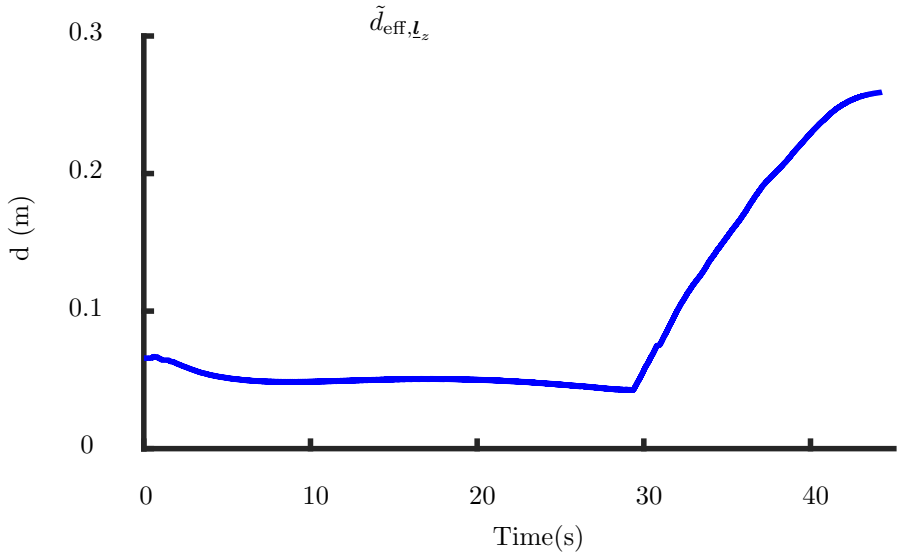


Figure 6.7: Constraint $\mathfrak{W}(l_z)$: signed distance $\tilde{d}_{\text{eff},\underline{\boldsymbol{l}}_z}$ between end-effector and robot z-axis infinite cylinder.

Besides the constraints to prevent collisions with objects in the workspace, there is also the line-cone constraint that limits the end-effector z-axis within a cone about a static line in the workspace (Quiroz-Omana and Adorno, 2019). Figure 6.8 shows that the angle between the end-effector z-axis and the static line remains within the safety angle.

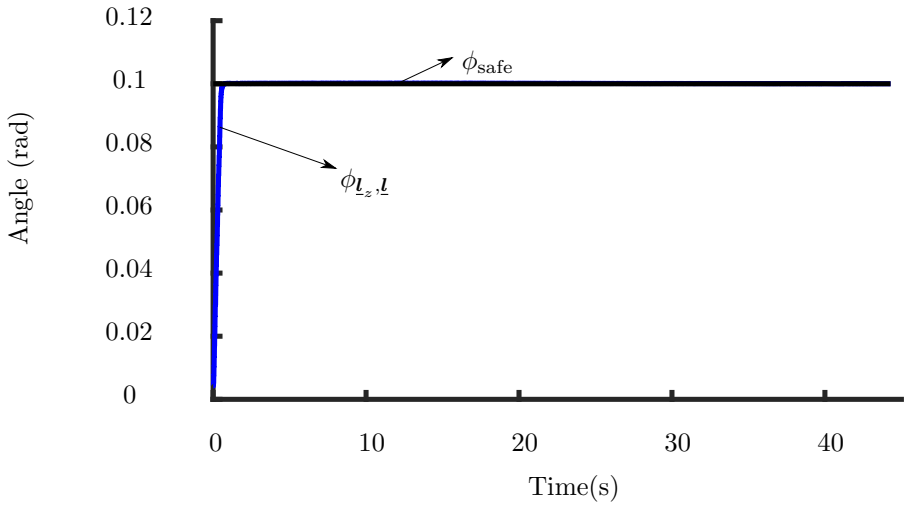


Figure 6.8: Constraint $\mathfrak{W}(l_{z,\text{cone}})$: angle $\phi_{\underline{\boldsymbol{l}}_z,\underline{\boldsymbol{l}}}$ between end effector z-axis and static line $\underline{\boldsymbol{l}}$ parallel to robot z-axis. The safe angle ϕ_{safe} is the cone maximum angle.

Finally, we have the joint limits constraint shown in Figure 6.9. It can be seen that the joint limits are satisfied. The fifth joint approaches the upper joint limit.

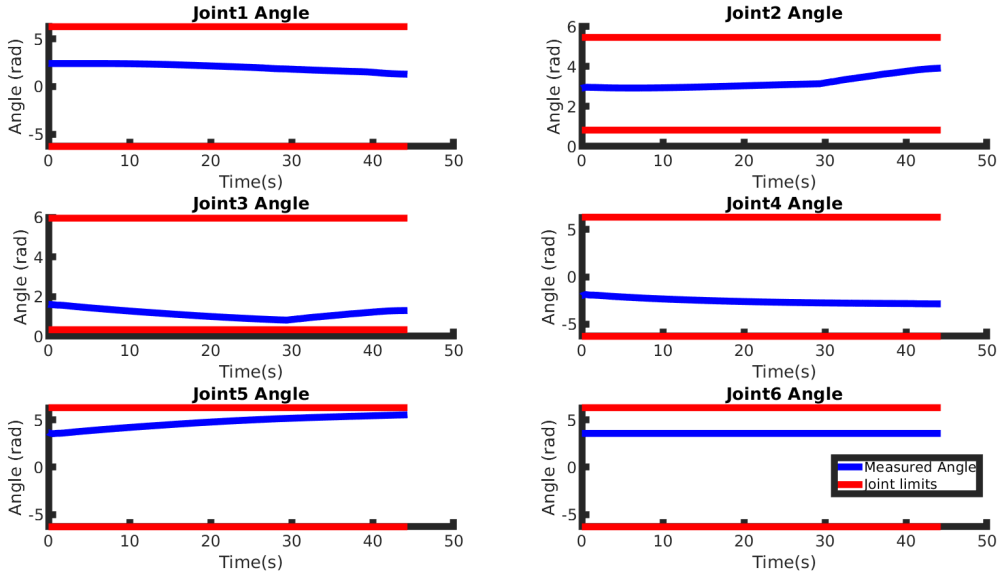
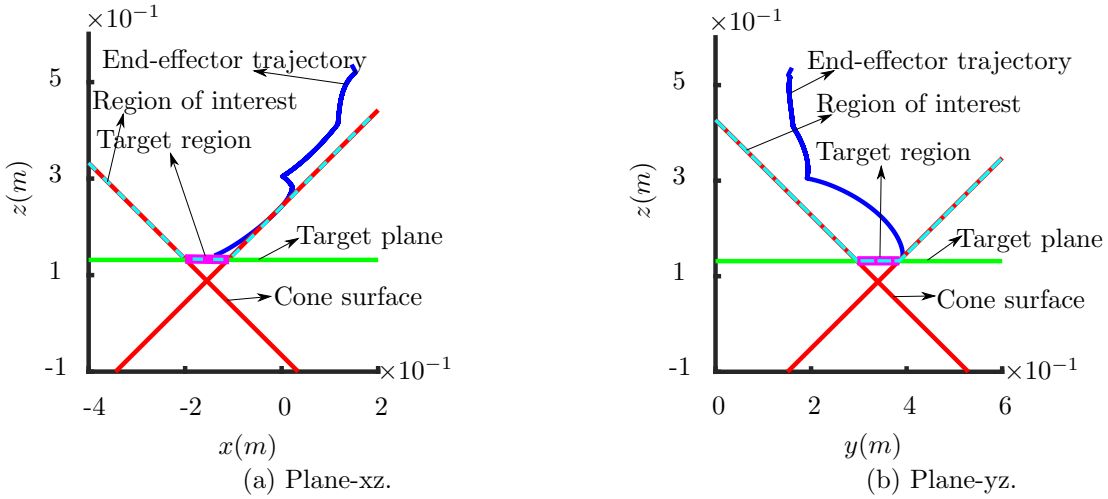


Figure 6.9: Constraint $\mathfrak{W}(\dot{\boldsymbol{q}})$: joint limits constraints.

Inverted cone trunk region of interest Analogously to the plane constraints, Figure 6.10 shows the result of using the point-cone constraint to make the end-effector move toward a circular target region while staying inside the inverted cone trunk region of interest 5.18. The result shows that the end-effector stays within the desired region of interest and moves towards the target plane.

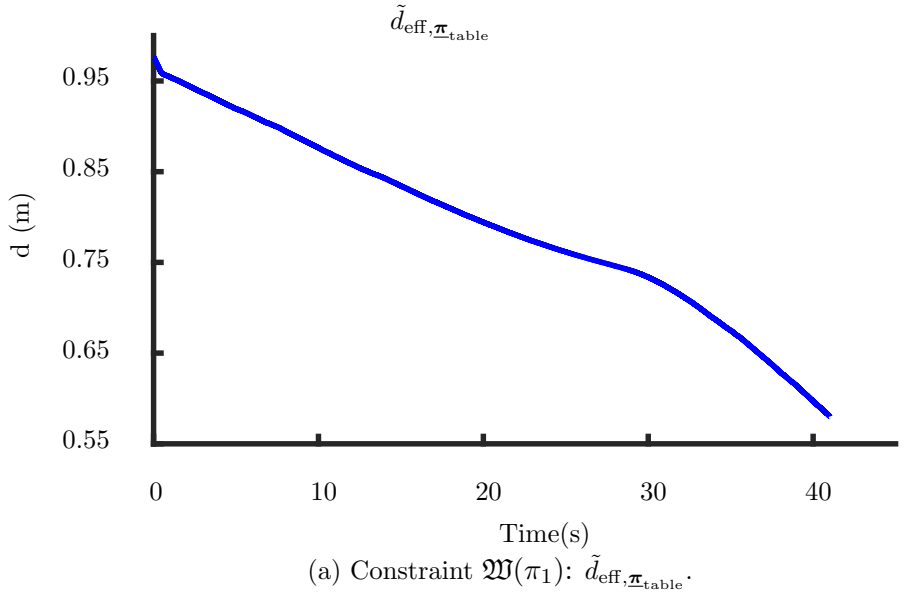


(a) Plane-xz. (b) Plane-yz.

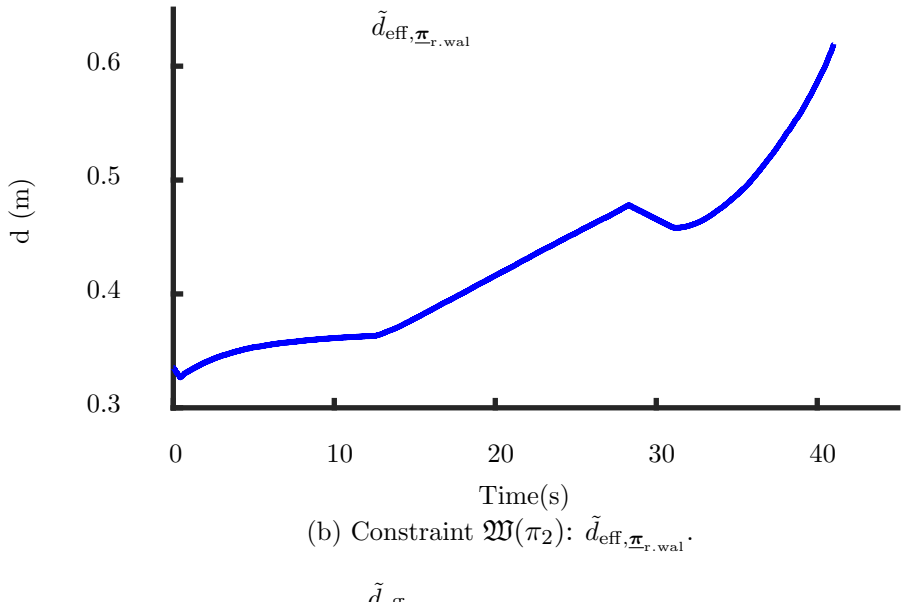
Figure 6.10: Lateral views of the end-effector trajectory using the point-cone constraint: moving the meat from the intermediate location to the heating location during task 1.

Additionally to the point-cone constraint, the other constraints from Table 6.2 were also satisfied. As in the case of the inverted pyramid trunk region of interest, we also analyzed the signed distance $\tilde{d}$ between the constrained elements. Figures 6.11-6.12 shows the signed distance between the end-effector and environment planes and the end-effector

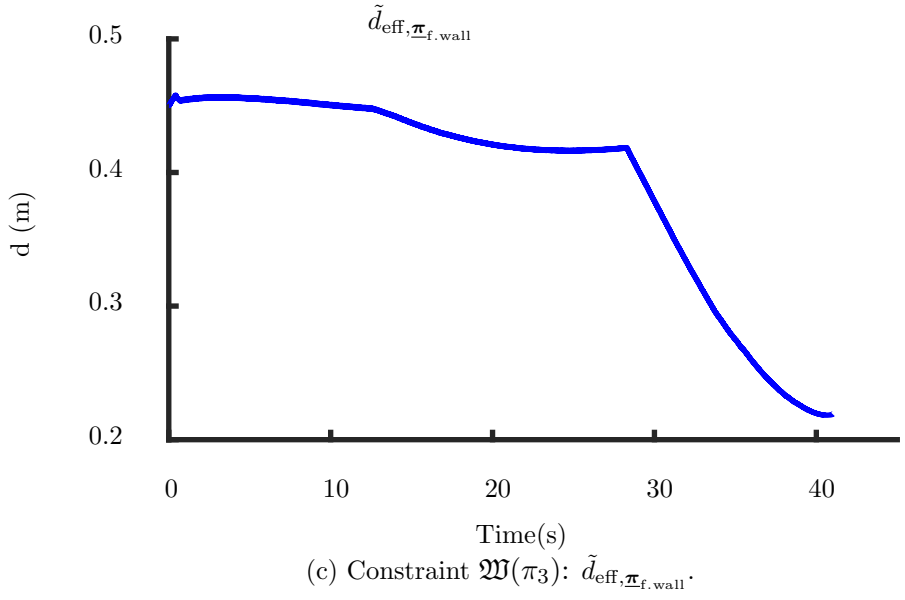
and the non-manipulated objects during the task. Also, Figure 6.13 shows the signed distance between the end-effector and the infinite cylinder about the robot z-axis. These figures show that no collision occurred and, hence, that the constraints to prevent collisions with objects in the workspace were satisfied.



(a) Constraint $\mathfrak{W}(\pi_1)$: $\tilde{d}_{\text{eff},\underline{\boldsymbol{\pi}}_{\text{table}}}$.

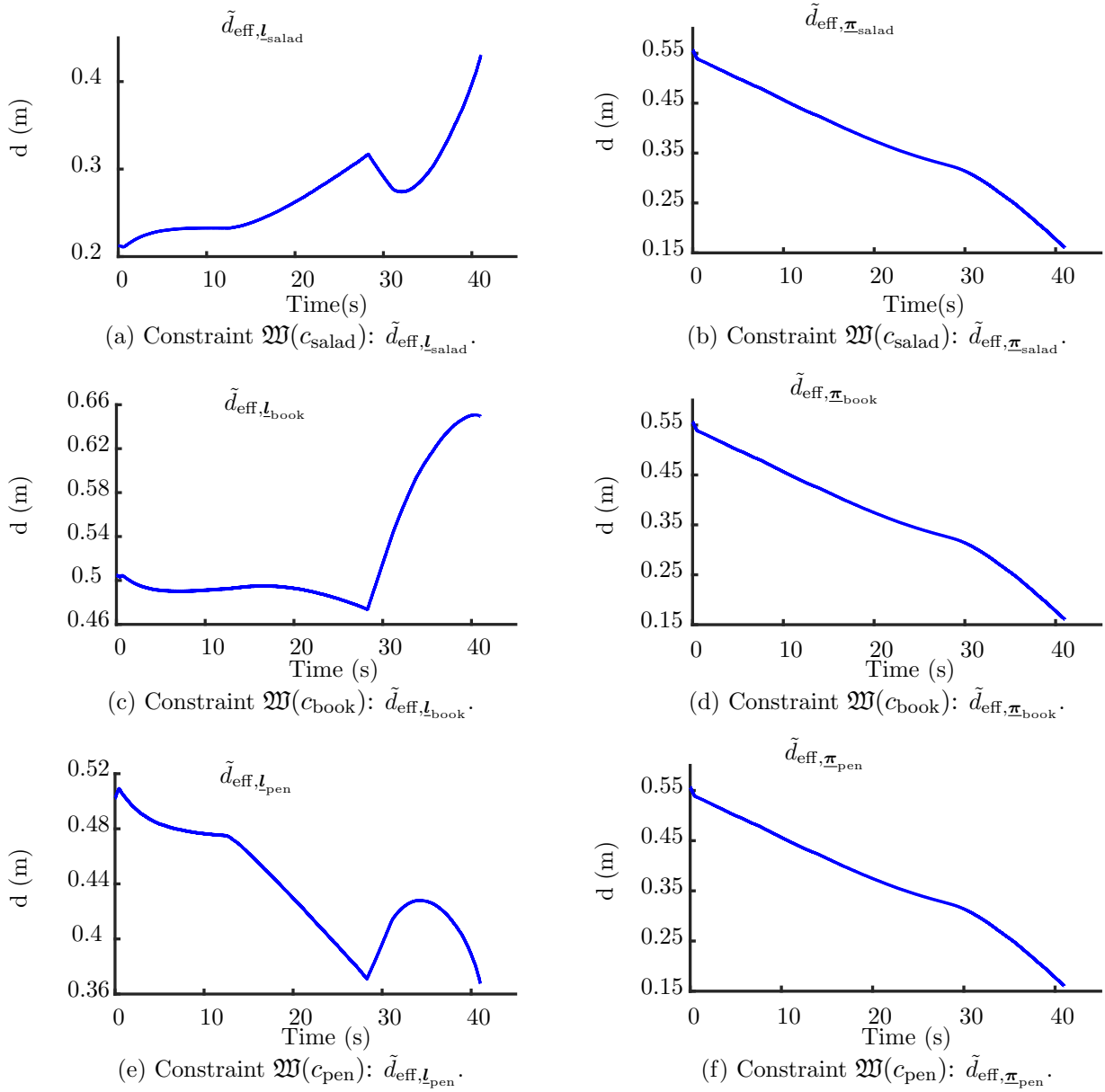


(b) Constraint $\mathfrak{W}(\pi_2)$: $\tilde{d}_{\text{eff},\underline{\boldsymbol{\pi}}_{\text{r.wal}}}$.



(c) Constraint $\mathfrak{W}(\pi_3)$: $\tilde{d}_{\text{eff},\underline{\boldsymbol{\pi}}_{\text{f.wall}}}$.

Figure 6.11: Signed distances between end-effector and environment planes: the table top, right wall (r. wall), and front wall (f. wall).



(a) Constraint $\mathfrak{W}(c_{\text{salad}})$: $\tilde{d}_{\text{eff},\underline{\boldsymbol{l}}_{\text{salad}}}$.

(b) Constraint $\mathfrak{W}(c_{\text{salad}})$: $\tilde{d}_{\text{eff},\underline{\boldsymbol{\pi}}_{\text{salad}}}$.

(c) Constraint $\mathfrak{W}(c_{\text{book}})$: $\tilde{d}_{\text{eff},\underline{\boldsymbol{l}}_{\text{book}}}$.

(d) Constraint $\mathfrak{W}(c_{\text{book}})$: $\tilde{d}_{\text{eff},\underline{\boldsymbol{\pi}}_{\text{book}}}$.

(e) Constraint $\mathfrak{W}(c_{\text{pen}})$: $\tilde{d}_{\text{eff},\underline{\boldsymbol{l}}_{\text{pen}}}$.

(f) Constraint $\mathfrak{W}(c_{\text{pen}})$: $\tilde{d}_{\text{eff},\underline{\boldsymbol{\pi}}_{\text{pen}}}$.

Figure 6.12: Signed distances between end-effector and object cylinder line and plane.

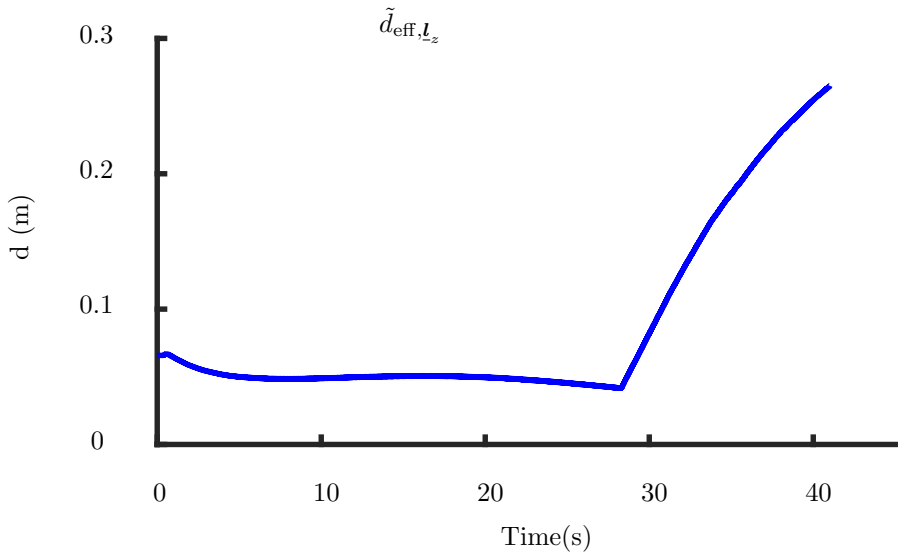


Figure 6.13: Constraint $\mathfrak{W}(l_z)$: signed distance $\tilde{d}_{\text{eff},\underline{\boldsymbol{l}}_z}$ between end-effector and robot z-axis infinite cylinder.

As in the case of the inverted pyramid trunk region of interest, there is also the line-cone constraint that limits the end-effector z-axis within a cone about a static line in the workspace (Quiroz-Omana and Adorno, 2019). Figure 6.14 shows that the angle between the end-effector z-axis and the static line remains within the safety angle.

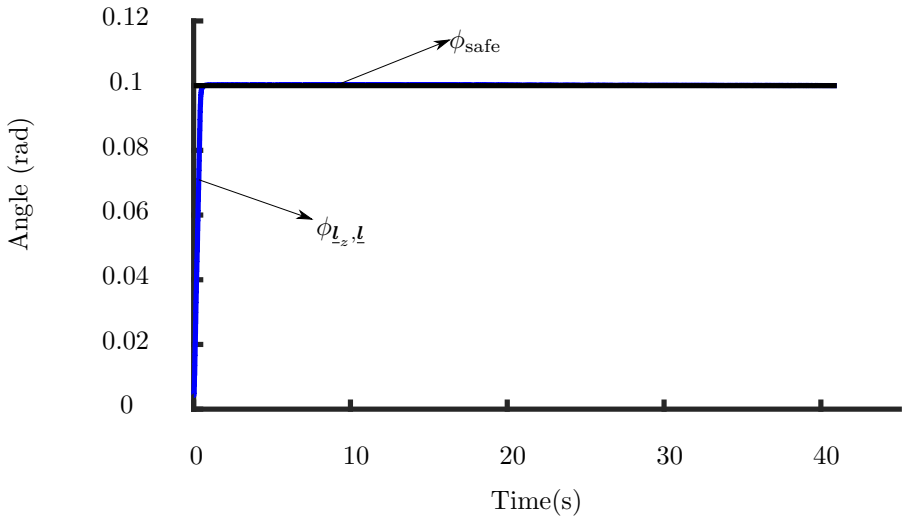


Figure 6.14: Constraint $\mathfrak{W}(l_{z,\text{cone}})$: angle $\phi_{\underline{\boldsymbol{l}}_z,\underline{\boldsymbol{l}}}$ between end effector z-axis and static line $\underline{\boldsymbol{l}}$ parallel to robot z-axis. The safe angle ϕ_{safe} is the cone maximum angle.

Lastly, we have the joint limits constraint shown in Figure 6.15. It can be seen that the joint limits are satisfied. The fifth joint approaches the upper joint limit.

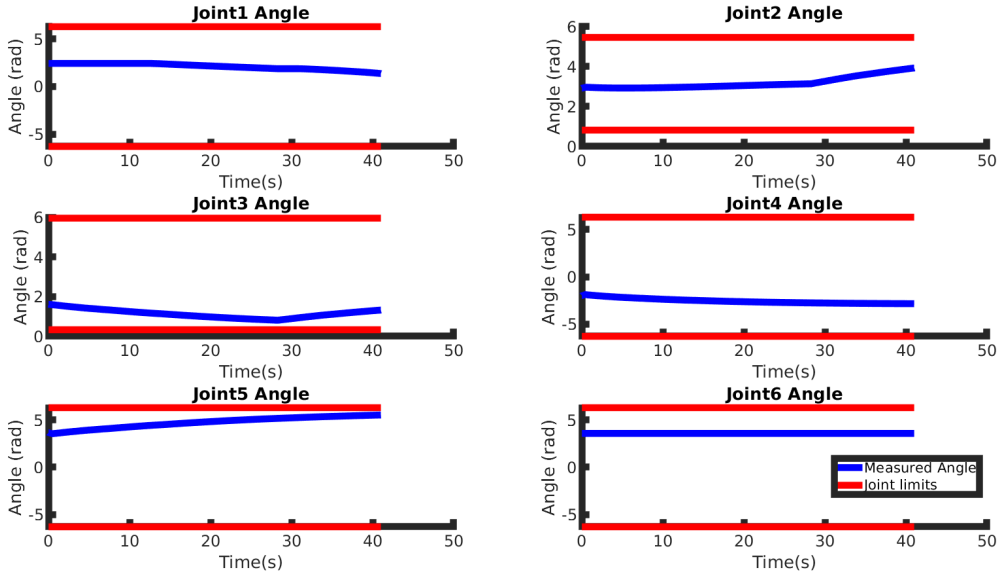


Figure 6.15: Constraint $\mathfrak{W}(\dot{\boldsymbol{q}})$: joint limits constraints.

Task 2: Holding the book from the intermediate location to the book region

In addition to analyzing the results of a snapshot of task 1, we also analyze the results of a snapshot of task 2. More specifically, we selected the action of holding the book from the intermediate location to the book region during task 2.

Inverted pyramid trunk region of interest Figure 6.16 shows the action of holding the book to the book region during task 2. The end-effector remains within the plane constraints and moves toward the target plane, as expected.

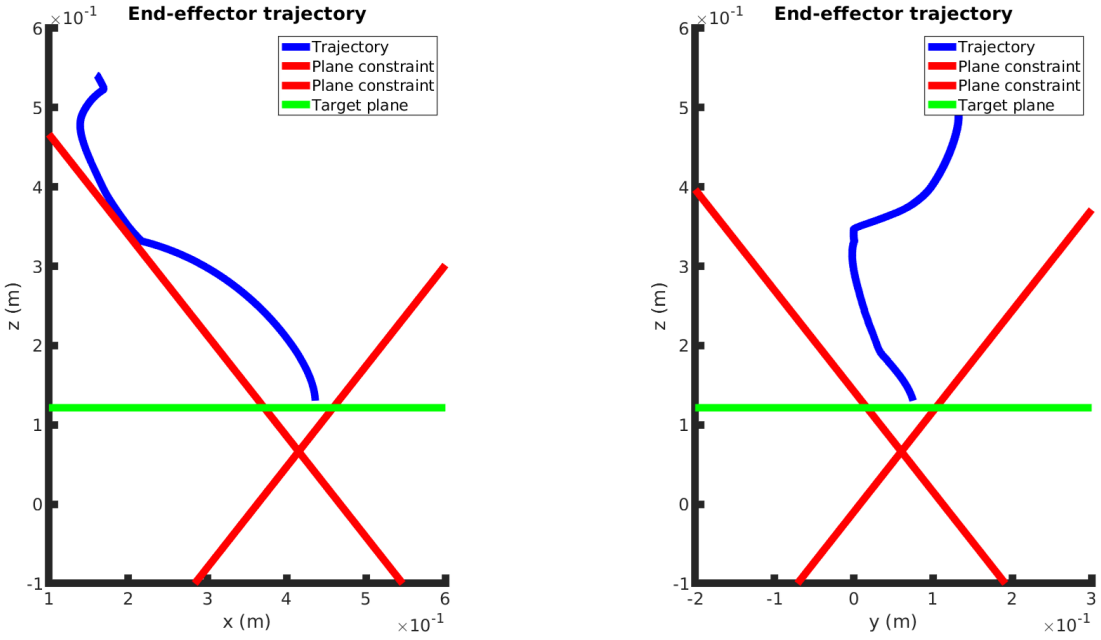


Figure 6.16: Lateral views of the end-effector trajectory using the plane constraints: holding the book from the intermediate location to the book location during task 2.

Inverted cone trunk region of interest Figure 6.17 shows the action of holding the book to the book region during task 2. It can be seen that the end-effector remains within the point-cone constraint and moves toward the target plane as expected.

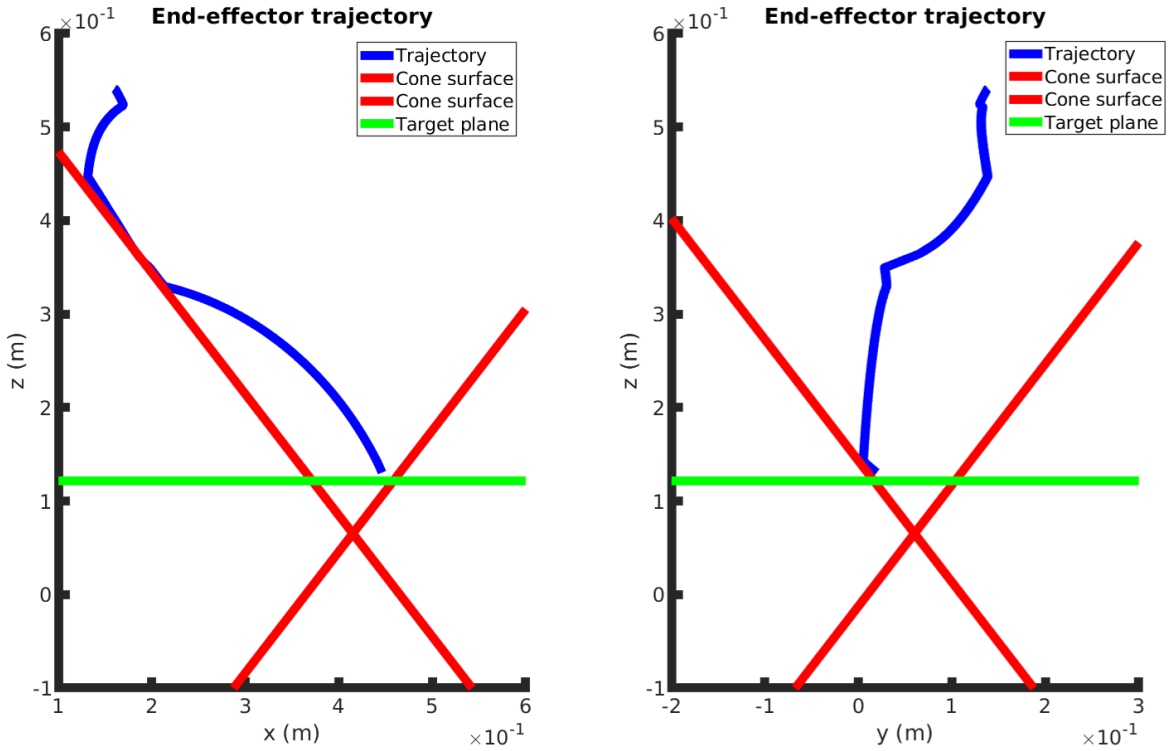


Figure 6.17: Lateral views of the end-effector trajectory using the point-cone constraint: holding the book from the intermediate location to the book location during task 2.

Task 2: Evaluation of constraints during the whole task execution

Now we evaluate the other constraints from Table 6.2 during the whole execution of task 2. As before, we analyze the signed distance $\tilde{d}$ between the constrained elements.

Inverted pyramid trunk region of interest Figures 6.18 shows the signed distance between the end-effector and environment planes. The signed distance remains positive during the whole task execution. Hence, there is no collision between the end-effector and the environment planes.

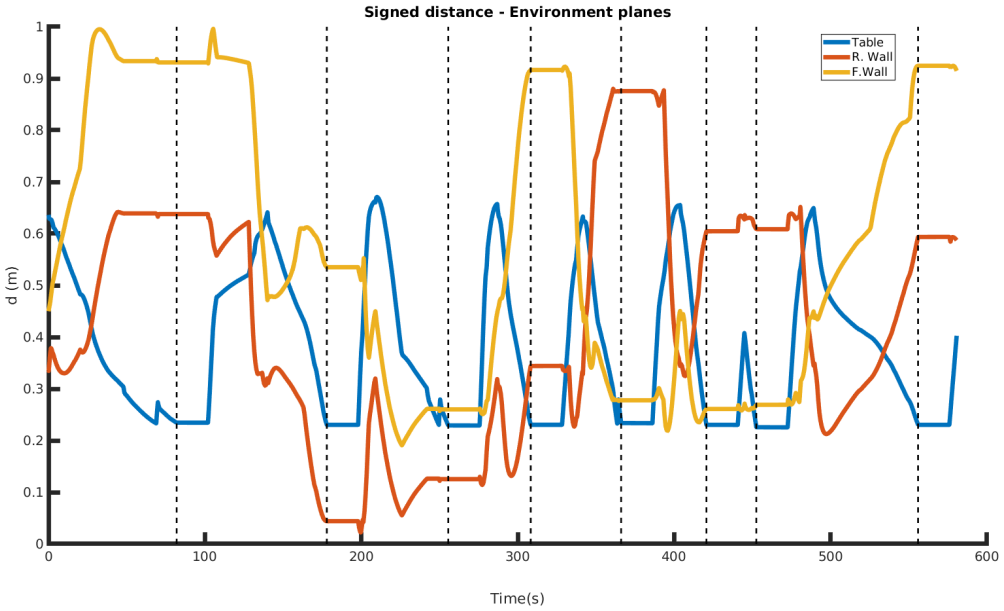


Figure 6.18: Plane constraints: signed distance between the end-effector and the planes in the environment (table top, right wall, and front wall). The dotted black lines indicate when an object is grasped or placed.

Also, Figure 6.19 shows the signed distance between the end-effector and the objects. In the first part of the task, the end-effector must grasp the book, hence, it approaches the book and the signed distance to the book becomes negative. However, the signed distance to the other non-manipulated objects remains positive indicating no collision occured. In the second part of the task, the end-effector manipulates the book (i.e. the signed distance remains negative, since we do not detect collisions with the manipulated object). Next, the end-effector places the book on the book region and moves away from the book (i.e. the signed distance becomes positive). In the sequence, the end-effector approaches the salad (i.e. the signed distance decreases and becomes negative). This procedure repeats until all the objects are manipulated and placed in the locations specified by the task. Also, the signed distance to the objects planes remains positive during the whole task execution indicating that the end-effector does not cross the objects planes.

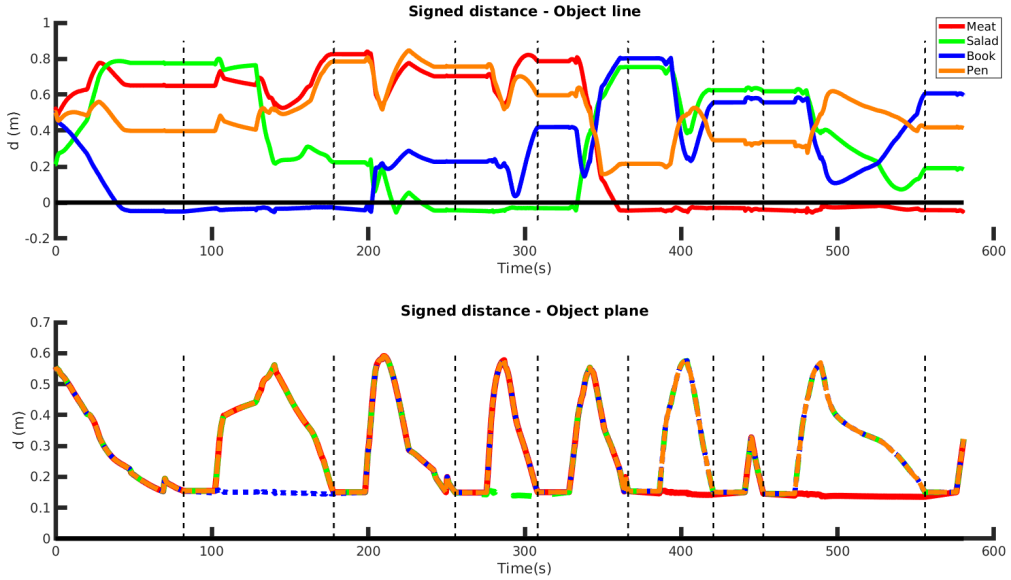


Figure 6.19: Plane constraints: signed distance between the end-effector and the objects. The dotted black lines indicate when an object is grasped or placed.

Figure 6.20 shows the signed distance between the end-effector and the robot z-axis. It remains positive during the whole task execution indicating that the constraint was satisfied.

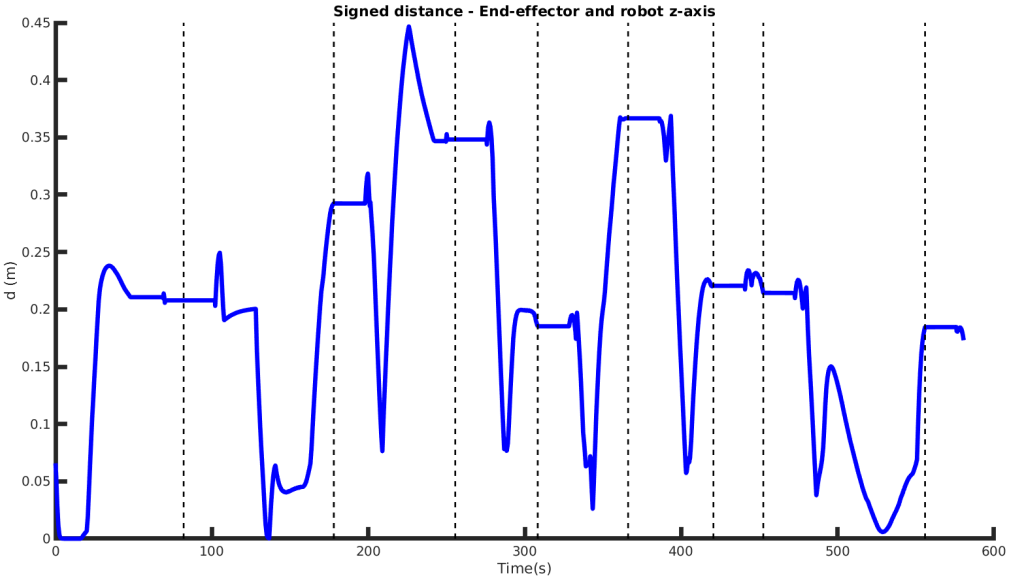


Figure 6.20: Plane constraints: signed distance between the end-effector and the robot z-axis. The dotted black lines indicate when an object is grasped or placed.

As before, there is also the line-cone constraint that limits the end-effector z-axis within a cone about a static line in the workspace (Quiroz-Omana and Adorno, 2019). Figure 6.21

shows that the angle between the end-effector z-axis and the static line remains within the safety angle.

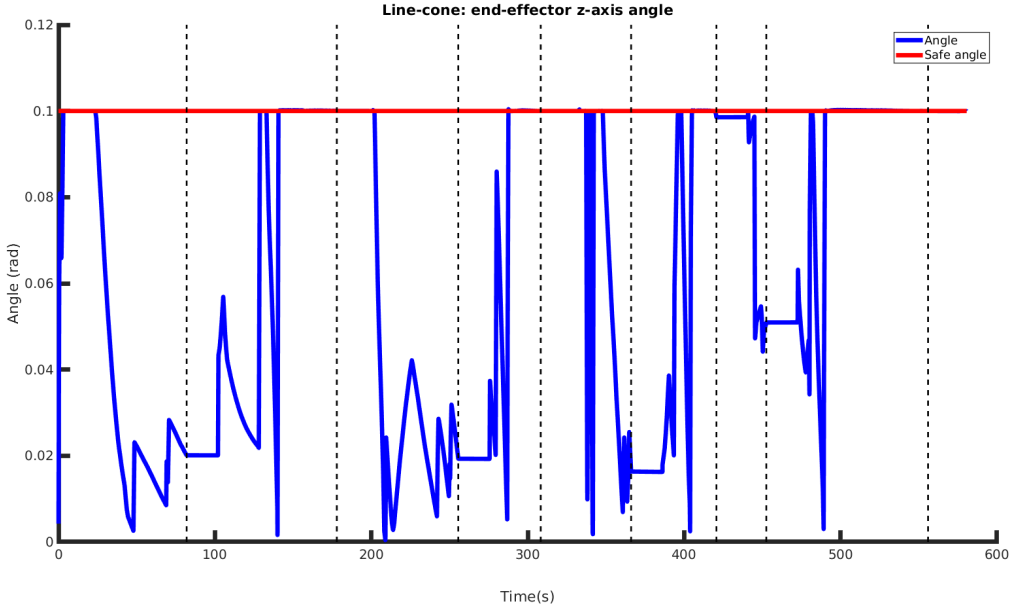


Figure 6.21: Plane constraints: line-cone constraint angle between end effector z-axis and static line parallel to robot z-axis. The dotted black lines indicate when an object is grasped or placed.

Last, Figure 6.22 shows the joints limits during the task. All joints angles are kept within the joints limits indicating that the constraint is satisfied.

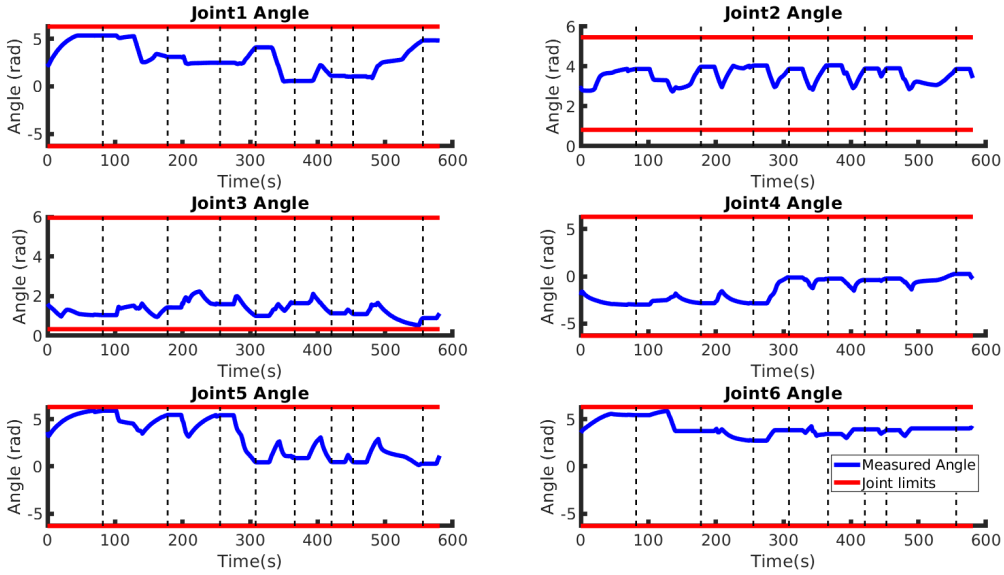


Figure 6.22: Plane constraints: joints limits constraint. The dotted black lines indicate when an object is grasped or placed.

Inverted cone trunk region of interest Analogously to the inverted pyramid trunk region of interest, the same analysis can be made for the inverted cone trunk region of interest using the point-cone constraints. Figures 6.23-6.25 shows that the signed distances remain negative when expected. Figure 6.26 shows that the line-cone constraint is satisfied. Last, all joints remains within the joints limits in Figure 6.27.

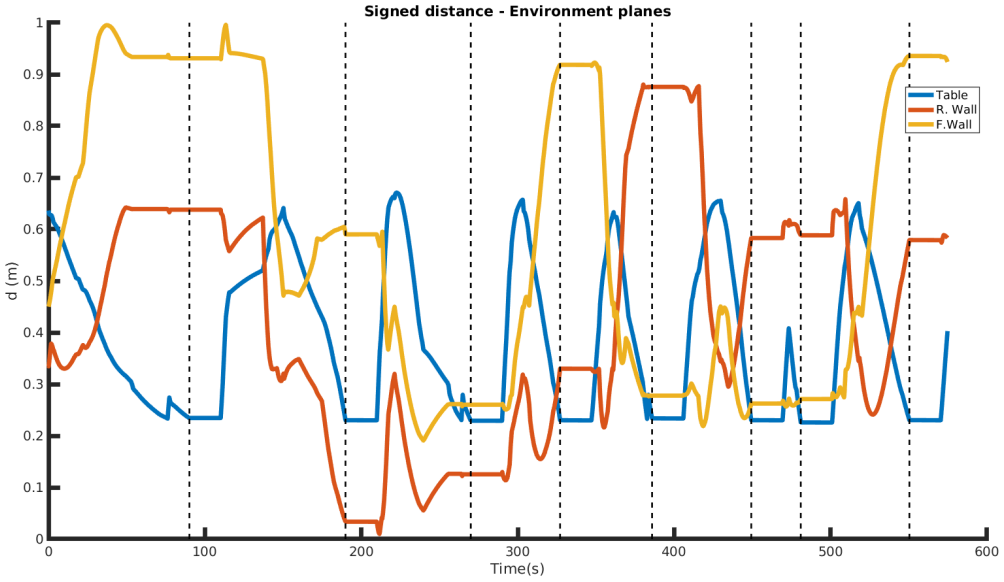


Figure 6.23: Point-cone constraint: signed distance between the end-effector and the planes in the environment (table top, right wall, and front wall). The dotted black lines indicate when an object is grasped or placed.

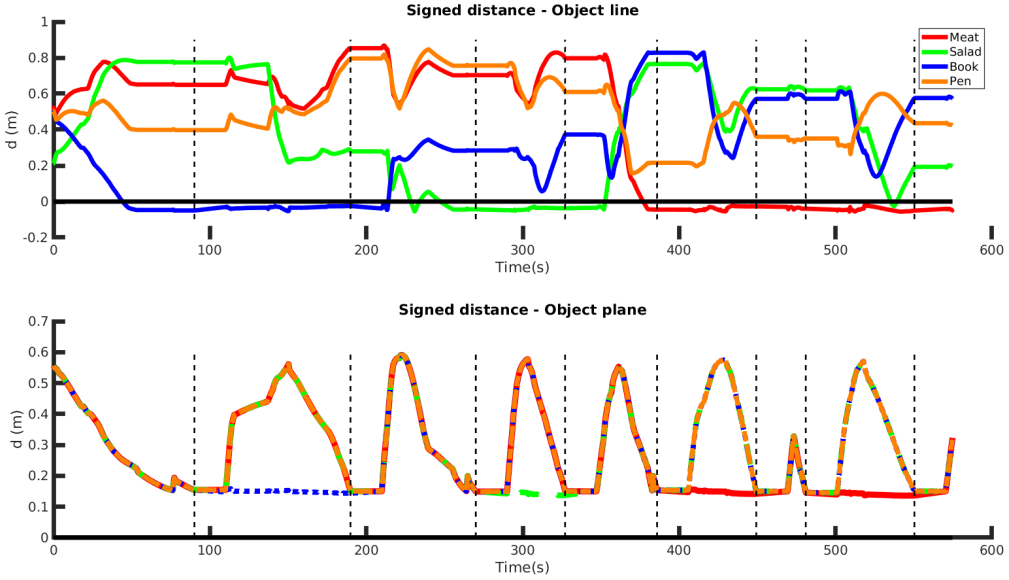


Figure 6.24: Point-cone constraint: signed distance between the end-effector and the objects. The dotted black lines indicate when an object is grasped or placed.

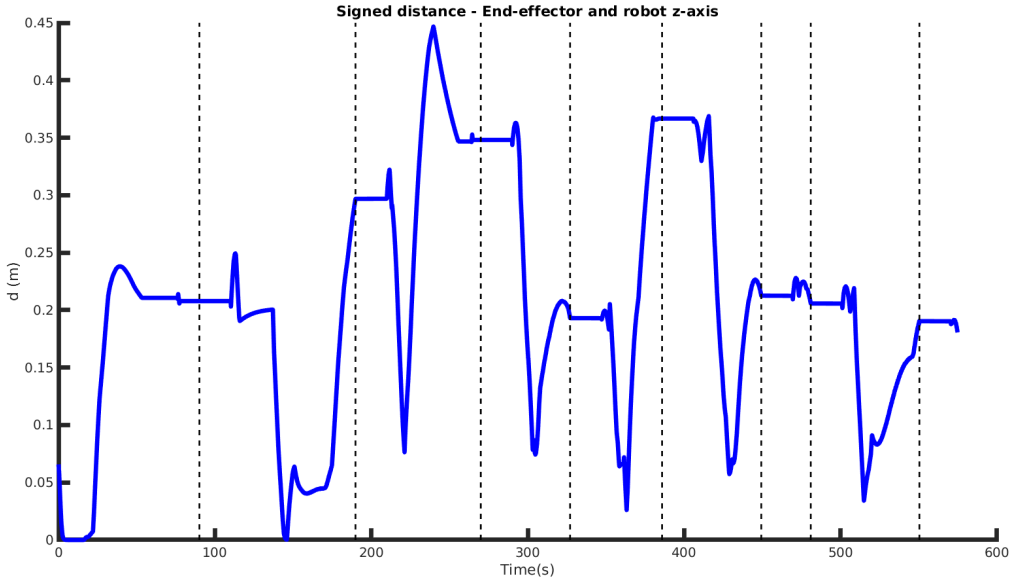


Figure 6.25: Point-cone constraint: signed distance between the end-effector and the robot z-axis. The dotted black lines indicate when an object is grasped or placed.

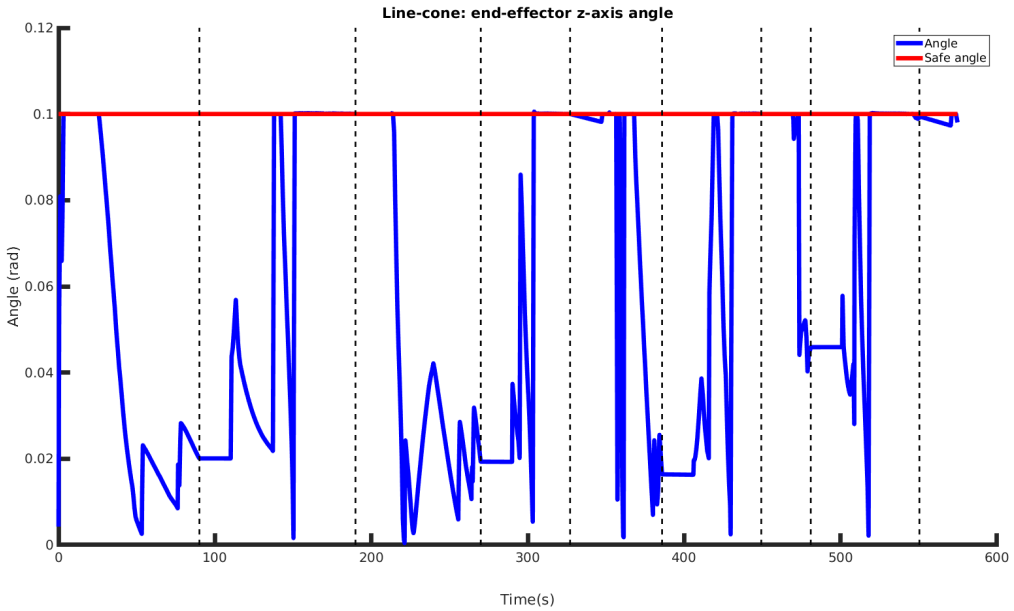


Figure 6.26: Point-cone constraint: line-cone constraint angle between end effector z-axis and static line parallel to robot z-axis. The dotted black lines indicate when an object is grasped or placed.

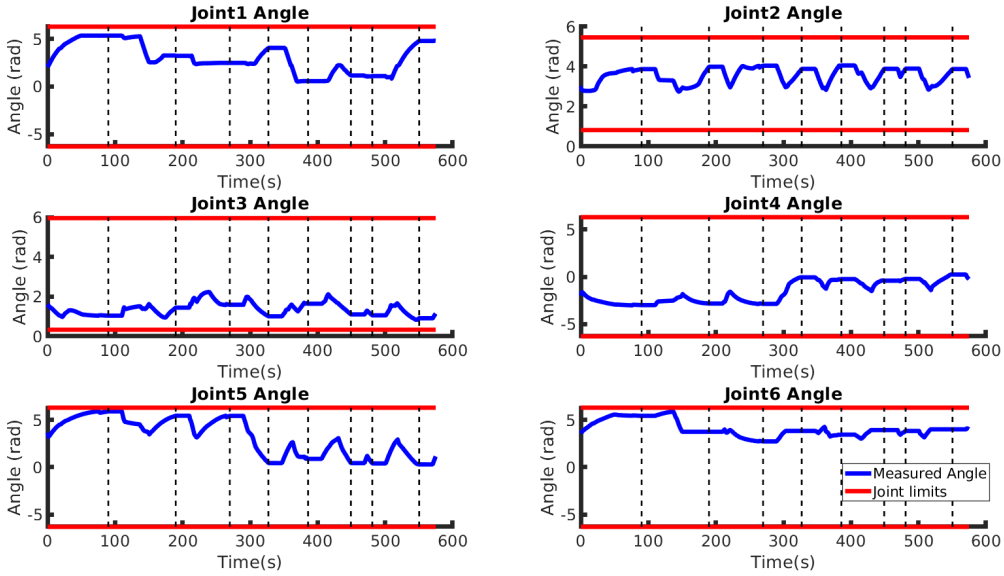


Figure 6.27: Point-cone constraint: joints limits constraint. The dotted black lines indicate when an object is grasped or placed.

6.3.5 Evaluation of Plane and Point-Cone Constraints Time Performance

At the end of Section 4.2.4 it was mentioned that the point-cone constraint requires less complex calculations during running time than the plane constraints. Recall that the point-cone constraint is used to define the circular target region and four plane constraints are used to define the squared target region. To verify the time performance of both constraints we calculated the constraint matrices and signed distances for each of them and measured the time duration. We calculated the Jacobian matrix $\boldsymbol{J}_{\text{cone},\boldsymbol{p}}$ and the signed squared distance $\tilde{D}_l$ for the point-cone constraint. For the plane constraints, we calculated four Jacobian matrices $\boldsymbol{J}_{p,n_{\underline{\pi}}}$ and four signed distances $\tilde{d}_{p,n_{\underline{\pi}}}$. Table 6.4 shows the time performance results for both constraints. The calculation time $T_{\text{calculation}}$ with the associated standard deviation (s.d.) corresponds to an average of 10000 calculations with random joint configurations. As expected, the calculation time for the point-cone constraint is lower than the calculation time for the four plane constraints.

Table 6.4: Constraints time performance.

Constraint	$T_{\text{calculation}}$(ms)
Plane constraints	0.180781 (0.0357274)
Point-cone constraint	0.0513916 (0.0105354)

$T_{\text{calculation}}$ (s.d.) is the total calculation time for the necessary constraint matrices and signed distances.

6.4 Conclusion

This chapter evaluated the adapted planning framework by devising four tasks with increasing complexity. The results show that the total planning time for the tasks is reduced in comparison to the original framework implementation since there is no need for additional low-level motion planning time. Moreover, the adapted approach greatly reduced the number of generated task plan nodes, which also reduced the planning time. Concerning task relaxation, the end-effector trajectory remained within the desired regions of interest and moved towards the target regions on the target plane while satisfying the imposed additional constraints. With regard to the time performance of the point-cone and plane constraints, the results show that the calculation time for the point-cone constraint is 71.57% lower than for the plane constraints. Lastly, since we define regions of interest instead of single locations of interest, the number of states in the planner is reduced thanks to a smaller discretization of the workspace.

7

Conclusion and Future Works

7.1 Conclusions

Motivated by the challenge of making a robot manipulator autonomously complete a task based on a given specification, in this work we studied how to specify, plan, and execute a manipulation task. This problem is called the integration of task and motion planning (ITMP). More specifically, we were concerned with the generation and execution of task plans from a given specification for pick-and-place of objects that can be placed anywhere within regions of interest. This kind of task constantly appears in the industry when it is needed to sort objects between locations. In some cases, the locations can be interpreted as a region as in the case of placing objects within boxes. Similarly, manipulators working at homes also face such situations when placing objects within drawers, boxes, countertops, etc. In this work, we focused on an assistive robotics application that requires a robot to do pick-and-place relaxed tasks for a person with motor disabilities. The person has limited or no lower limbs mobility and cannot reach farther objects in the environment. By using regions of interest instead of discrete poses, we focused on solving the ITMP problem for tasks that can be relaxed. To solve the ITMP problem we combine state of the art techniques presented in Chapter 2.

The task planning is done by using the manipulation planning framework proposed by He et al. (2015), which is described in Chapter 3. The framework receives a task specification based on linear temporal logic (LTL) that allows expressing the order of execution of actions by using temporal operators. In this sense, the chapter begins by

introducing the syntax and semantics of LTL. The second step in the framework is to create a suitable manipulation abstraction that describes how the manipulator can manipulate objects in the environment. Next, the LTL formula is converted into a deterministic finite automaton (DFA) that specifies all the ways the manipulator can complete the task. In the sequence, the abstraction is combined with the DFA into a product graph that contains all the possible sequences of actions the manipulator must execute to finish the task while satisfying the task specification. Afterward, Dijkstra's algorithm is used to search for the shortest task plan in the product graph. At this point, instead of using a coordinating and a motion planning layer as in the original framework, we propose to use the constrained motion controller proposed by Marinho et al. (2019) to execute the task plan. This way, no additional motion planning time is needed besides the task planning time.

The constrained motion controller (Marinho et al., 2019), presented in Chapter 4, allows, among other control objectives, to control the end-effector pose and the end-effector distance to a target plane. Furthermore, it also enables defining constraints such as obstacles in the workspace and joint limits. This way, we prevent collision with objects by using cylindrical constraints and constrain the end-effector to regions of interest while it moves towards a pose or a target plane by using plane and conical constraints. An additional cylindrical constraint was added to the z-axis of the manipulator coordinating frame to avoid twisted configurations. Four plane constraints were used to define a squared target region similar to the work of Quiroz-Omana and Adorno (2019) and a new approach to define conic constraints, called point-cone constraint, was used to define a circular target region. The advantage of the point-cone constraint is that it requires less complex calculations during runtime. Besides the aforementioned constraints, a line-cone constraint (Quiroz-Omana and Adorno, 2019) was added to constrain the end-effector z-axis within a cone and joint limits constraints were added to prevent saturation of actuators (Quiroz-Omaña and Adorno, 2018). After defining all constraints, the relaxed task regions of interest were defined in Chapter 5.

In Chapter 5, the relaxed regions of interest were defined using geometric primitives represented by elements of dual quaternion algebra. A brief introduction to dual quaternion algebra and distances between geometric primitives was presented. Basic geometric primitives such as lines, planes, and half-spaces were used to derive infinite cylinders, line segments, circles, and cones. Afterward, they were combined to obtain an inverted pyramid trunk in the case of the squared target region and an inverted cone trunk in the case of the circular target region. In this sense, the end-effector can be constrained to one of these regions of interest. We focused on relaxing the action of holding an object from the intermediate location to the target region of interest. However, by analyzing the other actions, more steps could also be relaxed. From the point of view of the task planner, the implemented planner task relaxation cannot be efficiently done by discretizing the workspace into multiple locations because an increase in the number of locations in the

workspace results in an increase in the size of the manipulation abstraction. An increase in the number of nodes in the abstraction results in longer planning time. Hence, the definition of regions of interest is a computationally efficient way to relax the tasks and still take advantage of the capabilities of the planner.

To evaluate both our approach to do task relaxation and the planner capabilities, simulations were done in the CoppeliaSim with a Kinova Jaco robot in an assistive environment, and the results were presented in Chapter 6. We devised four planning tasks for the planner with increasing complexity. The planner found solutions for all the tasks and used a reduced total planning time in comparison to the original planning framework proposed by He et al. (2015) since no additional motion planning time is needed. Moreover, there is no need to search for more than one task plan due to changes in the scene, in terms of the modeled geometric primitives, as long as there are mechanisms to track their changes. As a consequence, there is no increase in the number of generated planning nodes during the task planning phase, and Dijkstra's algorithm searches for a task plan on a static graph. With regard to task relaxation, all the tasks were executed relaxing the placing of objects to a target region. Both the squared and circular target region approaches were tested. It was shown that the manipulator remains within the defined region of interest and moves toward the target region on the target plane. Furthermore, the evaluation of the time performance of the constraints showed that the point-cone constraint requires less calculation time than the plane constraints.

As discussed above and evaluated in Chapter 6, the adapted framework presents advantages in comparison to the original framework concerning planning time, size of the abstraction and the possibility to relax tasks. Furthermore, the system is reactive. However, the task planner with the constrained motion controller also has disadvantages. By removing the motion planning layer, the method is not probabilistic complete. Moreover, the constrained motion controller may suffer from local minima. With regard to planning complexity although it is possible to relax tasks without incurring in an increase of computational complexity, if the number of objects and regions of interest are increased, the size of the abstraction quickly grows. Thus, there is an increase in the high-level planning time. Additionally, an increase in the complexity of the specification results in a larger automaton demanding more time in the conversion from the LTL formula to the automaton. From the application point of view, the system needs the information about all the objects poses in the environment. While in the original framework the information is used to build a map for the motion planner, in the adapted framework this information is used to correctly define the constraints. In this sense, the system is reactive to changes in the environment, in terms of the modeled geometric primitives, as long as there are mechanisms to track their changes.

During the development of this thesis other similar state of the art techniques also appeared. As mentioned in Section 2.2, there is a work that used the framework of He et al.

(2015) as a benchmark for ITMP. Dantam et al. (2018) propose an ITMP method that uses planning domain description language (PDDL) and satisfiability modulo theories (SMT) to solve the task planning problem and sampling-based motion planners to do motion planning. For scenarios with multiple objects, it achieves a better time performance than the framework of He et al. (2015). However, the method strongly couples the motion planner with the task planning phase by using the information of failed motion plans in the SMT solver. This way, it would not be possible to eliminate motion planning time. In this sense, one can choose between having less task planning time and needing motion planning time or having only a longer task planning time. We have chosen the latter and combined it with the constrained motion controller that allows alleviating the problem of increasing the number of locations by doing task relaxation. Concerning the planning domain description, LTL allows expressing tasks with temporal conditions for the objects without the need of considering the action to be executed while PDDL associates an action and effect on each object. Hence, the PDDL framework requires the whole task to be specified whereas the LTL framework requires only the specification of the locations for each object. Once again, we have chosen the latter to avoid needing to describe every action for each task.

Besides the work of Dantam et al. (2018), the work of He et al. (2019b) proposes a reactive synthesis planning framework that allows considering human interference on the task being executed by the manipulator. Instead of using a manipulation abstraction, the work uses PDDL to describe the planning domain, that is, the actions of the robot and the human. Also, the robot task specification is given by an LTL formula that is converted into a DFA. The novelty of that work is that the PDDL actions are converted into a binary decision diagram (BDD) and, afterward, the BDD is combined with the DFA. Since the LTL task is often much smaller than the planning domain, the only bottleneck becomes the conversion of the LTL to a DFA. Last, a symbolic synthesis tool computes a task plan on the product between the BDD and the DFA. The results show that the conversion from the planning domain to BDD is orders-of-magnitude faster than combining the planning domain (i.e. the actions the human and the robot execute) and the task specification (i.e. the location of each object along the task execution) into a single LTL formula. Nonetheless, because we were not considering human interference in the robot task, we have chosen the framework of He et al. (2015).

In addition to the works of He et al. (2015), Dantam et al. (2018), and He et al. (2019b), the work of Lindemann (2018) appears as a possibility to solve the task planning problem without the need for motion planning and an abstraction. Thus, it is more computationally efficient because it does not require the discretization of the system into an abstraction. Lindemann (2018) proposes to generate control laws that satisfy temporal logic specifications for dynamical systems. However, it is a more general approach for both single- and multi-agents systems and not specific to the manipulation problem. Hence, it

requires more steps to associate its control inputs to manipulation actions. Moreover, it uses signal temporal logic (STL) that allows not only the temporal ordering of tasks, but also timing duration of certain operators. The adaption of the work of Lindemann (2018) to manipulation task planning remains to future works.

7.2 Future works

To enhance the current framework, there are improvements and evaluations that can be made both in the implementation and in the used technique. In the short term, the following issues could be resolved:

- The conversion from the LTL formula into a DFA is currently generating a DFA with transitions that are not physically feasible. For instance, in the automaton from Figure 3.4, there are transitions that require an object to be into multiple locations at the same time. Hence, after the DFA is obtained, a possible improvement would be to add a pruning step to remove undesired transitions. Alternatively, the DFA construction could already take into account the not physically feasible transitions;

- Also regarding the transitions in the DFA: the way we implemented the application of the labeling function checks all propositions in each state of the automaton. From the application of the labeling function, a letter that causes a transition in the automaton is obtained. Next, the transition that accepts the letter occurs. However, there are transitions that depend only on a few propositions. If those propositions are valid, the transition occurs. Nonetheless, since we check all the propositions, there may be extra valid propositions that are not needed for the transition, but the transition still occurs. This may cause unwanted behaviors in task execution. For instance, if a transition requires only propositions p_0 and p_5, but propositions p_2 and p_3 are also valid, the objects that depend on p_2 and p_3 will also be moved in addition to the objects that depend on p_0 and p_5. Currently, we overcome this problem by being redundant in the task specification: we specify what the robot must do and must not do (see the tasks 2, 3, and 4 in Chapter 6). To fix this issue, only the propositions used in each transition should be checked for a transition to happen;

- The tasks evaluated in this work consider that each region is only occupied by one object at a time. Further experiments could be done to place multiple objects in the same region simultaneously.

On the long term, the following steps could be done:

- The framework could be tested on a real robot by starting with a similar scenario to Figure 6.1 as long as there are a vision system that can track all the locations and objects poses;

- Currently, when an object is being transferred between two locations, only one phase of the transferring process is being relaxed. However, a closer look reveals that most steps could be relaxed by controlling the end-effector distance to a plane (see Table 4.1 and Table 5.1);

- The LTL formula is converted into a DFA by the Spot library (Duret-Lutz and Poitrenaud, 2004) and the automata utilities from OMPL (Sucan et al., 2012) are being used. To reduce the dependency on external libraries, an automata library could be written to accommodate the DFA generated by the Spot library;

- The conversion from the LTL into a DFA could be studied. Depending on the conversion procedure difficulty, it could be implemented. This way, there would be no need of using the Spot library (Duret-Lutz and Poitrenaud, 2004);

- Other ITMP frameworks could be explored. Dantam et al. (2018) propose a framework with better time performance than the framework of He et al. (2015) and He et al. (2019b) propose a framework that is also LTL-based and considers human interference in the task.

- The use of LTL with abstraction-based methods usually discretize the system in space to obtain a finite state automaton. However, as can be seen from the current work, these methods suffers from the curse of dimensionality. With an increase in the number of objects and locations in the environment or the task complexity, the abstraction quickly grows leading to an intractable problem. A possible solution for this problem is to use abstraction-free frameworks as proposed by Lindemann (2018) in which control laws are derived from temporal logic specifications, thus, combining the expressiveness of temporal logic with the robustness of different control methods.

Bibliography

Adorno, B. V. (2017). Robot Kinematic Modeling and Control Based on Dual Quaternion Algebra-Part I: Fundamentals. Technical report, <hal-01478225>.

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

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

Bedrossian, N. (1990). Classification of singular configurations for redundant manipulators. In *Proceedings., IEEE Int. Conf. Robot. Autom.*, pages 818–823. IEEE Comput. Soc. Press.

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

Bryant (1986). Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Trans. Comput.*, C-35(8):677–691.

Cambon, S., Alami, R., and Gravot, F. (2009). A Hybrid Approach to Intricate Motion, Manipulation and Task Planning. *Int. J. Rob. Res.*, 28(1):104–126.

Cha, E., Forlizzi, J., and Srinivasa, S. (2015). Robots in the Home: Qualitative and Quantitative Insights into Kitchen Organization. In *Proc. 2015 Int. Conf. Human-Robot Interact. (HRI 2015)*.

Choset, H. M., Lynch, K., Hutchinson, S., Kantor, G. A., Burgard, W., Kavraki, L. E., and Thrun, S. (2005). *Principles of robot motion : theory, algorithms, and implementation*. MIT Press.

Dantam, N. T., Kingston, Z. K., Chaudhuri, S., and Kavraki, L. E. (2018). An incremental constraint-based framework for task and motion planning. *Int. J. Rob. Res.*, 37(10).

Dornhege, C., Gissler, M., Teschner, M., and Nebel, B. (2009). Integrating symbolic and geometric planning for mobile manipulation. In *2009 IEEE Int. Work. Safety, Secur. Rescue Robot. (SSRR 2009)*, pages 1–6. IEEE.

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

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

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

Figueredo, L., Adorno, B., Ishihara, J., and Borges, G. (2014). Switching strategy for flexible task execution using the cooperative dual task-space framework. In *2014 IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, pages 1703–1709. IEEE.

Figueredo C., L. F. (2016). *Kinematic control based on dual quaternion algebra and its application to robot manipulators*. PhD thesis, Universidade de Brasília.

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

Garrett, C. R., Lozano-Pérez, T., and Kaelbling, L. P. (2018). FFRob: Leveraging symbolic planning for efficient task and motion planning. *Int. J. Rob. Res.*, 37(1):104–136.

Gelfond, M. and Lifschitz, V. (1998). Action Languages. *Electron. Trans. AI*, 3.

Ghallab, M., Knoblock, C., Wilkins, D., Barrett, A., Christianson, D., Friedman, M., Kwok, C., Golden, K., Penberthy, S., Smith, D., Sun, Y., and Weld, D. (1998). PDDL - The Planning Domain Definition Language.

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

Hager, G., Okamura, A., Kazanzides, P., Whitcomb, L., Fichtinger, G., and Taylor, R. (2008). Surgical and interventional robotics: part III [Tutorial]. *IEEE Robot. Autom. Mag.*, 15(4):84–93.

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

He, K., Lahijanian, M., Kavraki, L. E., and Vardi, M. Y. (2017). Reactive synthesis for finite tasks under resource constraints. In *2017 IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, pages 5326–5332. IEEE.

He, K., Lahijanian, M., Kavraki, L. E., and Vardi, M. Y. (2019a). Automated Abstraction of Manipulation Domains for Cost-Based Reactive Synthesis. *IEEE Robot. Autom. Lett.*, 4(2):285–292.

He, K., Wells, A. M., Kavraki, L. E., and Vardi, M. Y. (2019b). Efficient Symbolic Reactive Synthesis for Finite-Horizon Tasks. In *2019 Int. Conf. Robot. Autom.*, pages 8993–8999. IEEE.

Iskandar, M., Quere, G., Hagengruber, A., Dietrich, A., and Vogel, J. (2019). Employing Whole-Body Control in Assistive Robotics. In *2019 IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, pages 5643–5650.

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

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

Khatib, O., Yeh, X., Brantner, G., Soe, B., Kim, B., Ganguly, S., Stuart, H., Wang, S., Cutkosky, M., Edsinger, A., Mullins, P., Barham, M., Voolstra, C. R., Salama, K. N., L'Hour, M., and Creuze, V. (2016). Ocean One: A Robotic Avatar for Oceanic Discovery. *IEEE Robot. Autom. Mag.*, 23(4):20–29.

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

Kloetzer, M. and Mahulea, C. (2015). LTL-Based Planning in Environments With Probabilistic Observations. *IEEE Trans. Autom. Sci. Eng.*, 12(4):1407–1420.

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

Kuban, D. P. and Martin, H. L. (1984). Advanced remotely maintainable force-reflecting servomanipulator concept.

Kuffner, J. and LaValle, S. (2000). RRT-connect: An efficient approach to single-query path planning. In *Proc. 2000 ICRA. Millenn. Conf. IEEE Int. Conf. Robot. Autom. Symp. Proc. (Cat. No.00CH37065)*, volume 2, pages 995–1001. IEEE.

Kundu, T. and Saha, I. (2019). Energy-Aware Temporal Logic Motion Planning for Mobile Robots. In *2019 Int. Conf. Robot. Autom.*, pages 8599–8605. IEEE.

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

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

Laryssa, P., Lindsay, E., Layi, O., Marius, O., Nara, K., Aris, L., and Ed, T. (2002). International Space Station Robotics: A Comparative Study of ERA, JEMRMS and MSS. In *7th ESA Work. Adv. Sp. Technol. Robot. Autom. 'ASTRA 2002'*, Noordwijk.

Latombe, J.-C. and Jean-Claude (1991). *Robot motion planning*. Kluwer Academic Publishers.

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

Liégeois, A. (1977). Automatic Supervisory Control of the Configuration and Behavior of Multibody Mechanisms. *IEEE Trans. Syst. Man. Cybern.*, 7(12):868–871.

Lindemann, L. (2018). *Robust and Abstraction-free Control of Dynamical Systems under Signal Temporal Logic Tasks*. PhD thesis, KTH, School of Electrical Engineering and Computer Science (EECS).

Lozano-Perez, T., Jones, J., Mazer, E., O'Donnell, P., Grimson, W., Tournassoud, P., and Lanusse, A. (1987). Handey: A robot system that recognizes, plans, and manipulates. In *Proceedings. 1987 IEEE Int. Conf. Robot. Autom.*, volume 4, pages 843–849. Institute of Electrical and Electronics Engineers.

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

Mansard, N. and Chaumette, F. (2009). Directional Redundancy for Robot Control. *IEEE Trans. Automat. Contr.*, 54(6):1179–1192.

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

McMahon, J. and Plaku, E. (2014). Sampling-based tree search with discrete abstractions for motion planning with dynamics and temporal logic. In *2014 IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, pages 3726–3733. IEEE.

Plaku, E. and Hager, G. D. (2010). Sampling-Based Motion and Symbolic Action Planning with geometric and differential constraints. In *2010 IEEE Int. Conf. Robot. Autom.*, pages 5002–5008. IEEE.

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

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

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

Sucan, I. A. and Kavraki, L. E. (2012). A Sampling-Based Tree Planner for Systems With Complex Dynamics. *IEEE Trans. Robot.*, 28(1):116–131.

Sucan, I. A., Moll, M., and Kavraki, L. E. (2012). The Open Motion Planning Library. *Robot. Autom. Mag.*, 19(4):72–82.

Vardi, M. and Wolper, P. (1994). Reasoning about Infinite Computations. *Inf. Comput.*, 115(1):1–37.

Vardi, M. Y. (1996). An automata-theoretic approach to linear temporal logic. pages 238–266. Springer, Berlin, Heidelberg.

W. Spong, M., Hutchinson, S., and Vidyasagar, M. (2005). *Robot Modeling and Control - Mark W. Spong, Seth Hutchinson, M. Vidyasagar*.

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