UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Centre de Formació Interdisciplinària Superior

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona

telecos
BCN

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat de Matemàtiques i Estadística

Massachusetts
Institute of
Technology

Bachelor's degree thesis

A Reactive Planning Framework for Dexterous Robotic Manipulation

José Andrés Ballester Huesca

Advised by:

Alberto Rodríguez García (MIT)
Maria Alberich Carramiñana (UPC)

In partial fulfillment of the requirements for the

Bachelor's degree in Mathematics

Bachelor's degree in Telecommunications Technologies and Services Engineering

May 2019

Abstract

A Reactive Planning Framework for Dexterous Robotic Manipulation

by José Andrés Ballester Huesca

This thesis investigates a reactive motion planning and controller framework that enables robots to manipulate objects dexterously. We develop a robotic platform that can quickly and reliably replan actions based on sensed information. Robotic manipulation is subject to noise due to uncertainty in frictional contact information, and reactivity is key for robustness. The planning framework has been designed with generality in mind and naturally extends to a variety of robotic tasks, manipulators and sensors. This design is validated experimentally on an ABB IRB 14000 dual-arm industrial collaborative robot.

In this research, we are interested in dexterous robot manipulation, where the key technology is to move an object from an initial location to a desired configuration. The robot makes use of a high resolution tactile sensor to monitor the progress of the task and drive the reactive behavior of the robot to counter mistakes or unaccounted environment conditions. The motion planning framework is integrated with a task planner that dictates the high-level manipulation behavior of the robot, as well as a low-level controller, which adapts robot motions based on measured tactile signals. We show that the proposed planning framework, based on the MoveIt! platform and the EGM controller setup from ABB, is able to plan simultaneous dual-arm trajectories that are continuous and successful at avoiding collisions as well as replanning and executing local adjustments at a frequency of up to 250 Hz.

Keywords: Collaborative Robotics, Contact Manipulation, Dexterous Robots, Manipulation Primitives, Tactile Sensing, Reactive Control.

Resum

A Reactive Planning Framework for Dexterous Robotic Manipulation

per José Andrés Ballester Huesca

Aquesta tesi investiga un marc de planificació de trajectòries (*motion planning*) i controlador que permet que robots manipulin objectes de manera destre. Presentem una plataforma robòtica que és capaç de replanejar de manera ràpida i fiable accions basant-se en informació detectada de l'entorn. La manipulació robòtica està subjecta a soroll a causa d'incertesa en la informació de contacte de fricció, i la capacitat de reacció és clau per a un sistema robust. Aquest marc ha estat dissenyat amb l'objectiu de ser estès naturalment a una varietat de tasques, manipuladors i sensors robòtics. El seu disseny ha estat validat experimentalment amb un robot col·laboratiu industrial ABB IRB 14000 de dos braços.

En aquesta recerca, estem interessats en manipulació robòtica destra, on la tecnologia clau és moure un objecte d'una localització inicial a una configuració desitjada. El robot fa servir un sensor tàctil d'alta resolució per monitorejar el progrés de la tasca i conduir el comportament reactiu per a contrarestar errades o condicions ambientals que no s'havien tingut en compte. Aquest marc s'integra amb un planificador de tasques que dicta el comportament manipulatiu d'alt nivell del robot, i amb un controlador de baix nivell que adapta els moviments del robot a partir de senyals tàctils mesurats. Mostrem com el marc planificatiu proposat, basat en la plataforma MoveIt! i la tecnologia EGM d'ABB, és capaç de planificar trajectòries simultànies per a dos braços que són contínues i tenen èxit en evitar col·lisions, així com de replanejar i executar ajustaments locals a una freqüència de fins a 250 Hz.

Paraules clau: robòtica col·laborativa, manipulació amb contacte, robots destres, primitives de manipulació, detecció tàctil, control reactiu.

Resumen

A Reactive Planning Framework for Dexterous Robotic Manipulation

por José Andrés Ballester Huesca

Esta tesis investiga un marco de planificación de trayectorias (*motion planning*) y controlador que permite que robots manipulen objetos de manera diestra. Presentamos una plataforma robótica que es capaz de replanear de manera rápida y fiable acciones basándose en información detectada del entorno. La manipulación robótica está sujeta a ruido a causa de incertidumbre en la información de contacto de fricción, y la capacidad de reacción es clave para un sistema robusto. Este marco ha sido diseñado con el objetivo de extenderse naturalmente a múltiples tareas, manipuladores y sensores robóticos. Su diseño ha sido validado experimentalmente con un robot industrial colaborativo ABB IRB 14000 de dos brazos.

En esta investigación, estamos interesados en manipulación robótica diestra, donde la tecnología clave es mover un objeto de una localización inicial a una configuración deseada. El robot utiliza un sensor táctil de alta resolución para monitorear el progreso de la tarea y conducir el comportamiento reactivo para contrarrestar errores o condiciones ambientales que no se habían tenido en cuenta. Este marco se integra con un planificador de tareas que dicta el comportamiento manipulativo de alto nivel del robot, y con un controlador de bajo nivel que adapta los movimientos del robot a partir de señales táctiles medidas. Mostramos como el sistema propuesto, basado en la plataforma MoveIt! y la tecnología EGM de ABB, es capaz de planificar trayectorias simultáneas para dos brazos que son continuas y evitan colisiones, así como de replanear y ejecutar ajustes locales a una frecuencia de hasta 250 Hz.

Palabras clave: robótica colaborativa, manipulación con contacto, robots diestros, primitivas de manipulación, detección táctil, control reactivo.

Acknowledgements

First of all, I would like to sincerely thank Prof. Alberto Rodríguez for giving me the opportunity to be an active part of the MCube Lab, and also for many insightful conversations about robotics and life. I would like to extend my gratitude to François and Bernardo, who I have had the pleasure to work side by side with, and the rest of the MCube team.

I would also like to acknowledge the helping hand of Prof. Maria Alberich back at UPC, who has closely followed my progress and offered great guidance during my stay at MIT.

This experience abroad could not have been possible without the support from CFIS, especially Prof. Miguel Ángel Barja and Prof. Toni Pascual, from MIT and from the Generalitat de Catalunya. A more than special appreciation to Fundació Privada Cellex, and the trust they have placed in me during all these years, which goes far beyond financial support.

This thesis represents the culmination of an intellectual and personal adventure that started five years ago and has brought me to meet great friends. I am really thankful for all I have learned from them, and especially my Cambridge family (Martí, Mik, Núria and Oriol) and my double degree mate, Carlos. This is one of the very few works done without him.

Finally, I would like to acknowledge the unconditional support and understanding from my family throughout all these years, and for all the experiences that still have to come.

José Andrés Ballester Huesca
Cambridge, May 17, 2019

Preface

This work has been developed during a nine-month research internship at the Manipulation and Mechanisms Laboratory (MCube Lab) at the Massachusetts Institute of Technology (MIT), led by Prof. Alberto Rodríguez. At the same time, Prof. Maria Alberich from UPC has been the link with my home university and has also supervised this work.

As a result, I have been able to conduct research in two main aspects of robotic manipulation:

1. Convex-combinatorial models for planar caging and grasping, and their application to provide success certificates for open-loop planar grasps with bounded pose uncertainty.

   This has led to a submitted paper [4] to the International Symposium on Robotics Research (ISRR), and has been useful to understand some concepts of this thesis.

2. A novel tactile dexterity framework for robust dual-arm manipulation based on contact rich primitives, used for efficient planning and tactile-based reactive control. This is an ambitious and exciting ongoing project that is expected to result in a conference paper and potentially extended to a journal paper.

   In particular, this thesis covers the development of a reactive dual-arm motion planning system and a high frequency robot controller, which are key to enable reactivity in the *tactile dexterity* framework but can be generalized to other setups. For example, the latter system has already been used in several other projects at the MCube Lab.

My coursework at UPC has been particularly useful to face the challenges listed above. This includes mathematical programming, numerical calculus, data structures and algorithms, linear algebra and geometry from the Degree in Mathematics; and also electronics, signal processing, networking and system integration, with a special mention to the Conceive, Design, Implement and Operate (CDIO) initiative, from the Degree in Telecommunications Technologies and Services Engineering.

Contents

List of Figures

List of Acronyms

CHOMP	Covariant Hamiltonian Optimization for Motion Planning
DLS	Damped Least-Squares
DOF	Degrees of Freedom
EGM	External Guided Motion
FK	Forward Kinematics
HMI	Human-Machine Interface
IDL	Interface Description Language
IK	Inverse Kinematics
I/O	Input / Output
IP	Internet Protocol
KDL	Kinematics and Dynamics Library
LMA	Levenberg-Marquardt Algorithm
MPC	Model Predictive Control
OMPL	Open Motion Planning Library
PRM	Probabilistic Roadmap Method
RDT	Rapidly-exploring Dense Tree
ROS	Robot Operating System
RRI	Robot Reference Interface
RRT	Rapidly-exploring Random Tree
SQP	Sequential Quadratic Programming
SRDF	Semantic Robot Description Format
STL	Stereolithography
STOMP	Stochastic Trajectory Optimization for Motion Planning
SVD	Singular Value Decomposition
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URDF	Unified Robot Description Format
XML	Extensible Markup Language

Chapter 1

Introduction

1.1 Motivation for reactivity

Historically, the field of robotic manipulation has heavily relied on the knowledge of environmental conditions to achieve a desired task. A typical approach is to perceive, reason and plan actions based on the available model prior to task execution and then operate the robot in an open-loop fashion, without taking into account any sensed information from external sources during execution (Fig. 1.1).

Such approaches have shown to be successful in applications within structured scenarios, where a developer programs open-loop algorithms to be executed by the robot. In such cases, an important assumption is that reality remains unchanged, and therefore any situation slightly out of the programmed initial conditions, such as the one from Fig. 1.1, can lead to complete system failure, for example due to collisions.

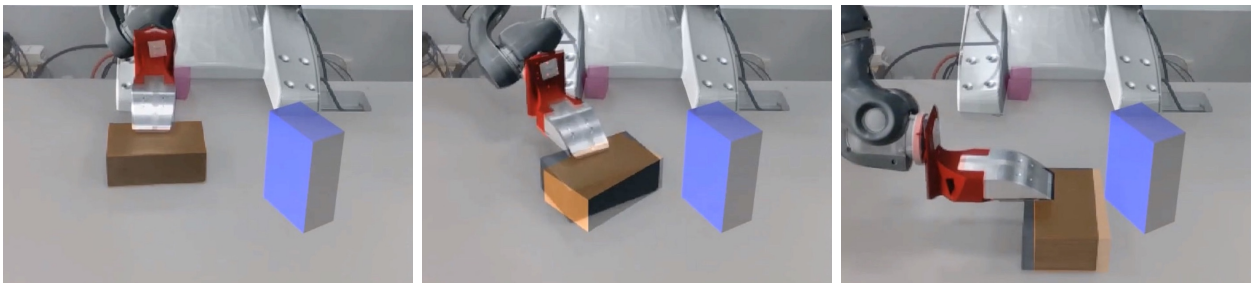


Figure 1.1: Open-loop execution of a manipulation task that leads to error in the object pose (black box) with respect to the planned pose (orange box).

Due to this, the development of reactive systems that can manage those previous steps of perceiving, understanding and planning is essential in order to achieve robust robotic skills. In other words, these systems would accept high level instructions and autonomously adapt their behavior to achieve a successful execution of those commands.

This challenge has received attention from multiple disciplines such as autonomous driving, locomotion or manipulation. In many cases, humans are taken as a reference when proposing solutions to it: not only we can easily understand our own behavior and model it, but also we are able to quickly approximate long term plans.

However, we are also able to process and react to information from our sensors in real time in order to compensate that original imprecision. We take this as a source of inspiration to design robotic manipulation skills that are approximate at a high-level but have reactive low-level capabilities to correct their behavior during execution time.

For example, collaborative robots, introduced in 1996 [27, 28], are intended to interact and perform manipulation tasks together with humans in a shared workspace. Therefore, they need to perceive their environment and react to uncertainty not only from their own executions, but also from the humans they are working with (Fig. 1.2).

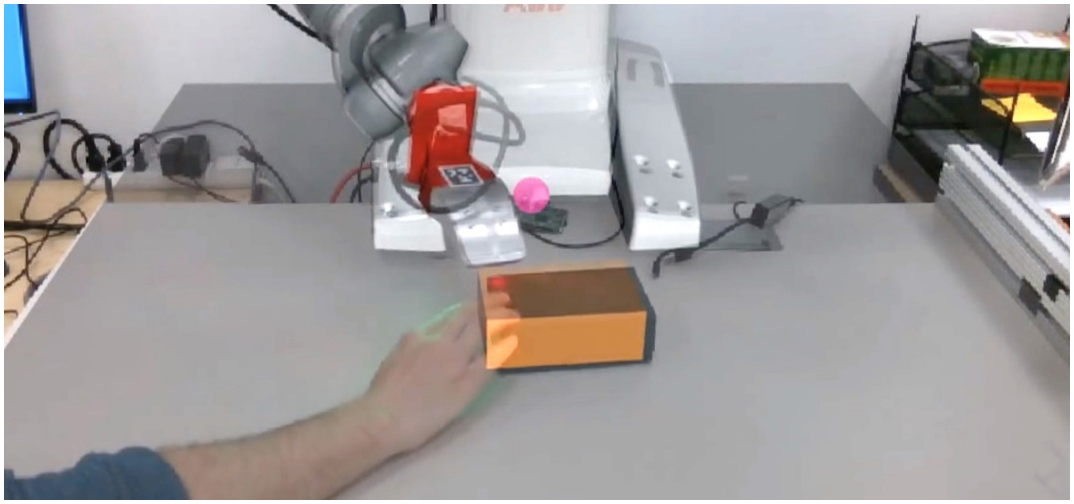


Figure 1.2: Collaborative robots should be able to perceive and react to uncertainty not only from their own executions, but also from the humans they are working with.

Perception has received an increasing interest from the robotics and computer vision communities, thanks to the latest advances in sensors, storage capacity and computation power. Nowadays, there exists a wide variety of input types, sensors that receive this information,

and algorithms that process it to obtain a relevant set of features.

In order to have robust robot interactions with the environment, it is important to have the ability of replanning motions according to new situations, which at the same time can be defined by several sensor data and/or processing pipelines. More specifically, robot motion planning and execution frameworks must include reactivity strategies to replan and execute motion with low delay and high frequency.

The aim of this thesis is to design, develop and implement a reactive motion planning and high frequency execution framework to enable dexterous robotic manipulation. This framework is developed for an ABB IRB 14000 (YuMi, Fig. 1.3) position-controlled dual-arm industrial collaborative robot with seven Degrees of Freedom (DOF) at each arm, although it can be easily exported for use in other ABB industrial robots.

Figure 1.3: Image of an ABB IRB 14000 (YuMi) dual-arm industrial collaborative robot.

This motion planning framework receives an original pose plan for the end-effector at each arm from which computes a full continuous motion plan for the robot's joints configuration. Both general and setup-specific limitations are considered, including self-collision and object collision awareness, joint limits and speed limits.

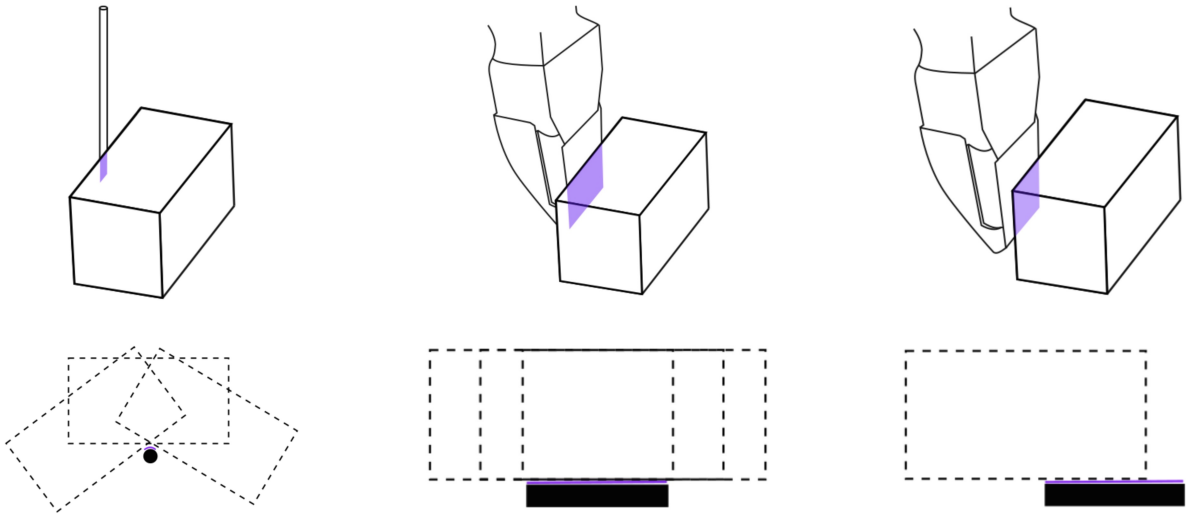
During execution, reactivity is made possible by the planner's ability of updating end-effector pose plans at a high frequency, leading to joint motion replanning using the original plan as a seed for online Inverse Kinematics (IK) computations. Finally, these reactive plans are executed in a high frequency loop by relying on an implementation of the External Guided Motion (EGM) controller setup developed by ABB.

1.2 Application to tactile dexterity

Our planning framework is designed to be easily generalized and incorporated into a wide variety of setups for many purposes, as it is independent of the nature of the end-effectors pose plan or the kind of sensor data used to reactively update those during execution.

From all sensing techniques, contact or tactile information is especially interesting in manipulation, as it offers a valuable perspective of the interaction between the manipulator and the object being manipulated. Examples of interesting interactions between robot and object include slip detection, force reconstruction, and object localization. More recently, vision-based tactile sensors with increasing resolution and decreasing size, such as GelSlim [35], have permitted the use of recent computer vision techniques to extract these contact features when available and take advantage of them for planning and control.

To this extent, extracting contact information from tactile imprints is not always possible and significantly depends on the contact *richness* of the robot-object interaction. At the same time, these interactions also influence how stable or determined some motions can be, e.g. pushing an object with a point or line pusher (Fig. 1.4).



(a) Point pushing resolves the object location along one dimension.

(b) Line pushing (centered) resolves the object location along two dimensions.

(c) Line pushing (corner) renders the state of the object fully observable.

Figure 1.4: Different object location resolution situations based on relative contact interactions between a pusher and an object. Reprinted from [42] with permission.

Therefore, aiming for robot motions that can lead to contact rich interactions seems a good strategy to make state estimation, planning and control easier. We coin this concept as tactile dexterity, the ability to control contact interactions and use tactile readings for extracting information and using feedback control.

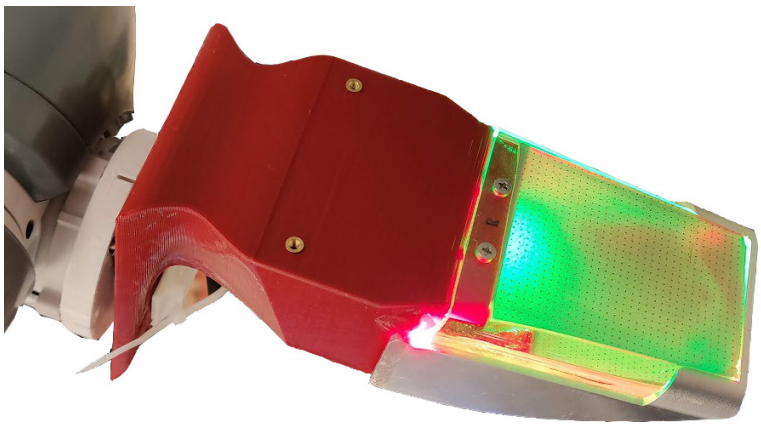


Figure 1.5: Detail of a GelSlim sensorized robotic palm used for tactile sensing.

Our robotic setup includes sensorized end-effectors at each arm: two robotic palms equipped with GelSlim tactile sensors (Fig. 1.5), that render high resolution images of the contact surface geometry (Fig. 1.6). When pressing an object against a GelSlim sensor, the camera located inside the palm captures the deformation of its external membrane. The raw image-based output can be processed via computer vision methods to obtain relevant contact features such as slip detection, deformation of the contact patch, etc.

This thesis also covers the integration of the developed framework with a high-level planner aiming at contact rich robot-object interactions. The goal of this project is to enable tactile dexterity by planning and exploiting contact rich interactions for state estimation and tactile-based reactive control.

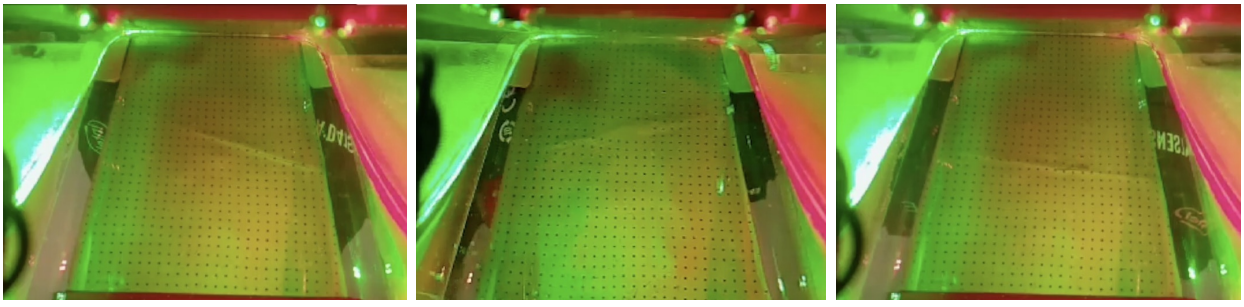


Figure 1.6: Raw outputs from a GelSlim sensor when pulling or grasping an object.

These big challenges can be decomposed into the following subproblems:

1. Offline planning framework:

   (a) Definition and modeling of several basic manipulation primitives to compute a stable original plan with contact rich interactions. Certain contact states and object interactions are assumed (and later enforced via the design of a tactile feedback controller) to output a pose plan for the two palms acting as end effectors.

   (b) Design and implementation of a high-level, long-term planner to decompose the original problem into a set of the previous manipulation primitives, ensuring compatibility and continuity during execution.

   (c) Implementation and evaluation of motion planning techniques to generate a complete, feasible plan for the robot joints resulting in the previous pose plan. This step requires continuity enforcement, awareness of YuMi's seven DOF and object and self-collision avoidance.

2. Online feedback controller:

   (a) Design of an object tracker (state estimation) based on GelSlim sensors output and feature extraction taking into account the original plan and the manipulation primitive being executed at every time.

   (b) Development of a reactive controller based on the previous information, to replan maximizing probability of success and updating future pose targets.

   (c) Implementation of an online motion planner based on a joint target updater from a fast IK solver, assuming a small pose deviation.

   (d) Development of a high frequency execution controller based on EGM, a recently launched communication interface offered by ABB, aimed to offer low delay commanding and possibilities for replanning.

This thesis covers the design and implementation of a reactive planning and execution framework for the tactile dexterity project. More specifically, solutions for steps 1(c), 2(c) and 2(d) are discussed to provide a functional operating framework for the project.

1.3 Approach and contributions

This thesis aims to design and provide a reactive motion planning and execution framework for an ABB IRB 14000 (YuMi) dual-arm collaborative robot.

Although this framework could later be used in many working scenarios to elaborate quickly updatable motion plans depending on the application, this thesis covers the application for enabling dexterity based on contact rich interactions and tactile sensing.

As a consequence, this project presents a theoretical overview and a technical implementation of the framework based on the following approach:

1. Implementation and evaluation of motion planning techniques to generate a plan for the robot joints leading to the desired end-effector pose plan output, provided at the same time by a high-level pose planner.

   Although heavily studied by the robotics community during the last few decades, fast motion planning remains challenging with no available closed-form solutions. Of particular interest to us are providing robust solutions to the following challenges:

   - Additional difficulty of having 7 DOF in every arm of the YuMi robot:

     The problem of finding a joint state leading to a desired end effector pose, named as IK, becomes significantly more difficult in this case due to an extra DOF (Cartesian space has 6 DOF, three for translation and three for orientation, while the space of possible robot joint configurations has now 7 DOF).

   - Consideration of continuity and non-punctual feasibility:

     As a consequence, the extra DOF now leads to a wide set of valid IK solutions (Fig. 4.1), some of which may be close to joint limits. For a punctual problem (a single timestep) an IK solution for a given pose may be valid, but it can make the following poses in the plan unreachable.[1] Hence, the motion planner cannot be a punctual IK solver, but a more complex program with a notion of future.

   - Collision awareness with the same robot and proper interaction with the object and/or the environment:

[1] In this context, an unreachable pose would be one that would imply a discontinuity or a significant variation of the current joint configuration. This could lead to a feasible solution not considered by the motion planner, or to an unfeasible solution due to the violation of other constraints.

In order to avoid errors and exceptions derived by collisions, collision avoidance has to be taken into account as a constraint, and becomes difficult when considering simultaneous motions for both of the robot arms. Furthermore, the possibility of having object motions due to robot motions (based on robot-object interactions) makes it necessary to enable or disable collision checks depending on the situation.

In this thesis, some well-known motion planning approaches are reviewed and adapted to set up a robust and efficient solution with the previous considerations.

2. Implementation of an online motion planner to dynamically update the joint targets according to the feedback-based pose targets.

   In this case, although the continuity and non-punctual feasibility constraints still hold, the implementation of a greedy IK solver can be enough. We assume that the pose deviation between the original pose plan and the feedback-based pose plan is small, and therefore using the offline joint plan as a seed for the IK solver will lead to similar solutions that satisfy the constraints as well.

3. Development of a high frequency execution controller based on EGM, a tool introduced by the robot manufacturer, ABB.

   This step covers the actual plan execution, that is the communication of instantaneous joint targets between a commanding computer and the industrial robot itself. In order to make this possible, a first introduction to the EGM framework is done, to understand the communication protocol and setup that is designed and implemented in both ends.

   On the computer side, a ROS robot node is developed in C++ to make communication with the robot possible, not only using the new EGM paradigm but also being backwards compatible with standard open-loop robot motion functions that have been used for years. On the robot side, an updated RAPID[2] server is implemented to be compatible with the computer's robot node and enable all desired features.

4. Integration of the reactive motion planning and execution framework with the high-level pose planner from the tactile dexterity project.

   More specifically, a proper interaction between both blocks is needed to enable or disable features like online replanning, collision avoidance, environment definition, single or multiple targets, pose or joint targets, etc.

[2]RAPID is the proprietary programming language from ABB that is used for programming their robots.

1.4 Document overview

The following chapters of this thesis are structured as it follows:

- Chapter 2 introduces the previous work related to the challenges faced in this thesis, as well as similar research contributions to the tactile dexterity project.
- Chapter 3 contains an overview of the tactile dexterity project, including the problem definition, the high-level planning process and the considered primitives modeling for planning and determining reactive behavior.
- Chapter 4 covers the definition and implementation of the motion planning issue, including the integration with the high-level planner to compute a fully executable plan.
- Chapter 5 includes the presentation of EGM, its advantages and drawbacks, its components and its implementations from both the computer and robot sides.
- Chapter 6 presents the full stack integration within the tactile dexterity project, from input definition to reactive execution, as well as its behavior and results.
- Chapter 7 discusses the developed framework and the concrete integration results for the presented application from a critical perspective, to finally conclude this thesis.

Chapter 2

Related work

This chapter presents previous work that could be related to the contributions of this thesis and could define somehow the current state of the art.

2.1 Motion planning for YuMi

As described in Section 1.3, motion planning is a common and widely studied problem within the robotics community, studied intensively during decades. This is reflected throughout many surveys [99, 68] and books [39, 26, 58, 59] covering all types of techniques and examples.

In any case, motion planning is a still pending challenge with no closed solution. More specifically, this thesis aims to provide a working and robust solution for motion planning applied to enable a reactive behavior for dexterous object manipulation, that implies a set of additional constraints like collision avoidance or specific contact modes with moving objects [95, 90] or 7 DOF and dual-arm robots [103, 77] like YuMi.

Of course, motion planning for YuMi can be implemented using generic algorithms for kinematic chains, e.g. sampling-based like the Open Motion Planning Library (OMPL) [109] or trajectory optimization-based like Covariant Hamiltonian Optimization for Motion Planning (CHOMP) [87] or Stochastic Trajectory Optimization for Motion Planning (STOMP) [51], potentially offered in frameworks like MoveIt! (Fig. 2.1) [108].

The YuMi Python interface project from the UC Berkeley AutoLab [11] features a similar integration as an experimental feature with some limitations, and more specifically without

considering dual-arm simultaneous planning, that is the main goal of this thesis.

Motion planners heavily rely on IK solvers, that find a valid joint configuration that leads to a desired pose. Either used for motion planning purposes or independently, they will be referenced many times throughout this thesis. Well-known solvers and libraries are Kinematics and Dynamics Library (KDL) [104], IKFast [32] or TRAC-IK [9].

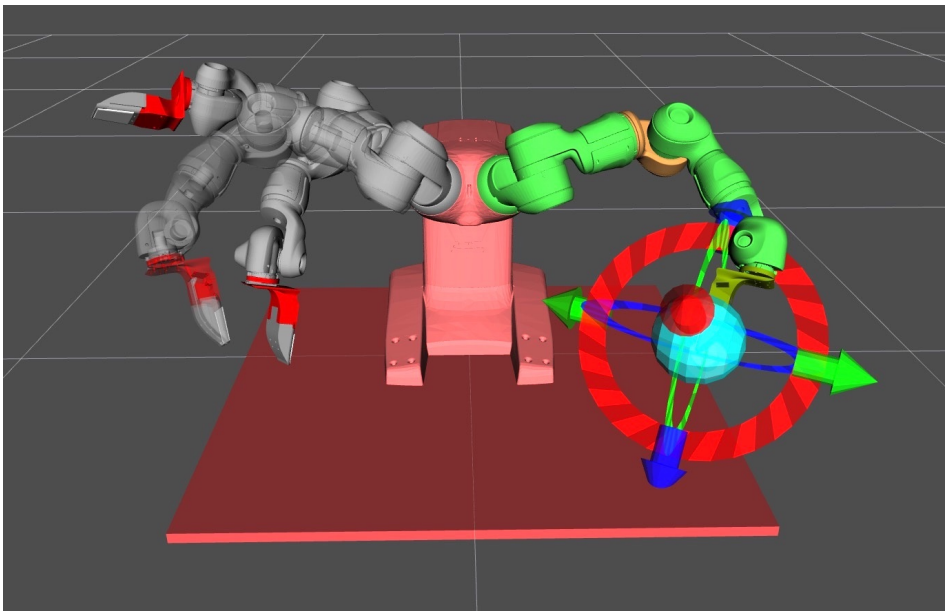


Figure 2.1: Screenshot of MoveIt!'s motion planning functionalities, including visual Cartesian target definition for individual arm planning.

2.2 High frequency execution for ABB robots

Historically, industrial robots have been used for well-defined applications that mostly required open-loop but precise motion in closed environments. However, both their precision and potential for technology transfer have made them attractive for the research community as well, that makes continuous efforts to interface them for high frequency teleoperation [41].

Within the ABB industrial robot atmosphere, standard motion functions were traditionally interfaced via a TCP/IP connection that enabled a reliable and robust teleoperation [30, 93, 78], but with a frequency around 5 Hz [14].

More recently, and after unofficial solutions like [70], ABB presented controller options for high frequency data exchange like Robot Reference Interface (RRI) [2] that was later reimple-

mented as EGM [3], enabling frequencies of up to 250 Hz. These protocols rely on Extensible Markup Language (XML) [17] and the more efficient Google protocol buffers [40] mechanisms for serializing structured data, respectively.

Despite its relevance for research purposes, EGM is poorly documented and there is not an unified, community-wide implementation project. The work in this thesis has been inspired by a previous working version from the MCube Lab, as well as Mæhre's thesis [73]. During the development of this work, other projects have progressed as well [111, 92].

2.3 Application to tactile dexterity

This thesis also covers the application of the developed framework for enabling tactile dexterity based on manipulation primitives like grasping but also pushing, pulling or levering. These primitives rely on the concept of nonprehensile manipulation, the relevance of which was described at first by Mason [76] to manipulate objects without full control over them.

To face this problem, there are two main possible approaches: designing sensorless manipulation tasks that try to maximize possible handled uncertainty with a same motion [74, 37], or developing sensors and algorithms to monitor and ensure success based on feedback while performing manipulation tasks like pushing [71, 120, 121], prehensile pushing [22], tumbling [96], pivoting [54, 46, 47], throwing and catching [47], and dynamic in-hand sliding [101].

The use of robotic palms is an interesting opportunity for both approaches and was explored in [36] considering the large mobility of dual-arm robotic manipulators. However, most works for scooping [115, 114], tilting [37], grasping [83], pushing [119], rolling [13] and collaborative manipulation [12] are performed in isolation to other primitives and are restricted to sticking contact interactions, difficult to achieve with open-loop control strategies.

However, there have been efforts to design planning algorithms that reason about multi-modal contact interactions that are natural in manipulation. For example, a nonlinear trajectory optimization framework that includes frictional forces and makes use of complementary constraints to encode different contact interaction modes is developed in [85]. Alternatively, a sampling-based planning algorithm for in-hand manipulation tasks that leverage contacts from the environment is presented in [22].

The application of model-based feedback control to contact rich interactions has been studied

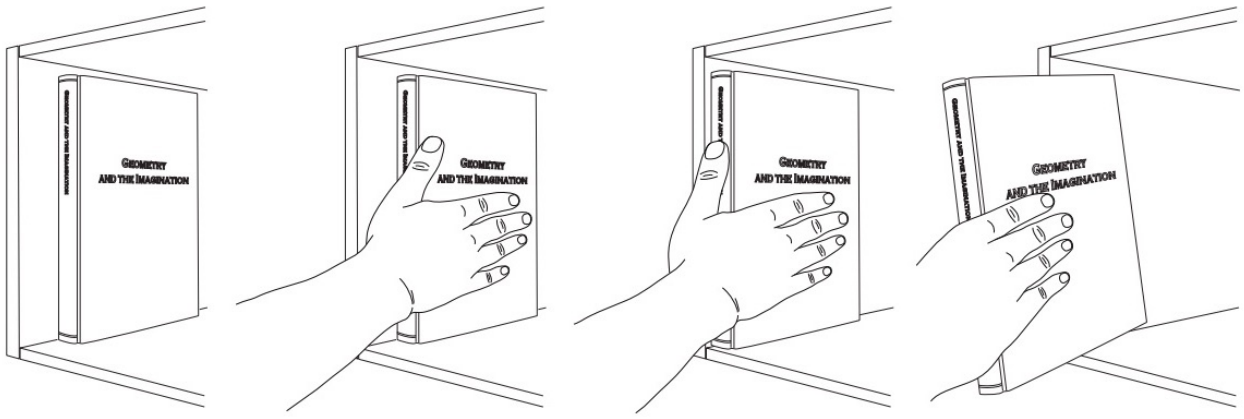


Figure 2.2: Animation of a manipulation task that uses several contact modalities: hand-book sticking, thumb-finger sliding and grasping. Reprinted from [43] with permission.

in some specific cases [94, 86, 117]. In these contributions, the control strategies are applied to systems that assume a previous knowledge of the contact mode sequencing.

In [43], a Model Predictive Control (MPC)-based feedback controller design is presented, where contact states (Fig. 2.2) are encoded using integer variables, leading to non-convex optimization programs that are challenging to solve in real-time.

In [31, 75, 44], approximate computational methods are explore to yield real-time control laws, assuming full state feedback of the object pose and knowledge about the physical properties of both the object and the environment.

Finally, simultaneous planning for high level task sequences and low level motion planning has been widely studied for robotic manipulation. For example, [112] introduces grasp placement tables for pick-and-place applications, that has later been followed by many works on regrasp planning such as [91, 110, 106, 24].

More recently, [116] proposes to use a graph search framework to effectively handle complicated mesh models and large-scale regrasp experiments. This planning framework decouples the graph search into finding first a sequence of stable object placements necessary to bring the object from its initial to final pose and then a sequence of regrasps that achieve the determined stable placement sequence, being flexible enough to extend task planning beyond regrasping, for example pivoting [47].

Chapter 3

Tactile dexterity: an overview

This chapter presents an overview of tactile dexterity, a framework that leverages tactile feedback for dexterous manipulation. We provide a brief formulation of the high-level manipulation planning that forms the backbone on which our motion framework is built.

3.1 Problem formulation

This thesis aims to contribute to the tactile dexterity project, i.e. the development of a high level planner that integrates tactile feedback and decomposes a complex tri-dimensional manipulation task into several basic contact rich manipulation primitives [42]. These simpler tasks can be modeled from first principles and are designed such that the resulting tactile readings are useful for feedback control purposes during execution.

More specifically, we consider the task of moving a rectangular prism (or cuboid) lying on a plane from a given to a target configuration (Fig. 3.1). Formally, the problem is defined by:

- A working plane (for example, a table attached to the robot).
- A mesh of the working object (rectangular prism), as a Stereolithography (STL) file.
- A tuple of initial conditions of the cuboid lying on the working plane, $q_0 = (x_0, y_0, \theta_0, f_0)$, where $(x_0, y_0) \in \mathbb{R}^2$ are the Cartesian coordinates within the working plane, $\theta_0 \in [0, 2\pi)$ is the object orientation w.r.t. the positive x semi-axis of the working plane's reference frame, and $f_0 \in \{0, 1, 2, 3, 4, 5\}$ is the numbering identifying the initial cuboid stable configuration (face in contact with the plane).

- A tuple of final conditions of the cuboid lying on the working plane, $q_f = (x_f, y_f, \theta_f, f_f)$, where the variables are defined analogously as above.

This task is executed using an ABB IRB 14000 (YuMi) position-controlled dual-arm industrial collaborative robot with seven DOF at each arm.

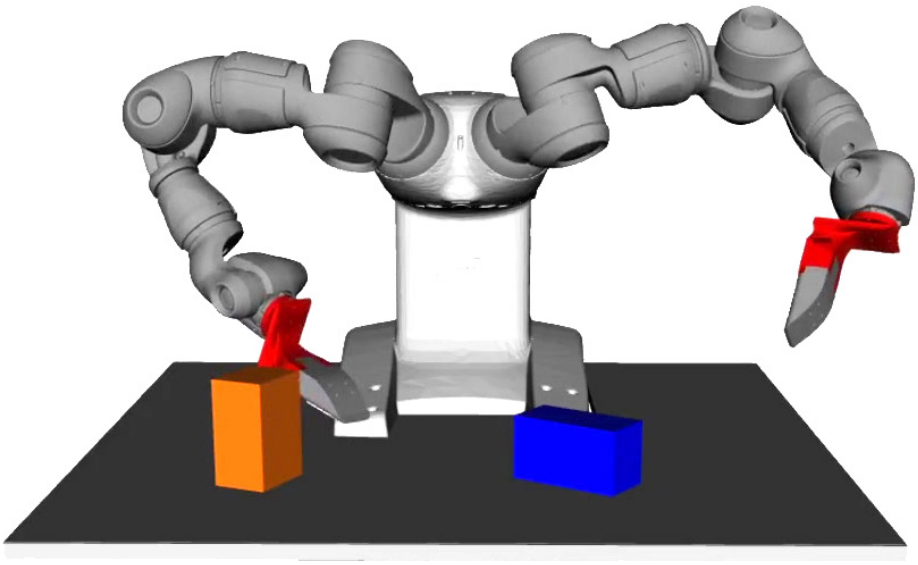


Figure 3.1: Problem example of moving a cuboid lying on a plane, from an initial configuration (orange) to a described target configuration (blue).

3.2 Description of primitives

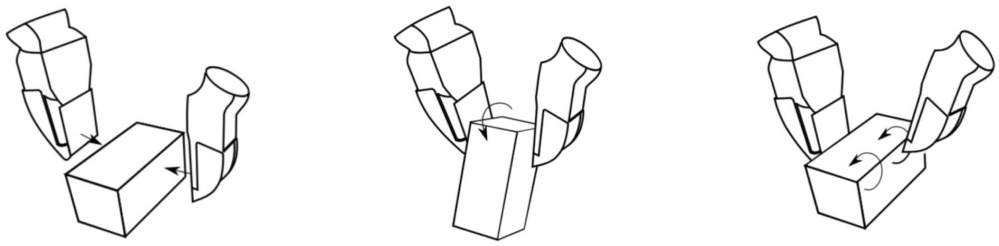
The manipulation task described above is decomposed into one or more basic primitives that can be kinematically modeled for planning and control purposes. Every primitive is designed to maximize the object motion stability and minimize the computational planning time, as well as rendering a useful tactile observation to be used for feedback control.

This decomposition of the problem is performed as to exploit the full dexterity made available by the dual robotic palms. Among many options, the combination of both palms can act as a parallel jaw gripper, work individually for pushing or pulling, or take advantage of the full motion workspace of both arms for more complex primitives like levering. For task planning purposes, these primitives are grouped into reconfiguration primitives and relocation primitives, as discussed in the following sections.

3.2.1 Reconfiguration primitives

Reconfiguration primitives are characterized for manipulating the object from one stable configuration to another, i.e. altering the face in contact with the plane. Even if both sets of all possible initial and final configurations are SE(2), the intermediate configurations of the object while being manipulated will lie in SE(3), making it a tri-dimensional task.

Examples of reconfiguration primitives include grasping (Fig. 3.2a), pivoting (Fig. 3.2b), levering (Fig. 3.2c), etc. Below, we give an overview of the approach taken to model the interactions between the object and the end-effector, plan for desired object motions, and feedback control strategy for the first three discussed primitives.



(a) Grasping (b) Pivoting (c) Levering

Figure 3.2: Examples of reconfiguration primitives for tactile-based dexterous manipulation. Reprinted from [42] with permission.

Grasping (Fig. 3.2a)

Mechanics: Force / form closure check.
Planning: Kinematic trajectories with grasp stability check.
Control: Enforce sticking relation between objects by reacting to incipient slip.

Pivoting (Fig. 3.2b)

Mechanics: 2D static equilibrium analysis.
Planning: Kinematic trajectories that satisfy static equilibrium.
Control: Enforce pure rotation of object relative to its center of rotation.

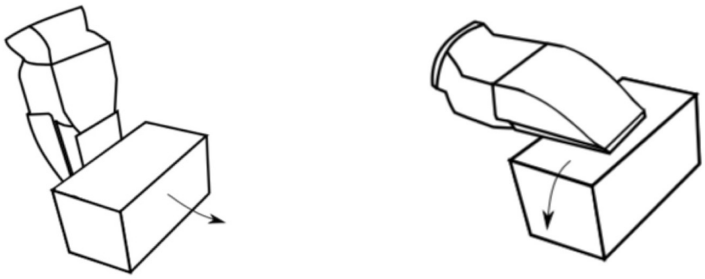
Levering (Fig. 3.2c)

Mechanics: 2D static equilibrium analysis.
Planning: Kinematic trajectories that satisfy static equilibrium.
Control: Enforce pure rotation of object relative to an edge.

3.2.2 Relocation primitives

Relocation primitives are characterized for manipulating (relocating and reorienting) the object within a given stable configuration, i.e. without altering the face in contact with the plane. In this case, all object configurations are manipulated within SE(2).

Examples of relocation primitives include pushing (Fig. 3.3a), pulling (Fig. 3.3b), corner rotation, push pulling, etc. The first two primitives are described below regarding object-palm interactions, planning and feedback control strategies.



(a) Pushing (b) Pulling

Figure 3.3: Examples of relocation primitives for tactile-based dexterous manipulation. Reprinted from [42] with permission.

Pushing (Fig. 3.3a)

Mechanics: Modeling of each palm as two point pushers. Quasi-static analysis of object-pusher interactions using limit surface.

Planning: Use of Dubins kinematic trajectories [19] to plan the trajectory of the object from initial to final poses.

Control: Enforce sticking relation between objects by reacting to incipient slip.

Pulling (Fig. 3.3b)

Mechanics: Modeling of each palm as a patch contact. Quasi-static analysis of object-pusher interactions using limit surface.

Planning: Kinematic trajectories checking feasibility (applied frictional forces lying within the limit surface).

Control: Enforce sticking relation between objects by reacting to incipient slip.

3.3 High-level task planning

The set of manipulation primitives described above present a wide variety of contact rich interactions that can be exploited for complex manipulation tasks, as well as facilitate state estimation, planning and control. However, this simplicity relies on a high level planner that can determine the optimal sequence of manipulation primitives that best achieves a task.

We formulate the search for sequences of manipulation primitives as a graph search problem, inspired in flexibility provided by the regrasp graphs proposed by Wan [116] and extended in [47]. In these cases, the search algorithm decouples the search into:

1. Finding a sequence of stable placements from the object's initial configuration to the final configuration.

2. Computing a sequence of manipulation primitives (regrasps in [116], regrasps and pivots in [47]) that achieve the determined stable placement sequence.

The tactile dexterity framework proposes two main variations to Wan's regrasp graph to allow for a more diverse set of primitives. First, only reconfiguration primitives (such as grasping or levering) are included as graph edges, in order to retain the structure of Wan's graph, where only object orientation is considered in the graph. Second, accounting for situations where reconfiguration actions can lead to IK infeasibility, several object positions are sampled to maximize the probability of finding feasible IK solutions.

Complementing reconfiguration primitives, relocation primitives (such as pushing or pulling) are responsible to bring the object to specific *safe* positions, from where reconfiguration primitives can be executed. At the same time, they are also used for placing the object from the resulting positions to the final desired one within a specified stable configuration.

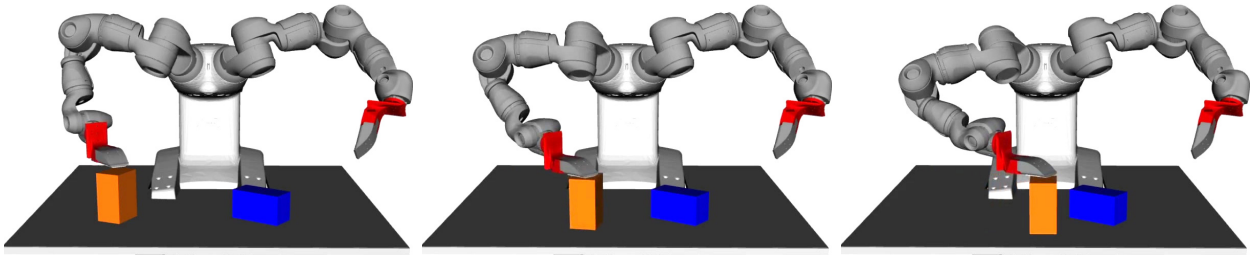
3.3.1 Illustrative example

For example, in Fig. 3.4 we consider moving a cuboid lying on a table when also changing the face in contact with it to a contiguous one. Such a problem would result in a sequence of three manipulation tasks:

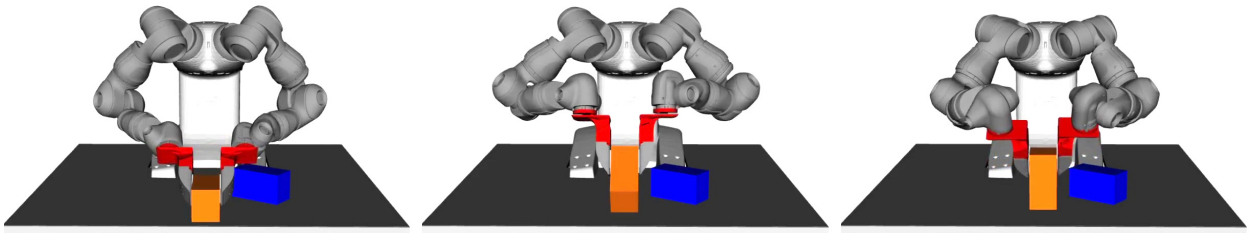
1. A relocation primitive (e.g. pulling or pushing) to move the object from the initial configuration to a safe starting point, maintaining the initial stable configuration.

2. A reconfiguration primitive (e.g. grasping, pivoting or levering) to reconfigure the object from the initial to the desired stable configuration.

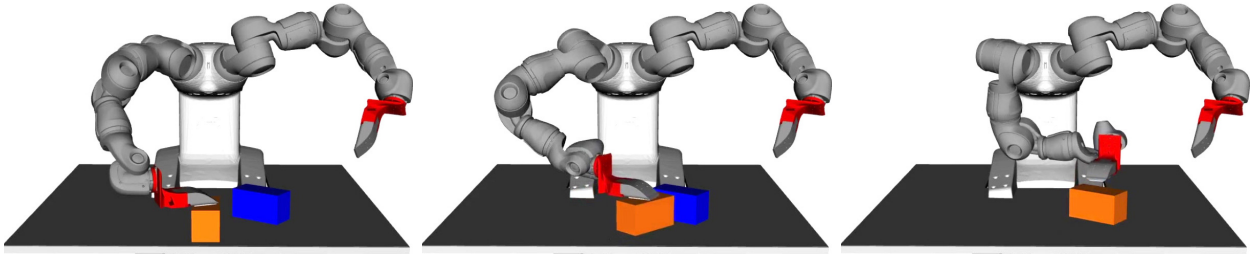
3. A relocation primitive to finally move the object from the resulting position of the previous step to the desired final configuration.



(a) First relocation primitive (pulling) from initial configuration to grasping initial position.



(b) Reconfiguration primitive (grasping) to change the object's stable configuration.

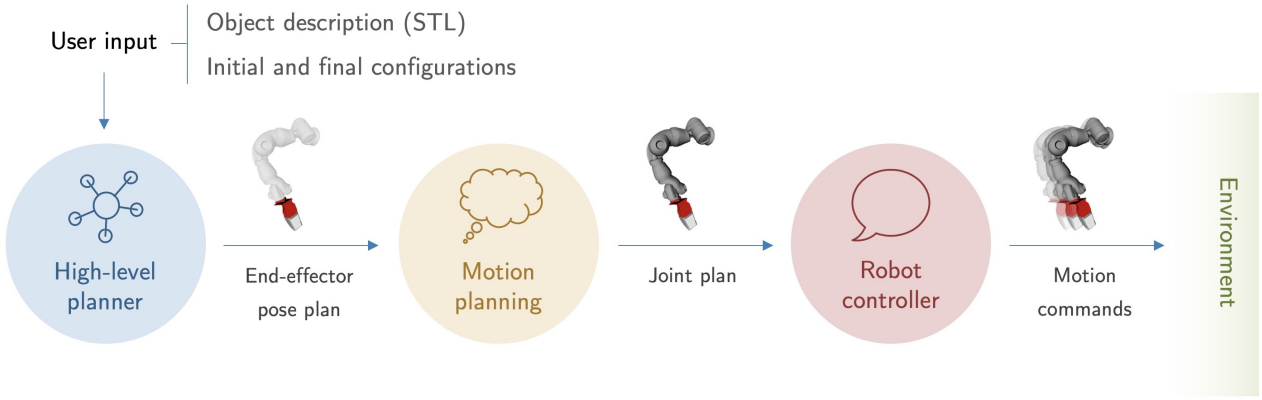


(c) Second relocation primitive (pulling) from grasping final position to desired configuration.

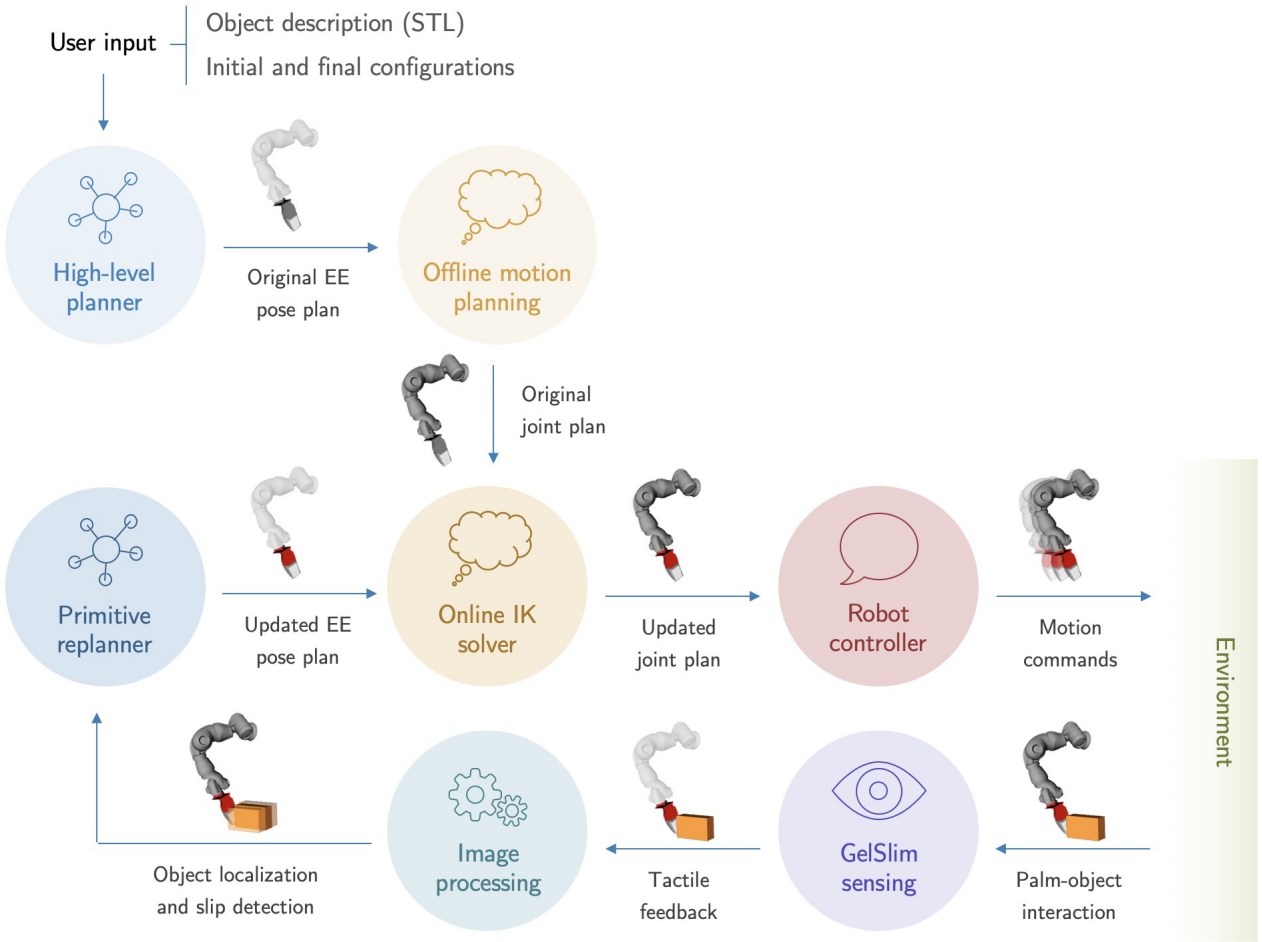
Figure 3.4: Example of manipulation task decomposition into basic primitives.

3.4 System architecture

In order to execute the high-level task plan, it needs to be fully integrated with a motion planning and execution framework. This is precisely the subject and aim of this thesis.



(a) Open-loop architecture without feedback and reactive control.



(b) Closed-loop architecture with tactile sensing, feedback processing and reactive control.

Figure 3.5: System architecture for open-loop and closed-loop executions.

The high-level task planner described above performs a graph search resulting in the sequence of manipulation primitives that best achieves a task. Once this sequence is found, every planned primitive within the sequence is described as paths for the palms. However, these paths define the behavior of the end-effectors, but not the entire robotic system.

As a consequence, the high-level planner output needs to be translated to a trajectory of joint configurations considering time, collisions, joint limits, maximum speeds, etc. that is finally executed. These processes are described later in Chapters 4 and 5, respectively, that make up the complete system architecture altogether with the high-level task planner.

3.4.1 Closed-loop integration

A first solution to this integration challenge would be an open-loop one, in such a way that the high-level planner output is executed without considering any feedback or reactive control (Fig. 3.5a). This motion planning framework would not require any specific frequency or delay specifications for execution.

However, as described in Section 1.2, the concept of tactile dexterity for robotic manipulation aims for robust and reactive manipulation skills, that can replan robot motions in real-time in response to tactile feedback. As a consequence, there is a need for a more complex system architecture that integrates the ability to replan quickly and robustly in all layers (Fig. 3.5b). This can be decomposed into the following pipeline:

1. Considering the problem formulation, the user inputs the object mesh and the initial and final configurations that define the manipulation task to be performed.

2. The high level task planner finds the sequence of stable placements and primitive executions that makes up the original plan. This plan include a sequence of poses for the robotic palms (end effectors), assuming certain contact modes and aiming for contact rich interactions that are used for state estimation and control.

3. The motion planning framework takes the end effector pose plan to compute an initial plan for the dual-arm robot joints that leads to those poses. This process also evaluates completeness, continuity enforcement, IK feasibility and collision awareness with the own robot, the environment and the manipulated object.

4. Starting from the original joint plan, two simultaneous threads keep running to perform

reactive control and execution of the plan:

(a) A replanning thread runs at the maximum possible frequency, only for the part of the primitive execution that involves interacting with the object. In this process, tactile feedback coming from the GelSlim sensors at each palm is obtained and processed for state estimation purposes. More specifically:

i. Contact mode verification is possible by monitoring the contact geometry made between the object and the end-effector, force distribution estimation [72] and incipient slip detection [34] using GelSlim.

ii. Specific contact features such as edges and lines enable the system to track the object pose during execution and account for small deviations of the expected plan [8], due to mistakes made or unaccounted environment conditions.

As a result of the received tactile feedback and primitive modeling, the high level planner replans palm poses to correct deviations from the original plan that have been found during execution.

(b) An execution thread keeps commanding the robot with a sequence of target joint configurations, initially set to the original joint plan from the motion planning framework, but potentially modified by using an online IK solver. This solver outputs the new sequence of target joint configurations, combining the new palm pose targets from the replanning thread with the original motion plan as a seed for solving IK. This loop runs at a frequency limited by either the IK solving rate or the EGM controller technology maximum frequency of 250 Hz.

The result of this system architecture integration is a robotic platform that can quickly and reliably replan robot actions, and therefore object motions due to robot-object interactions, based on tactile-based sensed information from GelSlim sensors. This replanning enables the system to drive the reactive behavior of the robot towards a successful execution of the manipulation task defined by the user.

Following an overview of the tactile dexterity project, Chapters 4 and 5 present more in depth the reactive motion planning framework (offline planning and online IK solving within the replanning thread) and the high-level execution framework (execution thread).

Chapter 4

Reactive motion planning

This chapter introduces a reactive motion planning framework that enables quick and reliable replanning based on sensed information. In particular, a short background review of well-known IK solvers and motion planners is presented, to later describe the application to robot-object interactions with YuMi. Finally, implementation details are discussed for both offline motion planning and online replanning systems.

4.1 Background

4.1.1 Forward and inverse kinematics

Kinematics is defined as the science of motion that treats motion without regard to the forces which cause it [29]. More specifically, kinematics studies position, velocity, acceleration and all higher order derivatives of the position variables, with respect to time or geometric variables. Hence, studying kinematics of manipulators allows to have a good understanding of geometrical and time-based properties of their motion.

Manipulators are formed by links that are assumed to be rigid, which are connected by joints that allow relative motion of neighboring links. These joints can be fixed, rotatory or revolute (if displacements are rotations measured in angles), or sliding or prismatic (if displacements are translations measured in linear distances, sometimes called *joint offset*).

In general, the number of Degrees of Freedom (DOF) is defined by the number of independent position variables that define a location of all parts of a mechanism. In the case of typical

industrial robots, the number of DOF is the number of joints, as manipulators are usually one or more kinematic chains with joints defined by a single variable. At the end of the link chain there is the end-effector, that can be a gripper, a palm, a welding torch or any other device depending on the intended application.

The first basic problem in manipulation is called Forward Kinematics (FK), the static geometrical problem of computing the pose, i.e. position and orientation, of the end-effector of a manipulator. This can be seen as a mapping from a joint space description of the manipulator (a set of joint angles) to one in the Cartesian space (a pose of the end-effector).

On the other hand, Inverse Kinematics (IK) refers to the following problem: given a pose of the end-effector, compute a possible set of joint angles that could be used to attain that pose. This is a fundamental problem for practical purposes, as usually applications just define the desired poses for the end-effector but manipulators are controlled by joint positions. In this case, IK can be seen as a mapping from feasible poses of the Cartesian space to the robot's internal joint space, considering joint limits.

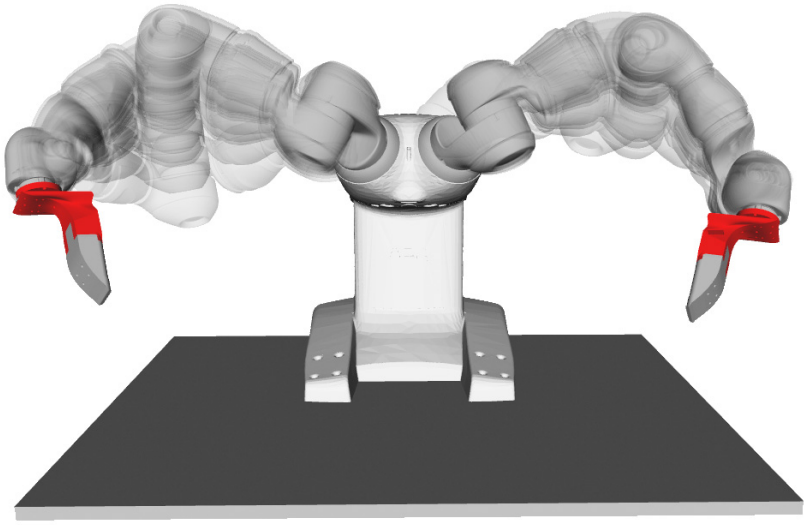


Figure 4.1: Example of the Inverse Kinematics problem: several joint configurations that lead to the same end effector pose.

It must be said that the IK problem is not as simple as the FK one. As kinematic equations are nonlinear, using them for FK is straight-forward but inverting them is not always easy, or even possible, in a closed form. Approaches like the Newton's method or gradient descent can be used, but issues like joint limits, positions close to singularities, different space dimensions or multiple solutions (Fig. 4.1) may arise within this process.

4.1.2 Review on IK solvers

In practical terms, many IK solvers have been developed with different strategies, either finding closed-form (algebraic or geometric) or numerical solutions. This section presents a short review of the most common IK solvers.

Orocos-KDL

The Kinematics and Dynamics Library (KDL) from the Orocos project is an application independent framework for modelling and computing of kinematic chains. It provides libraries for geometrical objects, kinematic chains (robots, biomechanical human models, computer-animated figures, machine tools, etc.) and their motion specification and interpolation [104]. It is usually used as the standard IK solver for most ROS or MoveIt! setups.

In particular, KDL features several algorithms to find numerical solutions for IK. By a simple inspection of its source code [105], several solvers can be found:

1. Based on the Newton-Raphson root-finding algorithm for the position kinematics equations, with and without considering joint limits;
2. Based on the Levenberg-Marquardt Algorithm (LMA), also known as Damped Least-Squares (DLS), to solve non-linear least squares problems applied to position equations. It includes cached computations of Jacobian kinematics, arbitrary weight specification for rotations and translations separately, and $\mathcal{O}(n)$ complexity;
3. Based on velocity transformations from Cartesian to joint spaces obtained from generalized pseudo-inverses, using Singular Value Decomposition (SVD);
4. Based on velocity transformations from Cartesian to joint spaces obtained from weighted pseudo-inverses with DLS, using SVD as well.

All solvers are numerical and iterative, and therefore lead to approximate solutions and can be given a maximum amount of iterations to execute.

The main inconvenient of KDL is its incompatibility with kinematic chains of less than 6 DOF, although this does not affect YuMi. On the other hand, KDL's convergence algorithms are based on Newton's method, which does not work well in the presence of joint limits, making the solve rate for YuMi around 76.88% [113].

IKFast

IKFast is a powerful analytical IK solver provided within the OpenRAVE motion planning software, developed by Rosen Diankov [32]. Unlike most IK solvers, IKFast aims for solving kinematic equations in an algebraic closed form, for any complex kinematics chain, including single-arm and dual-arm robotic manipulators, industrial robots and humanoids. This provides potentially faster solutions in comparison to numerical IK solvers, and also the computation of all feasible solutions (entire null space) instead of just a single one.

However, due to its algebraic nature, IKFast can only handle the same number of DOF than the considered target constraints (typically 6-dimensional pose, but sometimes combinations of only some position or orientation coordinates). For chains that contain more DOF, the user must set arbitrary values of a subset of the joints until both quantities match.

In the case of computing IK for 6-dimensional poses with YuMi arms (7 DOF each), this means that an arbitrary choice of a joint position should be done, that could limit the operating workspace when executing some tasks. This becomes an important drawback for manipulation purposes, for example in the tactile dexterity framework, as the motion planning system aims for being versatile enough to plan properly for several primitives that can potentially use a wide variety of different joint configurations.

TRAC-IK

TRAC-IK is an IK solver developed by Beeson and Ames from TRACLABS. It emerged as an attempt to improve KDL's performance on humanoid platforms, and its false-negative failures when constraining with joint limits [9].

In this case, TRAC-IK runs two simultaneous IK implementations and returns immediately when either of these algorithms converge to an answer [113]:

1. An extension of KDL's algorithm based on Newton's convergence method, that detects and mitigates local minima due to joint limits by random jumps.
2. A Sequential Quadratic Programming (SQP) nonlinear optimization approach that uses quasi-Newton methods to handle joint limits in a better way.

For YuMi, TRAC-IK achieves a success rate of 99.08 % with an average solution time of 0.56 ms [113], which makes it a good solution for computing IK in this project.

4.1.3 Motion planning

A fundamental problem within robotics is to plan collision-free motions for complex bodies from a start to a goal position among a collection of static obstacles, referred to as *motion planning* [55]. When considering Cartesian pose targets, this challenge can be seen a natural extension of the IK problem that also considers continuity in time, keeping away from joint limits and, of course, collision avoidance.

Despite it can be seen as a problem easy to understand, it has been deeply studied by the robotics community without any closed, unique solution. In fact, motion planning has been proved to be a PSPACE-hard problem [21], with a complexity that grows according to the number of the robot's Degrees of Freedom (DOF).

Configuration space

In order to describe a motion planning problem, it is necessary to have a complete description of a robot's geometry $\mathcal{A}$ and workspace $\mathcal{W}$, where usually $\mathcal{W} = \mathbb{R}^n$, $n \in \{2, 3\}$, is a static environment with obstacles.

A *configuration* $\boldsymbol{q}$ is defined as a complete specification of the location of every point on the robot geometry, and the *configuration space* or C-space $\mathcal{C}$ is the space of all possible configurations. Its dimension is the number of DOF of the robot system, or the minimum number of parameters to specify a configuration. The C-space concept is a useful way to abstract planning problems in a unified way, regardless of the manipulator's geometry [69], including cage or grasp planning [89, 5, 4].

Let $\mathcal{O} \subset \mathcal{W}$ represent the *obstacle region*, and $\mathcal{A}(\boldsymbol{q}) \subset \mathcal{W}$ the set of points occupied by the robot when its configuration is $\boldsymbol{q} \in \mathcal{C}$. Usually both $\mathcal{O}$ and $\mathcal{A}(\boldsymbol{q})$ are defined as closed sets and decomposed into piecewise-algebraic surfaces or sets enclosed by them.

Then, we define the *C-space obstacle region* as $\mathcal{C}_{\text{obs}} = \{\boldsymbol{q} \in \mathcal{C} \mid \mathcal{A}(\boldsymbol{q}) \cap \mathcal{O} \neq \emptyset\}$, i.e. the set of configurations that make the robot collide with the obstacle region. Conversely, the set of configurations that avoid collision is defined as the *free space* $\mathcal{C}_{\text{free}} = \mathcal{C} \setminus \mathcal{C}_{\text{obs}}$.

In case that $\mathcal{C} = \mathcal{W}$ (if the configuration is defined by coordinates within the workspace, e.g. an end-effector with fixed orientation) and the robot geometry is invariant by configuration changes, i.e. $\mathcal{A}(\boldsymbol{q}) = \mathcal{A}$, then $\mathcal{C}_{\text{obs}}$ can be easily computed as $\mathcal{C}_{\text{obs}} = \mathcal{O} \ominus \mathcal{A}$, where

$\ominus$ denotes a Minkowski difference.[1] This is common for cage or grasp planning, where the C-space is computed for the manipulated object rather than the manipulator, which is considered as a set of obstacles instead (Fig. 4.2).

In the general case, computing $\mathcal{C}_{\text{obs}}$ can be complicated and can result in complex and non-intuitive results [25, 60].

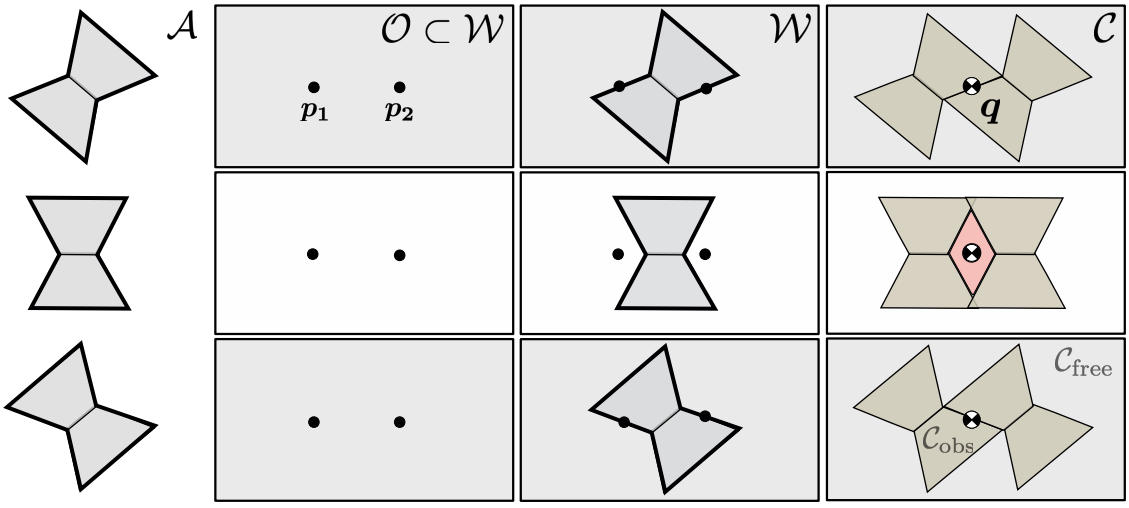


Figure 4.2: Example of C-space obstacle region computation for an object with geometry $\mathcal{A}$ and fixed orientations within $\mathcal{W} = \mathbb{R}^2$, with a $\mathcal{O} = \{\boldsymbol{p_1}, \boldsymbol{p_2}\}$ (manipulator with two point fingers). Then, $\mathcal{C}_{\text{obs}} = \mathcal{O} \ominus \mathcal{A}$. Reprinted from [4] with permission and modifications.

Basic problem

Using the definitions above, the basic motion planning problem is known as the *piano mover's problem* and is defined as follows [88]:

Given:

1. A workspace $\mathcal{W}$, usually $\mathcal{W} = \mathbb{R}^2$ or $\mathcal{W} = \mathbb{R}^3$,
2. An obstacle region $\mathcal{O} \subset \mathcal{W}$,
3. A mapping from robot configurations to robot geometries $\mathcal{A} : \mathcal{C} \to \mathcal{W}$, where $\mathcal{A}(\boldsymbol{q})$ is the set of points occupied by the robot when its configuration is $\boldsymbol{q}$,
4. The configuration space $\mathcal{C}$, from which the C-space obstacle region $\mathcal{C}_{\text{obs}}$ and the free space $\mathcal{C}_{\text{free}}$ are then defined,

[1] A Minkowski difference between two sets A and B is defined as $A \ominus B = \{a - b \mid a \in A, b \in B\}$.

5. An initial configuration $\boldsymbol{q_0} \in \mathcal{C}_{\text{free}}$, and

6. A goal configuration $\boldsymbol{q_G} \in \mathcal{C}_{\text{free}}$,

compute a continuous path $\tau : [0, 1] \to \mathcal{C}_{\text{free}}$ such that $\tau(0) = \boldsymbol{q_0}$ and $\tau_1 = \boldsymbol{q_G}$.

Although exact motion planning algorithms exist, they are difficult to implement, and show a poor performance with high-dimensional robots. A wide variety of algorithms have been published instead, based on different approaches that perform better or worse depending on the situation and requirements. A small selection is reviewed below.

4.1.4 Sampling-based planners

Sampling-based planners are one of the most well-known solutions, as they are able to solve very general problems. They rely on the latest advances in collision detection algorithms for single configurations, by sampling many configurations to construct a data structure with collision-free paths within the configuration space. As a consequence, these planners do not compute the C-space obstacle region directly, but only via the collision detector. This last tool is used as a *black box* that returns information of geometric contacts between objects according to their geometries and transformations [67, 50].

Sampling-based planners provide weaker completeness: they assume that if a solution path exists, it will be eventually found by the planner. Hence, they do not guarantee finding an exact solution in finite time, but are able to find an approximate one for complex systems where exact methods are not practical.

Typically, sampling-based planners are classified as multi-query planners (if they use a precomputed graph before) and single-query planners (if they build it during execution). More recently, several sampling-based planners have been proposed that build on one of the previous methods but provide some optimality guarantees, called optimizing planners. Many of these algorithms are integrated into the Open Motion Planning Library (OMPL) [109].

Multi-query planners

Multi-query planners construct a *roadmap*, an undirected graph G that is precomputed once and then used as a map of the connectivity properties of $\mathcal{C}_{\text{free}}$. This roadmap must be accessible, so that every configuration in $\mathcal{C}_{\text{free}}$ can be reached from G, and preserve the

connectivity of the free space.

A well-known example of multi-query planner that aims for approximating a roadmap in a computationally efficient way is the Probabilistic Roadmap Method (PRM) [56].

PRM-based algorithms work with an undirected graph $G = (V, E)$, where vertices are collision-free configurations and edges are collision-free paths between those, initially empty. During preprocessing, every time a sampled configuration from $\mathcal{C}_{\text{free}}$ is added to V, the algorithm tries to connect it to vertices already in V that are at a distance higher than certain threshold. In those cases, a local planning method will insert the new path as an edge to E after checking collisions. Vertices that are closer will be considered to be trivially connected. PRM will finish when a predefined number of collision-free vertices has been added.

The original PRM [56] uses random sampling and straight line paths in C-space for local planning. However, more specific sampling methods have been designed: at or near the boundary of $\mathcal{C}_{\text{free}}$, as in Obstacle-Based PRM [6]; as far as possible from it [45, 66]; using a Gaussian sampler that focuses on the difficult parts of $\mathcal{C}_{\text{free}}$ [16]; defining a deterministic sampling technique [62]; or many other approaches [102, 15, 49, 48, 65, 79, 20]. Some of these variants have been experimentally compared in [38].

Single-query planners

Single-query planners build tree data structures on demand, exploring the relevant part of the C-space to solve a planning query $(\boldsymbol{q_0}, \boldsymbol{q_G})$ as fast as possible.

Generally, single-query planners work with an undirected graph $G = (V, E)$ defined analogously to PRM search graphs, starting with some initial configurations in V and $E = \emptyset$. Then, the algorithm iterates by choosing a vertex $\boldsymbol{q_{cur}} \in V$ and trying to connect it to another vertex in V or a new sampled configuration $\boldsymbol{q_{new}} \in \mathcal{C}_{\text{free}}$. If successful, a local planning method will insert the new path as an edge to E after checking collisions, and if $\boldsymbol{q_{new}}$ was not already in V, will insert it. The algorithm iterates until G encodes a solution path or some termination condition is satisfied (a timeout, for example).

Depending on the strategy, G may consist of one or more trees, so single-query planners can be classified as: unidirectional methods, that involve a single tree and are usually initialized with $V = \{\boldsymbol{q_0}\}$; bidirectional methods, with two trees and usually starting with $V = \{\boldsymbol{q_0}, \boldsymbol{q_G}\}$; and multidirectional methods, which have more than two trees [61]. Using more than one tree is useful in situations with narrow openings, where a single tree can become trapped.

Well-known examples of single-query planners are the Rapidly-exploring Random Tree (RRT) and the more general Rapidly-exploring Dense Tree (RDT) [61]. Both algorithms choose to expand the vertex in V that is closer to $\boldsymbol{q_{new}}$ at every iteration, regardless of how this last configuration is selected. For example, using random samples, the probability of choosing a vertex is proportional to the volume of its Voronoi region, what is called *Voronoi bias*. In order to eventually reach $\boldsymbol{q_G}$, it is common to bias the choice of $\boldsymbol{q_{new}}$ such that $\boldsymbol{q_G}$ is frequently selected or, alternatively and more efficiently, growing two trees rooted at $\boldsymbol{q_0}$ and $\boldsymbol{q_G}$. These algorithms have been extended and adapted for several applications [18, 64, 10, 52, 107, 118].

Optimizing planners

The sampling-based algorithms presented above show a good performance in practice and have probabilistic completeness, but do not offer any guarantee on the quality of the solutions. Typically, this quality is a function of the path length, so the optimal solution is assumed to be the shortest path. New algorithms have been built on top of methods like PRM or RRT, that are provably asymptotically optimal and without big effects in terms of computational complexity. Some examples are PRM*, LazyPRM* and RRT* [53], RRT$^{\text{X}}$ [82], etc.

4.1.5 Optimization-based planners

In contrast to sampling-based planners, optimization-based planners aim for planning smooth trajectories for robot arms, avoiding obstacles and optimizing constraints or cost functions. Trajectory optimization is challenging in robotics, due to the non-convexity of of the collision-free regions and the high cost of collision checking.

A first optimization-based technique is Covariant Hamiltonian Optimization for Motion Planning (CHOMP) [87], a trajectory optimization procedure that is invariant to reparametrization and uses gradient descent for simultaneous numerical optimization and constraint enforcement, instead of separating in planning and optimization steps. CHOMP optimizes the initial trajectory, that can be feasible or not, using a functional that trades off between smoothness and obstacle avoidance components [122].

Stochastic Trajectory Optimization for Motion Planning (STOMP) [51], a stochastic trajectory optimization technique that explore noisy trajectories around an initial one, is considered as an alternative to CHOMP that is able to handle general cost functions for which gradients

are not available. At the same time, the stochastic nature of STOMP allows it to avoid local minima that gradient-based methods like CHOMP fall into.

Another optimization-based planner is TrajOpt, a sequential convex optimization procedure that penalizes collisions with a hinge loss and formulates the no-collisions constraint efficiently with continuous-time safety [97]. TrajOpt uses convex-convex collision checking while CHOMP uses Euclidian distances to the nearest obstacle; and also uses Sequential Quadratic Programming (SQP) instead of CHOMP's projected gradient descent, which is cheaper but is less flexible and cannot handle infeasible initializations [98].

4.1.6 Alternative approaches

Alternative approaches to the sampling-based and optimization-based methods exist, but are usually applicable only to particular cases. In those cases, efficient and elegant solutions are provided, but extensions for more complex scenarios have only theoretical interest.

Combinatorial roadmaps

For the case in which $\mathcal{C} = \mathbb{R}^2$ and $\mathcal{C}_{\text{obs}}$ is polygonal, combinatorial roadmaps have been shown to be efficient algorithms with polynomial complexity, but cannot be extended to higher dimensions.

A first algorithm is the maximum clearance roadmap or retraction method [81], which constructs a roadmap maximizing its distance with obstacles. The result is a sequence of Voronoi roadmap pieces between edge-edge (line), vertex-vertex (line) or vertex-edge (quadratic curve). Their naive computation has a complexity of $O(n^4)$ by generating all pieces for all possible pairs and their intersections. Better algorithms provide lower asymptotic complexity [63, 100], down to $O(n \log(n))$, where n is the number of roadmap pieces.

Variations of the retraction method are: the shortest-path roadmap [80], which allows the resulting path to touch the obstacles, a potentially necessary step to have an optimal path; or the vertical cell decomposition approach [23], that decomposes $\mathcal{C}_{\text{free}}$ into convex cells where planning is trivial, and sets vertices in the centers and boundaries of these cells.

Potential fields

This alternative for motion planning is inherited from real-time obstacle avoidance methods

[57], that create a differentiable function $U : \mathbb{R}^m \to \mathbb{R}$ called potential function. The key idea is that the potential function guides the motion of the moving object, due to attractive and repulsive components given by the goal and the obstacles, respectively.

The most naive algorithm is applying gradient descent, so the robot moves in a direction given by the gradient of U, but this does not guarantee a solution as the robot can end up in a local minimum of U that may not be $\boldsymbol{q_G}$. To solve this issue, randomized potential planners [7] combine this strategy with random walks when local minimum is reached and backtracking as a fallback when escaping from local minima fails.

4.1.7 Planning frameworks

Although a theoretical overview of motion planning algorithms is important, in practice most motion planners are seen as *black boxes* that are integrated by the community into planning frameworks. Thanks to these contributions, most applications just require the big effort of implementing a motion planning framework and choosing the IK solver and motion planner that best fits the considered use case.

The two most well-known robot motion planning frameworks are MoveIt! [108] and OpenRAVE [33]. Both of them feature a wide variety of plugins interfacing IK solvers, collision checkers, motion planners and sensors.

While OpenRAVE is primarily used as an environment for testing, developing and deploying motion planning algorithms in simulation, MoveIt! pretends to be a more general framework. In fact, MoveIt! is backed by the ROS community, has a stronger development and aims to provide functionalities for kinematics, motion planning and collision checking, but also 3D perception, robot interaction or control. In terms of motion planning, it has a longer list of compatible motion planners, including OMPL, CHOMP and STOMP, among others.

4.2 Problem formulation

In contrast to the basic motion planning problem presented in Section 4.2, this work aims to develop a versatile framework that can handle different motion planning problems while complying to the problem specifications stated below.

The assumed specifications for our framework are the following:

- A tri-dimensional workspace, $\mathcal{W} = \mathbb{R}^3$.
- An ABB IRB 14000 industrial robot model, that defines:
  - A robot configuration $\boldsymbol{q}$ is given by the set of 14 joints that correspond to the dual-arm setup (7 for each arm), $\boldsymbol{q} = (q_1, q_2, ..., q_{14}) \in \mathbb{R}^{14}$.
  - A set of position bounds at every joint: there exist two tuples $(q_{i,\min})_{i=1,...,14} \in \mathbb{R}^{14}$ and $(q_{i,\max})_{i=1,...,14} \in \mathbb{R}^{14}$ such that $q_{i,\min} \leq q_i \leq q_{i,\max}$ for every $i = 1, ..., 14$. As a consequence, the configuration space is $\mathcal{C} = [q_{1,\min}, q_{1,\max}] \times ... \times [q_{14,\min}, q_{14,\max}]$.
  - A set of maximum speeds at every joint: there exists $(\omega_{i,\max})_{i=1,...,14} \in (\mathbb{R}^+)^{14}$ such that $\left|\frac{dq_i}{dt}\right| \leq \omega_{i,\max}$ for every $i = 1, ..., 14$.
  - A robot geometry $\mathcal{A}(\boldsymbol{q})$ given by the geometries of every link in the two arms and their end-effectors as well.
- A static obstacle region $\mathcal{O} \subset \mathcal{W}$, including the workspace table, real or artificial obstacles within the environment, or the manipulated object in some cases. Consequently, $\mathcal{C}_{\text{obs}}$ and $\mathcal{C}_{\text{free}}$ depend on every situation as well.

While the basic motion planning problem aimed to find a continuous *path* from a current robot configuration $\boldsymbol{q_0}$ to a target, the motion planning problems of practical interest have the goal of computing a *trajectory*. Such a goal allows to include the notion of time in terms of speed or dynamic load limits, and therefore ensures a successful execution of the trajectory.

From a practical point of view, a trajectory seen as a continuous function is not executable, as robots cannot be commanded continuously but at some maximum frequency. In the case of the EGM controller setup, this frequency is set to 250 Hz. Therefore, trajectory discretization is a common pragmatic approach to express outputs from motion planners.

At the same time, motion planning targets can be expressed in several ways: as an explicit robot configuration $\boldsymbol{q_G}$ (for example, a joint configuration), a Cartesian pose $\boldsymbol{p_G}$ for one of the robot's links (usually the end-effector) in the workspace, or a sequence of more than one configuration space or workspace *waypoints*. Many times these targets are not achieved exactly, but within a certain tolerance that is defined by the user.

Note that planning for a sequence of several joint configurations is equivalent to plan for each of them one after the other, but the same does not hold for Cartesian targets. This is

due to the fact that a Cartesian target can be satisfied with several robot configurations (IK solutions), but only some of these can reach the next Cartesian target without implying a sudden motion that could make the trajectory unsmooth. Therefore, planning for a sequence of Cartesian targets will discard these combinations and result in smooth trajectories that combine the best IK solutions for every waypoint. This will be the case of most motion plans for precise manipulation tasks, such as the pushing task shown in Fig. 4.3.

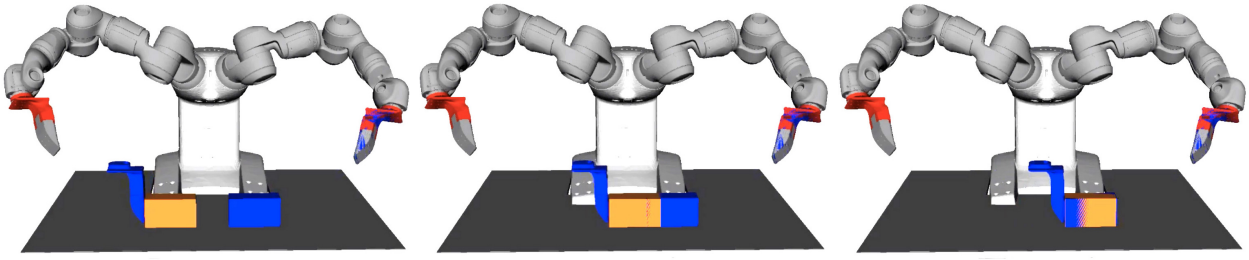


Figure 4.3: Example of pushing task described by a plan of end-effector Cartesian poses.

Another relevant point is collision avoidance. Current motion planners consider a static environment and therefore individual motion plans have to assume static obstacles. A variable obstacle region can be splitted into several time periods for which the environment can be considered static, and then plan individually with that assumption and join plans afterwards, but this may not be the preferred solution if slicing Cartesian pose plans is required.

With all these considerations in mind, we can formulate our extended motion planning problem for the ABB IRB 14000 robot as it follows:

Extended motion planning problem

Given a workspace $\mathcal{W} = \mathbb{R}^3$, joint position bounds $(q_{i,\min})_{i=1,...,14}$ and $(q_{i,\max})_{i=1,...,14}$, joint speed limits $(\omega_{i,\max})_{i=1,...,14}$ and a robot geometry function $\mathcal{A} : \mathcal{C} \to \mathcal{W}$ as specified above, and a static obstacle region $\mathcal{O} \subset \mathcal{W}$ that defines $\mathcal{C}_{\text{obs}}$ and $\mathcal{C}_{\text{free}}$, compute a *discretized trajectory* $\mathcal{T} = \{(t_k, \boldsymbol{q_k}) \mid k = 1, ..., N,\ t_k < t_{k+1}\}$, i.e. a set of N pairs of increasing timesteps and their corresponding robot configurations to command, such that:

1. All trajectory points satisfy joint position limits:

$$\forall\ k = 1, ..., N, \quad \forall\ i = 1, ..., 14, \quad q_{i,\min} \leq q_{k,i} \leq q_{i,\max}.$$

2. The trajectory satisfies joint speed limits:

$$\forall\ k = 1, ..., N-1, \quad \forall\ i = 1, ..., 14, \quad \left| \frac{q_{k+1,i} - q_{k,i}}{t_{k+1} - t_k} \right| < \omega_{i,\max}.$$

3. All trajectory points avoid collisions:

$$\forall\ k = 1, ..., N, \quad \boldsymbol{q_k} \in \mathcal{C}_{\text{free}}.$$

4. The trajectory is continuous:[2]

   (a) in configuration space: $\forall\ k = 1, ..., N-1,\ d_{\mathcal{C}}(\boldsymbol{q_k}, \boldsymbol{q_{k+1}}) < \varepsilon_{\mathcal{C}}$ for some arbitrary maximum step $\varepsilon_{\mathcal{C}}$ and some distance function $d_{\mathcal{C}}$ in configuration space, and/or

   (b) in workspace: $\forall\ k = 1, ..., N-1,\ d_{\mathcal{W}\times\mathcal{W}}(\pi(\boldsymbol{q_k}), \pi(\boldsymbol{q_{k+1}})) < \varepsilon_{\mathcal{W}\times\mathcal{W}}$ for the dual FK function $\pi : \mathcal{C} \to \mathcal{W} \times \mathcal{W}$,[3] some arbitrary maximum step $\varepsilon_{\mathcal{W}\times\mathcal{W}}$ and some distance $d_{\mathcal{W}\times\mathcal{W}}$ in $\mathcal{W} \times \mathcal{W}$.

5. The trajectory starts from the current configuration or an arbitrary one, $\boldsymbol{q_0}$: $\boldsymbol{q_1} = \boldsymbol{q_0}$.

6. The trajectory achieves the defined target(s) within a given tolerance:

   (a) a single configuration target: given a defined goal configuration $\boldsymbol{q_G} \in \mathcal{C}$, it must hold that $d_{\mathcal{C}}(\boldsymbol{q_N}, \boldsymbol{q_G}) < \delta_{\mathcal{C}}$ for some distance function $d_{\mathcal{C}}$ in configuration space and some tolerance $\delta_{\mathcal{C}}$, or

   (b) a single workspace target: given a goal pose pair $\boldsymbol{p_G} = (\boldsymbol{p_{G,1}}, \boldsymbol{p_{G,2}}) \in \mathcal{W} \times \mathcal{W}$, it must hold that $d_{\mathcal{W}\times\mathcal{W}}(\pi(\boldsymbol{q_N}), \boldsymbol{p_G}) < \delta_{\mathcal{W}\times\mathcal{W}}$ for the FK function $\pi : \mathcal{C} \to \mathcal{W} \times \mathcal{W}$, some distance function $d_{\mathcal{W}\times\mathcal{W}}$ in $\mathcal{W} \times \mathcal{W}$ and some tolerance $\delta_{\mathcal{W}\times\mathcal{W}}$, or

   (c) a sequence of workspace targets: given a sequence of pose waypoints $(\boldsymbol{p_j})_{j=1,...,M}$, it must hold that $\exists$ a subsequence of indexes $(k_j)_{j=1,...,M}$ such that:

   - k_j are indexes of the discretized trajectory: $k_j \in \{1, ..., N\}\ \forall\ j = 1, ..., M$;
   - Its last element is the last index of the original sequence: $k_M = N$;
   - Its elements are increasingly ordered: $k_j < k_{j+1}\ \forall\ j = 1, ..., M-1$;

[2] In this context, continuity does not refer to the discrete or continuous nature of the trajectory definition, but the imposition of small steps in robot configurations or Cartesian space between contiguous timesteps.

[3] The function π returns the poses of both end-effectors $\boldsymbol{p} = (\boldsymbol{p_1}, \boldsymbol{p_2})$ given a joint configuration.

- Its elements represent the indexes for which a desired pose pair waypoint is achieved (synchronously for both arms) by the computed discretized trajectory: $d_{\mathcal{W}\times\mathcal{W}}(\pi(\boldsymbol{q_{k_j}}), \boldsymbol{p_j}) < \delta_{\mathcal{W}\times\mathcal{W}} \ \forall \ j = 1, ..., M$, for the FK function $\pi : \mathcal{C} \to \mathcal{W} \times \mathcal{W}$, some distance function $d_{\mathcal{W}\times\mathcal{W}}$ and some tolerance $\delta_{\mathcal{W}\times\mathcal{W}}$.

Another important consideration is the reactive control implementation. The main idea is that external feedback is processed during execution and, altogether with the previous model of the task being executed, leads to a new target description for the motion planning problem. This description aims to correct mistakes made or unaccounted conditions.

However, since the previous plan is already being executed, it is not practical to consider the entire new target description and replan and restart execution from the beginning, as this could potentially lead to infinite loops. Instead, the previous plan can be used as a seed to speed up the replanning of the updated targets that have not been achieved yet. This leads to our reactive replanning problem, that we define as follows:

Reactive replanning problem

Given the conditions of the extended motion planning problem, an original discretized trajectory $\mathcal{T} = \{(t_k, \boldsymbol{q_k}) \mid k = 1, ..., N, \ t_k < t_{k+1}\}$ and a replanning initial condition $(t_l, \tilde{\boldsymbol{q}}_{\boldsymbol{l}})$, quickly compute an updated discretized trajectory $\mathcal{T}' = \{(t'_k, \boldsymbol{q'_k}) \mid k = l, ..., N, \ t'_k < t'_{k+1}\}$ such that:

1. Conditions 1, 2, 3 and 4 from the extended motion planning problem are satisfied.
2. The trajectory starts from the defined replanning initial condition, $\tilde{\boldsymbol{q}}_{\boldsymbol{l}}$: $\boldsymbol{q'_l} = \tilde{\boldsymbol{q}}_{\boldsymbol{l}}$.
3. The trajectory achieves the updated target(s) within a given tolerance:
   (a) a single configuration target: given a defined updated goal configuration $\boldsymbol{q'_G} \in \mathcal{C}$, it must hold that $d_{\mathcal{C}}(\boldsymbol{q'_N}, \boldsymbol{q'_G}) < \delta_{\mathcal{C}}$ for some distance function $d_{\mathcal{C}}$ in configuration space and some tolerance $\delta_{\mathcal{C}}$, or
   (b) a single workspace target: given a defined updated goal pose pair $\boldsymbol{p'_G} \in \mathcal{W} \times \mathcal{W}$, it must hold that $d_{\mathcal{W}\times\mathcal{W}}(\pi(\boldsymbol{q'_N}), \boldsymbol{p'_G}) < \delta_{\mathcal{W}}$ for the FK function $\pi : \mathcal{C} \to \mathcal{W} \times \mathcal{W}$, some distance function $d_{\mathcal{W}\times\mathcal{W}}$ in $\mathcal{W} \times \mathcal{W}$ and some tolerance $\delta_{\mathcal{W}\times\mathcal{W}}$, or
   (c) a sequence of workspace targets: given a sequence of updated pose waypoints $(\boldsymbol{p'_j})_{j=1,...,M'}$, it must hold that $\exists$ a subsequence of indexes $(k_j)_{j=1,...,M'}$ such that:
      - k_j are indexes of the discretized trajectory: $k_j \in \{l, ..., N\} \ \forall \ j = 1, ..., M'$;

- Its last element is the last index of the original sequence: $k_{M'} = N$;
- Its elements are increasingly ordered: $k_j < k_{j+1} \; \forall \; j = 1, ..., M' - 1$;
- Its elements represent the indexes for which a desired pose waypoint is achieved by the computed discretized trajectory: $d_{\mathcal{W}\times\mathcal{W}}(\pi(\boldsymbol{q'_{k_j}}), \boldsymbol{p'_j}) < \delta_{\mathcal{W}\times\mathcal{W}} \; \forall \; j = 1, ..., M'$, for the FK function $\pi : \mathcal{C} \to \mathcal{W} \times \mathcal{W}$, some distance function $d_{\mathcal{W}\times\mathcal{W}}$ in $\mathcal{W} \times \mathcal{W}$ and some tolerance $\delta_{\mathcal{W}\times\mathcal{W}}$.

4.3 Proposed solution

For these specifications, this thesis aims for a reactive motion planning framework that integrates into a system architecture like that of Section 3.4. Reactivity comes into play through a dual setup that combines offline planning and online replanning (Fig. 4.4):

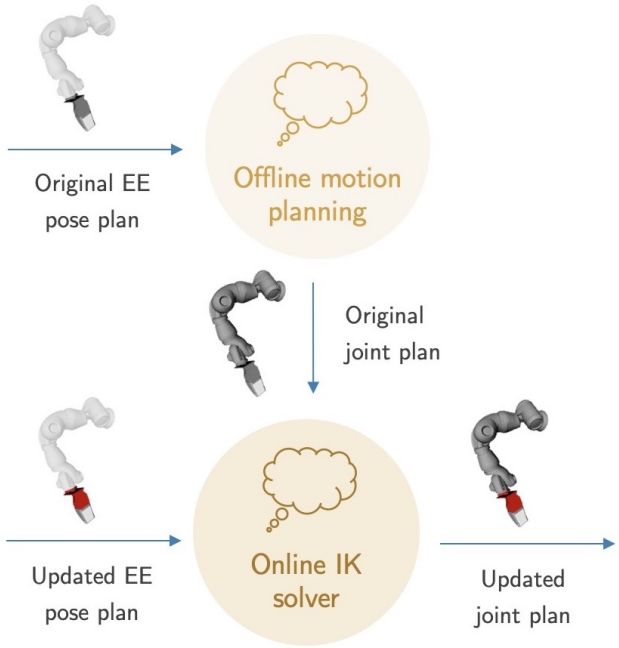


Figure 4.4: Part of the system architecture dedicated to reactive planning.

1. An offline motion planner accepts the extended motion planning problem defined above (for example, the original end-effector pose plan coming from the high-level planner from Section 3.3) and generates a complete, continuous, IK-feasible and collision-aware plan for all the 14 joints of the dual-arm robot.

2. An online IK solver, with a fallback motion planner, takes a reactive replanning problem (for example, the updated end-effector pose plan coming from tactile feedback processing as described in Section 3.4) and computes an updated joint plan for the two arms, using the original motion plan as a seed for faster computation. If we assume small deviations, we should expect that the output plans will inherit the desired properties from the original motion plan, although there are no formal guarantees for this.

Our proposed implementation builds on the MoveIt! [108] planning framework, that admits several of the IK solvers and motion planners described in Section 4.1. MoveIt! works with the Robot Operating System (ROS) environment, that is a standard tool in the community.

More specifically, a complete Python interface targeted to the YuMi robot description is implemented to enable easy handling of the motion planning problems defined in Section 4.2.

This framework is integrated into the tactile dexterity system architecture, source code of which will be available at the MCube Lab's GitHub repository.

4.4 Implementation

4.4.1 MoveIt! and robot description setup

The first implementation step consists on setting up the appropriate robot description and MoveIt! configuration for our ABB IRB 14000 (YuMi) robot.

A robot model description refers to information about its links and joints, how are they related, their limits, their collision geometry, their visual description, etc. Within ROS, this information is usually encoded in the XML-based Unified Robot Description Format (URDF), and Stereolithography (STL) files for the robot link geometries for both visual and collision avoidance purposes (Fig. 4.5). End-effectors, such as the GelSlim palms for the application to tactile dexterity, are usually considered as robot links and described in the same way.

Robot descriptions are required for almost any kind of project within robotics, even when only basic visualization and control is needed. Kinematics and planning have some extra requirements beyond what is encoded in an URDF. Motion planning frameworks such as MoveIt! use the Semantic Robot Description Format (SRDF) to describe semantic information about the robot structure: joints, links, chains and groups, and also information about

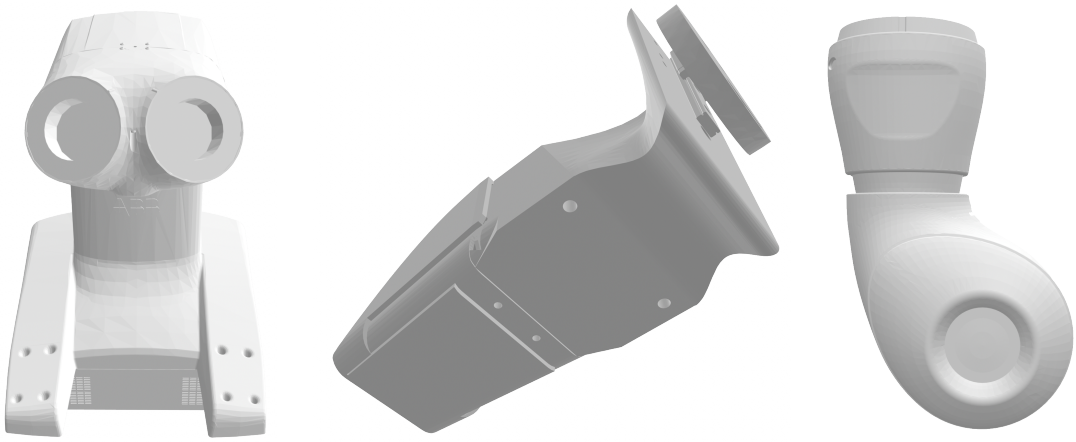


Figure 4.5: Geometries for some robot links and the end-effectors, defined by STL files.

collision checking (for example, to disable checks between adjacent links).

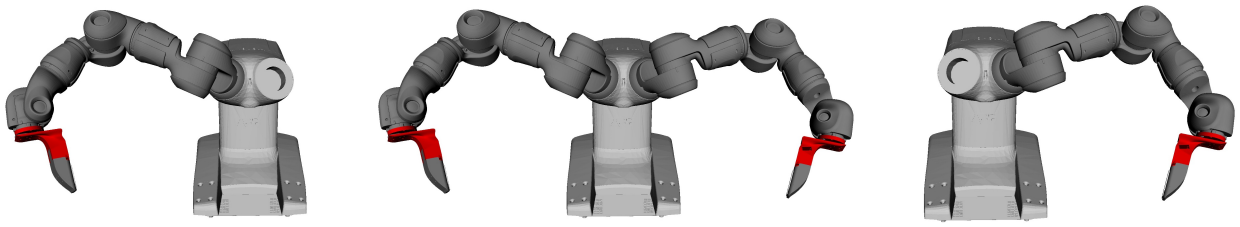
Once the robot description and semantics are defined in the URDF and SRDF files, respectively, these are included when launching the motion planning server, which is called `move_group` in the case of MoveIt!. At the same time, settings must be specified in order to choose a motion planning pipeline (such as OMPL, CHOMP or STOMP) and a kinematics solver (such as KDL, IK-Fast or TRAC-IK). Relevant parameters for every combination have to be set as well, such as timeouts, iterations, maximum attempts, weights, etc.

4.4.2 Planning groups

As a result of the SRDF description, a basic MoveIt! implementation (our starting point) is able to solve the previously defined extended motion planning problem for three possible groups (Fig. 4.6): every arm individually (`left_arm` and `right_arm`, named *left* and *right* according to the robot's body point of view) and both arms simultaneously (`both_arms`).

However, planning for Cartesian targets (one or more) is only available for individual arms, as all IK solvers only work for kinematic chains, while joint targets can be set for all three planning groups. As a consequence, the planning options are the following:

- For `left_arm` and `right_arm`: single joint configuration target, single Cartesian pose target or multiple Cartesian pose targets (waypoints).
- For `both_arms`: single joint configuration target.



(a) `right_arm` (b) `both_arms` (c) `left_arm`

Figure 4.6: Visual representation of considered joint groups for motion planning: (a) and (c) handle joint and Cartesian targets, while (b) only accepts joint configuration targets.

The most relevant consequence is that planning problems that have Cartesian targets need to be decomposed into subproblems for every arm, which are solved separately considering the other arm as a static obstacle. This leads to two main issues:

1. Self-collision avoidance is not guaranteed anymore, as every arm will just avoid collisions with the other arm at a specific configuration (typically initial, but any other could be arbitrarily chosen). If the other arm does not significantly change its configuration throughout the trajectory, this should not represent a big problem in practice, but there is no guarantee rather than post-checking and recomputing in case of collision.

2. Timing and synchronization issues may appear as well. As motions for every arm are planned individually, the resulting motion plans may not be synchronized even if the original pose plans were. In particular, every joint trajectory may have different time lengths, number of samples and times where targets are achieved (Fig. 4.8). This can happen when joint displacements are different, even if maximum joint speeds and dynamic limitations are the same, and is especially important when manipulating an object with two palms simultaneously.

Note that these two issues will not appear when targeting a joint configuration, as IK does not need to be solved and therefore the plan can be computed for the `both_arms` planning group, which will account for self-collision at every timestep and speed limits for all joints.

An important point is that using a greedy IK solver instead of a complete motion planning system would not work in general, because it would just return a feasible joint configuration for every Cartesian pose. Therefore, it would not manage any kind of *quality* criteria for joint configurations that are far or close to joint limits, and it would eventually return a joint configuration that would make following targets unreachable or only reachable through

a discontinuous or sudden motion. This possibility has been experimentally tested, and the results are an important motivation of the work presented here.

4.4.3 Original motion planning

This section presents the developed strategies to handle and solve extended motion planning problems for the dual-arm robot, exploiting the possibilities of MoveIt!'s planning groups.

If we consider the extended motion planning problem for two arms, as defined in Section 4.4.2, the three type of considered targets match with the ones handled by the planning groups: a single joint configuration, a single Cartesian pose or multiple Cartesian waypoints. However, only joint targets can be planned by the `both_arms` planning group, which is the only one that does not imply the self-collision avoidance and timing issues described in Section 4.4.2.

As a consequence, solving the extended motion planning problem for a joint configuration target consists in planning using the `both_arms` group for that same target:

Solving an extended motion planning problem (for a joint target $\boldsymbol{q_G}$)

1. Consider the original problem and solve it with the `both_arms` planning group.
2. Success: a complete, timed trajectory for all 14 joints with environment collision and self-collision awareness are generated as a result of the previous step.

On the other hand, planning for Cartesian poses becomes more difficult, as it requires to plan individually for each arm and solving the self-collision avoidance and synchronization issues to combine both trajectories, as discussed in Section 4.4.2.

Let $(\boldsymbol{p_j})_{j=1,\dots,M}$, with $\boldsymbol{p_j} = (\boldsymbol{p_{j,1}}, \boldsymbol{p_{j,2}}) \in \mathcal{W} \times \mathcal{W}$ and $M \in \mathbb{Z}^+$, the sequence of Cartesian pose target pairs, where $\boldsymbol{p_{j,1}}$ represents the end-effector pose target for one arm and $\boldsymbol{p_{j,2}}$ for the other. Without loss of generality, $\boldsymbol{p_{j,1}}$ and $\boldsymbol{p_{j,2}}$ are considered the left and right end-effector's poses, respectively. This includes the case of a single target pair, for $M = 1$.

To exploit MoveIt!'s planning groups, two separate planning problems have to be considered for the `left_arm` and `right_arm` planning groups, with targets $(\boldsymbol{p_{j,1}})_{j=1,\dots,M}$ for one arm and $(\boldsymbol{p_{j,2}})_{j=1,\dots,M}$ for the other (Fig. 4.7). Note that these problems have a lower dimensionality as they are now defined in $\mathcal{W} = \mathbb{R}^3$ (instead of $\mathcal{W} \times \mathcal{W}$) and $\mathcal{C} \subset \mathbb{R}^7$ (instead of $\mathbb{R}^{14}$). The starting configuration can be defined relative to all 14 joints, so the active seven are used for initializing the plan and the others are used to avoid collisions with the inactive arm.

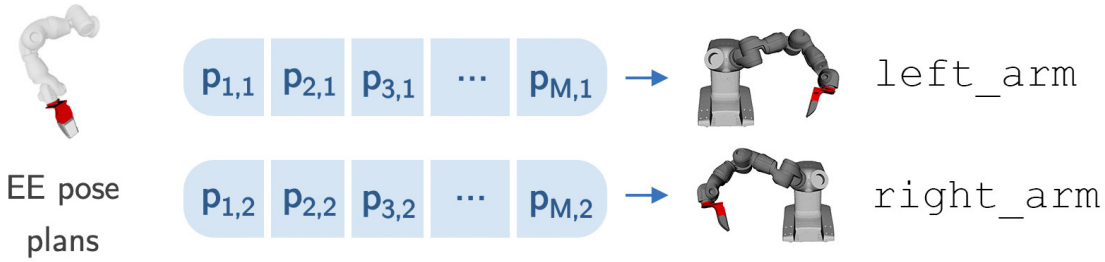


Figure 4.7: Separated extended motion planning problems are considered for each individual arm planning group, to then be combined strategically.

As a result from solving these problems, two separate discretized trajectories for each arm $\mathcal{T}_1 = \{(t_{1,k}, \boldsymbol{q_{1,k}}) \mid k = 1, ..., N_1\}$ and $\mathcal{T}_2 = \{(t_{2,l}, \boldsymbol{q_{2,l}}) \mid l = 1, ..., N_2\}$, with different amount of samples $N_1 \neq N_2$ and time length $t_{1,N_1} \neq t_{2,N_2}$ in general (Fig. 4.8), and collision avoidance only guaranteed with the environment (considered static) and between one active arm and the other inactive at its initial configuration.

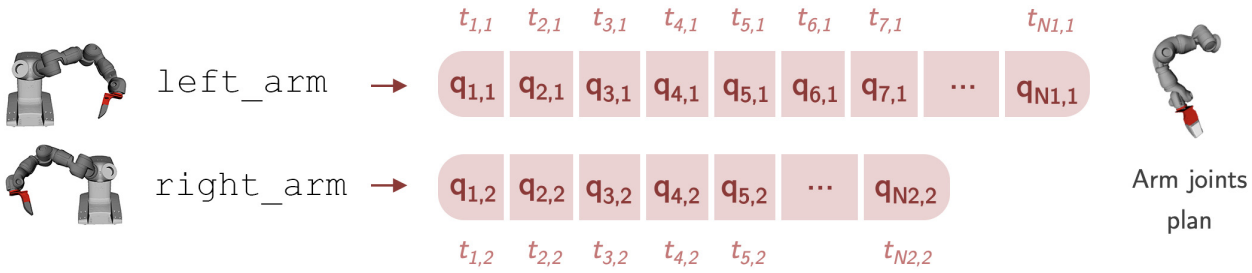


Figure 4.8: Resulting trajectories from each individual arm planning group have different length and achieve target waypoints in an unsynchronized fashion.

The implemented procedure to plan and combine $\mathcal{T}_1$ and $\mathcal{T}_2$ into the desired trajectory $\mathcal{T}$ for the dual-arm robot extended motion planning problem is as follows:

Solving an extended motion planning problem (for a pose sequence $(\boldsymbol{p_j})_{j=1,...,M}$)

1. Split the original target sequence $(\boldsymbol{p_j})_{j=1,...,M}$ into individual target sequences for every arm as discussed above, $(\boldsymbol{p_{j,1}})_{j=1,...,M}$ and $(\boldsymbol{p_{j,2}})_{j=1,...,M}$, and solve for those targets with the `left_arm` and `right_arm` planning groups, respectively.

2. Two trajectories $\mathcal{T}_1 = \{(t_{1,k}, \boldsymbol{q_{1,k}}) \mid k = 1, ..., N_1\}$ and $\mathcal{T}_2 = \{(t_{2,l}, \boldsymbol{q_{2,l}}) \mid l = 1, ..., N_2\}$, each for the 7 joints of a single arm $\boldsymbol{q_{1,k}} = (q_{1,k,1}, ..., q_{1,k,7})$ and $\boldsymbol{q_{2,l}} = (q_{2,l,1}, ..., q_{2,l,7})$, and starting at the same time $t_{1,1} = t_{2,1}$, are generated as a result of the previous step.

3. Use a resynchronization strategy to obtain trajectories $\mathcal{T}_1' = \{(t_n', \boldsymbol{q'_{1,n}}) \mid n = 1, ..., N\}$ and $\mathcal{T}_2' = \{(t_n', \boldsymbol{q'_{2,n}}) \mid n = 1, ..., N\}$ so they achieve every pair of waypoints at the same time. This is process is formalized and described more in detail in Section 4.4.4.

4. Finally, unify both retimed trajectories to obtain the desired trajectory $\mathcal{T}$ from the extended motion planning problem. We achieve so by defining

$$\mathcal{T} = \{(t_n, \boldsymbol{q_n}) \mid n = 1, ..., N\}, \quad \text{with} \quad \begin{cases} t_n = t_n', \\ \boldsymbol{q_n} = (q'_{1,n,1}, ..., q'_{1,n,7}, q'_{2,n,1}, ..., q'_{2,n,7}). \end{cases}$$

5. Using MoveIt!'s state validity check, verify that the sequence of configurations at each timestep, $\{\boldsymbol{q_n}\}_{n=1,...,N}$, is collision free. This should not give issues if planned motions do not vary significantly from the initial configuration, that is assumed to be collision-free. Otherwise, if the check fails, recompute a new motion plan (step 1).

6. Success: a complete, timed trajectory for all 14 joints with environment collision and self-collision awareness will be generated as a result of the previous step.

The algorithm presented above has two main assumptions: the existence of a resynchronization strategy to obtain synchronous trajectories and the *motion locality* of the planned motion so that it is possible to eventually obtain a plan without self-collision.

While the first condition is discussed in Section 4.4.4, it is important to note that the need of small or local motions can be handled by trying to plan larger motions as problems with targets described as joint configurations, which can be solved with self-collision avoidance for every timestep. These joint targets could be considered as *safe points* from which to start a manipulation task, and then reserve pose-targeted plans only for the manipulation itself.

4.4.4 Dual-arm trajectory resynchronization

As described above, solving an extended motion planning problem for a dual-arm robot with Cartesian pose targets requires a resynchronization strategy to make individual trajectories for each arm synchronous between them.

The aim of such a strategy is to modify a pair of original trajectories

$$\mathcal{T}_1 = \{(t_{1,k}, \boldsymbol{q_{1,k}}) \mid k = 1, ..., N_1\} \text{ and } \mathcal{T}_2 = \{(t_{2,l}, \boldsymbol{q_{2,l}}) \mid l = 1, ..., N_2\},$$

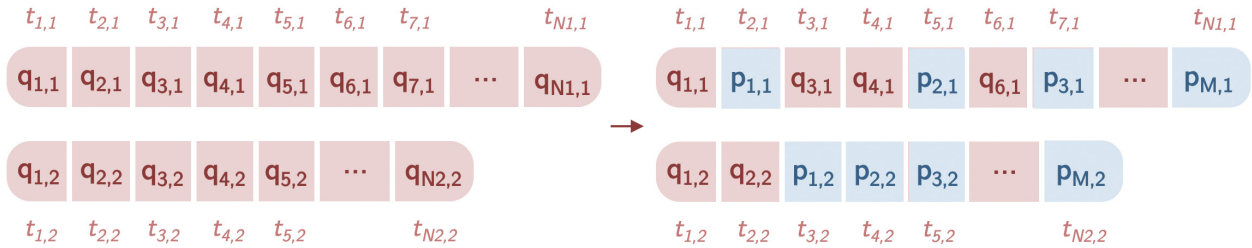
with $N_1 \neq N_2$ and $t_{1,N_1} \neq t_{2,N_2}$ in general, to obtain synchronized trajectories

$$\mathcal{T}_1' = \{(t_n', \boldsymbol{q'_{1,n}}) \mid n = 1, ..., N\} \text{ and } \mathcal{T}_2' = \{(t_n', \boldsymbol{q'_{2,n}}) \mid n = 1, ..., N\},$$

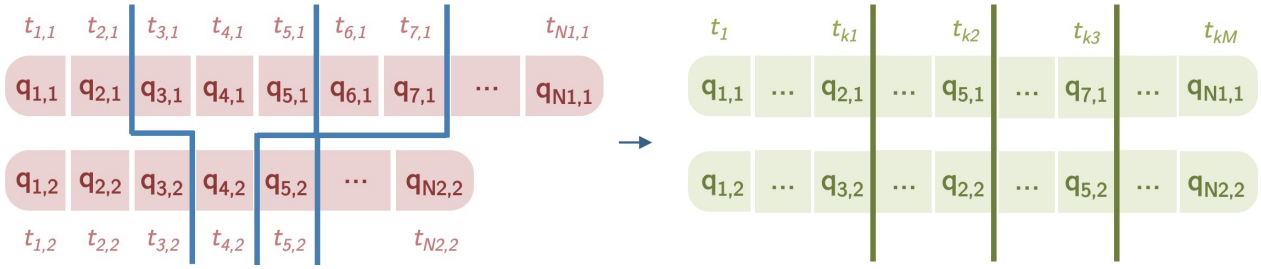
such that $\exists$ a subsequence of indexes $(n_j)_{j=1,...,M}$ satisfying that:

- k_j are indexes of the discretized trajectory: $n_j \in \{1, ..., N\}\ \forall\ j = 1, ..., M$;
- Its last element is the last index of the original sequence: $n_M = N$;
- Its elements are increasingly ordered: $n_j < n_{j+1}\ \forall\ j = 1, ..., M-1$;
- Its elements represent the indexes for which a desired pair of poses is achieved by every end-effector: $d_{\mathcal{W}}(\pi_1(\boldsymbol{q'_{1,n_j}}), \boldsymbol{p_{j,1}}) < \delta_{\mathcal{W}}$ and $d_{\mathcal{W}}(\pi_2(\boldsymbol{q'_{2,n_j}}), \boldsymbol{p_{j,2}}) < \delta_{\mathcal{W}}$ for all $j = 1, ..., M$, for the left and right end-effector FK functions $\pi_1 : \mathcal{C} \to \mathcal{W}$ and $\pi_2 : \mathcal{C} \to \mathcal{W}$, respectively; some distance function $d_{\mathcal{W}}$ and some tolerance $\delta_{\mathcal{W}}$.

The implemented algorithm to achieve this goal searches the indexes for which the next waypoint poses are reached at every sequence (Fig. 4.9a) and scales the duration of both segments of trajectory to make them reach that waypoint at the same time (Fig. 4.9b). The time length of the synchronized motion is determined by the arm that takes more time, and therefore the trajectory for the other arm is slowed down so both finish at the same time.



(a) Computation of Forward Kinematics (FK) to find the joint configurations leading to the defined targets.



(b) Time scaling within every subtrajectory to achieve synchronized target achievement.

Figure 4.9: Example of FK computation and interpolation stages for resynchronization.

Formally, the algorithm works as follows:

Dual-arm trajectory resynchronization algorithm

First of all, we initialize $\mathcal{T}_1' := \{(t_{1,1}, \boldsymbol{q_{1,1}})\}$, $\mathcal{T}_2' := \{(t_{2,1}, \boldsymbol{q_{2,1}})\}$, $N := 1$, $\tau_1 := t_{1,1} = t_{2,1}$, $j := 1$, $k_1 := 1$ and $l_1 := 1$.

1. If $j = M$, then take $k_2 := N_1$ and $l_2 := N_2$.

   Otherwise, search the indexes k_2 and l_2 for which the j-th left pose is reached by the left arm trajectory and the j-th right pose is reached by the right arm trajectory, respectively. They are found as:

   $$k_2 := \min\left(\arg\min_{k_1 \le k \le N_1} \|\pi_1(\boldsymbol{q_{1,k}}) - \boldsymbol{p_{j,1}}\|_2\right),\ l_2 := \min\left(\arg\min_{l_1 \le l \le N_2} \|\pi_2(\boldsymbol{q_{2,l}}) - \boldsymbol{p_{j,2}}\|_2\right),$$

   where $\pi_1, \pi_2 : \mathcal{C} \subset \mathbb{R}^7 \to \mathcal{W} = \mathbb{R}^3$ are the left and right end-effector FK functions, respectively. If a pose is reached multiple times, the first index is taken.

2. The motions from $\boldsymbol{q_{1,k_1}}$ to $\boldsymbol{q_{1,k_2}}$ (left arm) and from $\boldsymbol{q_{2,l_1}}$ to $\boldsymbol{q_{2,l_2}}$ (right arm) have to be synchronized, so they both have to start at time τ_1 and end at the same time, $\tau_2 := \max(t_{1,k_2}, t_{2,l_2})$. Therefore, sampled times during the motion will be scaled by factors $\alpha_1 := \frac{\tau_2 - \tau_1}{t_{1,k_2} - \tau_1}$ (left arm) and $\alpha_2 := \frac{\tau_2 - \tau_1}{t_{2,l_2} - \tau_1}$ (right arm), with $\alpha_1 = 1$ or $\alpha_2 = 1$. These rescaled times are then defined as:

   $$t'_{1,k} := \tau_1 + \alpha_1(t_{1,k} - \tau_1),\ k_1 + 1 \le k \le k_2; \quad t'_{2,l} := \tau_1 + \alpha_2(t_{2,l} - \tau_1),\ l_1 + 1 \le l \le l_2.$$

   Let γ an arbitrary set of times between τ_1 (not included) and τ_2 (included). A possible option could be the set of scaled versions of the times in both $\mathcal{T}_1$ and $\mathcal{T}_2$ that are part of those motions:

   $$\gamma = \left\{ t'_{1,k} \;\middle|\; k_1 + 1 \le k \le k_2 \right\} \cup \left\{ t'_{2,l} \;\middle|\; l_1 + 1 \le l \le l_2 \right\}.$$

   Given γ, add to $\mathcal{T}_1'$ and $\mathcal{T}_2'$ all times in γ and their corresponding interpolated joint configurations for those times. Let $t'_{N+m} \in (\tau_1, \tau_2]$ be the m-th time of γ when ordered in increasing order, for $m = 1, ..., |\gamma|$, then:

   $$\mathcal{T}_1' := \mathcal{T}_1' \cup \{(t'_{N+m}, \boldsymbol{q'_{1,N+m}})\}, \quad \mathcal{T}_2' := \mathcal{T}_2' \cup \{(t'_{N+m}, \boldsymbol{q'_{2,N+m}})\},$$

with the interpolated joints between the two nearest joint configuration samples in scaled time, and with weights given by differences in that scaled time:

$$\boldsymbol{q'_{1,N+m}} = \begin{cases} (1-\rho_{1,N+m}) \cdot \boldsymbol{q_{1,\tilde{k}_1}} + \rho_{1,N+m} \cdot \boldsymbol{q_{1,\tilde{k}_2}}, \ \rho_{1,N+m} := \frac{t'_{N+m} - t'_{1,\tilde{k}_1}}{t'_{1,\tilde{k}_2} - t'_{1,\tilde{k}_1}}, \ \text{if } \tilde{k}_1 \neq \tilde{k}_2, \\ \boldsymbol{q_{1,\tilde{k}_1}} = \boldsymbol{q_{1,\tilde{k}_2}}, \ \text{otherwise}, \end{cases}$$

$$\tilde{k}_1 := \max\{k_1 \leq k \leq k_2 \mid t'_{1,k} \leq t'_{N+m}\}, \quad \tilde{k}_2 := \min\{k_1 \leq k \leq k_2 \mid t'_{1,k} \geq t'_{N+m}\};$$

$$\boldsymbol{q'_{2,N+m}} = \begin{cases} (1-\rho_{2,N+m}) \cdot \boldsymbol{q_{2,\tilde{l}_1}} + \rho_{2,N+m} \cdot \boldsymbol{q_{2,\tilde{l}_2}}, \ \rho_{2,N+m} := \frac{t'_{N+m} - t'_{2,\tilde{l}_1}}{t'_{2,\tilde{l}_2} - t'_{2,\tilde{l}_1}}, \ \text{if } \tilde{l}_1 \neq \tilde{l}_2, \\ \boldsymbol{q_{2,\tilde{l}_1}} = \boldsymbol{q_{2,\tilde{l}_2}}, \ \text{otherwise}, \end{cases}$$

$$\tilde{l}_1 := \max\{l_1 \leq l \leq l_2 \mid t'_{1,l} \leq t'_{N+m}\}, \quad \tilde{l}_2 := \min\{l_1 \leq l \leq l_2 \mid t'_{1,l} \geq t'_{N+m}\}.$$

3. If $j < M$, update values for the next iteration: $N := N + |\gamma|$, $\tau_1 := \tau_2$, $k_1 := k_2$, $l_1 := l_2$ and $j := j + 1$. Otherwise, the algorithm has finished.

It is important to note that the algorithm assumes that there are no consecutive equal waypoints. As a result, it returns two synchronized trajectories with arbitrary samples for each subtrajectory between waypoints. This can also be useful for resampling already planned and synchronized trajectories, for example to execute them at certain frequencies.

4.4.5 Online replanning

Previous sections have presented solutions for the extended motion planning problem defined in Section 4.2, which also includes the formulation of the reactive replanning problem.

In the last case, given an original trajectory $\mathcal{T}$ and a replanning initial condition, the goal is to recompute an updated discretized trajectory $\mathcal{T}'$ that satisfies limits, collision avoidance and continuity, and achieves updated targets starting from the replanning initial condition. Those updated targets would probably come from a high-level planner that takes into account sensed feedback and has a reactive control policy, as in the tactile dexterity project.

If the updated target is also a joint configuration, the best option is to recompute the entire motion plan as an extended motion planning problem, as the updated plan will satisfy all conditions with guarantees and planning for joint targets is not so computationally expensive and fast to achieve (generally less than a second).

However, the computational cost and execution time of planning for a sequence of Cartesian poses are significantly higher. Therefore, attempting a smarter online replanning is preferable in these cases, combining a fast IK solver with cost thresholds for continuity and a complete motion planning fallback. Such an algorithm is presented below and works as follows:

Online replanning and execution algorithm (for pose targets)

This would be the algorithm running in parallel to a high-level pose replanning thread that listens for feedback, processes it and updates the pose targets. An example would be the two thread combination presented in the architecture from Section 3.4.1.

Let $\mathcal{T}_{\text{exec}} = \{(t_n, \boldsymbol{q_n}) \mid n = 1, ..., N_{\text{exec}}\}$ be a solution to the original extended motion planning problem, finely sampled for execution,[4] and $\mathcal{P}_{\text{exec}} = \{(t_n, \boldsymbol{p_n}) \mid n = 1, ..., N_{\text{exec}}\}$ and $\mathcal{P}'_{\text{exec}} = \{(t_n, \boldsymbol{p'_n}) \mid n = 1, ..., N_{\text{exec}}\}$ the original and dynamically updated plans of Cartesian pose pairs for each end-effector, with the same time sampling than $\mathcal{T}_{\text{exec}}$.[5] Initially, in the absence of feedback, $\mathcal{P}'_{\text{exec}}$ is initialized as $\mathcal{P}_{\text{exec}}$.

Note that $\pi(\boldsymbol{q_n}) = \boldsymbol{p_n}\ \forall\ n$, being $\pi : \mathcal{C} \subset \mathbb{R}^{14} \to \mathcal{W} \times \mathcal{W}$ the dual end-effector FK function.

First of all, we initialize $n := 1$ and then proceed:

1. Evaluate the updated pose plan $\mathcal{P}'_{\text{exec}}$ from the high-level replanning thread.
2. Compute the next executed joint configuration $\boldsymbol{q'_n}$ using IK, considering the next Cartesian pose target $\boldsymbol{p'_n}$ and the original joint configuration to execute $\boldsymbol{q_n}$ as a seed:

$$\boldsymbol{q'_n} = \pi^{-1}(\boldsymbol{p'_n}, \boldsymbol{q_n}),$$

   where $\pi^{-1} : \mathcal{W} \times \mathcal{W} \times \mathcal{C} \to \mathcal{C}$ returns an IK solution for the dual end-effector kinematics.
3. If the obtained result is close to the last executed joint configuration, $d_{\mathcal{C}}(\boldsymbol{q'_n}, \boldsymbol{q'_{n-1}}) < \theta_{\mathcal{C}}$, for some distance function $d_{\mathcal{C}}$ and some arbitrary threshold $\theta_{\mathcal{C}}$, execute it.

   Otherwise, go back to step 2. If the distance is higher than the threshold for more than N_a (arbitrarily defined) attempts, stop and recompute the entire motion plan with the updated following poses starting from the current joint configuration, and update $\mathcal{P}_{\text{exec}}$ to use it as a seed for IK.

[4] We should expect a sampling frequency $(t_{n+1} - t_n)^{-1}$ similar or equal to the robot commanding frequency. Resampling at a certain frequency is a particular case of resynchronization, as explained in Section 4.4.4.

[5] We assume this is possible because (i) original target poses can be interpolated and (ii) we expect updated targets to be small modifications of the original ones, and therefore not requiring higher joint speeds.

Chapter 5

High frequency execution using EGM

This chapter presents an implementation of the External Guided Motion (EGM) controller framework designed by ABB. In particular, the need of a high frequency communication protocol is introduced, and the proposed design and its development is explained.

5.1 Background

5.1.1 Introduction to EGM

Most commercial robots feature a controller system with an operating system, developed by the manufacturer, that defines all possible measurement and execution functions. This is the case of RobotWare, the robot controller system developed by ABB for its robots.

RobotWare offers a variety of functions for calibration, measurement, testing, error diagnostics and control. These functions can be manually executed through a FlexPendant unit, the Human-Machine Interface (HMI) developed by ABB, or integrated in an automatic workflow programmed using RAPID, ABB's high-level programming language.

RAPID allows development with routine parameters such as procedures and modules (subprograms), functions (admitting arguments and returning values) and trap routines (called by interruptions); automatic error handling and definition, and evaluation of arithmetic and logical expressions. Multiple added-value packages are available to enable multi-tasking, synchronized motion between different motion groups, sensor integration, etc.

One of these packages is External Guided Motion (EGM), a high frequency communication and control technology. Therefore, it enables external high frequency robot control by transferring most planning and safety checks to an external device, which is used to generate data that is processed during execution [1]. In particular, EGM offers two modes:

1. EGM Position Guidance, where the robot follows a path generated by an external device, instead of a programmed path in RAPID, and the robot just executes that path without additional planning features (e.g. interpolation or speed limits).

   In this case, the robot gets to the last received target as fast as possible, with no checks and following any kind of path (not necessarily linear), that depends on the start and end configurations. As a consequence, the robot can end up near a singularity and, if the target is not close, high dynamic load or joint torque limit errors may arise.

2. EGM Path Correction, for which there is a programmed path in RAPID that is modified or corrected using measurements from an external device. Only position corrections in the normal directions of the path coordinate system can be applied, but not in the tangent direction or in orientation.

Before EGM and its predecessor Robot Reference Interface (RRI), the ABB robot control paradigm was limited to the use of RAPID motion functions with complete planning and safety features, that can also be seen as overheads in terms of performance. In these cases, commanding a motion towards a final target blocks the system until the goal is reached.

The best attempts to emulate reactive teleoperation by slicing trajectories with intermediate updatable targets via TCP/IP made possible to control the robot with a frequency no higher than 5 Hz [14], making execution the frequency bottleneck in comparison to other sensors.

Conversely, EGM allows to communicate and execute instructions at a maximum frequency of 250 Hz and a theoretical control lag of 10-20 ms [1] that escalates to 40-60 ms in normal conditions. Communication is held via UDP, using Google's Protocol Buffers (Protobuf) based on XML, or I/O signals, while execution is performed through a simple velocity control law based on a position gain parameter and the current and target positions (if position controlled) or directly modifying the nominal velocity (if velocity controlled).

Since our reactive framework aims to accept any kind of updated target (not restricted to certain axes), the EGM Position Guidance mode is implemented. Therefore, the original and updated paths are handled by our motion planning framework instead of the EGM setup.

5.1.2 ROS communication features

Robot Operating System (ROS) is the most well-known set of software libraries and tools for robot applications among research and industry. This includes some functionalities already discussed in Section 4.4, such as URDF, SRDF or MoveIt! itself.

Another important feature from ROS is its low-level middleware infrastructure that allows inter-process communication, and in particular for our purposes:

- An asynchronous message passing system, based on *topics* to which different nodes publish and/or subscribe anonymously, to handle common message structures defined using the Interface Description Language (IDL).
- A synchronous remote procedure calling system, based on *services* that accept and return message structures as inputs and outputs, respectively, with a clearly defined request/response interaction between processes.

This communication framework is exploited in the presented implementation of the EGM technology. Services are used to enable and disable the EGM operation mode instead of the standard motion functions, and topics allow a fluent communication between the robot and the application. To this extent, the ROS robot controller node acts as an intermediary between the ROS network messages and the robot-computer network via UDP datagrams.

5.2 Implementation

As presented above, EGM represents simultaneously a communication protocol and an execution technology, that requires a dual implementation on the robot and controller sides, in order to ensure that commands from the end application are transferred and executed by the robot. Therefore, our implementation builds on the following blocks:

1. An updated ROS robot node that encapsulates all standard motion functions via TCP/IP, featuring an EGM mode that can be enabled on demand via a ROS service and operated through ROS topics. This implementation is performed on C++.
2. A RobotWare module on the robot side to interact with the ROS robot node in both standard and EGM operation modes. This implementation is based on RAPID.

This implementation's source code is available at the MCube Lab's GitHub repository.

5.2.1 ROS (C++) robot node

The ROS robot node implementation is based on its previous version from the MCube Lab [78], which offers many ROS services that encapsulate regular RAPID motion and information functions (Fig. 5.1), such as linear, joint or circular moves to Cartesian targets; joint moves to joint targets; setting and executing sequences of the previous moves; querying the current Cartesian pose or joint configuration; querying or setting parameters, etc.

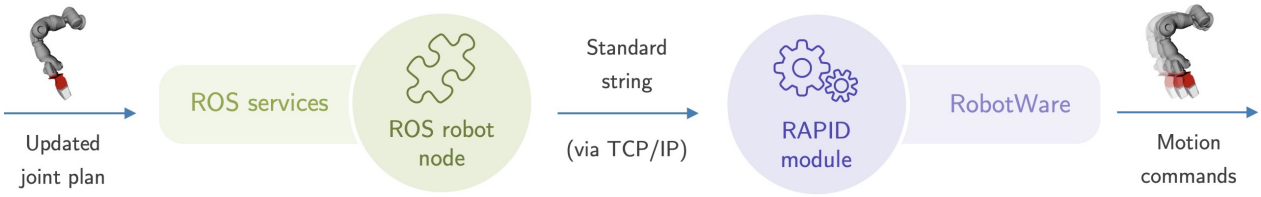


Figure 5.1: Communication flow between robot and controller for standard functions, through ROS services that map RAPID motion functions.

On the other hand, the updated version of the ROS robot node features compatibility with the EGM operation mode, as well as a cleaner structure and source code that makes the program easier to extend and update. In contrast to the previous version, the new ROS node is compatible with all ABB robots (with either 6 or 7 DOF) without no code modifications.

To integrate EGM, an additional ROS service `SetEGMMode` is implemented to enable or disable an EGM operation mode: (i) Cartesian position control, (ii) Cartesian speed control, (iii) joint position control and (iv) joint speed control. Currently, ABB robots with 6 DOF and a proper license are compatible with all four modes, while YuMi (IRB 14000, with 7 DOF) is only compatible with modes (iii) and (iv) with an experimental license from ABB. This extended compatibility is a particular difference with other EGM implementations.

Once an EGM operation mode is chosen, there is a continuous flow of information between the robot and the ROS node, which publishes to measured state topics and subscribes to target topics to send defined goals by the end application (Fig. 5.2). If there is no defined target by the user, the default behavior is sending the current position (for modes (i) and (iii)) or zero speed (for modes (ii) and (iv)).

This continuous flow is ensured by a thread running in parallel to the main program. When EGM is enabled, the thread creates a UDP connection and then periodically flushes data bidirectionally every 4 ms. On one hand, it reads from the ROS target topic, translates it

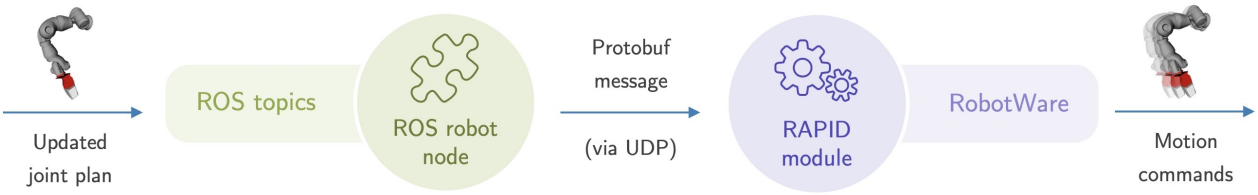


Figure 5.2: Communication flow between robot and controller for the EGM mode, through ROS topics that interface a continuous flow of information.

to a Protobuf message that can be parsed by the robot and sends it via UDP; on the other hand, it receives via UDP the measured data as a Protobuf message, translates it to ROS messages and sends it to the ROS network.

5.2.2 RobotWare (RAPID) module

As both ROS node and RAPID module are implemented to be compatible, the implemented RAPID code also builds on a previous version, that already integrated all regular motion and information functions for teleoperation. This is achieved by choosing the function to execute and its arguments based on received data via TCP/IP, coming from the ROS node services.

The updated version features a new accepted instruction code to enable or disable EGM, for which the proper functions to set up and run EGM are called. In addition, taking advantage of RobotWare multi-tasking capabilities, an additional thread is created to accept information queries from the ROS side while the main thread is blocked during EGM execution.

Chapter 6

System integration and results

This chapter presents the results of integrating the reactive motion planning and execution frameworks into the tactile dexterity system architecture described in Section 3.4.

Integrating such a framework has been possible by using common data structures for the plans throughout all the process of high-level pose planning, motion planning, feedback-based replanning and, finally, execution. This includes a hierarchical treatment of data based on the planning stage where it has been generated, and if it is common for several executions of a same task (offline planning output) or for individual executions only (e.g. replanning data).

Within this process, several systems work together to generate a final plan that is executable by a dual-arm robot and quickly updatable. However, these systems have different requirements or performance in terms of computation time, which can be taken as a measure of the current development state of our planning framework. Table 6.1 presents time performance estimates using RRT as motion planner and TRAC-IK as IK solver.

Stage	**Location**	**Max. frequency**	**Delay**
High-level pose planning	Offline	-	1-5 seconds
Motion planning	Offline	-	2-20 seconds
Feedback-based replanning	Replanning thread	20-30 Hz	20-30 ms
Online joint replanning (IK)	Execution thread	20-50 Hz	20-50 ms
EGM robot controller	Execution thread	250 Hz	40-60 ms

Table 6.1: Description of estimated computational requirements for the different stages.

These time performance estimates can be compared to other sensing techniques such as vision-based Vicon camera trackers (100-1000 Hz), vision-based AprilTags (around 10 Hz) or punctual force torque sensors (30-300 Hz). The first two approaches are focused at object localization and can fail in occluded environments, while the third returns quantitative sensed force, which could be potentially used for local object tracking as studied in [4]. Any of these three are able to report qualitative interaction features, potentially more useful for reactive control, such as slip detection or force pattern reconstruction.

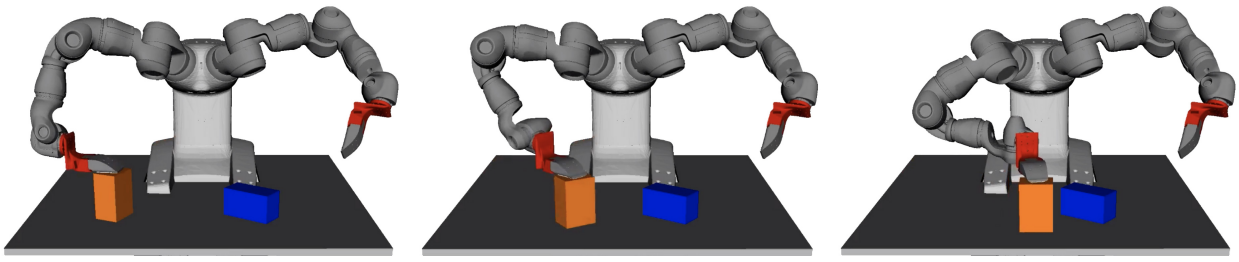
It is important to note that the performance of feedback-based pose replanning is given by the tactile feedback obtention and the different processing or feature extraction techniques used as criteria for replanning. Once an updated model is generated from feedback, the actual pose replanning takes around 100 ms due to pose computation and interpolations.

Additionally, online joint replanning (implemented through an IK solver) is performed in the execution thread instead of the replanning thread. This option represents a compromise between high execution frequency and low replanning delay, so the replanned pose plan starts to be executed as soon as it is generated but at a lower frequency, instead of waiting an additional time to compute all joint configurations and then executing them faster.

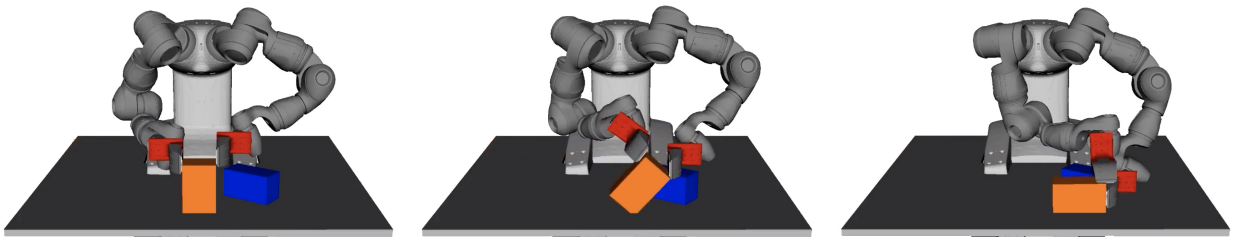
The integration with the tactile dexterity framework leads to the following execution types:

(1) Open-loop execution by a dual-arm robot of a complex manipulation task decomposed into several contact rich primitives, as defined in Section 3.1. To illustrate this result, a similar example to the one from Section 3.3.1 is considered.

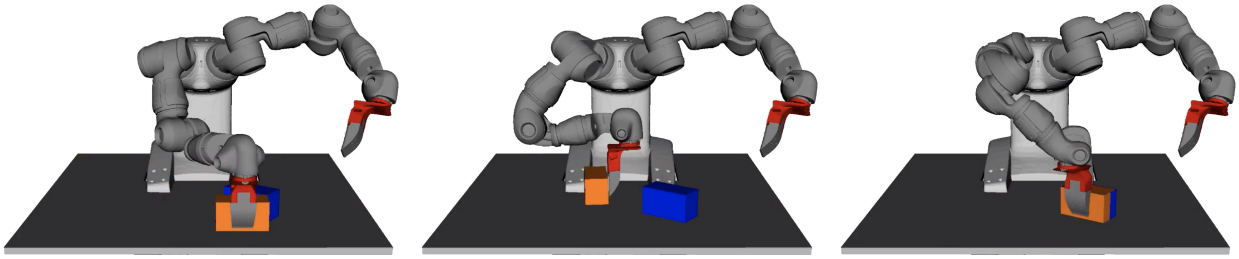
Figure 6.1 shows an example execution in a simulated environment, by using pushing (first) and pulling (second) as relocation primitives and levering as reconfiguration primitive. In this case, the simulated environment is assumed to be perfect and there are no errors, and it shows how the planning system deals with levering, an especially complex primitive that requires end-effectors to be close one to each other.



(a) First relocation primitive (pulling) from initial configuration to grasping initial position.



(b) Reconfiguration primitive (levering) to change the object's stable configuration.

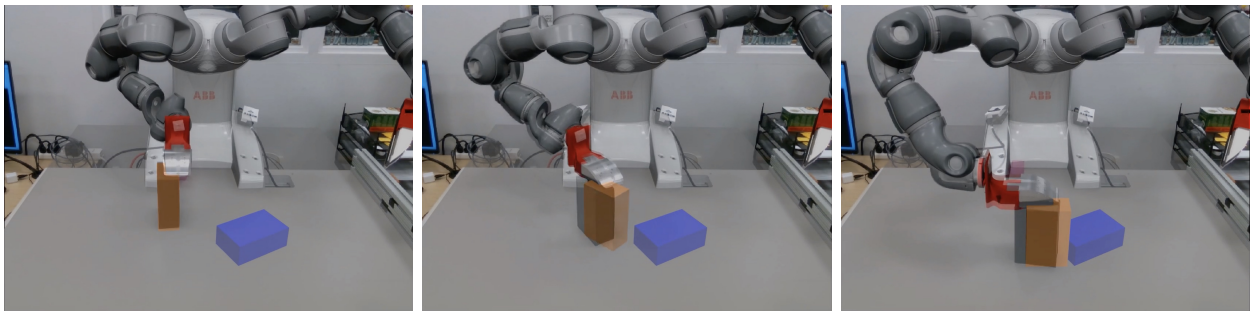


(c) Second relocation primitive (pushing) from grasping final position to desired configuration.

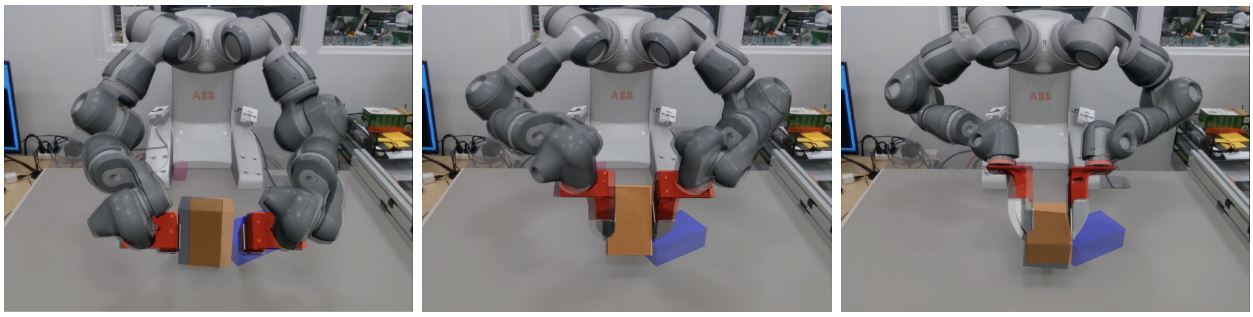
Figure 6.1: Execution of a manipulation task decomposition into pulling, levering and pushing in a simulated environment with no errors.

A similar simulated example using pulling only as a relocation primitive and grasping as a reconfiguration primitive, is shown in Figure 3.4 from Section 3.3.1.

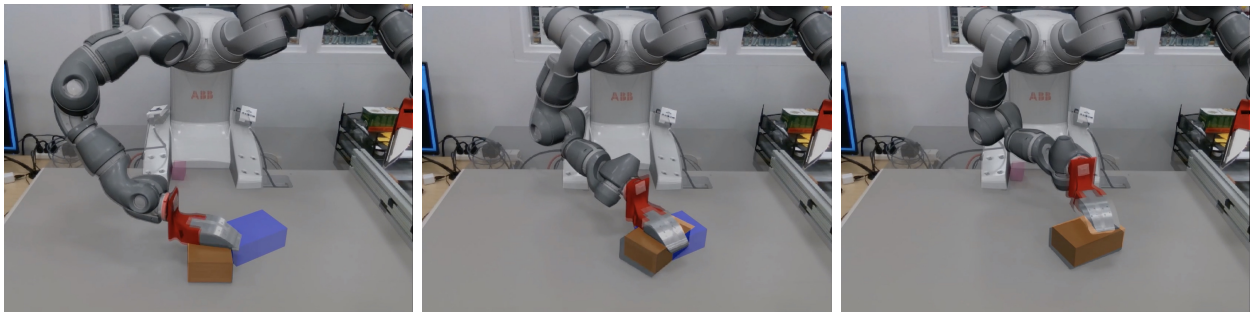
In contrast, Figure 6.2 shows a manipulation task in a real environment with two pulling executions as relocation primitives and an intermediate grasp as a reconfiguration primitive. In this case, the obtained results demonstrate the need for reactive control in manipulation, as uncertainty appears in multiple intermediate positions during the execution. Although this could potentially affect the system behavior and collapse it, this particular execution leaves the object at the desired configuration with a small error.



(a) First relocation primitive (pulling) from initial configuration to grasping initial position.



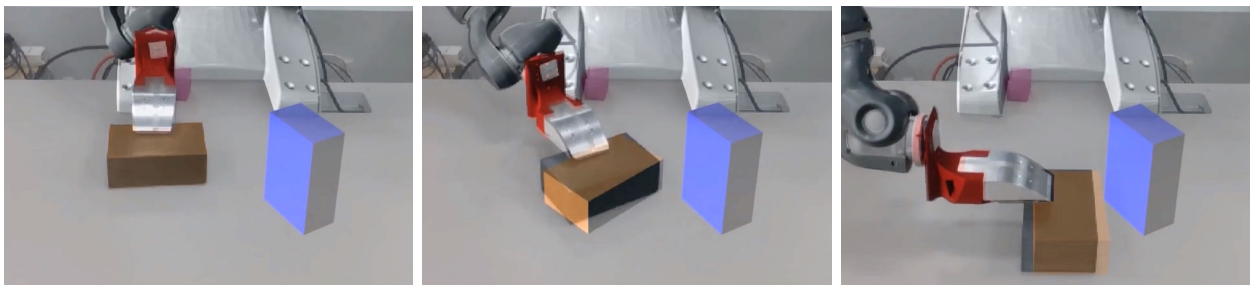
(b) Reconfiguration primitive (grasping) to change the object's stable configuration.



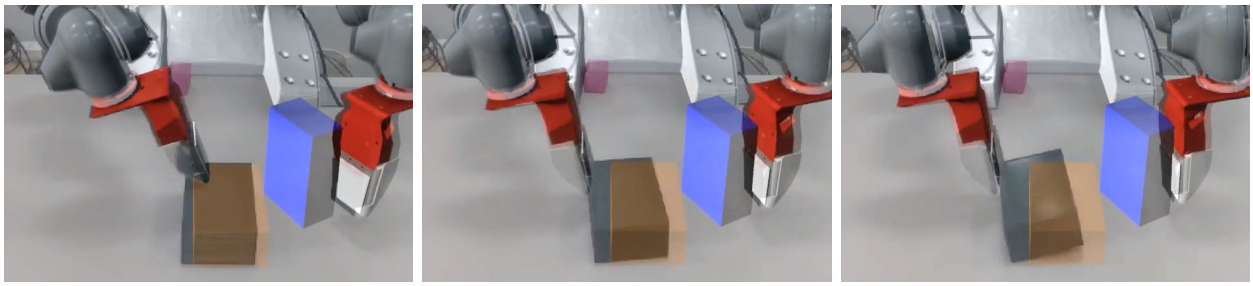
(c) Second relocation primitive (pulling) from grasping final position to desired configuration.

Figure 6.2: Open-loop execution of a manipulation task decomposition into pulling, grasping and pulling in a real environment. The black shape is the actual box, while the orange and blue overlays are the planned and final box poses, respectively.

Conversely, Figure 6.3 shows the result of an open-loop execution that leads to an error of a similar magnitude at an intermediate pose. However, in this case this small error ends up generating an unexpected collision between the object and a palm and, therefore, the failure of the manipulation task. This represents the risks of open-loop execution strategies without reactive control, which are especially important in manipulation due to object-robot interactions and dynamic environments, and motivates the development of a reactive planning framework like the one presented in this work.



(a) First relocation primitive (pulling) that leads to a small error in the final configuration.



(b) The attempt to start the reconfiguration primitive (grasping) fails due to the object pose error.

Figure 6.3: Open-loop execution of a manipulation task decomposition into pulling and grasping in a real environment, leading to failure due to collision. The black shape is the actual box, while the orange and blue overlays are the planned and final poses, respectively.

From the results above, we can conclude that open-loop executions may fail even if there are no external changes, but only due to mistakes from the own robot.

Of course, open-loop executions perform even worse when facing any kind of external perturbations that are not accounted in the original plan, as they cannot be handled properly due to the lack of sensing from the environment and reactive control.

This includes incorrect placing of the object within its expected initial configuration or a displacement of the object while being manipulated, which can lead to collisions as previously seen, dynamic load errors given by an unexpected resistance (Fig. 6.4) or losing the control of the object (Fig. 6.5). Even if these errors are enforced externally in the shown experiments, they are comparable to errors coming from unexpected conditions from the environment, such as unaccounted walls, surfaces with varying coefficients of friction, object weight distributions that are not uniform or imprecisions in the object shape.

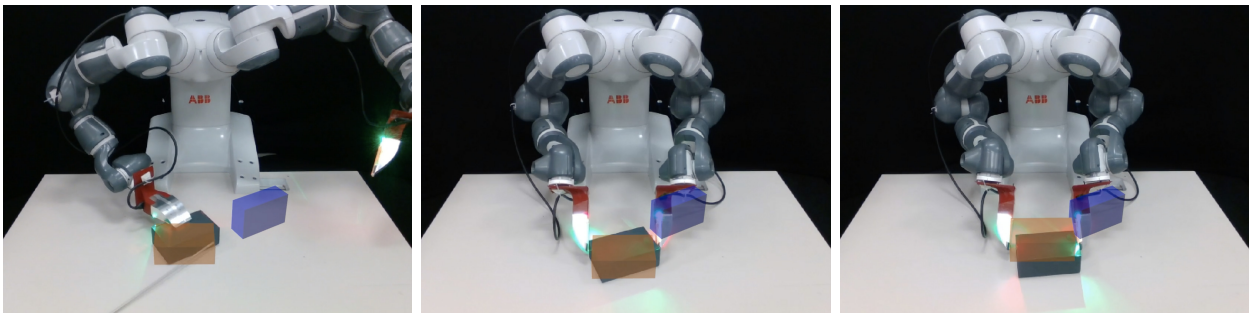


Figure 6.4: Open-loop execution of task that is externally perturbed while pulling the object, leading to failure due to a dynamic load error when achieving a grasp.

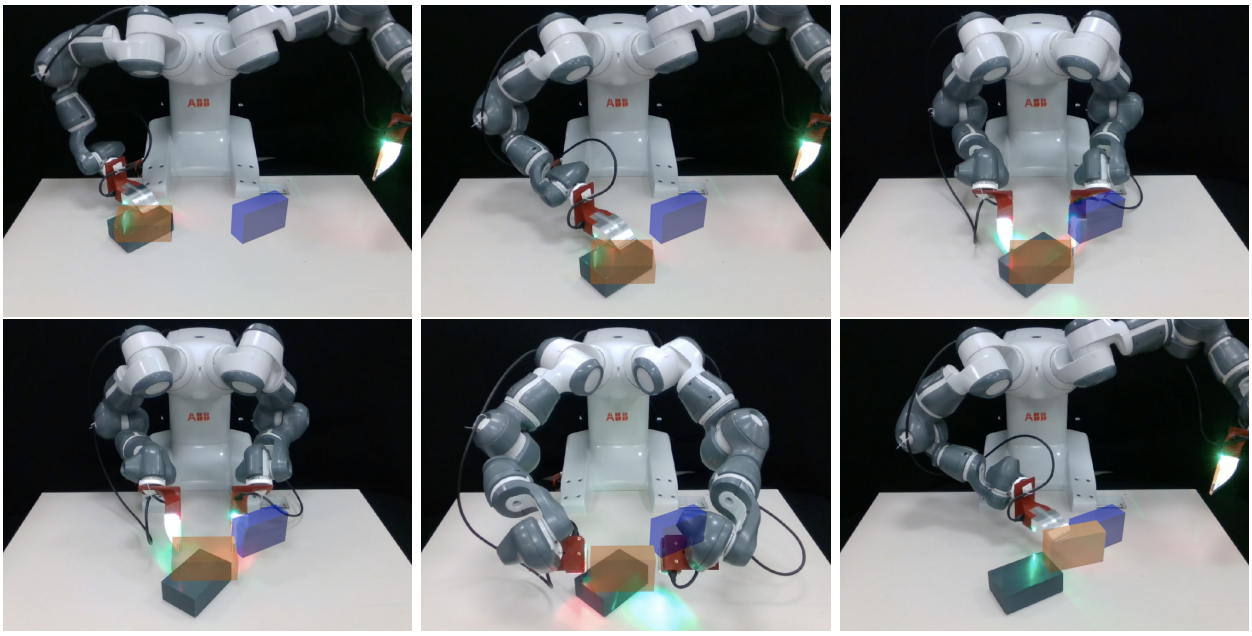


Figure 6.5: Open-loop execution of a manipulation task decomposition that starts from an incorrect position, which makes the system lose control of the object.

(2) Closed-loop execution of the task, with a reactive control given by tactile feedback and its processing through computer vision techniques for feature extraction.

Regardless of its nature, the reactive motion planning framework presented in this thesis is able to naturally integrate with any pose replanner without additional changes.

To this extent, Fig. 6.6 shows the reactive behavior of the framework in a simulated environment, without assuming a specific sensing technique, and its ability to react to an external perturbation in the object pose and correct it successfully.

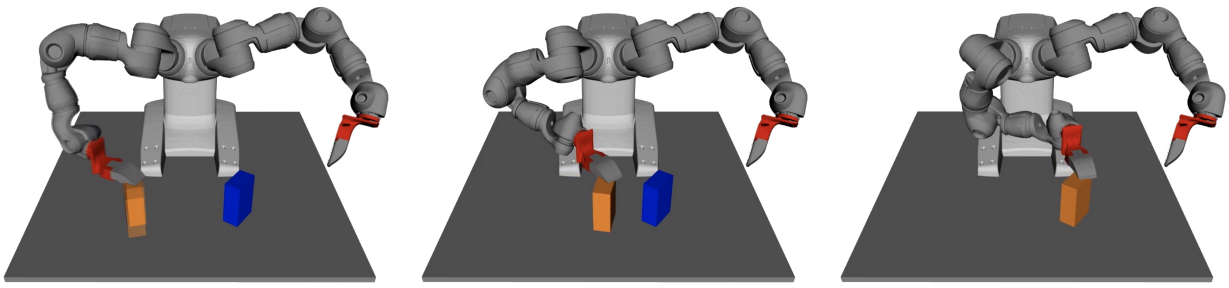
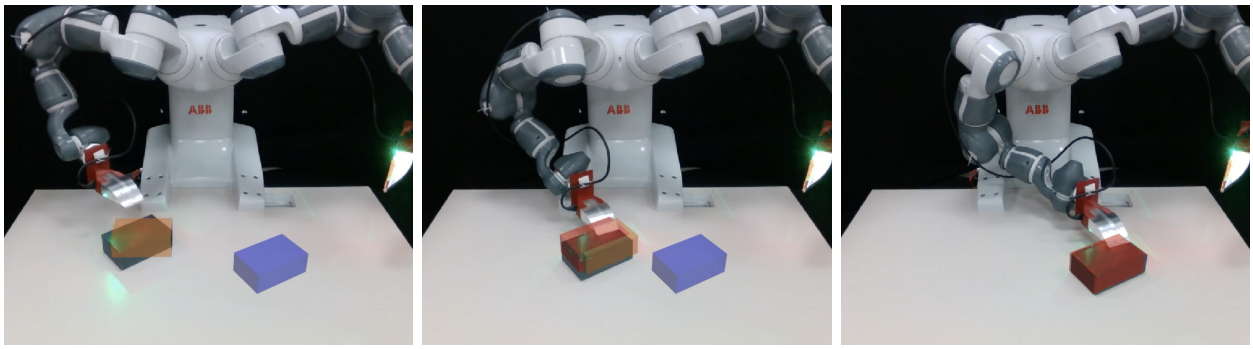


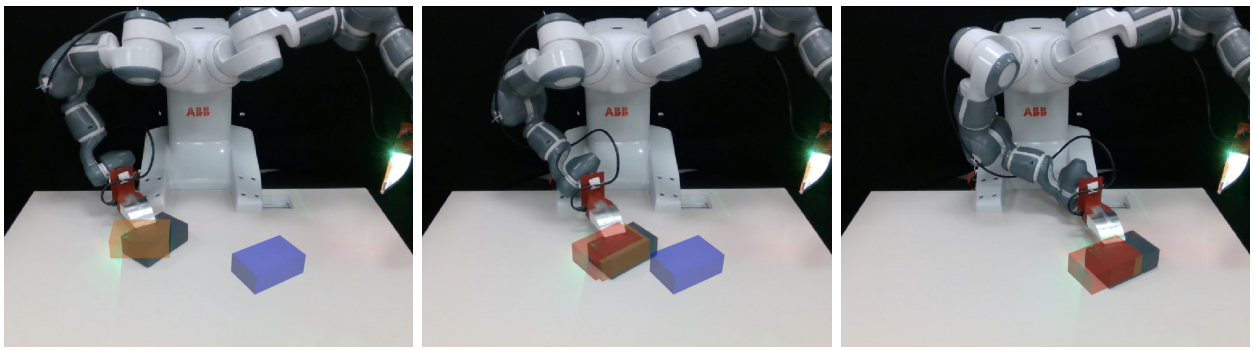
Figure 6.6: Closed-loop execution of task in a simulated environment. The manipulated object starts from a correct initial configuration, and is perturbed at a certain point (left). The end-effector uses its new relative position to bring the object to the final configuration.

These pose replanning strategies include previously mentioned algorithms using GelSlim sensors to extract features such as object localization, slip detection or force reconstruction and use them to update the end-effector trajectories according to the detected information.

For experimental validation, we consider the particular use of tactile feedback for object localization during the execution of the pulling primitive. In this case, we can see how the system is able to replan in order to drive the object to the final configuration, even if the initial configuration of the object is not correct (Fig. 6.7a) or in the presence of external disturbances while the object is being pulled (Fig. 6.8). Following plans are replanned to ensure continuity with the resulting joint configuration from the reactive execution.



(a) Successful correction in both planar axis thanks to tactile localization.



(b) Failed correction in the tangent axis to the detected line, due to the lack of observability within that axis.

Figure 6.7: Closed-loop executions of a task starting from a wrong initial configuration and partially corrected using line detection for tactile localization. The black shape is the actual box, while red, orange and blue are sensed, planned and final poses, respectively.

More specifically, object localization starts working when the GelSlim palms first touches the object and, through a line detection computer vision technique and an optimization problem, finds the best estimate of the current object pose, taking the original planned

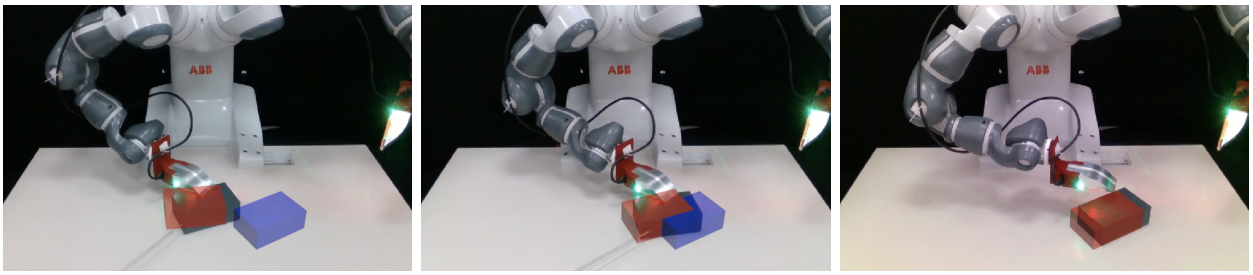


Figure 6.8: Closed-loop execution of a task in a real environment, that is externally perturbed while pulling the object. The end-effector uses its new relative position to bring the object to a final configuration that is aligned with the original one (due to the lack of observability of the detected line in one dimension).

pose as a reference in the first iteration. It is important to note that line detection has an important limitation in terms of observability, as any object pose lying in the axis of the detected line will lead to the same tactile imprint, but only the closest pose to the previous estimate will be returned. This can result in wrong final configurations such as in Fig. 6.7b, and could be improved by using, for example, corner detection techniques.

Chapter 7

Conclusions and future work

Motivated by the need of reactivity for dexterous manipulation, this thesis presents a motion planning and controller framework that enable robots to dynamically adapt their behavior to successfully achieve a task.

This is a particularly important challenge for dual-arm collaborative robotics. For this specific market, hardware development has significantly improved the available solutions in the last years, with the appearance of high-precision robots such as the ABB IRB 14000 (Fig. 1.3). However, planning and control frameworks are still not prepared for such complex setups, especially regarding simultaneous dual-arm planning and high frequency reactive control.

To this extent, the two main contributions of this work are:

1. A reactive motion planning framework for simultaneous dual-arm manipulation, based on the MoveIt! planning framework from ROS, that allows to quickly and reliably replan actions based on sensed information, which is critical for robustness in manipulation.

   This contrasts with current solutions that use arms as independent motion groups, e.g. setting an exclusive workspace for each or moving them in different periods of time, and therefore do not take advantage of potential synergies of dual-arm manipulation.

2. A high frequency execution controller for ABB robots, based on the EGM technology, which enables a control frequency of 250 Hz through position or velocity instructions in either Cartesian or joint space.

   This implementation of the EGM technology represents an improvement in comparison to the standard control technology that works at a maximum frequency of 5 Hz,

and is now publicly available at the MCube Lab's GitHub repository for use in any ABB industrial robot. As an example, the developed robot controller has already been integrated into several other projects at the MCube Lab.

In the same line, the presented robotic system has been developed with generality in mind such as to be extendable to different tasks, manipulators and sensors. This includes the EGM technology implementation just mentioned for any ABB robot, but also the reactive motion planning framework for dual-arm robots and any high-level pose planner.

In particular, this work has covered the integration into a general *tactile dexterity* framework that aims at performing the complex manipulation task of moving an object from an initial to a final configuration, by planning and exploiting contact rich interactions.

We experimentally validated our planning system on an ABB IRB 14000 dual-arm industrial collaborative robot, with high resolution GelSlim tactile sensors that are used to monitor the progress of the task and drive the reactive behavior of the robot to counter mistakes or unaccounted environment conditions in a closed-loop fashion.

However, there is still room for improving the presented framework in terms of extended capabilities, time performance, formal guarantees and robustness.

Regarding the reactive motion planning framework, using MoveIt! as a starting point presents a good opportunity to take advantage of its modular design. More specifically, MoveIt! and therefore our proposed implementation are compatible with many motion planners of different natures. A complete study of these could help to find a good compromise between planning time and path optimality, which is not critical for most applications.

In this same line, exploiting the static nature of some obstacles (e.g. tables or other environment features) through pre-cached motion plans or approaches like experience graphs [84] can significantly speed up offline motion planning, which is currently the slower process.

Another MoveIt! module to consider is the IK solver. To this extent, most well-known IK solvers only work for kinematic chains, which is the main reason why Cartesian targets can be accepted only for individual arms. In particular, this requires to apply a state validity check after resynchronizing joint trajectories, with the only option of planning again in case it fails. This strategy lacks of convergence guarantees and only relies on the randomness of numerical IK or motion planning. Therefore, a preferable option could be developing our own IK solver as a non-linear optimization problem for both arms simultaneously with self-

collision avoidance. This could be integrated as a MoveIt! module and would allow us to solve the extended motion planning problem for Cartesian targets and both arms directly.

This same lack of guarantees takes place in replanning, as our assumption of *local* or *small* adjustments is useful in practice, but not formally precise. Further analysis considering numerical stability of the Jacobian matrix could determine when online IK is enough.

Another relevant point are contact modes with the manipulated object. For example, while approaching the object requires collision avoidance, contact is required when actually manipulating the object. Currently, since only static obstacle regions can be considered, collision avoidance is disabled if any subplan requires contact. Therefore, developing tools to handle dynamic obstacle regions and ensuring correct contact modes could be an interesting path.

Finally, we consider that the ultimate purpose of our framework is to offer robustness in offline and online planning. Hence, further development is required to ensure compatibility with multiple object configurations, manipulation primitives and end-effector plans.

Bibliography

[1] ABB. Application manual: Controller software IRC5 [Online]. URL here.

[2] ABB. Application manual: Robot reference interface [Online]. URL here.

[3] ABB. Latest robot controller software from ABB combines flexibility and productivity for developers [Online], December 2014. URL here.

[4] B. Aceituno-Cabezas, J. Ballester, and A. Rodriguez. Certified Grasping. *International Symposium on Robotics Research (ISRR)*, 2019 (under review).

[5] B. Aceituno-Cabezas, H. Dai, and A. Rodriguez. A convex-combinatorial model for planar caging. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019 (under review).

[6] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo. OBPRM: An obstacle-based PRM for 3D workspaces. *International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 1998.

[7] J. Barraquand and J.-C. Latombe. Robot motion planning: A distributed representation approach. *The International Journal of Robotics Research*, 10(6):628–649, 1991.

[8] M. Bauza, O. Canal, and A. Rodriguez. Tactile mapping and localization from high-resolution tactile imprints. *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[9] P. Beeson and B. Ames. TRAC-IK: an open-source library for improved solving of generic inverse kinematics. *IEEE-RAS International Conference on Humanoid Robots*, 2015.

[10] K. E. Bekris, B. Y. Chen, A. Ladd, E. Plaku, and L. E. Kavraki. Multiple query probabilistic roadmap planning using single query primitives. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2003.

[11] Berkeley Automation Lab. YuMi Python Interface on GitHub [Online]. URL here.

[12] A. Bicchi, C. Melchiorri, and D. Balluchi. On the mobility and manipulability of general multiple limb robots. *IEEE Transactions on Robotics and Automation*, 11(2):215–228, 1995.

[13] A. Bicchi, R. Sorrentino, et al. Dexterous manipulation through rolling. *IEEE International Conference on Robotics and Automation (ICRA)*, 1995.

[14] A. Blomdell, I. Dressler, K. Nilsson, and A. Robertsson. Flexible application development and high-performance motion control based on external sensing and reconfiguration of ABB industrial robot controllers. *IEEE International Conference on Robotics and Automation (ICRA)*, 2010.

[15] R. Bohlin and L. Kavraki. Path planning using Lazy PRM. *IEEE International Conference on Robotics and Automation (ICRA)*, 2000.

[16] V. Boor, M. H. Overmars, and A. F. van der Stappen. The Gaussian sampling strategy for probabilistic roadmap planners. *IEEE International Conference on Robotics and Automation (ICRA)*, 1999.

[17] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible Markup Language 1.0 (Fifth Edition). W3C Recommendation [Online]. URL here.

[18] J. Bruce and M. Veloso. Real-time randomized path planning for robot navigation. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2002.

[19] X.-N. Bui, J.-D. Boissonnat, P. Soueres, and J.-P. Laumond. Shortest path synthesis for Dubins non-holonomic robot. *IEEE International Conference on Robotics and Automation (ICRA)*, 1994.

[20] B. Burns and O. Brock. Sampling-based motion planning using predictive models. *IEEE International Conference on Robotics and Automation (ICRA)*, 2005.

[21] J. F. Canny. *The Complexity of Robot Motion Planning*. MIT Press, 1988.

[22] N. Chavan-Dafle and A. Rodriguez. Sampling-based planning for in-hand manipulations with external pushes. *International Symposium on Robotics Research (ISRR)*, 2017.

[23] B. Chazelle. Approximation and decomposition of shapes. In J. T. Schwartz and C. K.

Yap, editors, *Algorithmic and Geometric Aspects of Robotics*, pages 145–185. Lawrence Erlbaum Associates, 1987.

[24] K. Cho, M. Kim, and J.-B. Song. Complete and rapid regrasp planning with look-up table. *Journal of Intelligent and Robotic Systems*, 36(4):371–387, 2003.

[25] H. Choset. *Robotic Motion Planning: Configuration space*. Course notes, The Robotics Institute at Carnegie Mellon University, 2007. URL here.

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

[27] J. Colgate, M. Peshkin, and W. Wannasuphoprasit. Nonholonomic haptic display. *IEEE International Conference on Robotics and Automation (ICRA)*, 1996.

[28] J. Colgate, W. Wannasuphoprasit, and M. Peshkin. Cobots: robots for collaboration with human operators. *ASME International Mechanical Engineering Congress and Exhibition (IMECE)*, 1996.

[29] J. J. Craig. *Introduction to Robotics: Mechanics and Control*. Pearson, 2005.

[30] M. Dawson-Haggerty. Open ABB Driver on GitHub [Online]. URL here.

[31] R. Deits, T. Marcucci, L. Manuelli, T. Koolen, and R. Tedrake. Approximate explicit model predictive control for push recovery using mixed-integer convex optimization. *Dynamic Walking Conference*, 2017.

[32] R. Diankov. IKFast: The robot kinematics compiler [Online]. URL here.

[33] R. Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute, August 2010. URL here.

[34] S. Dong, D. Ma, E. Donlon, and A. Rodriguez. Maintaining grasps within slipping bound by monitoring incipient slip. *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[35] E. Donlon, S. Dong, M. Liu, J. Li, E. Adelson, and A. Rodriguez. GelSlim: a high-resolution, compact, robust, and calibrated tactile-sensing finger. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[36] M. Erdmann. An exploration of nonprehensile two-palm manipulation. *The International Journal of Robotics Research*, 17(5):485–503, 1998.

[37] M. A. Erdmann and M. T. Mason. An exploration of sensorless manipulation. *IEEE Journal on Robotics and Automation*, 4(4):369–379, 1988.

[38] R. Geraerts and M. Overmars. Sampling techniques for probabilistic roadmap planners. *International Conference on Intelligent Autonomous Systems*, 2004.

[39] H. H. Gonzalez-Banos, D. Hsu, and J. C. Latombe. *Autonomous Mobile Robots: Sensing, Control, Decision-Making and Applications*. CRC Press, 2006.

[40] Google Developers. Protocol Buffers [Online]. URL here.

[41] G. Hirzinger, J. Bals, M. Otter, and J. Stelter. The DLR-KUKA success story: robotics research improves industrial robots. *IEEE Robotics Automation Magazine*, 2005.

[42] F. Hogan. *Tactile Dexterity with Robotic Palms*. PhD thesis proposal, Massachusetts Institute of Technology, 2018.

[43] F. Hogan and A. Rodriguez. Feedback control of the pusher-slider system: a story of hybrid and underactuated contact dynamics. *International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2016.

[44] F. Hogan, E. Grau, and A. Rodriguez. Reactive planar manipulation with convex hybrid MPC. *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[45] C. Holleman and L. E. Kavraki. A framework for using the workspace medial axis in PRM planners. *IEEE International Conference on Robotics and Automation (ICRA)*, 2000.

[46] Y. Hou, Z. Jia, A. M. Johnson, and M. T. Mason. Robust planar dynamic pivoting by regulating inertial and grip forces. *International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2016.

[47] Y. Hou, Z. Jia, and M. T. Mason. Fast planning for 3D any-pose-reorienting using pivoting. *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[48] D. Hsu, T. Jiang, J. Reif, and Z. Sun. The bridge test for sampling narrow passages

with probabilistic roadmap planners. *IEEE International Conference on Robotics and Automation (ICRA)*, 2003.

[49] P. Isto. Constructing probabilistic roadmaps with powerful local planning and path optimization. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2002.

[50] P. Jiménez, F. Thomas, and C. Torras. Collision detection algorithms for motion planning. In J.-P. Laumond, editor, *Robot Motion Planning and Control*, pages 305–338. Springer-Verlag.

[51] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal. STOMP: Stochastic trajectory optimization for motion planning. *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[52] M. Kallmann and M. Mataric. Motion planning using dynamic roadmaps. *IEEE International Conference on Robotics and Automation (ICRA)*, 2004.

[53] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[54] Y. Karayiannidis, C. Smith, D. Kragic, et al. Adaptive control for pivoting with visual and tactile feedback. *IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

[55] L. Kavraki and S. M. LaValle. *Handbook of Robotics*. Springer, 2008.

[56] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):556–580, 1996.

[57] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5(1):90–98, 1986.

[58] J. C. Latombe. *Robot Motion Planning*. Kluwer, 1991.

[59] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[60] S. M. LaValle. Motion planning: Living in C-space. *IEEE International Conference on Robotics and Automation (ICRA)*, 2012. URL here. Tutorial.

[61] S. M. LaValle and J. J. Kuffner. Rapidly-exploring random trees: Progress and

prospects. In B. R. Donald, K. M. Lynch, and D. Rus, editors, *Algorithmic and Computational Robotics: New Directions*, pages 293–308. A K Peters.

[62] S. M. LaValle, M. S. Branicky, and S. R. Lindemann. On the relationship between classical grid search and probabilistic roadmaps. *The International Journal of Robotics Research*, 23(7/8):673–692, 2004.

[63] D. Leven and M. Sharir. Planning a purely translational motion for a convex object in two-dimensional space using generalized voronoi diagrams. *Discrete and Computational Geometry*, 2:9–31, 1982.

[64] P. Leven and S. A. Hutchinson. Real-time motion planning for agile autonomous vehicles. *AIAA Journal of Guidance and Control*, 25(1):116–129, 2002.

[65] P. Leven and S. A. Hutchinson. Using manipulability to bias sampling during the construction of probabilistic roadmaps. *IEEE Transactions on Robotics and Automation*, 19(6):1020–1026, 2003.

[66] J.-M. Lien, S. L. Thomas, and N. M. Amato. A general framework for sampling on the medial axis of the free space. *IEEE International Conference on Robotics and Automation (ICRA)*, 2003.

[67] M. C. Lin and J. F. Canny. Efficient algorithms for incremental distance computation. *IEEE International Conference on Robotics and Automation (ICRA)*, 1991.

[68] S. R. Lindemann and S. M. LaValle. Current issues in sampling-based motion planning. *Robotics Research: The Eleventh International Symposium*, 2005.

[69] T. Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, C-32(2):108–120, 1983.

[70] R. Lundqvist and T. Soreling. *New Interface for Rapid Feedback Control on ABB-robots*. MSc thesis, Linkoping Institute of Technology, 2005.

[71] K. Lynch, H. Maekawa, and K. Tanie. Manipulation and active sensing by pushing using tactile feedback. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1992.

[72] D. Ma, E. Donlon, S. Dong, and A. Rodriguez. Dense tactile force distribution estimation using GelSlim and inverse FEM. *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[73] Ø. Mæhre. *Following Moving Objects Using Externally Guided Motion (EGM)*. MSc thesis, University of Stavanger, 2016.

[74] M. Mani and R. D. W. Wilson. A programmable orienting system for flat parts. *North American Manufacturing Research Conference*, 1985.

[75] T. Marcucci, R. Deits, M. Gabiccini, A. Bicchi, and R. Tedrake. Approximate hybrid model predictive control for multi-contact push recovery in complex environments. *IEEE-RAS International Conference on Humanoid Robots*, 2017.

[76] M. Mason. Mechanics and planning of manipulator pushing operations. *The International Journal of Robotics Research*, 1986.

[77] S. S. Mirrazavi, N. Figueroa, and A. Billard. Coordinated multi-arm motion planning: Reaching for moving objects in the face of uncertainty. *Robotics: Science and Systems (RSS)*, 2016.

[78] MIT MCube Lab. ABB ROS Node on GitHub [Online]. URL here.

[79] D. Nieuwenhuisen and M. H. Overmars. Useful cycles in probabilistic roadmap graphs. *IEEE International Conference on Robotics and Automation (ICRA)*, 2004.

[80] N. J. Nilsson. A mobile automaton: An application of artificial intelligence techniques. *International Conference on Artificial Intelligence*, 1969.

[81] C. O'Dunlaing and C. K. Yap. A retraction method for planning the motion of a disc. *Journal of Algorihtms*, 6:104–111, 1982.

[82] M. Otte and E. Frazzoli. RRTX: asymptotically optimal single-query sampling-based motion planning with quick replanning. *The International Journal of Robotics Research*, 35(7):797–822, 2016.

[83] E. Paljug and X. Yun. Experimental results of two robot arms manipulating large objects. *IEEE International Conference on Robotics and Automation (ICRA)*, 1993.

[84] M. Philips, B. Cohen, S. Chitta, and M. Likhachev. E-Graphs: Bootstrapping Planning with Experience Graphs. *Robotics: Science and Systems (RSS)*, 2012.

[85] M. Posa, C. Cantu, and R. Tedrake. A direct method for trajectory optimization of rigid bodies through contact. *The International Journal of Robotics Research*, 33(1): 69–81, 2014.

[86] M. Posa, S. Kuindersma, and R. Tedrake. Optimization and stabilization of trajectories for constrained dynamical systems. *IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

[87] N. Ratliff, M. Zucker, J. A. D. Bagnell, and S. Srinivasa. CHOMP: Gradient optimization techniques for efficient motion planning. *IEEE International Conference on Robotics and Automation (ICRA)*, 2009.

[88] J. H. Reif. Complexity of the mover's problem and generalizations. *20th Annual Symposium on Foundations of Computer Science*, 1979.

[89] A. Rodriguez, M. T. Mason, and S. Ferry. From caging to grasping. *The International Journal of Robotics Research*, 31(7):886–900, 2012.

[90] S. Rodriguez, J.-M. Lien, and N. M. Amato. A framework for planning motion in environments with moving obstacles. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2007.

[91] F. Rohrdanz and F. M. Wahl. Generating and evaluating regrasp operations. *IEEE International Conference on Robotics and Automation (ICRA)*, 1997.

[92] ROS Industrial Consortium. ABB EGM library on github [Online], . URL here.

[93] ROS Industrial Consortium. ABB meta-package on GitHub [Online], . URL here.

[94] J. Ryu, F. Ruggiero, and K. Lynch. Control of nonprehensile rolling manipulation: Balancing a disk on a disk. *IEEE International Conference on Robotics and Automation (ICRA)*, 2012.

[95] T. Sakaguchi, S. Sato, and F. Miyazaki. Robot motion planning for moving objects. *IFAC 12th Triennal World Congress*, 1993.

[96] N. Sawasaki and H. Inoue. Tumbling objects using a multi-fingered robot. *Journal of the Robotics Society of Japan*, 9(5):560–571, 1991.

[97] J. Schulman, J. Ho, A. Lee, I. Awwal, H. Bradlow, and P. Abbeel. Finding locally optimal, collision-free trajectories with sequential convex optimization. *Robotics: Science and Systems (RSS)*, 2013.

[98] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel. Motion planning with sequential convex optimization and convex

collision checking. *The International Journal of Robotics Research*, 33(9):1251–1270, 2014.

[99] J. T. Schwartz and M. Sharir. A survey of motion planning and related geometric algorithms. *Artificial Intelligence Journal*, 1988.

[100] M. Sharir. Algorithmic motion planning. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, pages 1037–1064. Chapman and Hall/CRC Press.

[101] J. Shi, J. Z. Woodruff, P. B. Umbanhowar, and K. M. Lynch. Dynamic in-hand sliding manipulation. *IEEE Transactions on Robotics*, 33(4):778–795, 2017.

[102] T. Siméon, J.-P. Laumond, and C. Nissoux. Visibility based probabilistic roadmaps for motion planning. *Advanced Robotics*, 14(6), 2000.

[103] C. Smith, Y. Karayiannidis, L. Nalpantidis, X. Gratal, P. Qi, D. V. Dimarogonas, and D. Kragic. Dual arm manipulation - a survey. *Robotics and Autonomous Systems*, 2012.

[104] R. Smits. KDL: Kinematics and Dynamics Library [Online]. URL here.

[105] R. Smits and other contributors at GitHub. Orocos Kinematics and Dynamics C++ library on GitHub [Online]. URL here.

[106] S. A. Stoeter, S. Voss, N. P. Papanikolopoulos, and H. Mosemann. Planning of regrasp operations. *IEEE International Conference on Robotics and Automation (ICRA)*, 1999.

[107] M. Strandberg. Augmenting rrt-planners with local trees. *IEEE International Conference on Robotics and Automation (ICRA)*, 2004.

[108] I. A. Şucan and S. Chitta. MoveIt! [Online]. URL here.

[109] I. A. Şucan, M. Moll, and L. E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, 2012. URL here.

[110] H. Terasaki and T. Hasegawa. Motion planning of intelligent manipulation by a parallel two-fingered gripper equipped with a simple rotating mechanism. *IEEE Transactions on Robotics and Automation*, 14(2):207–219, 1998.

[111] J. Tjerngren. Ease-of-Use Packages between ROS and ABB Robots. *ROS-Industrial Conference*, 2018. URL here.

[112] P. Tournassoud, T. Lozano-Pérez, and E. Mazer. Regrasping. *IEEE International Conference on Robotics and Automation (ICRA)*, 1987.

[113] TRACLabs. TRAC-IK ROS package [Online]. URL here.

[114] J. C. Trinkle. On the stability and instantaneous velocity of grasped frictionless objects. *IEEE Transactions on Robotics and Automation*, 8(5):560–572, 1992.

[115] J. C. Trinkle, J. M. Abel, and R. P. Paul. An investigation of frictionless enveloping grasping in the plane. *The International Journal of Robotics Research*, 7(3):33–51, 1988.

[116] W. Wan, M. T. Mason, R. Fukui, and Y. Kuniyoshi. Improving regrasp algorithms to analyze the utility of work surfaces in a workcell. *IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[117] J. Woodruff and K. Lynch. Planning and control for dynamic, nonprehensile, and hybrid manipulation tasks. *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[118] A. Yershova, L. Jaillet, T. Simeon, and S. M. LaValle. Dynamic-domain RRTs: Efficient exploration by controlling the sampling domain. *IEEE International Conference on Robotics and Automation (ICRA)*, 2005.

[119] X. Yun. Object handling using two arms without grasping. *The International Journal of Robotics Research*, 12(1):99–106, 1993.

[120] J. Zhou, R. Paolini, J. Bagnell, and M. Mason. A convex polynomial force-motion model for planar sliding: Identification and application. *IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

[121] J. Zhou, J. Bagnell, and M. Mason. A fast stochastic contact model for planar pushing and grasping: Theory and experimental validation. *Robotics: Science and Systems (RSS)*, 2017.

[122] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa. CHOMP: Covariant hamiltonian optimization for motion planning. *The International Journal of Robotics Research*, 32(9-10):1164–1193, 2013.