



**HAL**  
open science

# Planning of visual servoing tasks for robotics

Alexis Nicolin

► **To cite this version:**

Alexis Nicolin. Planning of visual servoing tasks for robotics. Automatic. INSA de Toulouse, 2022. English. NNT : 2022ISAT0003 . tel-03659668v2

**HAL Id: tel-03659668**

**<https://laas.hal.science/tel-03659668v2>**

Submitted on 23 May 2022

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Université  
Fédérale

Toulouse  
Midi-Pyrénées

# THÈSE

En vue de l'obtention du  
**DOCTORAT DE L'UNIVERSITÉ DE  
TOULOUSE**

Délivré par :  
*l'Institut National des Sciences Appliquées de Toulouse (INSA de Toulouse)*

---

---

Présentée et soutenue le 17 Février 2022 par :  
**Alexis Nicolin**

**Planning of visual servoing tasks for robotics**

---

---

## JURY

SAMER ALFAYAD	Professeur des universités	Président du jury
JEAN-BERNARD HAYET	Investigador titular B	Rapporteur
WAELE SULEIMAN	Associate professor	Examineur
VIVIANE CADENAT	Maîtresse de conférences	Examinatrice
OLIVIER STASSE	Directeur de Recherche	Directeur de thèse
FLORENT LAMIRAUX	Directeur de Recherche	Co-directeur de thèse
SÉBASTIEN BORJA	Ingénieur R&D	Encadrant entreprise

---

**École doctorale et spécialité :**

*EDSYS : Robotique 4200046*

**Unité de Recherche :**

*LAAS - Laboratoire d'Analyse et d'Architecture des Systèmes (UPR 8001)*

**Directeur(s) de Thèse :**

*Olivier Stasse et Florent Lamiraux*

**Rapporteurs :**

*Samer Alfayad et Jean-Bernard Hayet*





---

## Remerciements

Je souhaiterais en premier lieu remercier mes directeurs de thèse, Olivier Stasse et Florent Lamiraux, pour m'avoir proposé cette thèse et guidé au long de ces 3 années. Je remercie aussi Sébastien Boria et Damien Van Damme qui ont été leurs pendants côté entreprise. J'ai pu grâce à eux continuer à découvrir et progresser dans ce vaste domaine qu'est la robotique, passion découverte un peu par hasard au détour d'un couloir, pas très loin du laboratoire. Merci aussi à Viviane Cadenat et Philippes Souères, membres du comité du suivi de ma thèse, pour leur conseils m'ayant rassuré lors de moments de doute.

Merci à Samer Alfayad d'avoir accepté les rôles de rapporteur mais aussi de président du jury de ma soutenance. Et merci à Jean-Bernard Hayet pour avoir accepté d'être second rapporteur, membre du jury, et pour la correction d'une grande partie des coquilles qui avait échappées aux relectures. Merci aussi à Viviane Cadenat et Wael Suleiman d'avoir été membres de mon jury de thèse.

Un grand merci à tous les membres de l'équipe, passées ou présents, étudiants ou diplômés, gepettistes ou adoptés. Grâce à vous, le LAAS est bien plus qu'un endroit où l'on travaille et étudie, c'est aussi un lieu où on partage et on vit (de pâtisseries). Et merci aussi pour toutes ces soirées et ces week-ends hors du laboratoire, et en espérant y être toujours convié. Une mention spéciale pour le bureau B181, soutien quotidien de ces années de thèse.

Merci à ma famille, et à tou.te.s mes ami.e.s, pour ces soirées à créer des robots, des radeaux, et de bons moments. Enfin, un merci tout particulier à Aurélie et à Poe pour avoir été présents chaque jour à mes côtés.

---

# Contents

<b>List of Figures</b>	<b>v</b>
<b>List of Tables</b>	<b>vii</b>
<b>List of Algorithms</b>	<b>vii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Context . . . . .	2
1.2 Robots and software . . . . .	3
1.3 Objectives and contributions of the thesis . . . . .	5
1.4 Organisation of this manuscript . . . . .	6
1.5 Publications . . . . .	7
<b>2 State of the art</b>	<b>9</b>
2.1 Motion planning . . . . .	10
2.2 Constrained motion planning . . . . .	17
2.3 Task planning . . . . .	22
2.4 Manipulation planning . . . . .	27
2.5 Control laws for robots . . . . .	30
2.6 Visual servoing . . . . .	32
<b>3 Agimus</b>	<b>35</b>
3.1 Introduction . . . . .	37
3.2 Concepts . . . . .	37
3.3 Software . . . . .	53
3.4 Experiments . . . . .	62
3.5 Conclusion . . . . .	73
<b>4 Calibration</b>	<b>75</b>
4.1 Introduction . . . . .	76
4.2 Calibration using HPP's Newton-Raphson projection . . . . .	77
4.3 Calibration as several manipulator arms . . . . .	80

---

4.4	Whole-body elasto-geometric calibration of a TALOS robot . . . . .	82
4.5	Conclusion . . . . .	83
<b>5</b>	<b>Agimus: Visual Servoing</b>	<b>85</b>
5.1	Introduction . . . . .	86
5.2	Visual Servoing . . . . .	86
5.3	Experiments . . . . .	91
5.4	Conclusion . . . . .	99
<b>6</b>	<b>Middle Sized Drilling Robot</b>	<b>101</b>
6.1	Introduction . . . . .	102
6.2	Presentation of the robot . . . . .	102
6.3	Drilling process and its challenges . . . . .	105
6.4	Conclusion . . . . .	107
<b>7</b>	<b>Conclusion</b>	<b>109</b>
	<b>Bibliography</b>	<b>113</b>
	<b>Glossary</b>	<b>131</b>
	<b>Acronyms</b>	<b>133</b>

# List of Figures

1.1	Some industrial robots. . . . .	2
1.2	The robots used during this thesis. . . . .	4
2.1	The piano mover’s problem . . . . .	10
2.2	Comparison between Rapidly-exploring Random Tree and Probabilistic Road-Maps. . . . .	13
2.3	Robots assemble an IKEA chair . . . . .	17
2.4	Humanoid motion planning using a decoupled approach. . . . .	21
2.5	Example of a STRIPS tree. . . . .	24
2.6	Rearrangement planning . . . . .	28
2.7	A Romeo robot grasps a soda can using visual servoing . . . . .	33
3.1	TALOS manipulates a wooden plank . . . . .	37
3.2	The four handles on the plank manipulated by TALOS. . . . .	39
3.3	Simple graph of constraints with three states . . . . .	40
3.4	Graph of constraints with waypoints . . . . .	42
3.5	Initial estimation projected onto the graph of constraints . . . . .	43
3.6	Transitions taken in the graph of constraints . . . . .	45
3.7	Visual Servoing Platform’s logo . . . . .	53
3.8	An overview of ros_control and its place in a robotics software . . . . .	58
3.9	Architecture of Agimus . . . . .	62
3.10	TALOS humanoid robot developed by PAL Robotics . . . . .	63
3.11	Some of the humanoid robots used in the DARPA Robotics Challenge	64
3.12	Humanoid robots in the years following the DARPA Robotics Challenge	66
3.13	A typical run of the plank manipulation experiment . . . . .	69
4.1	Comparison of the joints’ offsets of TALOS before and after calibration	76
4.2	TALOS humanoid robot in a Motion Capture system setup . . . . .	79
4.3	Illustration of the Denavit-Hartenberg parameters . . . . .	81
4.4	The various deformations on an industrial robot. . . . .	82
4.5	RMS error of the tracking of the robot before and after calibration . .	83

5.1	The ambiguity problem in pose estimation . . . . .	87
5.2	TALOS robot with AprilTag markers on its wrists . . . . .	89
5.3	TALOS robot placing a plank on a table . . . . .	92
5.4	Tracking error between the gripper and the plank . . . . .	92
5.6	TIAGo robot, made by PAL Robotics . . . . .	93
5.5	TALOS manipulates a plank with visual servoing . . . . .	94
5.7	REEM-C (left) and REEM (right) . . . . .	95
5.8	3D printed mockup of a subcomponent of an Airbus A380 engine pylon	96
5.9	Representation of deburring tasks in Agimus . . . . .	97
5.10	TIAGo experiment with deburring holes on an aircraft's part . . . . .	98
6.1	Airbus A321 automated fuselage assembly line in Hamburg Finken- werder . . . . .	103
6.2	M-800iA/60 manipulator from Fanuc . . . . .	104
6.3	The Middle Sized Drilling Robot . . . . .	105
6.4	The Middle Sized Drilling Robot uses reference holes to adapt its offline programming . . . . .	106

# List of Tables

3.1	Descriptions of some robots used in the DARPA Robotics Challenge .	65
-----	--	----

# List of Algorithms

1	Projection on implicit constraints . . . . .	44
2	Manipulation Rapidly-exploring Random Tree . . . . .	45





# Chapter 1

## Introduction

### Contents

---

<b>1.1</b>	<b>Context . . . . .</b>	<b>2</b>
1.1.1	ROB4FAM joint laboratory . . . . .	3
<b>1.2</b>	<b>Robots and software . . . . .</b>	<b>3</b>
1.2.1	Software . . . . .	3
1.2.2	Robots . . . . .	4
<b>1.3</b>	<b>Objectives and contributions of the thesis . . . . .</b>	<b>5</b>
<b>1.4</b>	<b>Organisation of this manuscript . . . . .</b>	<b>6</b>
<b>1.5</b>	<b>Publications . . . . .</b>	<b>7</b>

---

## 1.1 Context

Robotics is the latest of the industrial revolutions, which began in the 1950s with Unimate, a robot conceived by George Devol. Starting in 1961, this robot was used by General Motors to move die castings and weld them to cars' bodies. Today, it is estimated that more than 3 million robots are used in factories throughout the world, mainly in automotive and electronics industries.

They serve multiple purposes centred around the goal of assisting or even replacing human operators in repetitive, dull or dangerous tasks. Many forms of robots exist, from the well-known industrial robot manipulators, fig. 1.1a, to delta robots, fig. 1.1b, two-wheeled mobile robots, and humanoid robots, etc. This last type of robot easily impress the public, but its complexity has kept it far from the factory floors for a long time. They indeed require considerable computing power, battery capacity and software ingenuity to function correctly and be able to react quickly to their environment. On the other hand, industrial robot manipulators are simple and often fixed into the ground. Their programming has been done offline for decades, first manually, and more recently via modelling software or interactive commands.



©Fanuc



CC BY-SA 4.0 Marc Audélas

(a) A 6-axis robot

(b) A delta robot

Figure 1.1: Some industrial robots.

### 1.1.1 ROB4FAM joint laboratory

This thesis is part of a joint laboratory between the Laboratory for Analysis and Architecture of Systems (LAAS-CNRS) and the aeronautic company Airbus Operations. The aim of this partnership is to develop innovative solutions to offer more responsiveness and decision autonomy to industrial robots in the context of aeronautic construction. The long-term research target is to have a humanoid robot able to climb into a plane on the assembly line, drill some holes and come back, all without human assistance. This goal is currently thought to be many years into the future, so there are short-term goals more in line with the industrial wills.

Those short-term goals are structured around two CIFRE theses, two academic theses, and several post-doctorate positions to provide guidance and experience. The first goal is for the robot to locate itself robustly in an industrial environment, which is at the same time very structured and organised, for safety reasons, but also subject to drastic change over the course of weeks due to business needs. A second target, which is covered by the second CIFRE, is the development of a new form of control for the robots, based on measured forces and torques, to increase their intrinsic safety when working near human operators. The third goal and subject of this thesis is the demonstration of the viability of state-of-the-art automated planning methods for industrial tasks. This also includes the simultaneous formulation of control schemes able to react to slight changes of the plan, such as holes already drilled, without human intervention. Finally, all those targets needs to be modular, in the sense that we want to use the same concepts and software on a research humanoid robot and on an industrial arm, with as few modifications as possible.

## 1.2 Robots and software

The concepts and ideas developed during this thesis were based on existing software that will be shortly presented in the following paragraphs. More in-depth presentations will be done in the corresponding chapters. Those developments were then tested on three very different robots, that we will also describe broadly thereafter, before a complete introduction in their respective chapters.

### 1.2.1 Software

#### **Visual Servoing Platform (ViSP) [3]**

It is a visual servoing framework developed by the Rainbow team at INRIA in Rennes, France. In our projects, we are using it to provide computer vision algorithms to our robots.

#### **Pinocchio [4]**

This library implements state-of-the-art rigid body algorithms for articulated systems. It was created by the Gepetto team at LAAS-CNRS in Toulouse, France. Pinocchio is one of the fundamental brick in our other software.



Figure 1.2: The robots used during this thesis.

### Humanoid Path Planner (HPP) [5]

It is a motion planning library developed by the Gepetto team. It is able to handle the planning of manipulation motions for complex robots such as bi-handed humanoids. But HPP is also quite modular and able to plan for robotic manipulator arms, mobile robot or even cable robots.

### Stack-of-Tasks (SoT) [6]

This is a software implementing a control architecture for redundant robots. It is based of the notion of tasks from Samson et al. [7] and uses a strict hierarchy of them to compute the whole body control commands sent to the robot.

### Robot Operating System (ROS) [8]

It is an open-source middleware suite that became the *de facto* standard framework for light robotics applications. This software serves as an abstraction of the robot's hardware and provides services such as inter-process messaging for the users. ROS is at the core of our two robots made by PAL Robotics.

## 1.2.2 Robots

### Pyrène: The first TALOS Humanoid Robot [9]

The TALOS robot, fig. 1.2a, has been developed by the Spanish company PAL Robotics, based on the requirements provided by the Gepetto team from the LAAS-CNRS. It is a 1.75 m humanoid robot weighing around 100 kg. Several laboratories around the world have acquired this robot to perform research on walking, manipulation, human interaction, etc.

**TIAGo [10]**

This is a two-wheeled mobile robot dedicated to research, also manufactured by PAL Robotics, and displayed fig. 1.2b. It is based on a logistics robot, which provides autonomous navigation among obstacles. On that base, a torso equipped with a head and one or two arms has been added. Research done on this robot include human interaction, manipulation, like its bipedal brother, without the hassle of moving on two legs.

**Middle Sized Drilling Robot (MSDR)**

Finally, this robot is an industrial manipulator arm manufactured by the Japanese company Fanuc, with additional tooling specified by Airbus robotics team. The arm itself has 6 axes, and can reach out up to 2m with a 60 kg payload. Its purpose is the autonomous drilling of fasteners' holes in airframe to assemble a fuselage. To handle this task, it is equipped with an end-effector comprised of a drill and multiple sensors to ensure the high quality of the holes. The in-progress prototype can be seen fig. 1.2c with some of its equipment.

### 1.3 Objectives and contributions of the thesis

This thesis is a CIFRE, realised within a partnership between the LAAS-CNRS laboratory and the Airbus Operations company. Its objectives are therefore oriented towards an industrial target.

The first objective of the thesis is to conceptualise links between the automated planning of a robot's manipulation movements and the executive control of those motions. With those links established, the second contribution should be to automate the formulation of the controllers themselves, in a way similar to the planning. For the execution, the accent should be put on using all perceptive capabilities of the robots, including vision and force sensing. This is aimed at enabling the possibility for the robot to react to slight changes in its tasks or the environment: obstacles, tasks already done, etc. By reacting, we mean to be able to still achieve the goals without having to re-plan its movements, which is a slow process.

Those objectives should be done while keeping in mind the final application domain which is aeronautic construction. Thus, the targeted tasks include drilling, deburring, or riveting, which are all requiring a high precision. In the same way as the joint laboratory it is part of, this thesis has to produce modular software that can be adapted on the three different robots presented in the previous section.

Because of the large scope of the work, which encompasses planning, control and sensing, this thesis is heavily based on the pre-existing software presented previously. This makes the overall work of this thesis a challenging integrative task that enables communications between those components.

## 1.4 Organisation of this manuscript

Chapter 2 is a presentation of the state of the art in the various domains of robotics that this thesis dealt with. The first sections present motion planning and task planning, beginning with a short history of those fields, followed by manipulation planning, which is a combination of those two aspects of planning. The last two sections are overviews of robotics whole-body control and visual servo control.

Chapter 3 is the introduction of the Agimus framework. This is a software formulated and developed during this thesis, which incorporate my main contributions. It builds upon both the motion planner and the hierarchical control stack of the Gepetto's team to allow the automated formulation of both manipulation trajectories and the real-time control schemes to ensure their execution on robots. For the work of this chapter, the framework was demonstrated on the TALOS humanoid robot.

Chapter 4 deals with the calibration's issues of our humanoid robot. Those shortcomings are discussed, alongside the hypothesis of their origin. I will explain the two solutions that we tried to solve the problem, without success, then quickly introduce the work of a colleague who finally designed a process to quickly and precisely calibrate such a complex robot.

Chapter 5 describes an expansion of the Agimus framework on the control side, with the addition of visual servoing. Indeed, even with a better calibration of the robot, the first formulation of Agimus, which computes all trajectories before the start of the manipulation, is not able to deal with slight changes of the environment during execution. The chapter deals with the process of adding visual servo control in the framework and presents two experiments to validate the system. The first experiment is a more robust version of the one from chapter 3, on the humanoid robot TALOS. The second experiment lean towards a more industrial side and introduce the TIAGo robot, tasked with deburring of holes drilled in an aircraft part.

Chapter 6 is focused on an industrial robot designed for hole drilling in aircraft's panels, the Middle Sized Drilling Robot (MSDR). The work presented in this chapter is simpler and more adapted to the safeness and robustness goals of an industrial environment. There are however many parallels between the research presented in the previous chapters and the long-term targets of the industry.

Finally, chapter 7 is a general conclusion about the work completed during my thesis. It will be followed by perspectives and opening about future endeavours, and about how all this can be adapted from the research to the industrial world.

## 1.5 Publications

The work achieved during my thesis was presented in two international conferences:

- **Alexis Nicolin**, Joseph Mirabel, Sébastien Boria, Olivier Stasse, and Florent Lamiroux. “Agimus: A new framework for mapping manipulation motion plans to sequences of hierarchical task-based controllers”. In: *2020 IEEE/SICE International Symposium on System Integration (SII)*. IEEE. 2020, pp. 1022–1027
- Joseph Mirabel, Florent Lamiroux, Thuc Long Ha, **Alexis Nicolin**, Olivier Stasse, and Sébastien Boria. “Performing manufacturing tasks with a mobile manipulator: from motion planning to sensor based motion control”. In: *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*. IEEE. 2021, pp. 159–164





# Chapter 2

## State of the art

### Contents

---

<b>2.1</b>	<b>Motion planning</b>	<b>10</b>
2.1.1	Notions	10
2.1.1.1	How to represent a planning problem?	10
2.1.1.2	Configuration space	11
2.1.2	Motion planning	11
2.1.2.1	Deterministic planning	12
2.1.2.2	Sampling-based planning	12
2.1.2.3	Optimisation-based planning	15
2.1.3	Towards variants of the classical motion planning problem	16
<b>2.2</b>	<b>Constrained motion planning</b>	<b>17</b>
2.2.1	Point-wise constraints	18
2.2.2	Constraints along a path	19
2.2.3	Humanoid motion planning	20
<b>2.3</b>	<b>Task planning</b>	<b>22</b>
2.3.1	Definition of a task planning problem	22
2.3.1.1	Planning Domain Definition Language (PDDL) and its successors	23
2.3.2	Solving a task planning problem	24
2.3.2.1	Subsequent approaches	25
<b>2.4</b>	<b>Manipulation planning</b>	<b>27</b>
2.4.1	Different flavours of manipulation	27
2.4.2	Manipulation planning problem	28
<b>2.5</b>	<b>Control laws for robots</b>	<b>30</b>
<b>2.6</b>	<b>Visual servoing</b>	<b>32</b>

---

## 2.1 Motion planning

Motion planning is the process of finding a path, devoid of collision, between a starting robot configuration and a goal. The first classical version of this problem is called **the piano mover's problem**, illustrated in fig. 2.1. The 3D models of an environment and a rigid object, in this case respectively a house and a piano, are given as inputs, along with the starting and goal poses of the piano. The objective of the problem is to find a path to move the instrument while avoiding contact with the walls of the house, or prove that such a path does not exist. This problem was discussed at length by Schwartz and Sharir in the 1980s [11, 12, 13, 14, 15], with extensions to articulated bodies.

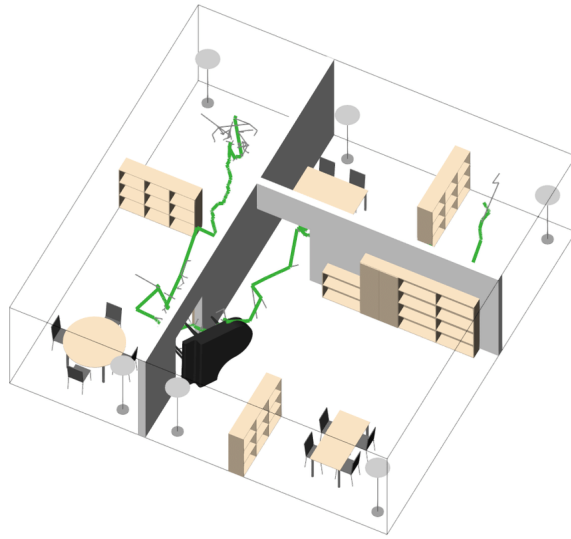


Figure 2.1: A representation of **the piano mover's problem**, from [16].

Over the years, those subjects have migrated from the pure research field towards industrial applications surrounding assembly, disassembly [17, 18, 19] and part moving [20]. However, research is still ongoing for multi-body robots, which we are using in this thesis. In this section, we will introduce the fundamental notion of configuration space, then we will go over various algorithms of motion planning, separated into three classes, each with their strengths and limitations.

### 2.1.1 Notions

#### 2.1.1.1 How to represent a planning problem?

##### Links and joints

The most common representation of a robot is as a set of rigid **links** articulated by moving **joints**. The two elementary joint types are the prismatic joint, which allows the linear sliding of a link relative to another, and the revolute joint, which constrains the movement between two links to a pure rotation. A combination of those main joint types, with or without links between them, allows the creation of more complex joints, such as ball joints or screw joints.

Using links and joints, one can describe the schematic structure of a robot. The main format, based on XML, used in Robot Operating System (ROS) [8] for those descriptions is called Unified Robot Description Format (URDF) [21]. In such a file, a robot is described from a root link, and then links and joints are gradually added in a tree-like structure to describe the whole system. Useful information, such as the characteristics of the joints (range, speed, friction, etc.), or those of the links (collision model, inertia, etc.) are embedded inside the description. Other data, like 3D models or colours, can also be added for the needs of computer vision simulations.

### 2.1.1.2 Configuration space

#### Definition 1 — A configuration

It is a set of values — one for each Degree of Freedom (DoF) — necessary and sufficient to define the position of every point of the system, composed of the robot and, if applicable, movable objects. The Configuration Space ( $\mathcal{CS}$ ) contains all the configurations achievable by the system. This concept comes from Lagrangian mechanics and was introduced into the robotic field by Lozano-Pérez and Wesley [22, 23].

The  $\mathcal{CS}$  for multiple robots and objects is the Cartesian product of their respective  $\mathcal{CS}$ . This space is composed of two subsets:

- $\mathcal{CS}_{\text{obs}}$ : the subset of  $\mathcal{CS}$  where at least one body of the system collides with the environment or another body of the system.
- $\mathcal{CS}_{\text{free}}$ : the subset of  $\mathcal{CS}$  devoid of collision;  $\mathcal{CS}_{\text{free}} = \mathcal{CS} \setminus \mathcal{CS}_{\text{obs}}$ . This is the subset where we plan the robot's trajectories.

### 2.1.2 Motion planning

Mathematically, the motion planning problem can be formulated as such:

#### Definition 2 — Motion planning problem

Given an initial configuration  $q_{\text{init}}$  and a set  $q_{\text{goals}}$  of suitable objective configurations fulfilling some desired properties, the problem is to find a trajectory  $p(t)$ , continuous, and ideally sufficiently differentiable (to avoid jerk), such that:

- $\forall t \in [0; 1], p(t) \in \mathcal{CS}_{\text{free}},$
- $p(0) = q_{\text{init}},$
- $p(1) \in q_{\text{goals}}.$

Extensive studies of the problem have been performed since the 1980s, and while major algorithms date back to the end of the 1990s and the beginning of the 2000s, it is still an active field of research. The most important results are compiled into the books written by Latombe [24], Choset, Lynch, Hutchinson, Kantor, and Burgard [25] and LaValle [26]. There are multiple methods to undertake this problem, which can be separated into three broad classes, detailed in the coming sections.

### 2.1.2.1 Deterministic planning

The main property of those approaches is that they always provide the same path as an answer for a given problem. Or, in the case where no path can join the initial and goal poses while avoiding obstacles, they can clearly answer that it cannot be planned. Diverse methods exist to transform an explicit description of  $\mathcal{CS}_{\text{obs}}$  into an abstracted representation of the connectivity of  $\mathcal{CS}_{\text{free}}$ , in the form of a graph or a roadmap. We can cite techniques such as cellular decomposition [11], Voronoi diagrams [27], Canny’s algorithm [28], etc. Other methods rely on a simpler discretisation of  $\mathcal{CS}_{\text{free}}$ .

Later developments allowed the planning of trajectories in a time-varying environment, where some obstacles are mobile. Kant and Zucker [29] use a two-steps planning to solve this problem: first, find a path to avoid static obstacles, then adjust the velocity along this path to avoid the mobile objects. At the end, the motion planning in itself is accomplished by the use of a graph path search algorithm such as Dijkstra’s shortest path algorithm [30], A\* algorithm [31], or their more recent variants. A second sort of approach is based on the use of potential fields to guide the movements of the robot [32, 33]. The use of potential fields also offers feedback to avoid mobile objects. Recent works by Zhang et al. [34] combine the previous methods, using a cell decomposition to approximate  $\mathcal{CS}_{\text{free}}$  and vector fields to provide a reactive and smooth motion planning.

However, deterministic approaches suffer from some limitations. The need for a complete representation of the connectivity of  $\mathcal{CS}_{\text{free}}$  implies a combinatorial explosion when the dimension of the problem increases. Yet, the increase in dimension comes directly from the addition of DoFs to the robot and sometimes the objects they manipulate. In the case of the humanoid robot used during the thesis, there are 38 DoFs including its free-floating base, which is intractable for any of the aforementioned algorithms in a reasonable time-frame. The potential fields methods are able to deal with the increasing dimension of the problem with a greater success, but they have a tendency to fall into local minima and be trapped there.

### 2.1.2.2 Sampling-based planning

In order to solve problems of higher dimension, a new algorithmic approach was necessary. To increase the performance, the whole  $\mathcal{CS}$  cannot be used any more. Instead, a random sampling of this space provides an incomplete graph that approximates the connectivity of  $\mathcal{CS}_{\text{free}}$  using collision-free paths.

The algorithms in this approach are usually separated into two categories:

- Single query algorithms. The typical goal being a single-shot use, they do not seek to construct a good representation of the  $\mathcal{CS}$ . Instead, they progress randomly from the initial configuration, towards the goal, by creating a tree of samples. In some cases, the reverse operation, from goal configurations towards the initial state, is performed at the same time to increase the chances of quickly finding a suitable path. The archetype of those algorithms is Rapidly-exploring Random Tree (RRT) [35], illustrated in fig. 2.2a.
- Multiple queries algorithms. The typical use is to provide multiple paths in a quasi-static environment. This allows to take some time to construct a good

enough approximation of  $\mathcal{CS}_{\text{free}}$ , in the form of a graph rather than a tree, to be able to provide answers for paths between any two configurations in  $\mathcal{CS}_{\text{free}}$ . Then, for each query from an initial towards a goal configuration, the algorithm only has to pass through this graph to find a suitable path, with little to no need for further exploration, thus leading to a fast response time. The most popular of those algorithms are of the family of Probabilistic Road-Maps (PRM) [36], illustrated fig. 2.2b. The main difference with RRT is that each new sample is connected to its closest neighbours. Subsequent research have improved the exploration capabilities of these algorithms to reduce the size of the road map, like Visibility-PRM (V-PRM) [37].

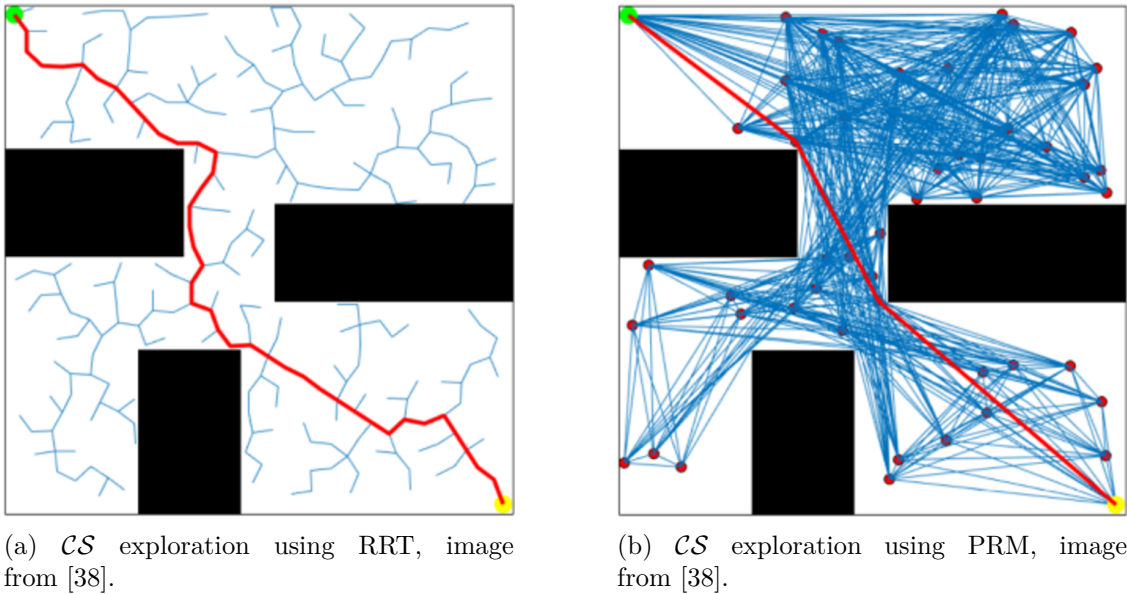


Figure 2.2: Comparison between Rapidly-exploring Random Tree and Probabilistic Road-Maps.

These algorithms are able to deal with larger problems than their deterministic counter-parts. However, they are not devoid of flaws:

- They are only probabilistically complete, meaning that if a solution exists, the probability of finding it converges to 1 as the number of iterations increases. However, in the case where no solution exists, they are not able to conclude and will run as long as they are allowed.
- Narrow passages are a problem for those algorithms, because the entrance and exit of such passages is not likely to be explored by the random expansion of the graph. Some techniques aim to reduce this problem [39, 40].
- For closed-loop systems, such as dual-arm robots, or even humanoids, the volume of the configuration space that is achievable is zero. Therefore, the probability of a random configuration being sampled in such sub-manifold is also zero. Constrained variants of the aforementioned algorithms exist to counter this deficiency [41].
- Due to the underlying randomness of the configuration chosen by the algorithms, the resulting paths can be very erratic. A post-processing path is therefore often needed before the plans can be played on real robots. Another

approach as been the creation of variants of the previously mentioned algorithms which are guaranteed to converge towards the globally optimal solution of the motion planning problem, for example Optimal PRM (PRM<sup>\*</sup>) and Optimal RRT (RRT<sup>\*</sup>) [42]. The former attempts to link the last sample with the closest neighbouring configurations. The latter mainly work by rewiring the tree of sampled configurations to reduce the overall cost of following the trajectory.

The most recent works, from the last 10 years, are more focused on practical improvements of the algorithms. In 2014, Gammell, Srinivasa, and Barfoot presented a new method: Informed RRT<sup>\*</sup> [43]. This algorithm improves RRT<sup>\*</sup> using the fact that only new random samples from a certain ellipsoid around the current path can improve the length of the path. This method retains the optimality and completeness of the base RRT<sup>\*</sup> algorithm, while outperforming it in rate of convergence and being less dependent on the dimension of the planning problem.

Fast Marching Tree (FMT<sup>\*</sup>) [44] is advantageous over RRT<sup>\*</sup> and PRM<sup>\*</sup>. In an obstacle-free space, it will provide the same optimal answer as PRM<sup>\*</sup>, but in the presence of obstacles it will gain an edge over it because it only lazily evaluates collisions. This causes sub-optimality against PRM<sup>\*</sup> in some cases, but the rarity of those occurrences gives a practical advantage to FMT<sup>\*</sup>. It is also simpler to tune than RRT<sup>\*</sup>, having only two parameters instead of four. The same year, the authors of Informed RRT<sup>\*</sup> published Batch Informed Trees (BIT<sup>\*</sup>) [45]. It works iteratively in batches. Configuration samples are randomly and uniformly distributed in the free space between the start and the goal of the system. An explicit tree of configurations is then grown from the starting configuration towards the goal, using only collision-free edges. The construction stops when a solution is found or when the tree cannot be expanded further. This concludes a batch.

The free space either stays the same if no solution has been found, or it is reduced into an ellipsoid of around the same size as the solution's length. The following batches use increasingly denser sampling strategies, which will yield more solutions, and asymptotically converge towards the optimal path between the start and the goal. The authors claim to find better solutions than all the previously mentioned algorithms, while being faster.

In [46], the authors provide boundaries for the convergence rate of PRM<sup>\*</sup>, and demonstrate that a good tuning can bring this algorithm towards a  $O(n)$  computational and space complexity for its asymptotic optimality. They extend their results to more recent sampling-based algorithm, like FMT<sup>\*</sup>. A compromise between RRT, which is fast to the detriment of the paths' quality, and RRT<sup>\*</sup>, slower but yielding better paths, is given in [47]. Finally, we can cite a recent axis of the motion planning research, towards learning and neural networks.

In [48], sampling distributions are learnt from demonstrations, and this information is used to bias the sampling of the otherwise classical algorithm. The results are the improvement of an order of magnitude of the success rate.

### 2.1.2.3 Optimisation-based planning

The motion planning problem can be written as a trajectory optimisation problem. Those approaches require an initial trajectory as input, but it can be very naive, or even in collision with the environment. In broad terms, an optimisation problem solver seeks the reduction of a cost given by the user. This cost can aggregate multiple properties of the problem:

- Collision avoidance, which is necessary for an achievable path. It is always formulated as a signed distance function. The distance corresponds to the shortest distance between two measured objects, generally the robot and a part of the environment. The measure is defined as positive when the objects are non-colliding, zero at the point of contact, and negative when the objects are inter-penetrating. The goal of the solver is then to maintain all those distances above a positive threshold which is the margin of security between the robot and each and every object.
- Length or energy. Those are optional, but often appreciated. The cost of the motion planning problem is penalised with the length of the path, or the quantity of energy needed to execute it.
- Other properties can be rewarded, such as the smoothness of the movement, or penalised, like the use of some joints, etc.

The main techniques are Covariant Hamiltonian Optimisation for Motion Planning (CHOMP) [49] and Stochastic Trajectory Optimisation for Motion Planning (STOMP) [50]. CHOMP reduces the user-defined cost using covariant gradient information. It is able to find collision-free trajectories starting with a naive, straight-line initial solution which might be in collision with the environment. STOMP has a very different approach, generating random trajectories around the current trajectory to explore the  $\mathcal{CS}$  in search of cheaper parts of the trajectory, which is then updated before the next iteration of the algorithm.

[51] presents a new algorithm that improves on the original one that was implemented in TrajOpt [52]. It has the same capabilities as CHOMP and uses a sequential optimisation routine to penalise the collisions. It also considers the continuous-time collision safety with an efficient formulation of the avoidance constraint. Another approach, Gaussian Process Motion Planner (GPMP) [53], represents continuous-time trajectories using sparse Gaussian processes. The planning algorithm itself relies on gradient-based optimisation exploiting this sparsity efficiently. The authors propose various extensions, such as GPMP2, which uses numerical optimisation, or iGPMP, an incremental version able to re-plan the motions if the problem evolves slightly. Those variants are much faster than the original algorithm while retaining its robustness. Optimisation-based motion planning has many advantages over the deterministic and random sampling approaches described previously.

By essence, it converges directly towards an optimum given by the equations of the cost, instead of randomly and asymptotically approaching it. It should therefore be faster than the random sampling based motion planning algorithms. However, in reality, the optimisation can fall into a local optimum that will trap further progress, preventing the algorithm to ever reach the global optimal path. Various



relaxation techniques, or hybrid approaches using random sampling combined with optimisation, are able to explore  $\mathcal{CS}_{\text{free}}$  more efficiently to ensure that the result of the planning will be the global optimal path.

But it comes as no surprise that such solutions are computationally costly compared to the purely optimisation-based approaches.

### 2.1.3 Towards variants of the classical motion planning problem

In the previous parts of this section, robots were supposed to be able to directly control their movements along all the dimensions of their  $\mathcal{CS}_{\text{free}}$ . They are called **holonomic** robots. But in reality, some robots are not holonomic, and they cannot access immediately all their degrees of freedom. For example, differential robots only have two wheels that can roll forwards or backwards, independently, so they have 2 DoFs directly available for the control, and cannot move sideways. But they move on a floor, which is a plan, so their pose is defined by three independent parameters:  $(x, y, \theta)$ . This implies that, in order to reach the whole  $\mathcal{CS}_{\text{free}}$ , relations between the controlled DoFs must be taken into account. Staying with our example of differential robots, for a pure rotation movement, one wheel must roll forwards while the second rolls backwards at the same speed.

To deal with this, we use what is called a **steering method**. It is a local planner that is able to guide the robot between two configurations, while respecting the dependency between its DoFs. A simple example for our differentiable robot would be the use of a three steps movements:

- A rotation to orient the robot towards the goal  $(x, y)$  coordinates,
- A straight line to reach those coordinates,
- A last rotation to control  $\theta$ , while  $(x, y)$  stay constant.

A more complex **steering method** would be able to follow curves to reduce the overall time necessary to execute the movement.

Another point of importance is that the whole  $\mathcal{CS}$  is not accessible for all robots. Indeed, even an industrial robot bolted to the floor can have its reach reduced for heavier loads. Moreover, robotic systems that contain closed loops, such as humanoids, parallel robots or cable-driven robots, are subject to internal constraints. Those kinds of robots can also have to take into account constraints like equilibrium or cable tension[54]. This reduces  $\mathcal{CS}_{\text{free}}$  to a sub-manifold of **feasible** configurations. More details about constrained systems and the motion planning approaches necessary to deal with them will be given in the next section.

Finally, up until here, a goal was given to the motion planner as a set of configurations in  $\mathcal{CS}$ . But often, there is an infinity of configurations that produce the desired effect. Therefore, instead of goal configurations, we prefer the concept of **task** to reach an objective. For example, a task could be defined as **grab the bottle on the table**. Many goal configurations can succeed in doing this task, so it is more convenient than specifying the exact goal pose. Moreover, if a pose was given but occupied by another object, the goal would become unfeasible. The planning of the tasks will be the subject of section 2.3.

## 2.2 Constrained motion planning

In the classical motion planning framework, the planner is expected to use the whole  $\mathcal{CS}_{\text{free}}$  to find a suitable path for the robotic system. While this approach is often successful for 6+ DoFs industrial manipulators performing tasks on their own, other systems or other tasks may imply some constraints on the planner.

- A closed-loop robotic system, when two or more grounded manipulators are interacting with the same object at the same time, is subject to drastic positioning constraints relative to each other. Indeed, for a single manipulator, the precision of the trajectory taken by the end-effector might be important, especially in the presence of numerous obstacles. But for multiple interacting manipulators, even small errors along the trajectory will affect the other robots and the feedback control could cause damage to the actuators. An illustration of closed-loop kinematic chain is provided fig. 2.3, where two robots assemble an IKEA chair together.
- Other robots may have physical constraints not related to obstacles. Cranes, manual or automated, are especially subject to weight constraints, limiting their range of operation, or wind speed, which can limit both their range and precision. Mobile robots, not bounded to the floor like industrial arms, have to respect equilibrium constraints.
- Humanoid robots, which are one of the main subjects of interest in this thesis, cumulate several of those constraints. They need to ensure the equilibrium of their pose at all time, and are nearly always subject to closed-loop constraints. Indeed, at least the feet are simultaneously on the ground, and multi-contact locomotion involves even more contacts, using both feet and hands.

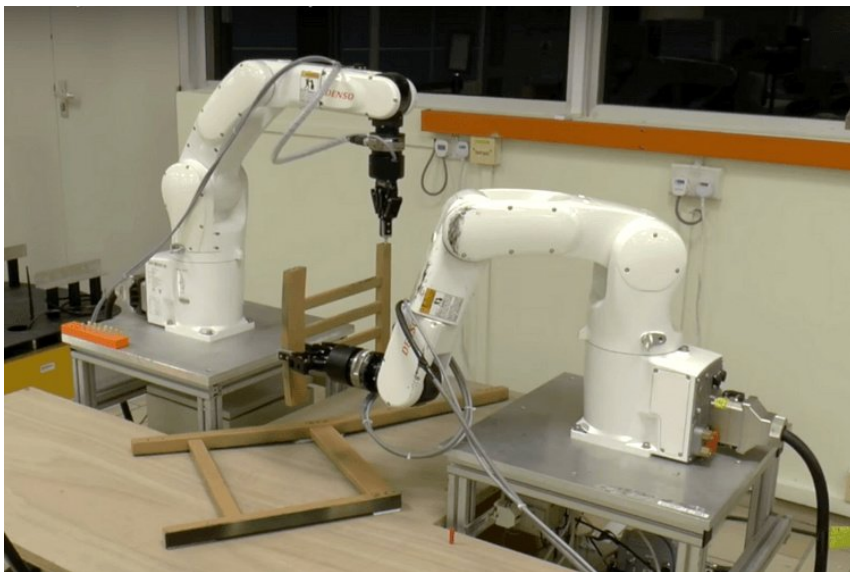


Figure 2.3: Two robots interact to assemble an IKEA chair, thus forming a closed-loop kinematic chain.

### 2.2.1 Point-wise constraints

In order for the robots to respect the constraints, the configuration space that is feasible is of the form  $\{q \in \mathcal{CS} \mid f(q) = 0\}$ , where  $f$  is the function modelling the constraints of the system. This mathematical expression of the feasible  $\mathcal{CS}$  clearly highlights the fact that it is a manifold that is only implicitly defined, and that for non-trivial constraints, the measure of this set is zero in the whole  $\mathcal{CS}$ . The consequences of those properties is the impossibility to directly use sampling methods in order to generate a plan. Indeed, the probability of sampling a feasible configuration is zero. To deal with this, a new problem, defined thereafter, must be solved when sampling configurations, to guarantee that the constraints are fulfilled.

**Definition 3 — Constraint Satisfaction Problem (CSP)**

Given a configuration  $q_0 \in \mathcal{CS}$  and a constraint function  $f : \mathcal{CS} \mapsto \mathbb{R}^n$ , the problem is to find a new  $q \in \mathcal{CS}$ , close to  $q_0$  and such that  $f(q) = 0$ , meaning the constraint is satisfied by this configuration.

In order to find the new  $q \in \mathcal{CS}$  that respects the constraint, we use a projector, defined here:

**Definition 4 — Projector**

Given a constraint function  $f : \mathcal{CS} \mapsto \mathbb{R}^n$ , we call **projector** a function  $p : \mathcal{CS} \mapsto \mathcal{CS}$  such that  $\forall q \in \mathcal{CS}, f(p(q)) = 0$ , *i.e.* the projected configuration respects the constraint, and  $\forall q \in \mathcal{CS}, p^2(q) = p(q)$ .

In the next paragraphs, we will discuss various projection methods proposed over the years.

As is nearly always the case when dealing with computer science, quantities are not continuous and sometimes subject to some computing imprecision. To account for those imperfections, a practical solution is to slightly relax the problem using  $\|f(q)\| \leq \epsilon$ .

Stilman [55] compares multiple algorithms regarding the CSP. Each of them keeps a sampling-based planner at its root, but introduces a mechanism to project the random sample  $q_s$  onto the manifold satisfying the constraints. Randomised Gradient Descent (RGD) is a naive yet successful approach, giving a nice baseline for the performance of cleverer algorithms. It consists in randomly sampling the neighbourhood of the configuration to reduce the distance to the constraints manifold. It has been used to plan motion for closed kinematic chain systems using PRM as the planner [56]. Because of its random nature, a lot of computational power is wasted sampling configurations farther from the feasible manifold.

The same authors proposed the Tangent-space Sampling (TS) method to reduce this waste by starting from  $q_{near}$ , the configuration closest to  $q_s$  that is already part of the motion planner exploring tree. Because it is part of the tree, it satisfies the constraints of the system. A small displacement from  $q_{near}$  to  $q_s$  is projected onto the space tangent to the manifold at the configuration  $q_{near}$ , by computing the constraint Jacobian at this point. Because the displacement is small and happens in the tangent space of the manifold, it is expected that the resulting configuration lies close to the manifold. A gradient descent may be applied from this point to reduce the error further if needed.

The Newton-Raphson (NR) approach proposed in [55] is similar to TS method, but the Jacobian of the constraints is computed at  $q_s$  to improve the convergence speed toward the feasible manifold. The drawback is the classical tendency of the pseudo-inverses of Jacobians to become unstable close to singularities. Of course, those methods require that the constraints be differentiable, to allow the computation of the Jacobians, but the majority of the constraints in robotics are geometric and are therefore explicitly differentiable. Recent studies are following similar paths, but improve the initial sampling strategy by creating approximations of the implicitly defined constraints manifold [57, 58, 59].

## 2.2.2 Constraints along a path

Multiple algorithms have been presented to project a randomly sampled configuration onto the manifold defined implicitly by the constraints. But those are only points, and not the continuous path constructed on the manifold that a real robotic system would be able to follow.

The simplest way to approach this continuity problem is the use of a linear interpolation in  $\mathcal{CS}$ . For each pair of consecutive feasible configurations, a direct line is computed between the two. This form a continuous polygon chain in  $\mathcal{CS}$ , or, if a higher-order interpolation or post-processing is applied, a smoother curve, which may be gentler for the real robot. This approach, very simple and computationally efficient, is however subject to drawbacks, both theoretical and practical.

Indeed, only the endpoints of the chain are guaranteed to be close enough to the manifold to respect the constraints, while the rest of the curve may cross into parts of  $\mathcal{CS}$  that are not feasible. A naive solution to the problem would be to discretise the path and project the point outside the constraint error range onto the manifold, then re-interpolate the path. A serious drawback for the general case is the absence of any information about constraints violation outside the intermediate points' immediate neighbourhood.

A more theoretically grounded solution, Recursive Hermite Projection (RHP), proposed by Hauser [60], is aimed at solving the problem of generating  $C^1$  paths satisfying a set of non-linear constraints. To reach the  $C^1$  continuity goal, the interpolation is a cubic Hermite spline, able to link points by matching both their values and their derivatives. The rest of the process is similar to the previously mentioned discretisation and interpolation method, but provides a guarantee on the maximum value of the constraint violation along the interpolation curve. For this guarantee to exist, the constraints' function must be Lipschitz continuous, with a constant  $M$  such that  $\forall(q_0, q_1) \in \mathcal{CS}^2, \|f(q_1) - f(q_0)\| \leq M\|q_0 - q_1\|$ . Then, the path is recursively split and the projection of the intermediate points is done, until the distance between two consecutive points is  $d \leq \frac{2\epsilon}{M}$ , where  $\epsilon$  is the maximum constraint violation that is acceptable.

This distance and the Lipschitz continuous property of the constraints' function ensure that the whole path is close enough to the manifold implicitly defined by the constraints. While this method provides a better constrained path than the naive approach, it also requires generating and projecting sheer numbers of intermediate points, which dramatically impact the motion planner performance.

To increase the efficiency of continuous path projection onto the manifold defined

by the constraints, Mirabel and Lamiroux [61, 62] proposed a method similar to RHP, but only  $C^0$  and using fewer intermediate points in the general case. They first demonstrate how to compute the radius of the neighbourhood of a point where the NR algorithm is continuous, and use that information to create intermediate points at the edge of this volume. Far from singularities in  $\mathcal{CS}$ , the neighbourhood of a constrained point where NR is continuous can be several orders of magnitude larger than the  $\epsilon$  distance on which RHP depends. The result is a  $C^0$  path between  $q_0$  and  $q_{goal}$ , where the intermediate points  $q_i$  are projected onto the manifold, and the sub paths between them are certified to be continuously projectable onto the same manifold during a subsequent post-processing of the trajectory.

Recently, Kingston, Moll, and Kavvaki [63] proposed a framework to decouple the choice of a motion planning algorithm and of the constraint adherence method used. Their approach preserves the probabilistic completeness and asymptotic optimality of modern motion planning algorithm, and allows the choice of different methods to plan on the implicit manifolds coming from the constraints. The results show that the decoupling between those two key parts of constrained planning can improve performance by enabling pairings that were difficult to test before. This work is available in Open Motion Planning Library (OMPL).

### 2.2.3 Humanoid motion planning

The generation of full body motions for a humanoid robot is especially difficult. They possess a large number of DoFs, often around 30, separated into 4 limbs, and thus have a larger part of their  $\mathcal{CS}$  unfeasible due to self-collisions. Their equilibrium needs to be maintained at all times. They can only move themselves through the environment by applying contact forces onto obstacles with their end-effectors, namely their feet and hands. Finally, their mobile nature implies various limitations on the actuators, to reduce their weight and power consumption, which can itself be capped by the on-board battery. For instance, an HRP-2 robot needs to use a handrail to help itself climb 15 cm stairs, because its legs' motors are not powerful enough for this kind of movement [64].

Because of those properties, the direct computing of all the variables necessary for a humanoid robot movement was rarely addressed until the last ten years. To make the problem manageable, the complexity had to be decreased. A simplified model can be proposed, such as a cart running on a table with a small foot [65] to represent the Zero Moment Point (ZMP) of a biped humanoid robot. Those methods based on simplified models and the ZMP often require an optimisation pass to improve the plan and especially its stability [66]. Other approaches use a decomposition into smaller, and easier, sub problems. With the presentation of the physics engine MuJoCo [67], it became possible to deal with the complete problem at once. This engine has been open-sourced by the company DeepMind at the end of 2021, and one can only hope that it will lead to a general improvement around planning and control of complex robots.

In [68], the authors show that a humanoid robot performing a dynamic walk, *i.e.* oscillating its ZMP between its two feet, is small-space controllable. It means that the robot is able to grossly follow an arbitrary path providing it respects the conditions needed to perform the dynamic walk. The problem of generating motions

to make a humanoid robot walk on a floor is then decoupled into two steps. First, a collision-free path, at constant Center of Mass (CoM) height, is computed, ensuring that the feet will always be able to lie flat on the ground. Second, the whole-body motion that follows this path is generated using the dynamic walking pattern. The path search was usually done using graph search algorithm such as A\* algorithm, with obstacles only at the foot level. This complements another approach that makes use of motion planning algorithms and approximation of the volume occupied by the feet during motion [69], which allow the avoidance of above-ground objects.

Recent work [70] increased the flexibility of the planning by removing the hypothesis of a flat floor, allowing the crossing of 3D clutter. Because the floor is not flat any more, the small space hypothesis does not hold and other execution schemes are used. A popular method is the pre-computation of small primitive motions to move the CoM along trajectories that are then concatenated to closely follow a path [71, 72].

Another approach reduces the dimensionality of the humanoid path planning problem by first looking only for a path of the CoM of the robot close enough to the environment to make contact with it, but far enough for the trunk of the robot not to be in collision [73]. A second step plans the contacts using the end-effectors of the robot, and creating the corresponding whole-body configurations in static equilibrium. The last step is to link those static configurations with a dynamic movement that ensures equilibrium, contact forces and collision avoidance. A simplified view of those steps is illustrated fig. 2.4. The whole pipeline is described in details in [74].

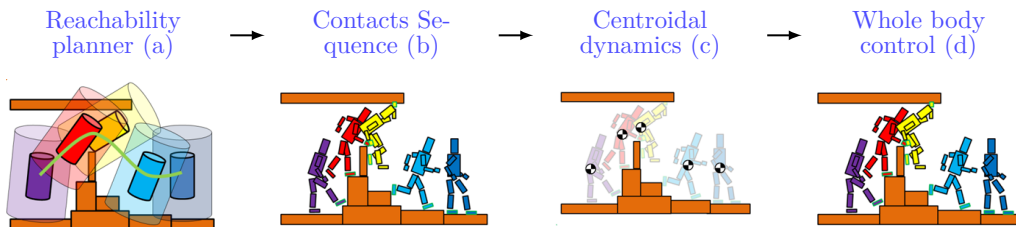


Figure 2.4: Humanoid motion planning using a decoupled approach.

Later work by the same authors improved the original publications with continuous formulation of the first step using Bezier curves [75], whose convexity allows for optimisation to increase the success rate of the later steps of the process. It also allows for dynamic transitions between the contacts, instead of ensuring quasi-static equilibrium at each step. In [76], they use the exact dynamics of the robot, instead of a reduced model, to compute the configurations. Offline learning of the relation between the CoM position and the contacts applied to the environments also allowed to find better states in the first steps of the pipeline, in turn improving the overall success of the method. Recent works keep a similar decomposition into the sequential computing of a centroidal path, then contact forces and finally whole-body configurations, with the introduction of mixed integers optimisation over a short horizon, and learning, to provide better centroidal paths from the beginning [77, 78].

## 2.3 Task planning

Automated planning and scheduling is a branch of artificial intelligence concerned with the planning of sequences of actions to reach a goal. In a known and static environment, where the models of the robots and the objects involved are well-described, it can be done offline beforehand. But in an unknown or dynamic world, online adaptation, or even full online planning and error-dealing strategies become necessary.

Early development emerged from the works around automated reasoning. Those programs aimed at reaching a goal given a starting hypothesis and logic rules allowing specific transitions between states. Seminal work include the logic theory machine [79], the General Problem Solver (GPS) [80]. However, they suffered from combinatorial explosion, even on simple examples like the Tower of Hanoi problem. Stanford Research Institute Problem Solver (STRIPS) [81] was an early automated planner more oriented toward real-world task execution. Its language was the basis for almost all the existing action languages, specifying state transition systems.

### 2.3.1 Definition of a task planning problem

#### Definition 5 — A state

It is a conjunction of logical atoms, or fluents, that are true. All non-mentioned fluents are assumed to be false.

*e.g.*  $At(robot, kitchen) \wedge Hold(robot, knife)$  is the state when the robot is in the kitchen, holding a knife.

#### Definition 6 — An action

It is a transition function between two states. Mathematically, it is defined as a triple  $(pre, add, del)$  which describes the realisation of an action in the system:

- $pre$  is the conjunction of pre-conditions, *i.e.* the set of functions dependent on the problem's fluents that must be true for the action to be executable.
- $add$  is the set of fluents that the action will set to **true** when executed.
- $del$  is the set of fluents that the action will set to **false** when executed.

*e.g.*  $GoFromRoom1ToKitchen = (At(room1), At(kitchen), At(room1))$  is the action to go from room1 to the kitchen. It is executable only if the robot is present in the room1 at the time at the start of the action. The new state reached by executing an action will then be defined as such:

$$S_{new} = result(S_{old}, (pre, add, del)) = \begin{cases} (S_{old} \setminus del) \cup add & \text{if } pre \in S_{old}, \\ = S_{old} & \text{otherwise.} \end{cases}$$

#### Definition 7 — STRIPS

A STRIPS instance is a quadruple  $(P, O, I, G)$  with the following components:

- $P$  is the set of fluents (or primitives) that defines the system,
- $O$  is the set of actions doable in this world,

- $I$  is the initial state of the system, *i.e.* the conjunction of true fluents, the others being assumed as false,
- $G$  is the goal state, which specifies which fluents must be true and which must be false. Some fluents can be in either state and will not be mentioned in those two sets.

### 2.3.1.1 PDDL and its successors

PDDL [82] is a tentative of standardisation of the various artificial intelligence planning languages that existed at the end of the 1990s. It was elaborated mainly to allow the creation of the International Planning Competition (IPC) [83]. This language enables the comparability of the different teams' propositions, and enhances the re-usability of their work. It is separated into two parts. The first part describes the domain of the planning, or in other terms, the world. It contains the definitions of objects that will be present in each problem, the possible actions executable in this world, and their effects. The second part describes the planning problem which has to be solved using this input. It contains the initial conditions of the world, the goal states and possibly other objects and actions specific to this instance.

Subsequent developments for each new competition introduced notions to express other, often more complex, planning problems. Examples of those additions are numeric fluents, long-term actions, and timed initial literals [84]:

- **Numeric fluents**, as their name indicates, can have a range of numerical values, and enables the modelling of non-binary resources such as energy level or weight.
- **Long-term actions** have effects that take place not only at the time of execution of the action, but also afterwards. Opening a tap could be modelled as a long-term action, because it is a one-time action that has a continuous effect: adding water into a bucket at each time-step.
- **Timed initial literals** can model events occurring at a given time independently of the execution of the plan.

In parallel to the effort to update the main language to model more problems with an increased precision, variants and extensions were developed to cater to specific needs. We can cite:

- **PPDDL**, the probabilistic extension, which allows uncertainty in the transitions,
- **NDDL**, from the **NASA**, with its accent on activities' duration, a critical aspect for space missions,
- **MA-PDDL**, adding the capability to plan with multiple agents solving the same problem, or different aspects of it, at the same time.

All those languages enabled researchers and engineers to express complex task planning problems that needed automated solving. The next step is the resolution of those problems, *i.e.* finding the sequence of actions that leads from the initial state of the world to a state which respects the specifications of the goals. In the



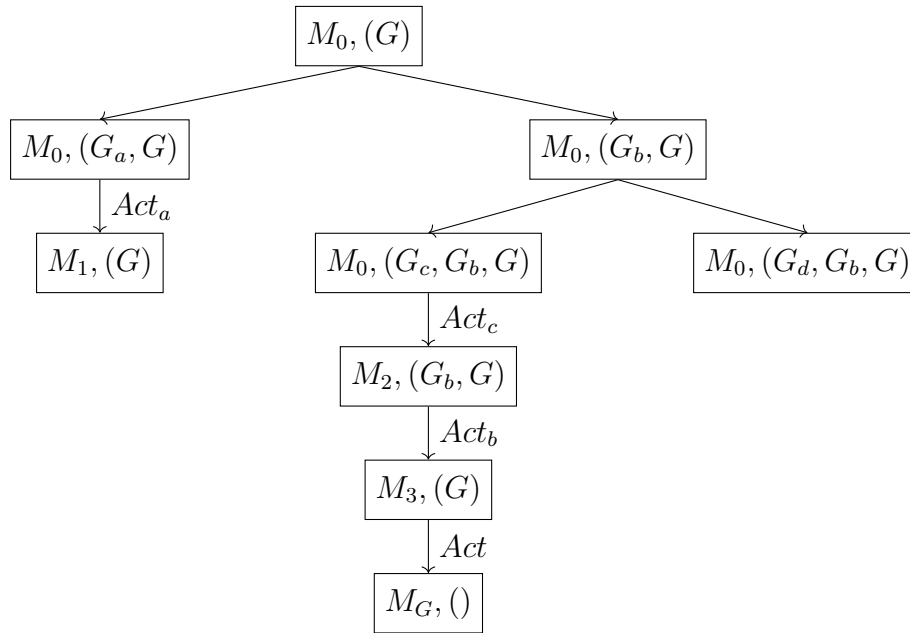


Figure 2.5: Example of a STRIPS tree.

following paragraphs, we will give some details about the methods used over the years to tackle those problems.

### 2.3.2 Solving a task planning problem

The original method proposed by the authors of STRIPS is based on the strategy used in GPS. The main principle is to extract the differences between the fluents of the current state and a goal state, and to identify the actions that are the most relevant to reduce those differences. A search tree is generated to solve the problem, where each node is a pair comprised of the current state of the world and a list of goals. An example of such a tree is given on fig. 2.5.

At the root, we have the initial state of the world, and the goal of the task planning problem. At each node, an automated theorem proving algorithm checks if the current state of the world respects the pre-conditions necessary to reach the first goal of the list in one action.

- If it can, the action  $Act_i$  is applied, and a child node is appended to the current node. It contains the new state of the world, obtained after the execution of the action, and the list of the remaining goals.
- If no action can reach the first goal, the theorem proving algorithm is able to provide the differences between the current state of the world and the first goal. Using those differences, a list of relevant actions, which could bring the state of the world closer to the first goal, is established. For each of those actions, a new child node is created, comprised of the current state of the world (because no action were taken), and the list of goals, extended with a new goal that represents the pre-conditions of the action.

When the new nodes have been generated, the system immediately checks if an action can be executed to reduce the list of goals, for each of them. After this

reduction of the lists has ended, if the problem is not solved, a heuristic mechanism has to select one of the nodes to extend the tree. The choice of this heuristic is not trivial for a good solving performance and multiple formulas have been proposed to enhance the STRIPS original method, depending on the precise nature of the task planning problem at hand.

### 2.3.2.1 Subsequent approaches

In the 1990s, new approaches emerged, increasing the efficiency of task planning systems.

#### GraphPlan

Proposed by Blum and Furst [85], GraphPlan uses a planning technique based on planning graphs. Instead of a tree, the method consists in the building of an alternating levels forward graph. Level 0 contains the values of the fluents in the initial state of the world. Level 1 contains all the actions which pre-conditions are fulfilled at the level below, including a **do nothing** action. Some may be mutually exclusive, because their pre-conditions or effects are not compatible, *e.g.* **do nothing** and **eat cake**, and a list of those exclusivities is kept alongside the graph. Level 2 contains all the possible values of the fluents if the action at the level above are executed. Once again, mutual exclusivities can exist, *e.g.* **cake eaten** and **cake not eaten**, and a list is kept.

The graph construction progresses by repeating the alternating process of checking the actions pre-conditions and applying those actions. The existence of a **do nothing** action ensures that the number of possible actions and fluents is never decreasing. Moreover, this reduces the number of mutual exclusivities. After some time, an odd level may contain all the fluent values of the goal, without mutual exclusivity between them. A solution could then be searched by going through the graph. At the end, the graph will level off, *i.e.*  $level_{i+1} = level_i$ , which means that no action can create other fluents' values or reduce the exclusivity's lists. If the goal's fluents values are not present, or if there are mutual exclusivities between some of them, the task planning problem is not doable.

#### Satisfiability

A second approach is the translation of the task planning problem into a Propositional Satisfiability Problem (SAT) [86]. A SAT problem consists in determining if there exists a set of boolean values that satisfies a formula, and returns such a set. Because such a formula is time and order independent, the authors had to use a trick to formulate a task planning problem as a SAT problem. The idea is to encode all the possible plans of a given length into boolean propositions with explicit time of execution. For example,  $GoFromTo(A, B, 3)$  is the proposition encoding that the robot will go from room A to room B at time 3. This example proposition is only executable in conjunction with the presence of the robot in room A at that time, which in turn is only true if the robot has gone to the room at a previous time, etc.

When the task planning problem has been encoded as a SAT problem, a propositional satisfiability solver is used to determine if the goal is reachable, and if so, how. Because the maximum size of a plan is fixed during encoding, the authors suggest

a binary search to determine the smallest plan reaching the goal. The authors' assumptions were that SAT solver's research would advance much faster than research in planning systems. Later works by the same authors [87] unified this scheme with the previously described planning graph concept, to reduce the number of propositions to check and limit the trial and error phase around the minimal length of the solution plan.

### Heuristic-search

This approach introduced with Heuristic Search Planner [88] uses a heuristic function to approximate the distance between the current state and the goal. Using this heuristic as a guide, the planner moves forward inside the state space. Fast-Forward [89] proved that this approach was very competitive, when in 2000 it outperformed all the other algorithms during the AIPS-2000 planning competition. Those approaches use a relaxed form of the task planning problem to enhance the performance of the solver.

Fast-Forward's heuristic is based on the same idea as GraphPlan, only considering the *add* lists of actions and not their *del* lists. It also uses a method called **hill-climbing** where the algorithm tries to improve its results by only going forward, guided by its heuristic, and appending actions to its plan. Backtracking on its choices is excluded, and the only way for the algorithm to undo an action is by applying, if it exists, an action having the inverse effect. As we have seen, this method has been successful, but can provide plans with a lot of **actions** followed by their **inverse actions**. It is also susceptible to the presence of dead-end states, from which no **inverse action** exist to backtrack on the current plan; *e.g.* throwing the key after closing the door.

Other works of interest include:

- Goal partitioning [90], where the main goal is divided into less complex sub-goals to enhance the performance of the planning algorithm by using a divide and conquer approach.
- Downward planning [91], which starts at the goal and reverses all actions to reach the initial state of the world. This can reduce the combinatorial explosion of the state space research.
- Causal graphs [92], aimed at improving the detection of possible dead-end before the execution of an algorithm like Fast-Forward.

Classical task planning is purely a symbolic problem, which either assumes that actions are always feasible, or assigns a percentage of failure to each of them. It does not take into account the reality of the underlying physical system that will execute the sequence of actions that are planned. Therefore, it is not directly usable for the guidance of robotic systems. We will present in the next section the domain of manipulation planning, which combines the motion planning seen in section 2.1 with the higher level task planning of this section.

## 2.4 Manipulation planning

In the first section, we had an overview of motion planning, which enables a robot to move from a configuration to another while avoiding obstacles. The second section was dealing with task planning, a branch of artificial intelligence aiming to find sequences of actions to reach a goal. In this section, the combination of motion planning and task planning will be presented. It is called manipulation planning, and deals with robotic systems interacting with their environment to reach a goal state.

The problem of manipulation planning has been researched for around 40 years, often by the same authors that dealt with pure motion planning [93]. Among the first authors tackling the subject, we can cite the works of Wilfong [94] and Alami, Simeon, and Laumond [95]. They originally considered the simpler problem of unarticulated robots and objects translating in their environment. Over the years, the subject gained in complexity, especially with the expansion of the humanoid research and the will to include more autonomous robots among human workers [96].

### 2.4.1 Different flavours of manipulation

The field of manipulation planning research is often times divided into multiple categories, each with its challenges:

- Navigation Among Movable Obstacles (NAMO), where a robot may need to move obstacles along its path to reach the goal configuration [97]. Propositions from the end of the 2000s include probabilistically complete approaches, using explicit representations of the  $\mathcal{CS}$  [98, 99, 100]. Later additions allowed the presence of uncertainties in the environment to better model the challenges encountered by a real robot [101, 102]. After being mainly theorised on wheeled robot, this domain was extended to humanoid robots [103].
- Rearrangement planning, aiming at moving several objects from their initial configurations toward a goal. [104, 105, 106, 107]. This may use difficult re-grasp movements for object reorientation [108, 109]. Given the initial and goal poses of an object, the re-grasp planning component finds a sequence of robot postures and grasp configurations that reorients the object from the initial pose to the goal. This is illustrated in fig. 2.6 where a Baxter robot moves coloured cubes. Those moves may be necessary to reorient an electric drill to drill holes, or workpieces for assembly. Recent extensions include the use of learning to address the problem of rearrangement planning among obstacles without prehension [110]. Their method provides a doubling of performance compared to the state-of-the-art planner by combining a sampling-based motion planner and a reinforcement learning scheme. The robot learns how it can interact with the objects, and the planner provides a path along the actions to reach the goal state.
- Finally, the demand for automation of human tasks, which often necessitates two arms, has driven the research around multi-robot manipulation planning forward. A successful approach decomposes this problem into sub-problems containing less DoFs, that can be solved separately and executed sequentially [111]. Another approach uses the same kind of decomposition, but then,

the road-maps of the sub-problems are combined into an efficient graph to plan multi-arms motions [112]. This has led to the formulation of discrete-RRT [113], and its asymptotically optimal variant dRRT\* [114, 115].

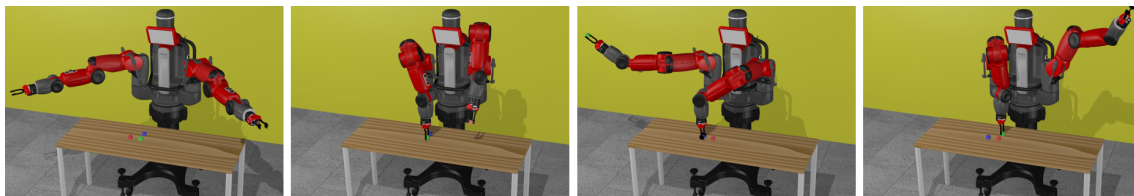


Figure 2.6: A Baxter robot performs rearrangement planning of cubes.

## 2.4.2 Manipulation planning problem

More generally, the manipulation planning problem takes place in a known environment. In that environment, a set of robots and a set of movable objects are placed at an initial configuration. A set of rules or constraints define goal's configurations for this instance of the problem. The goal of the manipulation planning problem is to find a sequence of actions, each corresponding to a collision-free geometric path, that starts at the initial configuration and reaches a goal.

Unlike in the case of motion planning, the constraints applied to the robots and objects are not necessarily the same during the whole problem duration. Indeed, when an object is manipulated by a robot, it moves alongside the end-effector holding it, and therefore, its position is constrained by the configuration of the robot. Conversely, an object not held by a robot should be at rest, and the robot may have other constraints in order to avoid moving this object. Thus, the manipulation planning problem has a hybrid structure, with discrete states, defined by the set of activated constraints, which are themselves continuous. Those states are linked by manipulation paths, and that creates a graph structure [116]. The manipulation paths can be separated into two types of manipulation motions: **transit** paths, where the robot moves and the objects are still, and **transfer** paths, where objects are manipulated by the robot [95].

In the previous section, we discussed task planning, and concluded that it does not take into account the reality of the physical system that may be controlled. A motion planner, on the contrary, deals with the physical system but is not able to find sequences of actions to reach a goal efficiently. But combining a task planner with a motion planner is not simple, because the former needs information that the latter is not able to give.

Indeed, in order to plan a sequence of actions, a task planner needs to know if the said actions are feasible or not, depending on the computed state of the world. However, we have seen that the most successful motion planner at the moment are sampling-based motion planner, that are only probabilistically complete and therefore cannot determine if a motion is not feasible. In order to plan manipulation motions, one needs to address this problem and provide geometric information to the task planner while keeping the computation time at a reasonable level.

Srivastava et al. [117] proposed a framework that uses off-the-shelf task and motion planners with little to no modification. The only requirement is that the motion planner should be able to either return a successful plan, and return a failure to do so to the task planner after some time. The basic principle of the system is simple, with the task planner calling the motion planner for each task in order to check if they are physically feasible or not, and creating a sequence of actions from this information.

### Transit paths, transfer paths and the reduction property

When dealing with robots and manipulable objects, the  $\mathcal{CS}$  is the Cartesian product of the respective  $\mathcal{CS}$  of each robot and of each object. This can easily amount to a space with a dimension of several tens, or even hundreds, and is computationally prohibitive to sample efficiently. However, we can recall some important notions from [118]. First, only two sub-manifolds are of interest in the whole  $\mathcal{CS}_{\text{free}}$  for manipulation planning:

- $\mathcal{CP}$ , the sub-manifold of  $\mathcal{CS}_{\text{free}}$  where the objects are at a valid and stable placement, not grasped by the robot. This is the manifold where the **transit paths** lie. However, one may note that the reciprocal is false. A path where an object slides on a table is contained in  $\mathcal{CP}$ , but is not a **transit path**, because the object is moving. Each possible position of an object therefore defines a sub-manifold of  $\mathcal{CP}$ , giving it a foliated structure.
- $\mathcal{CG}$ , the sub-manifold of  $\mathcal{CS}_{\text{free}}$  where a robot is holding an object, which therefore moves alongside the end-effector. The **transfer paths** are in this manifold. In the same spirit as above, a foliation of  $\mathcal{CG}$  exists, induced by the relative position of the object relative to the end-effector.

Because the goal is not to throw objects or release them at unstable positions, but instead to manipulate them, all the movements of interest for our purpose are constrained to those manifolds. The intersection  $\mathcal{CP} \cap \mathcal{CG}$  is the space where a robot grabs or releases an object. Because of the foliations mentioned above, this is a space consisting of connected components. An important property, recalled below, was shown by Dacre-Wright, Laumond, and Alami [119].

**Theorem 1** (Reduction property). *Any path lying in  $\mathcal{CP} \cap \mathcal{CG}$  where the robot is not in collision with static obstacles can be transformed into a finite sequence of transit and transfer paths.*

Thanks to this, the manipulation problem can be reduced to the discovery of the connectivity of this space, via pure transit or transfer paths. This is useful, because task planners are dealing with symbolic variables while motion planners use continuous variables with a physical meaning. *e.g. take the spoon on the table* is symbolic, but depending on the exact position of the spoon, the motion planner could succeed or fail. However, using the reduction property, if one can prove that the whole tabletop is a single connected component of  $\mathcal{CP} \cap \mathcal{CG}$ , the symbolic variable is sufficient. This property is used by Cambon, Alami, and Gravot [120] in their work integrating a task and a motion planner for several robots and objects. It allows them to use continuous grasps and placements with ease.

A recent and successful development in the domain of manipulation planning is the algorithm FFrob, proposed by Garrett, Lozano-Perez, and Kaelbling [121]. As its name indicates, it has ties with the task planning algorithm Fast Forward, which it extends to handle geometrical information. They use an offline sampling of the robot configurations and useful objects poses to create a reachability graph that will serve as a heuristic for the Fast Forward algorithm (FF) task planner. In the case where no solution is found, a new sampling batch is created to update the graph and retry. While the offline sampling improves performance of this algorithm, especially when keeping the same graph over multiple instance of problems in the same environment, this algorithm may be difficult to tune. Indeed, one needs to select the initial number of samples for objects' poses, grasps, the iterations' limit of the RRT, etc..

Last years' approaches have taken a similar road of providing geometric and sometimes constraints information [122] directly to the task planner to guide it. There are also attempts to create heuristics using machine learning, with a classifier to quickly decide if a motion is feasible or not [123], or to predict the constraints that will be encountered during the solving of the manipulation problem, to choose an appropriate strategy [124]. Finally, a recent extension of the motion planning library MoveIt! [125] was proposed with the goal of using task planning in combination with motion planning.

During the last four sections, we have discussed the steps necessary to plan the movements of a robot in an environment in order to reach a goal configuration. The first section was dealing with pure motion planning, from its historical beginning at the end of the 1960s to the modern approach using sampling and machine learning in order to plan optimal motions. The second section was about adding constraints to those planner, in order to plan motions for more complex robots, like humanoids, or simply interact with the environment. The third section dealt with task planning, where the goal is to find a sequence of elementary actions to reach a desirable situation. It is one of the oldest problems in computer science and is still improved today because of its ubiquity in the modern world's functioning. Finally, the current section combined all of the above to plan manipulation motions for robotics systems. But now that we have plans, we need to ensure that the robot will be able to carry them out, and we will discuss the methods used to control a robot in the next section.

## 2.5 Control laws for robots

In the four previous sections, we presented and discussed methods to generate plans and paths to guide robotic systems in order to perform tasks. In an ideal world, without unforeseen forces impacting the robot, those paths could be followed blindly by the system to reach their goals. However, we do not live in this ideal world, and as we have seen, planning may use simplifications of the robot's model in order to compute the paths efficiently. A robot may also need to react on the fly to a perturbation or even to a new command, which may not be part of the original planned path. Because of all of this, open-loop control is often insufficient to guide a robot, and a form of feedback is necessary to robustly follow the planned paths.

For a complex system such as a robot, there exist two main levels for the control. The first one is specific at the level of the actuators, to ensure that they are able to follow the orders they are given. Those actuators are equipped with sensors, for example the angular sensors or torque sensors on a rotating joint, or current, temperature, distance sensors, depending on the kind of actuators. Using the values provided by the sensors, and a model of the dynamic of the actuator, one is able to formulate control laws to ensure a quick, robust and precise following of the orders sent by the higher level control system of the robot.

A good knowledge of the actuator is a necessary step to build a good model that in turn leads to good performance of the controlled actuators. A first method to acquire this knowledge is to build the actuators from the ground up, testing their reactions along the way and adapting the conception to reach the desired performance [126, 127, 128]. Others have to use an already provided actuator, and need to identify the dynamics of the system, by sending orders, getting information through the sensors, and modelling what is in-between. This work was for example done on a HRP-2 robot, in order to propose torque control at the joint level, despite the large and unknown frictions of the strain wave gears, and most importantly, the absence of a dedicated torque sensor [129]. On a more recent robot, TALOS, which is equipped with torque sensors on each of its joints, identification of the actuators was done to provide a control law that respects the mechanical limits of the system [130].

A second level of control deals with the multiplicity of the actioned joints, and the constraints the system is subject to. Indeed, even if each actuator follows its orders, something has to issue those commands to perform a sound motion. The simplest method to reach a position with a manipulator is to use inverse kinematics [131]. It consists in finding the position or angle of each joint of a robot that will result in the desired position of the end-effector. However, this has severe limitations when the robot has some redundancy in its actuation chain, with more DoFs than what is required to reach the goal, or when a joint is close to a singularity, in which case the commands given by the inverse kinematics scheme may diverge. In the 1980s, solutions were proposed to handle those singularities for robotic manipulators [132], and to deal with redundancy [133, 134]. In this last publication, Siciliano and Slotine begin to use the notion of control task, that will be defined mathematically in more detail by Samson, Espiau, and Le Borgne [7] and will be used during the rest of this manuscript.

A subsequent development in the control field is the simultaneous presence of multiple objectives for the control laws. For example, we saw in the previous section that humanoid robots have to maintain their equilibrium at all times while performing other actions. Two major approaches are used to include multiple objectives in the control scheme of a robot [129, 135]. The first one is to add a weight to each goal, and tune those weights in order to prioritise one task over another. This method is highly versatile for specific tasks but tuning can become quite a hard problem when one tries to generalise a controller for different tasks or environments.

A second approach is the use of a strict hierarchy of tasks. This method will be described in more details in chapter 3, where we use the Stack-of-Tasks (SoT) [6], a framework implementing this kind of control scheme, based on the Generalised Inverted Kinematics (GIK) proposed by Nakamura et al. [133]. The principle is to



try to solve a task of lower importance in the null space of the previously solved tasks, *i.e.* using the DoFs that do not affect those higher priority tasks.

Whatever the choice of control scheme, the mathematical basis often involves the solving of quadratic problems [136, 137, 138], sometimes with non-linear constraints [139, 140, 141]. The development of those solving methods and the rapid increase in embedded computational power in robots has allowed the control in real-time of full-sized humanoid robots [142], and the handling of difficult tasks such as pulling a fire hose [143]. Recent advances in the control field aim at reducing the dichotomy between weighted control and strict hierarchy by implementing a generalised projector able to deal with both strict and non-strict prioritisation of tasks [144].

During the last decade, computers have reached sufficient capabilities to be able to deal with model-based approaches in real-time. This form of control uses a model of the system to simulate its reactions to commands, using a physical simulation of the environment [67], in order to refine them before they are sent to the real robot. The model used for the control can be simplified to reduce the computational burden, for example by only using the centroidal dynamics and the feet poses of a humanoid robot to control its walk in real-time [145, 68, 146].

More recent works use a more complete model of the robots, and are able to deal with multiple contacts with optimisation methods such as Differential Dynamic Programming (DDP) [147, 148], Model Hierarchy Predictive Control (MHPC) [149] or mixed integer solving [78]. Another direction of research enabled by the increase of computation capabilities is that of the solutions based on learning. Some approaches are based on the use of demonstrations, by way of video clips or motion capture, of movements to be reproduced by a simulated actor [150, 151, 152]. Others are bypassing the demonstration step to directly use the simulations to generate more robust control laws for the locomotion of robots [153, 154]. Finally, some studies are going further by learning from scratch the movements of a hand in simulation, that are then transferred to a real robot in order to achieve complex tasks such as solving a Rubik's Cube [155].

In this section, we saw an overview of the huge domain of control schemes used in robotics, in order to provide commands to actuators in reaction to information transmitted by sensors. One aspect that is used in this thesis is more precisely the use of visual information to steer the robot during the execution of its tasks. This is called visual servoing and will be presented in the following section.

## 2.6 Visual servoing

Visual servoing, also called visual servo control, is a control method that uses data obtained from computer vision to guide the movements of a robotic system. A nice overview of the field and some well established algorithms of servo control are presented in tutorials from Chaumette and Hutchinson [157, 156, 158]. Because this thesis deals with the planning of visual servoing tasks and not the servo control itself, I remained a simple user of those existing algorithms, implemented in the software suite Visual Servoing Platform (ViSP) [3]. Thus, this section will be short and only presents the few bricks that were used by Agimus, so as not to write a poor

copy of those tutorials. Details about the mathematics behind the control schemes we used will be given in chapter 5.

The first step to perform visual servo control is of course to capture images using a camera. However, cameras are equipped with lenses, used to focus the image onto the photographic sensors, but that can also introduce slight distortions. In order to gather useful data from those images, one needs to calibrate the cameras used, to determine the parameters of the lenses and if needed, compensate for them. Multiple methods have been proposed over the years, most notably by Tsai [159] and later by Zhang [160] and Sturm and Ramalingam [161].

The second step is to detect the objects or part of the environment that we are interested in, and track their position relative to the robot. In order to do that, important visual features of objects are extracted from the image using detectors. Indeed, an image is basically a huge matrix composed of pixels and a reduction into a few important components makes the rest of the problem computationally tractable. The features of interest can be colours, corners, edges [162, 163, 164], etc. An illustration of the features is provided at the top left of fig. 2.7, where a Romeo robot is required to grasp a soda can using visual servoing.

The results of those detectors can be used directly in Image-Based Visual Servoing (IBVS), while a second method, called Position-Based Visual Servoing (PBVS) calls for an estimation of the position of the object from those features. Many algorithms exist for this estimation, depending on the features used, the computing power available, the desired precision of the results, etc. Most notable estimators based on a model of the object are the works of Fischler and Bolles [165] and later DeMenthon and Davis [166].

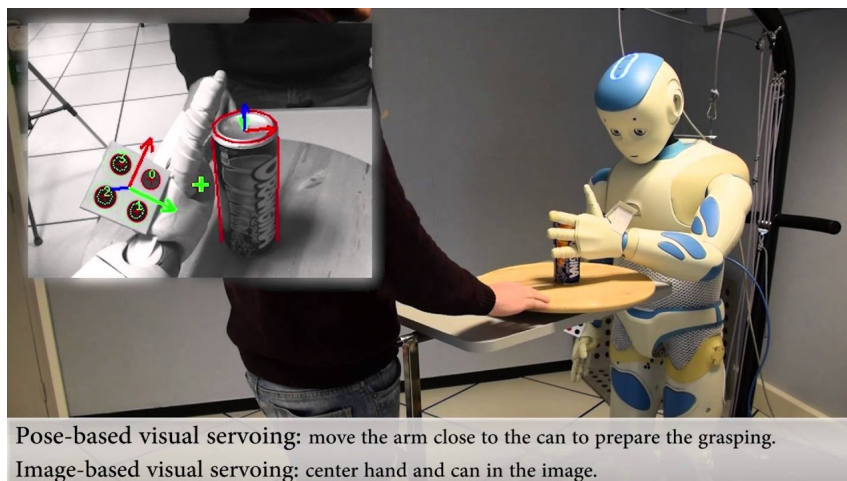


Figure 2.7: A Romeo robot grasp a soda can using visual servoing. Image from the ViSP team.

The extraction of the features is often the most computation-intensive part of a visual servo control scheme. Therefore, when the situation allows it, it may be advantageous to stick an artificial marker on the object, designed to be easily and quickly detected using computer vision [167]. This trick is widely used in the realm of augmented reality [168], and the markers have been improved over the years, for example to include embedded information [169] or to improve their robustness against occlusion [170, 171]. Dedicated algorithms allow the estimation of the po-

sition of those markers using the few features that they provide, often the 4 points that are their corners [172, 173, 174, 175, 176].

Features or positions of objects having been extracted from the captured images, a control scheme can make use of them to guide a robotic system toward a goal. Those goals can be diverse, and the precise control methods will therefore vary. One of the most common use of visual servoing is the tracking of an object [177], in order to reach it and sometimes grab it [178, 179, 180]. Another usage is the use of those visual information to guide the locomotion of a robot, be it with wheels [181, 182, 183] or on legs [184, 185]. Whatever the goal, one of the difficulty of those control methods are that a line of sight has to be maintained between the robot and the features used for the control [186]. Recent developments are using machine learning to enhance the possibilities offered by visual servoing. For example, the simulation of multiple points of view of the scene to improve the success rate of servoing [187], or the memory of past scenes to use them as a locomotion goal [188].

# Chapter 3

## Agimus

### Contents

---

<b>3.1</b>	<b>Introduction</b>	<b>37</b>
<b>3.2</b>	<b>Concepts</b>	<b>37</b>
3.2.1	Constraints Graph	38
3.2.2	Estimation of the initial state	41
3.2.3	Manipulation planning	44
3.2.4	Hierarchical control	46
3.2.4.1	Task-functions	46
3.2.4.2	Dealing with multiple tasks	47
3.2.4.3	Admittance control	49
3.2.5	Controllers' generation and execution	50
3.2.5.1	Generation of the real-time controllers	50
3.2.5.2	Finite State Machine to control the execution	52
3.2.6	Conclusion	52
<b>3.3</b>	<b>Software</b>	<b>53</b>
3.3.1	Visual Servoing Platform	53
3.3.2	Pinocchio	54
3.3.3	Humanoid Path Planner	55
3.3.4	The Dynamic Graph and the Stack-of-Tasks	56
3.3.5	Ros_control	57
3.3.6	The Agimus framework	58
3.3.7	Conclusion	62
<b>3.4</b>	<b>Experiments</b>	<b>62</b>
3.4.1	The TALOS humanoid robot	63
3.4.1.1	Position against other humanoid robots	64
3.4.2	Course of the experiment	66
3.4.3	Presentation of a typical run	69
3.4.4	Discussion	71

3.4.4.1	Failures' detection . . . . .	72
3.4.4.2	Limitations caused by the poor kinematic cali- bration of TALOS . . . . .	72
<b>3.5</b>	<b>Conclusion . . . . .</b>	<b>73</b>

---

## 3.1 Introduction

In this chapter is the first contribution of my thesis, in the form of a new framework to plan and execute manipulation tasks. The major interest of this new framework is the automation of the generation of real-time controllers, and the simultaneous building of a Finite State Machine (FSM) that orchestrates the execution of the plan. The controllers are derived from the specifications of the manipulation tasks. An introductory illustration of our framework capabilities is presented in fig. 3.1, where the robot manipulates an object using both its grippers.

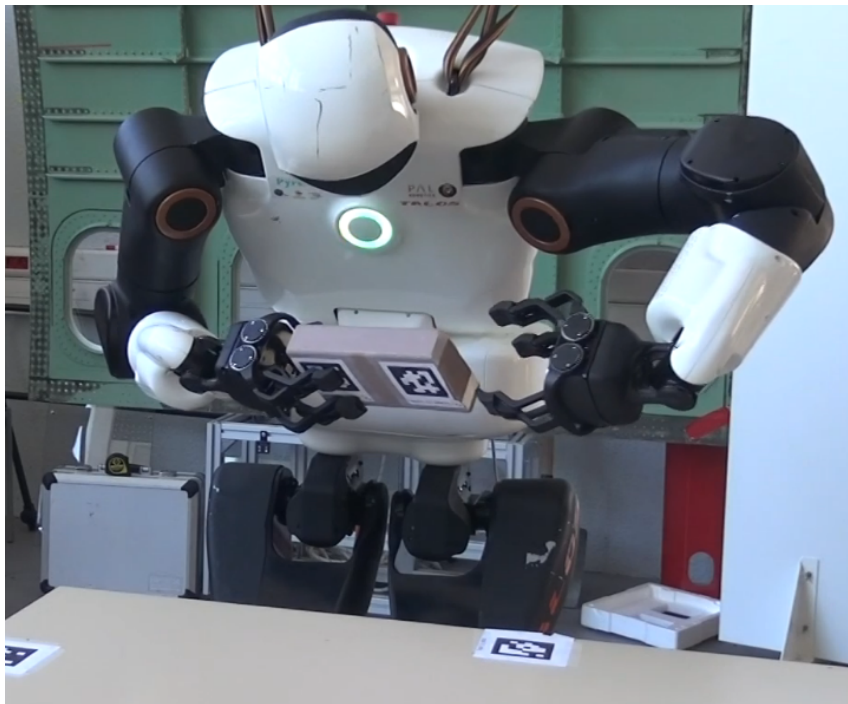


Figure 3.1: Example of a manipulation task: a TALOS humanoid robot is requested to turn a wooden plank upside-down and place it back on a table.

In the first section, the concepts underlying the whole system will be presented and explained. The second section deals with the software implementing those concepts, and specifically the ones we used during our development and the subsequent experiments that validated our ideas. The last part of this chapter is about the experiment itself, beginning with a description of the robot used to perform it, a human-sized TALOS humanoid robot. It is followed by the description of the experiment itself, where the robot had to grab a plank of wood on a table and place it back upside-down. This chapter is concluded by a discussion about the upsides and downsides of our system.

## 3.2 Concepts

This section presents the concepts and algorithms used in the framework *Agimus*. At the core of those is the graph of constraints, which is explained in details in the first subsection. Following this is a part about the estimation of the initial state of

the robot and its environment, and then a presentation of the manipulation planning algorithm. Next, the fourth section is the introduction to the hierarchical control schemes that ensure the real-time execution of the planned motions. Finally, the last part presents the main contributions of Agimus. First, the automated creation of the control stacks, based on the requirements extracted from the graph of constraints. Second, the Finite State Machine (FSM) generated to orchestrate the switching between those controllers, and more generally to order the overall progress of the demonstration.

### 3.2.1 Constraints Graph

The upcoming paragraphs deal with the graph of constraints, a representation of the manipulation rules originally presented in [189], extended in [5], and that is at the core of Agimus. To improve the clarity of this chapter, the concepts behind this representation will be recalled here, with the addition of concrete examples coming from the experiment described in the introduction, performed on the humanoid robot TALOS. Note that all of this can easily be adapted to a robot with only one arm, like a classical industrial manipulator, or conversely to a robot with more limbs.

#### Numerical constraint

Constraints were loosely introduced in the previous chapter, more specifically in section 2.2. Because in this chapter they are used extensively, a more mathematically sound definition of the constraints encountered in motion planning will be given.

Let  $\mathcal{CS} = \mathcal{CS}_{robot_1} \times \mathcal{CS}_{robot_2} \times \dots \times \mathcal{CS}_{object_1} \times \dots$  be the cartesian product of the Configuration Space ( $\mathcal{CS}$ ) of all the robots and objects involved.

A numerical constraint is a mapping:  $f : \mathcal{CS} \mapsto \mathbb{R}^m$ , where  $m \in \mathbb{N}$ .

We say that a configuration  $q \in \mathcal{CS}$  satisfies the equality (resp. inequality) constraint  $f$  if and only if  $f(q) = 0$ , (resp.  $f(q) \leq 0$ ).

A constraint can also be parametrised by a value  $g_0 \in \mathbb{R}^m$ . In this case, it is said to be satisfied, in the equality case (resp. inequality case), if and only if  $g(q) = g_0$  (resp.  $g(q) \leq g_0$ ).

#### Projection on numerical constraint

A projector onto a constraint is a mapping:  $proj_f : \mathcal{CS} \mapsto \mathcal{CS}$ . For a suitable input configuration  $q$ ,  $proj_f(q)$  satisfies the numerical constraint  $f$ . In section 2.2.1, we presented various methods to project a configuration onto manifolds defined by some numerical constraints. In Agimus, the projection is done using the Newton-Raphson algorithm.

#### Gripper, handle, and grasp

In the framework Agimus, for the manipulation planning step, a **gripper** is reduced to a virtual frame which represents the point that the robot will grasp when closing its gripper. On the other side, objects are described alongside one or several virtual frames, called **handles**, that represent reference frames (position + orientation) that a robot can grasp. Those frames are defined manually at the moment to ensure their feasibility. Thus, in order to pick up an object, the robot superposes a gripper

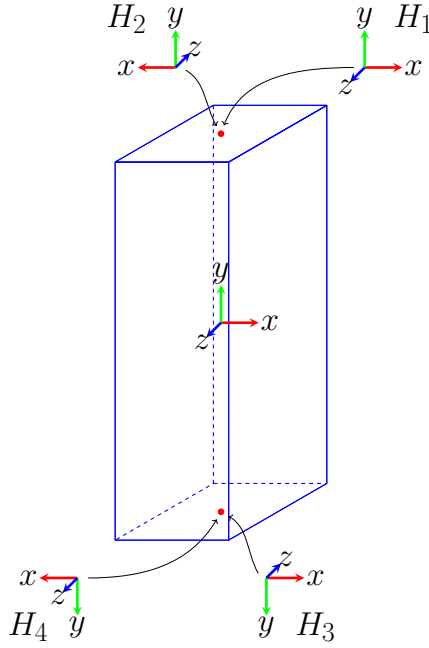


Figure 3.2: The four handles on the plank manipulated by TALOS.

frame onto a handle, and closes its gripper. This superposition of a gripper frame and an object handle is called a **grasp**.

In our experiment, presented in the introduction and that will be detailed later on, 4 handles denoted by  $(H_1, H_2, H_3, H_4)$  are attached to the object. The fig. 3.2 illustrates the different handles and their positions on the plank: two handles on the top, rotated  $180^\circ$  from each other, to allow grasping from the front and from behind ; two handles on the bottom of the object, to allow grasping even if the plank is placed bottom-up.

### State

A state is a subset of  $\mathcal{CS}$  defined by a set of numerical constraints. If a configuration satisfies all those constraints, it belongs to this state. The largest state we considered in our experiment, in the sense that it is defined by the smallest set of constraints, is the state were the robot is at equilibrium with its feet lying flat on the ground. We call it the **free state**, and the corresponding set of constraints is  $g_{free} = g_{CoM} \cup g_{FL} \cup g_{FR}$  (resp. the constraints of equilibrium, left foot position and right foot position). All other states in the problem add some constraints on top of these. In the experiment, the states over which we were planning the manipulation demonstration add constraints linked to the grasping of the plank:

- Position of the hands,
- Relative position between the hand and the plank,
- Torque control to grip the plank firmly.



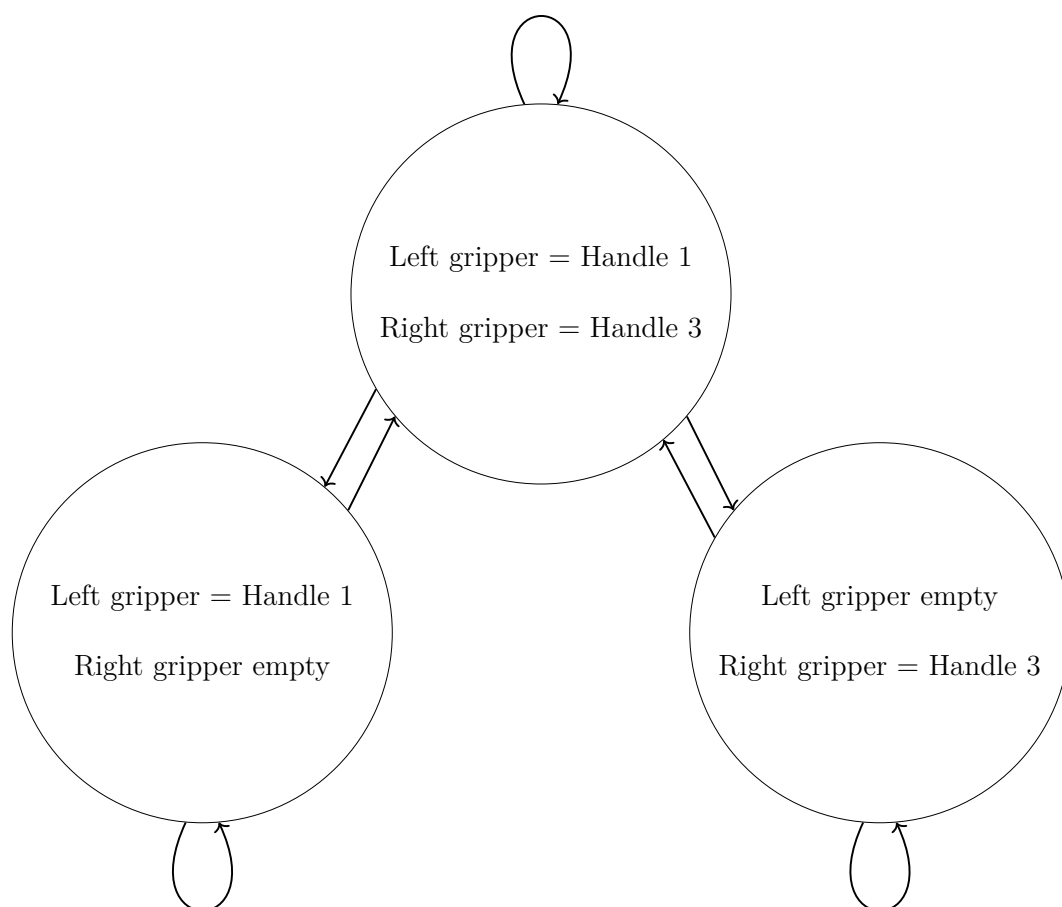


Figure 3.3: A simple graph of constraints with three states. The = sign means that the gripper's and the handle's frames are superposed, and describes how the robot has grabbed the object. This graph represents the state to pass an object from one hand to the other. No transition exists between the states on the sides because it would require to instantaneously change the grippers' from open to closed and *vice-versa*.

### Transition

States are connected by **transitions**, which contain additional parametrisable numerical constraints. A transition is defined by three states: its origin, its destination and its owning state ; the latter defining the overall set of constraints that apply along the whole transition. Often, the owning state is either the origin or the destination. A transition exists between two states  $S_0$  and  $S_1$  if there is a path  $\gamma : [0, 1] \mapsto \mathcal{CS}$  such that:

- $\gamma(0) = q_0 \in S_0$  and  $\gamma(1) = q_1 \in S_1$ ,
- $\forall t \in [0, 1], g(\gamma(t)) = g(q_0)$ , where  $g$  is the set of parametrised constraints applying to the transition.

Conversely, the existence of a transition between two states does not imply the existence of a feasible path between configurations belonging to those states. Indeed, a transition can exist between a state with no grasp and a state where the robot

is grasping an object, but the object could be out of the reach of the robot, and therefore, a physical path cannot exist. The same is true if a collision prevents the motion planner from finding a feasible path.

In the experiment, there were two main types of transitions. Those of the first type link two states together, and represent the creation or deletion of a **grasp**. The transitions of the second type are the ones that loop over the same state, and represent the possibility for the system to move while keeping the existing grasps.

### Graph of constraints

The combination of the states and transitions defined above forms a graph that we call the **Graph of Constraints**. It represents the manipulation rules for a given problem. An example of a graph of constraints is given fig. 3.3. It represents the states necessary to pass an object from the left hand of a humanoid robot to its right hand, and conversely.

### Waypoint state

The existence of a transition between two states does not imply the existence of an achievable path for the robotic system. But, even if a path exists, the motion planning algorithm has to find it in a reasonable amount of time. The framework being aimed at the planning of manipulation motions, it implies interactions with movable objects. Any interaction between a robot and an object lies at the frontier between  $\mathcal{CS}_{\text{free}}$  and  $\mathcal{CS}_{\text{obs}}$ . Moreover, to avoid the difficulties of the online checking of the viability of the grasps of the robot, we explained previously that the handles of an object are in finite number and at manually-specified positions. Therefore, classical motion planning methods, by random sampling of configurations and their subsequent projections onto the constraints, is unlikely to give sufficiently quick results so close to  $\mathcal{CS}_{\text{obs}}$ . The solution we use is the addition of waypoint states on transitions creating or deleting a **grasp**. We call them **pre-grasps** states. Those states are defined by the same set of constraints as the state the transition lies in, with the addition of a fixed pose constraint for the **gripper**, a short distance away from the **grasp** pose itself. The previous graph of constraints with the addition of waypoints is presented fig. 3.4. This allows the motion planning algorithm to more easily, and therefore quickly, reach the waypoint, and then a straight path can connect the waypoint configuration with the **grasp** configuration. The fig. 3.1, in the introduction, displays a configuration in a waypoint state that makes the left gripper less likely to be in collision during the subsequent grasping.

## 3.2.2 Estimation of the initial state

In the previous section, we explained the concept of the **Graph of Constraints**, which is a representation of the rules and constraints applied to a manipulation problem. However, in order to plan the tasks and motions necessary to reach the goal of a specific instance of the problem, one must first know the initial configuration of the system. The naive and time-tested solution is to always start in the same known configuration, with all the objects at precise positions. Because one of the goal of our framework is to allow more leeway in the programming of robotic systems, this solution is unacceptable.

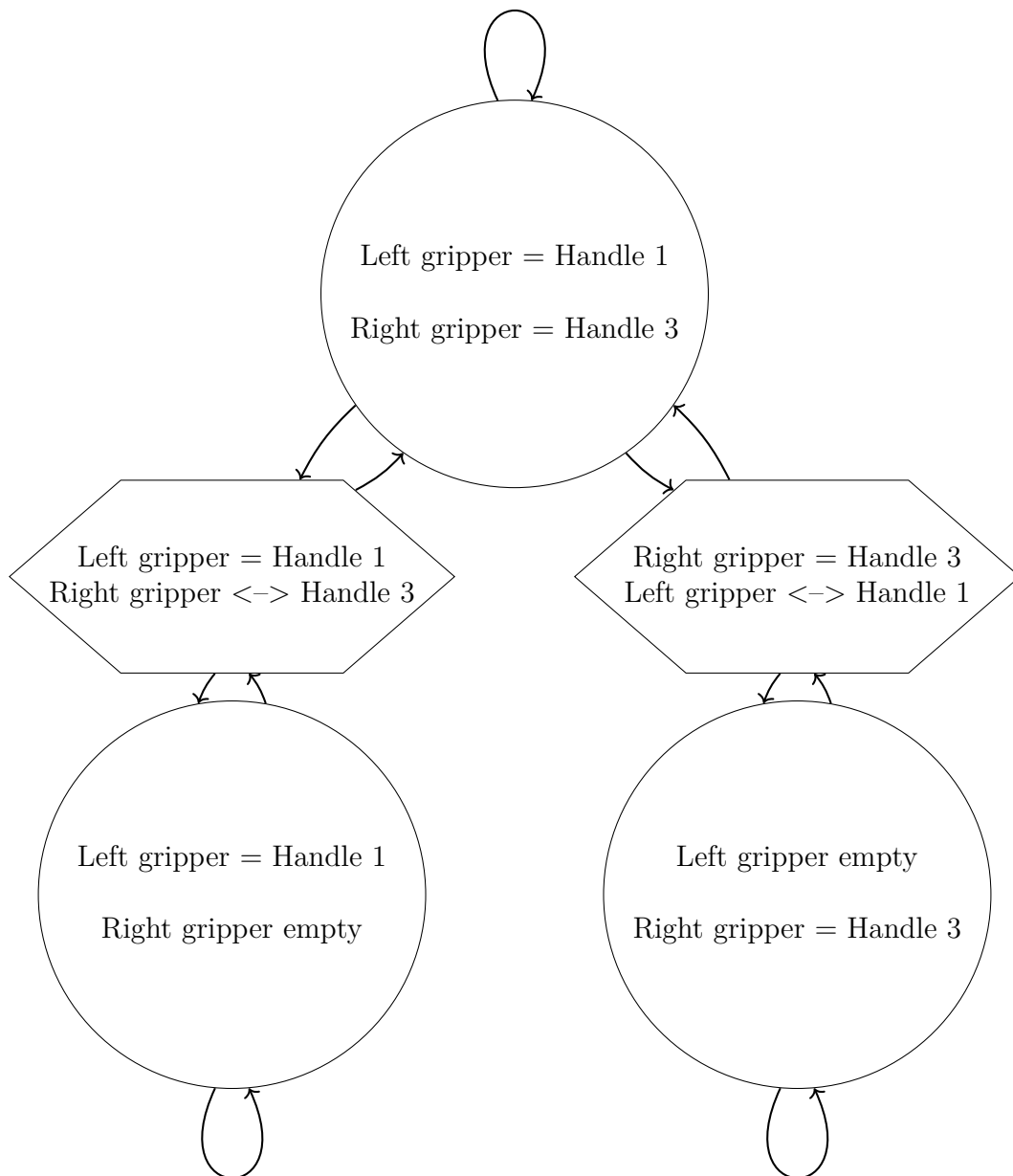
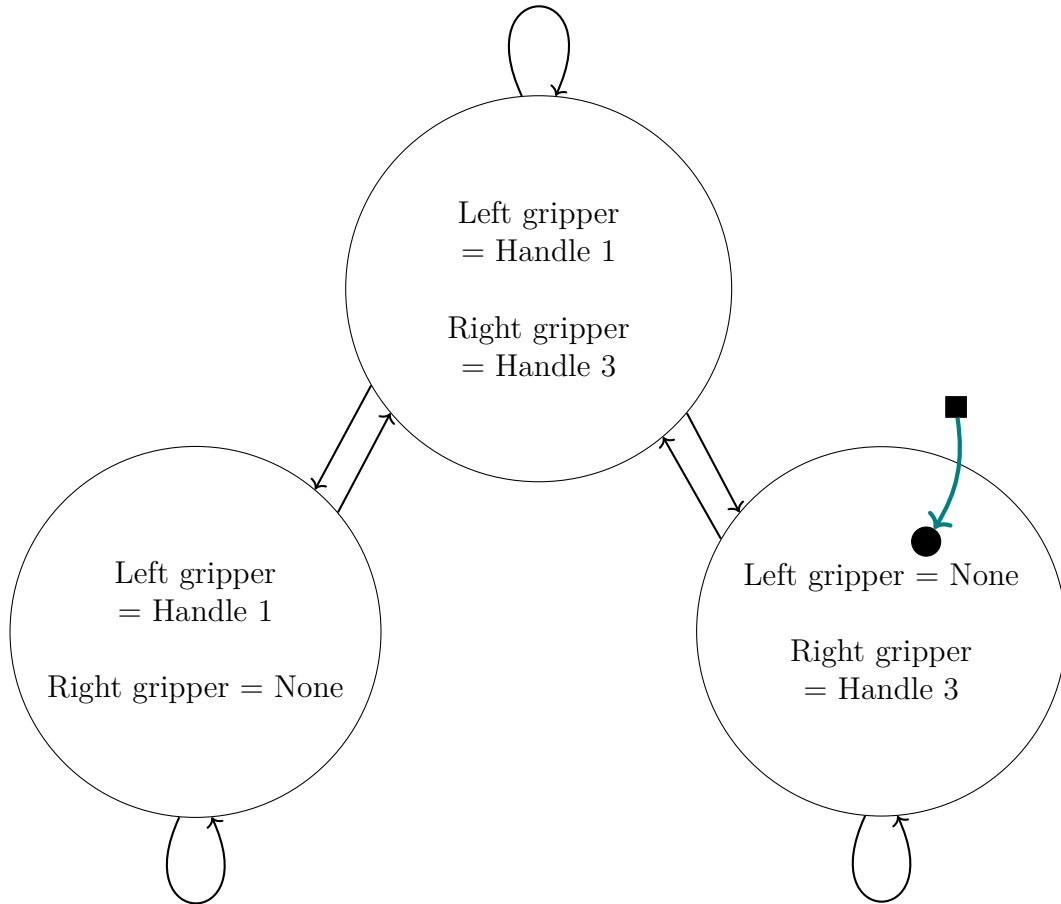


Figure 3.4: The graph of constraints to pass an object from one hand to the other, with added waypoints to improve the motion planner performance. Circles are states and hexagons are waypoints. The  $\leftrightarrow$  sign means that the gripper and object are close, and the gripper either approaches the object or recedes from it, depending on the states of origin and destination.

The approach we chose instead is the online estimation of the initial configuration by the robot itself. Robotic manipulators are equipped with various sensors in order to be able to figure out at least their own configuration. For the purpose of manipulating object, we also use on-board cameras to assess the situation of the rest of the environment. To ease the detection of the relevant objects, we opted for the use of markers, specifically AprilTags [171], positioned at known coordinates on those objects. Indeed, this thesis is not oriented toward computer vision, and this

solution was acceptable for the contribution we aimed to propose.



- Configuration computed from the encoders and cameras
- Configuration projected onto the nearest state

Figure 3.5: The robot computes the configuration of itself and the world using its sensors. This **raw** configuration is then projected onto the nearest state of the graph of constraints, to become valid for the motion planner.

Thus, we obtain the configuration of the robot via the embedded encoders in its joints, and the configuration of the rest of the environment in relation to the robot using its cameras. However, due to the finite precision of those sensors, the perceived configuration of the whole system can never precisely satisfy the constraints of any state in the **graph of constraints**. The configuration of the system therefore does not belong to any state of the graph of constraints and the manipulation planning algorithm is unable to plan a path. To overcome this issue, we project the configuration measured by sensors (AprilTags and joint encoders) onto a state using the Newton-Raphson method as described in [190] Sec. II.B, and reproduced in algorithm 1 to improve the clarity and the fluidity of the reading. The result of this projection is illustrated schematically on the simple graph of constraints presented previously, fig. 3.5.

A benefit of this approach, in addition to the extended latitude on the placement of manipulated objects, is the ability to detect when the system is already in a latter

**Algorithm 1** Projection on implicit constraints

---

```

procedure PROJECT( $q, g, \epsilon$ )
   $\alpha = .1, \alpha_{max} = .95, it = 0, it_{max} = 20$ 
  while  $\|g(q)\| < \epsilon$  do
     $q \leftarrow q - \alpha \left(\frac{\partial g}{\partial q}(q)\right)^+ g(q)$ 
     $\alpha \leftarrow \alpha_{max} - .8 * (\alpha_{max} - \alpha)$ 
     $it \leftarrow it + 1$ 
    if  $it > it_{max}$  then
      return Failure
    end if
  end while
  return  $q$ 
end procedure

```

---

step of the process. This is important for us because one of the long term goal of this framework is to allow the collaboration of multiple robots or even robots and human operators. An example of this, that will be developed in further chapters, namely chapter 5 and chapter 6, is about robots drilling holes in aircraft parts. This process has multiple steps, some of which will be made by either humans or robots, depending on their availabilities and the skill needed (holes near supports may need a clearance that prevent robots from operating). Having one program capable of adaptation to the situation and the advance of the process it is placed in would be a vast improvement over manual initialisation.

After this estimation step, the initial state of the manipulation problem is known, and the goal state can be defined even if it depends on the initial state, as was the case in our demonstration. The next phase will be the elaboration of a guide path in  $\mathcal{CS}_{free}$  to reach this goal.

### 3.2.3 Manipulation planning

As we have seen in the section 3.2.1, a path can only be found between two configurations  $\mathcal{C}_{init} \in S_i$  and  $\mathcal{C}_{goal} \in S_g$ , if transitions exist between those states. However, the existence of the necessary transitions is not enough to guarantee that a path can be found in  $\mathcal{CS}$ . This path could be prevented to exist by collisions or unreachable spaces. Therefore, a motion planning algorithm has to be used in order to create a continuous guiding motion between the initial and goal configurations.

To solve the manipulation planning problem, we run a variant of RRT, that has been proposed in [189] Sec. III. It is called **Manipulation-RRT**, and is able to explore the states defined in the graph of constraints. While the precise operation of this algorithm can be found in the cited publication, an abstracted version is provided in algorithm 2.

Upon completion of the algorithm, either a path has been found in  $\mathcal{CS}_{free}$ , that goes from the initial configuration towards the goal configuration specified in the problem, or no complete path could be created. In the first case, the returned path is expected to be of low quality. Indeed, the RRT algorithm is based on random sampling of configurations. Thus, even if the variants of RRT we use are asymp-

**Algorithm 2** Manipulation Rapidly-exploring Random Tree

---

Draw a random configuration  $\mathbf{q}_{rand}$ ,  
 Find the closest node  $\mathbf{q}_{near}$  in the current roadmap,  
 Find the state of this node in the constraint graph,  
 Sample a transition getting out of this state,  
 Extend  $\mathbf{q}_{near}$  along the transition up to  $\mathbf{q}_{new}$ ,  
 Try to connect  $\mathbf{q}_{new}$  to other connected components of the roadmap.

---

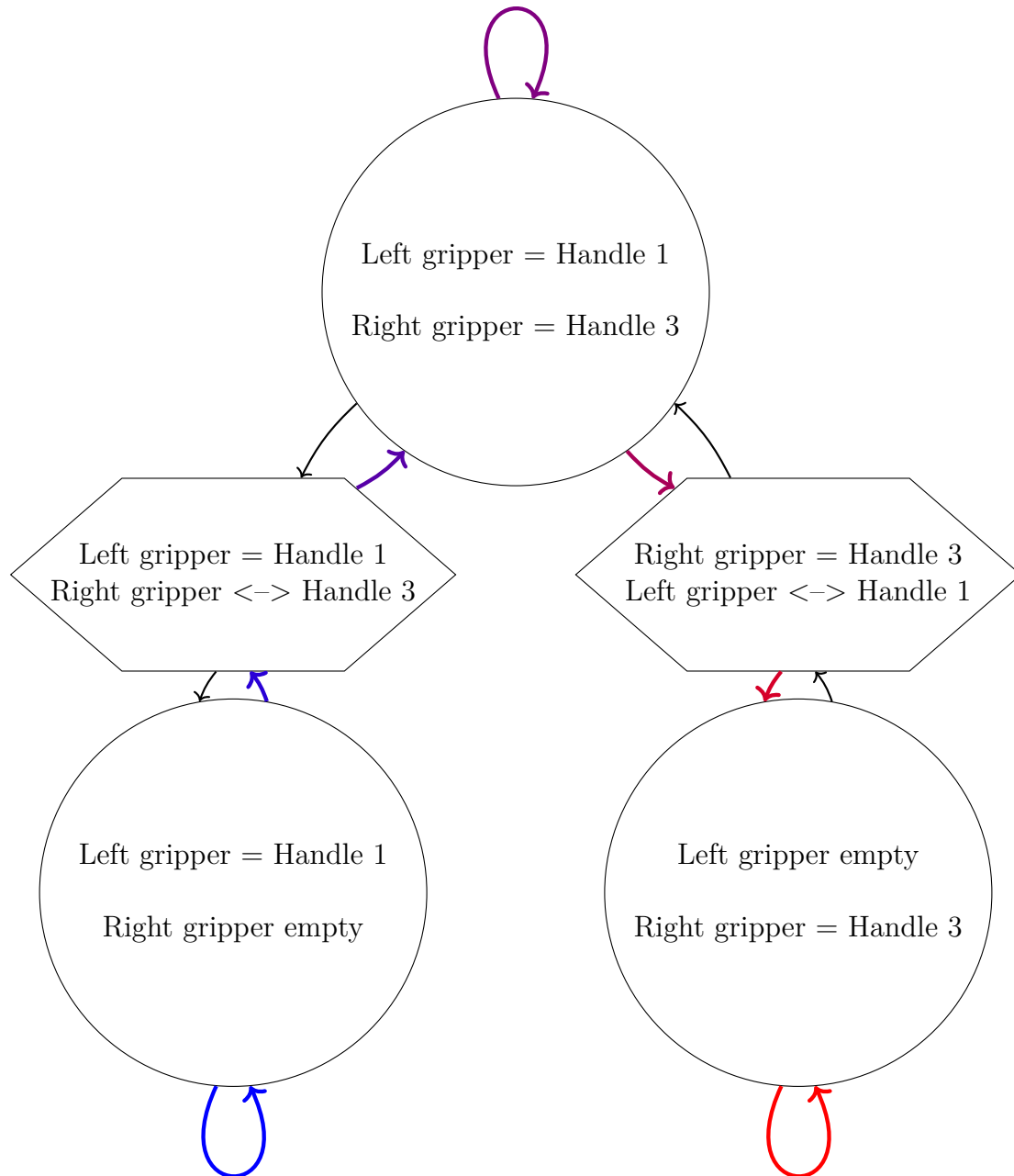


Figure 3.6: The transitions chosen by the motion planner in the graph of constraints to pass the object from the left hand to the right hand. The order of the transitions is from blue to red.

totically optimal, they are not run long enough to reach this optimality. Therefore, the resulting path will be punctuated by seemingly random movements that do not contribute to the overall goal of the problem's instance. A refinement step has to be taken, in our case with the use of the random shortcut method and gradient-based optimisation algorithms [191, 5].

The output of manipulation planning is a sequence of paths linking the initial and goal configurations. Each segment of the sequence lies in a **transition** of the graph of constraints. A schematic illustration of the choice of transitions to reach the goal can be seen in fig. 3.6. In the following paragraphs, we will describe the control strategy that is used to respect those constraints.

### 3.2.4 Hierarchical control

When reaching this step, we are in possession of a guide path for all the actuators of the system, *i.e.* in  $\mathcal{CS}$ . However, the blind following of this path is not sufficient to ensure that the goal of the manipulation will be reached. Indeed, someone or something could perturb the actuators trajectories, and some form of feedback is necessary to ensure that the overall movement still respect the physical constraints of the system and achieve the goal.

Another obvious interest of control is the ability to handle objects that are moved between the planning step and the execution step of the movements, or even during the execution itself. This important part will be treated specifically in a later chapter, chapter 5.

The result of the control step is a vector of commands for the actuators of the robot. This can come in multiple forms, such as position, velocity, acceleration, force or current control. In the Agimus framework, commands are expressed kinematically, using both position and velocity control, to ensure smooth execution of the trajectories. In order to compute those, multiple source of information can be used.

Broadly, those sources can be separated into two groups: feed-forward and feedback data. Feed-forward data are given a priori to the system, and some, like the trajectories of the joints of the robot, can be pre-computed. It includes the path data created during a motion planning step, model-based commands, etc. Feedback data are coming from the robot's sensors, and are used to check whether the paths provided to the robot are followed, and adjust them in case of perturbations.

Between the input data and the output commands, there are multiple mathematical objects that will be presented in the following paragraphs. Their implementation will be detailed in section 3.3.4.

#### 3.2.4.1 Task-functions

We explained in section 2.5 that during this manuscript, we are using the concept of task-function defined by Samson, Espiau, and Le Borgne [7]. I recall his definition here for clarity:

**Definition 8 — A task-function**

is a  $\mathcal{C}^2$   $n$ -dimensional function  $e(q, t)$  that is usually regulated to zero during the duration  $[0, T]$  of the task.

By abusing the notation, in this work and the related and cited publications, task and task-function are often used interchangeably.

**Feature**

$e(q, t) = s(q) - s^*(t)$  is the general form of a task-function, where  $s$  is a measured, or estimated, feature of the system, and  $s^*$  its desired value. A feature can be any measurable value, such as the position of a end-effector, the angle of a joint or a visual cue that the robot has to maintain at the center of its field of view.

Those feature are generally separated into two types:

- Those formulated in  $\mathcal{CS}$ :  $e(q, t) = q - q_{des}(t)$  where  $q_{des}$  is the desired trajectory for some joints of the robot.
- Others whose formulation is in task space, which often coincide with the world coordinates:  $e(q, t) = x(q) - x_{des}(t)$ , where  $x(q)$  is a pose, and  $x_{des}(t)$  its desired trajectory.

In the end, because the commands are given at the joint level, everything happens in  $\mathcal{CS}$ . This makes objectives formulated in task space generally non-linear, because the relation between a chain of articular angles and the 6D pose of an end-effector is not linear.

**Exponentially decaying error**

The task is therefore seen as the error between a current feature measurement and its desired value. It is possible, without loss of generality, to impose a desired trajectory for the reduction of this error, and we choose in our software the exponential decay:

$$\dot{e}_i^* = -\lambda e_i \tag{3.1}$$

where  $\lambda$  is the gain applied to the task.

We define the Jacobian matrix  $J_i$  of the task  $e_i$  as follows:

$$\dot{e}_i = J_i \dot{q} \tag{3.2}$$

In our framework, the robot is controlled purely with kinematics variables, in other words by commanding the velocities of the motors. If we assume the Jacobian of the task to be of full rank, we obtain:

$$\dot{q} = J_i^+ \dot{e}_i^* \tag{3.3}$$

$$= -\lambda J_i^+ e_i \tag{3.4}$$

**3.2.4.2 Dealing with multiple tasks**

Complex systems, such as humanoid robots, can have multiple tasks, often with a notion of priority between them. For example, for a humanoid robot, maintaining contact with the ground and equilibrium is of the utmost importance, while reaching



for an object is only a secondary objective. Therefore, we have to find the joint velocities for a set of tasks that may interact with each other, while keeping in sight their ordering.

One way to realise that is to assign weights to the multiple tasks, and sum them inside a large task-function that execute everything at the same time. However, such systems may be hard to tune. Too large a weight for the important task may prevent the execution of the secondary objectives, even if the robot would be able to achieve them. Too low, and the equilibrium task could be disrupted by those secondary targets, in our case leading to the fall of the robot.

One solution implemented in the SoT is thus a strict hierarchy of tasks, that ensures that objectives of lower importance cannot disturb higher level tasks. The main idea of this approach is to use the whole  $\mathcal{CS}$  for the first task, and then compute the null space of this task, meaning the DoFs that do not impact the task. Then, the next task is solved inside this reduced space, and we again compute the null space of both this task and the higher level ones. We progress recursively like this until all tasks are accounted for, or the null space is reduced to zero.

In a nutshell, the SoT solves iteratively this set of equations:

$$\dot{q}_{i+1} = \dot{q}_i + (J_{i+1}P_i)^+ \left( \dot{e}_{i+1} - J_{i+1}\dot{q}_i + \frac{\delta e_{i+1}}{\delta t} \right) \quad (3.5)$$

$$P_{i+1} = P_i - (J_{i+1}P_i)^+(J_{i+1}P_i) \quad (3.6)$$

where:  $i \in \{1, \dots, n\}$   
 $q \in \mathcal{C}_{rob}$ , the configuration vector of the robot  
 $\dot{q}$  = the related velocity  
 $e_i$  = the  $i^{\text{th}}$  task  
 $J_i = \frac{\delta e_i}{\delta q}$ , its Jacobian  
 $\dot{e}_i = -\lambda_i e_i$   
 $\dot{q}_o = 0$   
 $P_0 = I$

### Computation cost

While this approach is effective to deal with multiple objectives for our robot, the computation cost of the pseudo-inverse in eq. (3.5) and eq. (3.6) is prohibitive. Escande et al. [142] showed that it is more efficient to iteratively compute a basis of the null space of the tasks.

Therefore at iteration  $i + 1$  the null space of the previous control task is given by an orthogonal matrix  $K_i$  such that  $J_i K_i = 0$  and  $K_i^T K_i = I$ , then using a SVD decomposition:

$$J_{i+1}K_i = [U_{i+1} \ V_{i+1}] \begin{bmatrix} S_{i+1} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} Y_{i+1}^T \\ Z_{i+1}^T \end{bmatrix} = U_{i+1} S_{i+1} Y_{i+1}^T \quad (3.7)$$

Then the null space of  $J_{i+1}K_i$  is given by  $Z_{i+1}^T$ . Therefore any new control  $u_{i+1}$  is such that:

$$\dot{q}_{i+1} = \dot{q}_i + K_i u_{i+1} \quad (3.8)$$

to not perturb  $\dot{q}_i$ . Therefore we slightly rewrite the control eq. (3.5) and the update eq. (3.6) as:

$$\dot{q}_{i+1} = \dot{q}_i + (J_{i+1}K_i)^+ \left( \dot{e}_{i+1} - J_{i+1}\dot{q}_i + \frac{\partial e_{i+1}}{\partial t} \right) \quad (3.9)$$

$$K_{i+1} = K_i Z_{i+1} \quad (3.10)$$

Updating  $K_{i+1}$  is simpler than computing  $P_{i+1}$  and the number of columns of  $K_i$  decreases after each iteration. This allows us to have this decomposition working at 1 kHz on the embedded computer of a TALOS humanoid robot.

The method of computing the commands in the null space of the previous tasks is however not devoid of downsides. Indeed, in the presence of computational singularities, that can for example appear when an arm is nearing its full extension, the null space can be reduced too quickly and prevent most of the lower importance tasks to be achieved. But we are following a motion planned path, so even if singularities are approached, the path provides a reliable reference to follow. Difficulties could of course arise with the use of visual servoing, later in this manuscript, but the margins used to avoid singularities during motion planning were sufficient for it not to appear in our experiments.

Nevertheless, if this become a real issue, one could implement controllers able to deal with the problem. A possible solution would be using manipulability ellipsoids [192] to guide the end-effector along the principal axis for the prioritised tasks and keep a larger field of motions available for the secondary objectives.

### 3.2.4.3 Admittance control

In order to successfully grab an object and most importantly keep it steady in its gripper, we have to control said gripper not only in position but also in torque. However, the gripper of the TALOS robot is one of the few joints that is not equipped with a torque sensor and thus, native torque control is not available. We had to design what is called an admittance control scheme in order to command the position of the gripper by simulating a dampened spring system as output, in order to exert the desired force on the object.

The torque applied by the gripper's motor is estimated using the intensity of its supplied current. Neglecting the electrical losses, there is a proportional relation between input current and output torque for a DC motor:

$$\tau_{motor} = I \cdot k_{\tau_{motor}} \quad (3.11)$$

Then, still neglecting frictions and losses, the torque output of a reduction gear is also proportional to the input torque supplied by the motor:

$$\tau = \tau_{motor} \cdot k_{gear} \quad (3.12)$$

And finally, the force exerted by the gripper on the object is also linear with respect to the distance between the axis of rotation and the point of contact with the object:

$$\mathcal{F} = \tau \cdot d_{axis-gripper} \quad (3.13)$$

The values of  $k_{\tau_{motor}}$  and  $k_{gear}$  are given by the manufacturer of the robot, while  $d_{axis-gripper}$  has been measured roughly on the robot itself. We decided to aim for a gripping force of around 5 N at first to steady the object. The measurements of the current intensity pass through a first-order low-pass filter, with cut-off frequency of 5 Hz, in order to eliminate the high frequency variations of the 10 kHz control at the joint level.

The control is then switched between pure position control and admittance control. At first, we slowly command a gripper position in collision with the object. At the moment of contact between the tip of the gripper and the object, this position will not have been reached, thus the internal PID loop of the joint will command higher currents. Above a selected threshold, slightly below the desired gripping force, the commands are switched over to the admittance control scheme, parametrised empirically to ensure both the continuity and smoothness of the command. When releasing an object, the inverse process occurs, first with an admittance control, then a position control to fully open the gripper.

### 3.2.5 Controllers' generation and execution

In the previous part, we presented the control scheme that ensure the correct execution of the manipulation. There is still two missing pieces for the functioning of our framework. The first one is the generation of the controllers themselves, based on the information provided by the graph of constraints and a manually designed correspondence between the constraints and the tasks of the controllers. The second is the creation of a FSM that supervises the progress of the manipulation and orchestrates the various motions and associated real-time controllers. Those two steps are described in the following paragraphs.

#### 3.2.5.1 Generation of the real-time controllers

Tasks are the execution counterpart of the constraints present in the graph and at the manipulation planning level. Often, they can be formulated in a similar manner for those two parts of the software. Equality constraints in the manipulation planner could be automatically translated into task-function regulated to zero in the controllers. An example of this is the pose of an end-effector relative to another.

However, things like the position of the CoM or the constraints of the gaze of the robot are formulated as inequalities for the manipulation planning, to allow some freedom of movement. Because of the task-function approach of our controllers, a direct translation is not possible.

Hierarchical controllers able to deal with inequalities exist, but the software used by the team had not been maintained for several years and were not tested on the current robots. Therefore, inequality constraints in graph of constraints are also formulated as task-functions following the planned path at the control level.

In order to deal with incompatibilities and still be able to generate multiple controllers automatically given a graph of constraints, a manual correspondence between each type of constraint and a task has been created. This is presented in more details in the section dealing with the *Agimus* software itself. Controllers are generated for each transition of the graph of constraints that might be crossed

during the execution of the experiment.

The controllers contain tasks that can be separated into three groups, by order of priority, from the most important to the less important:

- Common tasks that ensure the balance of the robot
- Transition-specific tasks that mainly deal with the grasping of the manipulated objects
- A posture regulation task whose goal is to track the reference configuration path resulting from the motion planning phase

The first and last groups are therefore put into each and every controller, but follow different references during the execution of the experiment. The addition of the second group of tasks depends on the transitions' position inside the graph of constraints.

To automatically generate the intermediate tasks, we pass through the graph of constraints. In a nutshell, there are only three main type of transitions: motions, creation of a grasp, deletion of a grasp. Motion's transitions between two states do not induce the creation or deletion of a constraint, therefore the controller can keep the same stack of tasks as the one existing in the origin state.

A simple grasp, *i.e.* using only one gripper, is dealt with as follows:

- The origin state  $S_0$  has no transition-specific task in its controller, thus only the balancing and posture tasks are present.
- Between  $S_0$  and  $S_1$ , the goal state of the transition, there exist a waypoint state,  $Sw_{0-1}$ . This state has the same controller as  $S_0$ . It only adds a pose goal for the motion planner, a few centimetres above the grasp, to help reduce the computational cost of planning. Thus, the transition between  $S_0$  and  $Sw_{0-1}$  keeps the controller used for  $S_0$ . At this time, the transition between  $Sw_{0-1}$  and  $S_1$  also uses this controller, but a visual servoing task was added for the experiment described in chapter 5.
- We pass through a final transition, from  $S_1$  into  $S_1$ , that adds the position and admittance tasks handling the gripping force of the robot. This is the transition where the robot closes its gripper and grab the object.

The resulting SoT at the end of this procedure is then kept during all motions involving this grasp, to prevent the object from sliding out of the gripper.

A double grasp, *i.e.* using both grippers, involves an additional task between  $Sw_{0-1}$  and  $S_1$ , that makes the pose of the free gripper relative to the pose of the gripper already grasping the object. In this chapter, the relative positioning of the two grippers is only managed at the path planning level. Loop closure based on visual information was added for chapter 5.

The inverse operation, ungrasping, is done in a similar fashion, but in reverse:

- The origin state  $S_1$  has at least one gripping force task, and maybe the relative pose of the second gripper, in addition to the balancing and posture tasks.
- The first transition, from  $S_1$  to  $S_1$ , opens a gripper, so the gripping task has to remain and be controlled, as do the relative pose task, if it exists.

- Between  $S_1$  and  $S_2$ , there is a waypoint,  $Sw_{1-2}$ , with the same tasks as  $S_2$ . For the transition from  $S_1$  to  $Sw_{1-2}$ , the gripping force task is removed. Then, between  $Sw_{1-2}$  and  $S_2$ , if a relative pose task was present, it is also removed.

### 3.2.5.2 Finite State Machine to control the execution

While browsing the graph of constraints, along with the generation of the controllers, Agimus also builds a FSM that will orchestrate the execution of the manipulation. For each transition in the graph of constraints, a corresponding state is built in the FSM. This FSM has two key goals: handling the passage from one controller to another, and using the robot's sensors to detect important events.

The changeover of controllers is a critical step for a real robot, because any discontinuity in the commands can have dire consequences on the hardware. Therefore, we check the state of the robot both at the start of a new state, *i.e.* before going through a transition in the graph, and at the end of the state, after the transition has been passed.

The starting function confirms that the commands of the next controllers are close enough to the one of the current controllers, before the actual switching. At the end, we verify that the reference trajectory has been sent in its entirety, and that the associated commands have converged. There might still exist non-zero commands for some tasks such as visual servoing, but if the next controllers keep this task, it should not be an issue.

The second goal of the FSM is to trigger reactions to events detected by the robot and its software. Those come from various origins, such as the sensors of the robot, the controllers' state or internal data. The principal source of events is simply the manipulation planner informing that it has sent the whole planned trajectory for the current transition. It triggers the switching towards the end function of the state that, as mentioned above, waits for the controllers to converge, then switches over to the next state in the FSM.

Another event monitored during our experiment is the force exerted by the robot onto its environment. It allows us to quickly stop execution in case of impromptu contact, which could harm someone or damage the robot. But it is also used to detect an imprecisely planned contact, for example to grab an object. In this case, the transmission of the reference trajectory is interrupted because the current goal has already been reached.

## 3.2.6 Conclusion

In this section were presented the concepts and algorithms that are at the basis of the functioning of the framework Agimus. It all begins with the graph of constraints, which models what the robot can or cannot do, both in term of physical and of manipulation abilities. A second step uses an estimator to determine the initial state of the world, and optionally refine the goal based on those information. Then we succinctly presented the manipulation and motion planner that browses the graph of constraints in order to travel from the initial state to the goal while respecting the constraints and avoiding any collision with the environment. The last part

dealt with the main contributions of the Agimus framework in the field of objects manipulation by a robotic system. First, the automated generation of the real-time controllers, in the form of strict hierarchies of tasks, based on the transitions taken by the manipulation planner through the graph of constraints. Finally, the simultaneous, and also automated, creation of a complete FSM that regulates the switching between the controllers, reacts to external or internal events, and more broadly ensures the good execution of the experiment.

### 3.3 Software

In this section are presented the software that implement the concepts discussed previously. The first one, Visual Servoing Platform (ViSP) [3], is a computer vision and control software that we use simply to detect and locate AprilTags in the environment and on the objects manipulated by the robot during our demonstrations. The second, Pinocchio [4], is a library implementing efficient computation of the kinematic and dynamic parameters of robots. It is at the core of the two following software. Humanoid Path Planner (HPP) [5] is the motion planner created by the Gepetto team. It is based on the concept of the **graph of constraints** and uses mainly random sampling planning methods to provide paths for robotic systems in constrained problems. The fourth piece of software in Agimus is the Stack-of-Tasks (SoT) [6], which implements a real time hierarchical control scheme for complex robots. The following is **ros\_control** [193], providing the link between the low level control of the robot and the higher level control mentioned just above. Finally, our contribution, Agimus itself, aims at creating the hierarchical control stack automatically, based on the **graph of constraint**.

#### 3.3.1 Visual Servoing Platform

ViSP is an open-source modular software library aimed at the fast prototyping of visual tracking and servoing applications. It was first proposed in 2005 by the Rainbow team (then known as the Lagadic team) at the French laboratory INRIA. ViSP is still improved to this day by the same team, with contributions from various students hosted in the lab, and end users from all over the world.



Figure 3.7: ViSP's logo

At the time of its inception, available visual servoing software were nearly always tied to specific hardware, be it the robot, the camera or the computing system. Therefore, one of the goal of ViSP is to be cross-platform and as independent from the targeted hardware as possible. A strong emphasis is also placed on the documentation and simplicity, to let new users be quickly efficient and able to prototype their ideas. The software itself is a C++ library made of various modules mostly independent of each other for an easier composability. The two main targets of those modules are visual tracking on one side, and visual servo control on the other.

Within the *Agimus* framework, it is used for the detection and tracking of artificial landmarks called *AprilTags*. Simpler and smaller software could certainly have been integrated in the framework for this purpose. However, the good packaging of *ViSP* into the platform we use, Linux Ubuntu, combined with the high quality documentation, made it the most efficient choice for us. Besides, the modular architecture of *ViSP* reduced the burden of the integration because we only had to deal with a few modules.

### 3.3.2 Pinocchio

*Pinocchio* is a software library implementing state-of-the-art rigid body algorithms for poly-articulated systems. It was introduced in 2019 by the *Gepetto* team at Laboratory for Analysis and Architecture of Systems (LAAS-CNRS), but has been used for several years and in various project before this official presentation. It is now maintained and actively improved by both the *Willow* team at INRIA and the *Gepetto* team, along with contributions from other users.

Rigid body dynamics is an extremely useful tool in the robotic field. Despite the theory being more than two centuries old, algorithms are still periodically revised to improve their efficiency and adapt them to modern hardware's computing acceleration strategies. One of the last revision of those algorithms has been proposed by Featherstone [194], and is the main basis upon which *Pinocchio* is built.

*Pinocchio* provides efficient implementations of those algorithms for systems composed of rigid bodies linked via joints of several types: revolute, prismatic and spherical, or even complex compositions of the aforementioned joints. The systems themselves can be fixed in the world, like industrial manipulators, able to navigate on the floor, or even free-floating. Among the main algorithms available are Recursive Newton-Euler Algorithm (RNEA), Articulated Body Algorithm (ABA), Composite Rigid Body Algorithm (CRBA), etc. They allow the user to easily compute **forward** and **inverse kinematics**, **forward** and **inverse dynamics**, **kinematic jacobian** or **joint space inertia matrix**, among others. *Pinocchio* also handles geometric collision detection by incorporating the Flexible Collision Library (FCL) [195].

At the time of creation of this library, two contrasting implementation approaches of the computing of rigid body dynamics were at the core of the available software. On the one hand, there were software based on code generation, trading flexibility for an important gain in performance. On the other hand, more recent libraries loaded robot models at runtime, increasing the versatility.

*Pinocchio* follows the two paradigms at once, being able to load dynamic models at runtime, but also to generate code and carry out automated differentiation, also during the program execution, to improve performance. It also exploits at best the sparsity induced by the kinematic trees of the modelled systems, which are mostly robotics, human or animal, and therefore devoid of too many closed kinematic chains.

To further boost the performance, the main author added analytical derivatives [196] for the main algorithms. Finally, thanks to the use of modern programming language paradigms, *Pinocchio* is able to perform most of the computations directly at compile time. Benchmarks available in [4] demonstrate that *Pinocchio* outperforms all other existing frameworks at the time of presentation.

Pinocchio is now at the core of several robotics software, among which we can cite Crocodyl [147, 148], an open-source and efficient Differential Dynamic Programming solver for robotics, the Stack-of-Tasks, an open-source and versatile hierarchical controller framework or the Humanoid Path Planner, an open-source software for Motion and Manipulation Planning. The last two libraries are themselves part of the Agimus framework and are presented in the upcoming sections.

### 3.3.3 Humanoid Path Planner

Humanoid Path Planner (HPP) [5] is a C++ software library that implements path planning and manipulation planning algorithms for articulated systems moving in environments cluttered with obstacles. It is the result of a major refactoring of the work around path planning software developed since the 1990s by the robotic department of the LAAS-CNRS. It was open-sourced at the end of 2013, originally presented in [116], extended in [5], and is still actively used and improved by the Gepetto team to this day.

The main characteristic of HPP over other path planning software is its native integration of the features necessary to handle the constraints associated with humanoid robots. Those non-linear constraints, such as quasi-static equilibrium or position constraints of end-effectors, are indeed modelled at the core level of the software and handled by the RRT algorithm used as the default planning method.

Also at the basis of HPP is the concept of the graph of constraints, explained in details in section 3.2.1, which adds a structure to the manipulation problems, by expressing the various grasps possible and their connectivity. Despite its origin mainly oriented toward humanoid robotics, this planner is versatile enough to tackle other kind of robots, ranging from simple 6 DoFs arms up to cable robots [54].

On the technical side, HPP is now based upon Pinocchio for the handling of kinematic and dynamic properties of the kinematic chains describing the robots. We can also note that, like its ancestor Move3D [17], HPP is using a single implementation of the concept of roadmap, seen in section 2.1, that can then be employed successively by several path planning algorithms.

Indeed, multiple path planning algorithms have been proposed over the years. However, at the moment, none of them is clearly superior to the others in all situations. So there exists various state-of-the-art algorithms with different forces and weaknesses that are useful for different kinds of problem, and being able to use them in the same framework is quite practical.

Finally, like other software developed by the Gepetto team, its core is written in C++ for high performance, but accessible via a Python interface to ease the burden of prototyping new algorithm's implementations, or in our case, new demonstrations of the already existing ones.

In the Agimus framework, HPP is used extensively. The core of the framework is indeed based on the graph of constraints, originally presented for its use in HPP. Then, HPP is used for the initial estimation of the system configuration and subsequent evaluation of the origin state in the graph. Finally, it naturally provides the motion planner used to compute the guide path for the system.



### 3.3.4 The Dynamic Graph and the Stack-of-Tasks

Dynamic-Graph is a software framework that implements an efficient data flow programming structure. It is similar to the better-known MatLab library Simulink. It was created by the teams of the joint laboratory between Japan and CNRS called AIST, and is now mainly supported by the Gepetto Team at LAAS.

One of its goal is to provide real-time enabled computations with the use of small components, named **entities**, connected by input and output signals, thus forming a graph structure. The other objective is to do that while offering an easy approach to build the graph itself, by the use of a scripting language such as **Python**.

A Dynamic graph is composed of different entities which each provides a more or less elementary operation. Among other, we can cite:

- **Mathematical operators:** addition, subtraction, multiplication, and their matrices counterparts.
- **Device entity:** a special entity that directly controls either real hardware or a simulation of it. It is generally both one of the point of entry of a graph, transferring the data from the robot's sensors, and the exit point, sending commands to the actuators.
- **Solvers:** Larger entities aggregating several signals to compute commands. This part is discussed in more details later in this section.

The efficiency of the overall framework is based on two main pillars. First, small components that can be well-tested and optimised. Second, a flow of computation based around data caching, where outputs are only computed when inputs change. This allow multiple entities to consume time-hungry results, such as matrices' inverses, at the cost of only one inversion per execution of the complete graph.

The Dynamic graph is used in our framework to implement the SoT in a way that is both real-time safe and can be interacted with using **Python** scripting and commands.

The Stack-of-Tasks (SoT) [6] is a software implementing a control architecture for redundant robots, and more specifically humanoid robots. As the name suggests, it is based on the notion of task formalised by Samson et al. [7], and solves GIK, an approach proposed by Nakamura and Hanafusa [133]. It has been developed by the Gepetto team at LAAS-CNRS for more than ten years, but is recently being replaced by other control schemes such as the optimal control library Crocoddyl [147, 148].

Like several other control implementations, the SoT is solving a hierarchy of quadratic programs to compute the instantaneous whole body control for the robot. Different versions of the solver are available:

- A hierarchical scheme with equality constraints and tasks, formulated in velocity: **sot-core**, <http://github.com/stack-of-tasks/sot-core>
- A hierarchical scheme with equality and inequality constraints and task, formulated in acceleration: **sot-dyninv**, <http://github.com/stack-of-tasks/sot-dyninv>

- A weighted sum control scheme formulated using the robot's joints' torques, **sot-torque-control**, <http://github.com/stack-of-tasks/sot-torque-control>

In *Agimus*, we are using the kinematic controller, **sot-core**, to control the robots. The formulation of those various controllers is automated, based on the information structured by the graph of constraints.

Internally, the SoT makes heavy use of Pinocchio for its state-of-the-art computation algorithms for articulated-body dynamics. It is carefully made up of real-time enabled entities in a **Dynamic Graph**, avoiding at runtime operations that are not real-time safe such as memory allocations or disk access. Overall, the control scheme is able to run at 1 kHz on a real humanoid robot equipped with a desktop-grade computer's processor.

The SoT uses the inputs coming from the robot's sensors in order to compute the commands that are sent to its actuators. This bidirectional communication is however not direct, as it passes through a `ros_control` plugin, aptly called **ros\_control-sot**, to easily integrate a SoT within the `ros_control` framework that the robots uses. Because this control architecture is a crucial part of the robots we are using, it is presented in more details in the next paragraphs.

### 3.3.5 `Ros_control`

ROS is an open-source robotics middleware suite, that has become the *de facto* standard framework in light robotics. It provides services such as hardware and software abstraction, message-passing and packaging aimed at easing the development of functionalities in robotics. An application based on ROS is made of several modules, called **nodes**, that communicate between them by passing information or providing and calling services. Modules are typically small and specialised: an image grabber to manage a camera; a Simultaneous Localisation And Mapping (SLAM) component to build maps; a module abstracting a range-finder's interface to gather data for this map building, etc.

One of the most important component available in the ROS community is **ros\_control**. It provides a mean to quickly and easily implement robot's controllers, and later manage them. It focuses on guaranteed real-time performance, and abstraction from the real robot hardware. `Ros_control` was started in 2012 by Chitta et al. [193] to gain independence from the controller manager of the PR2 robot made by Willow Garage, the company behind the creation of ROS. This effort was later joined by many young robotics companies that based their robots on ROS to benefit from the growing community and compatible software.

`Ros_control`'s main job is to start, set up, run in real-time, stop and finally tear up robotics controller software provided by the user or the community. During those steps, it deals with the resource management of the underlying robot, preventing multiple controllers to send conflicting commands to a motor.

This allows developers to concentrate either on the logic of a new innovative controller, without dealing with those dubious yet crucial aspect about control of a real system, or on their real complete applications, by reusing existing controllers and components. Indeed, `ros_control` is robot-agnostic, meaning that, using the

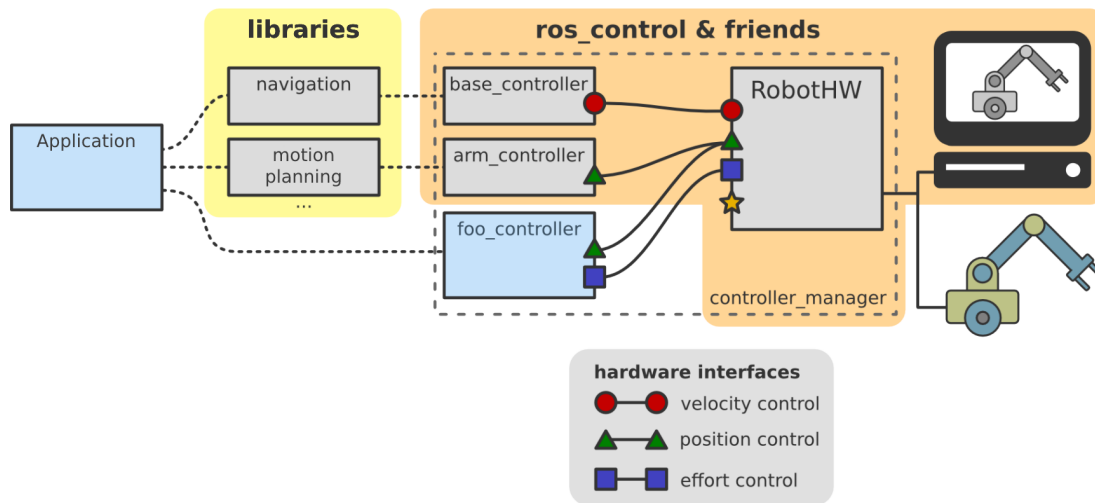


Figure 3.8: An overview of `ros_control` and its place in a robotics software

right configuration files, a controller such as a PID can be used as easily for a 6 DoFs arm that it can be for a delta pick-and-place robot. This independence from the hardware is also advantageous, as it permits to use the same code either on a real robot or on a simulation.

Moreover, several libraries like the ROS navigation stack are able to interface with `ros_control` and use the controllers. Finally, part of `ros_control` can exist between the controllers and the system, in order to check, and if necessary modify, the commands sent. This is a crucial step in ensuring the safety of the real hardware by setting limits on the actuators' positions or velocities. A schematics summarising those various functionalities and their interactions can be found in fig. 3.8.

The robots that we use for our experiments are manufactured by the company PAL Robotics, and their software is heavily based on ROS. Moreover, the company was involved in the development of the `ros_control` component, but is now providing a slightly different version that is able to expose the torque and current sensors present in their robots' joints. Indeed, due to initial design choices, those changes could not be introduced in the main version of `ros_control`, but will be part of the next generation of ROS. In order to combine our robots' software with our hierarchical control stack, the SoT, presented in the previous subsection, a `ros_control` plugin integrating the SoT has been created.

### 3.3.6 The Agimus framework

Agimus itself is composed of several semi-independent packages, each dealing with a different facet of the overall software. The software is mainly made of **Python** scripts, and some **C++** files for the performance-critical parts or to interact with some of the accompanying software. The overall goal of the Agimus framework is to solve a manipulation problem using a real robot which can be as complex as a full-sized humanoid robot.

In order to achieve this, the software needs to first figure out a graph of con-

straints, described in section 3.2.1, that models the ways the robot can manipulate the objects mentioned in the problem. It then computes the initial state the robot and the environment are in. Next, the motion planner HPP is called to compute a path both in the graph and in the  $\mathcal{CS}$  of the robot. After that, Agimus creates SoT real-time controllers for the graph's transitions that will be passed through during the manipulation experiment. Finally, an automatically generated supervisor handles the execution of the motions, switching controllers as needed.

This section presents the various packages forming Agimus, and details how they are contributing to the steps mentioned above.

### **Agimus**

This package implements the user interface of the framework. For the technical details, it provides plugins for the **Gepetto viewer**, the Graphical User Interface used by HPP. This allows the user to manage all aspects of the experiments: graph of constraints, estimation of the initial state of the system, manipulation planning, and finally the execution, from a single graphical interface.

### **Agimus-Vision**

This part of the framework deals with the localisation of 2D markers in the environment of the robot, using its camera. It is mainly a wrapper around the ViSP software suite, presented in section 3.3.1.

While at first the goal was to locate simple objects by direct recognition of their geometric features, it quickly appeared that the tuning of the detector was not trivial and the precision was not consistent depending on the ambient luminosity of the experimentation's room. For those reasons, we migrated towards a simpler and more reliable system, based on AprilTags markers, at the cost of having to place them on the objects manipulated by the robot. Since our contributions are in the domain of planning and controllers' generation, and not in the field of computer vision, that solution was perfectly acceptable for us.

The package is made of a ROS node that captures the video stream of the robot's camera, detects and locates AprilTags, and publishes their poses relative to the robot. The poses are published over ROS transform library 2 (TF2) [197] topics, for a better integration with ROS and its tools. In this package, there are also some tools for the intrinsic and extrinsic calibration of the camera.

### **Agimus-HPP**

This is the bridge between the Agimus software and the motion planner HPP. It provides the required adaptation to make the two software work together.

Indeed, there is at first the trouble of the different middleware used by those two programs. On one side, HPP uses CORBA [198], while Agimus is heavily relying on the ROS messaging system. Agimus-HPP handles the translation of the commands and information between those two systems.

Secondly, HPP's motion paths are continuous, but the robot awaits discretised commands at a frequency of 2kHz. This software package handles the sampling of the path before the transmission to the controllers. During this step, it also calls the Pinocchio library to add some information needed by the controllers but not

directly available in HPP, which only returns the configurations of the robot along the path. For example, the positions of the CoM and important parts of the robot, such as the hands, are computed and streamed during this stage of the process.

Finally, this part of the software hosts the **Python** scripts that conduct the estimation of the initial state of the robot and its environment. As seen in section 3.2.2, this is an essential step to make the information compatible with the graph of constraints and the motion planner. In order to do this, the raw state returned by the robot's sensors is projected into the initial state of the problem, and then a second step of optimisation is performed to reduce the error between the theoretical model of the robot and the sensors' measurements.

### Agimus-SoT

In this package is the main contribution brought by the Agimus framework, which is the automated generation of the real-time controllers corresponding to the planned motions, and their subsequent execution. This part of Agimus has two roles.

The first goal is the automated formulation of the SoT's controllers, matching the constraints applying to the transitions in the graph of constraints. The second is the building of a FSM that handles the switching between those controllers and the reactions to internal or external events during the execution. The concepts underlying those objectives were discussed in section 3.2.5, and here we only deal with the technical side.

The controllers generation is handled by the **factory**. It is not *stricto sensu*, as presented in the concept, a script that reads the graph of constraints to formulate the controllers, but a script that inherits from the graph builder in HPP and modifies some functions to build the controllers instead of a graph. Eventually, the result is the same.

The tasks are defined manually as classes in **Python**. They are made of several **Dynamic Graph** components linked together. There are also metadata to inform the **factory** of the diverse inputs and outputs needed by those tasks. During the pseudo graph building, this script instanciates the classes automatically, adds them to one SoT per transition, and links the inputs and outputs as needed. These input-s/outputs are automatically linked to the associated source of data either internally, via the **Dynamic Graph**, or externally, using the ROS messaging system.

The tasks<sup>1</sup> that appear in our experiments are:

- **CoM**, which controls the pose of the Center of Mass of the robot
- **Foot**, which controls the pose of a foot of the robot
- **End\_effector**, controlling the pose of the grippers, and also the gripper<sup>2</sup>. The gripper control is using either a pure position control, generally for simulation or empty handed tests, or a force control, to effectively grab an object. Indeed, in the first implementation, we used only the position control, knowing the size of our object. However, it often led to one of two problems. In some cases the gripper was slightly too largely opened and the object slid, sometimes

---

<sup>1</sup>Code folder found at [https://github.com/agimus/agimus-sot/tree/master/src/agimus\\_sot/task](https://github.com/agimus/agimus-sot/tree/master/src/agimus_sot/task)

<sup>2</sup>Code related to the gripper's control: [https://github.com/agimus/agimus-sot/tree/master/src/agimus\\_sot/control](https://github.com/agimus/agimus-sot/tree/master/src/agimus_sot/control)

falling out of the gripper. In other cases, with a gripper more firmly closed, the motors were excessively loaded and the robot's securities were triggered, stopping the motor, thus releasing the object. Therefore, a force control in admittance was added, using the current sensors of the motors, because the grippers are one of the few joints of our robot not equipped with direct torque sensing.

- **Grasp**, detailed in section 3.2.1, a special task to handle the grasping of the object on the table or during the handover. It was added to handle the addition of visual servoing to the framework, which is discussed in chapter 5. Its job is mostly to ensure that the second gripper is correctly aligned with the object, even if the latter has slid in the first gripper during the preceding motions.
- **Posture**, the lowest importance task that follows the guide path computed by the motion planner. Without perturbation, all the tasks should be compatible and therefore the robot gladly follows this guide. However, in the presence of perturbations, the hierarchy prevails and the robot ensures its equilibrium, then the following of its visual inputs, and finally, it tries to follow the guide path. This task acts as a regularisation term in the control's quadratic program.
- Tasks dealing with visual servoing schemes are introduced in the chapter 5.

The supervision of the whole system is done by the aptly named **supervisor**, which emulates a FSM. It is this component that selects the SoTs during execution, and connects them to the rest of the system. Its most important job is to check if the next SoT is compatible with the previous one, *i.e.* that the commands that would be sent to the robot are nearly the same, to avoid discontinuities that create jerk and are harmful to the mechanics of the robot.

The **supervisor** receives orders from the user interface, presented at the beginning of this section, but also from some events, such as the grippers' current sensors or the arm torque sensors. At the moment of experimentation, the orders were limited to user's authorisations to proceed with the next action, in order to allow some time to check if everything is going smoothly. Further envisioned extensions are discussed at the end of this chapter.

### **Agimus-Demos**

This package hosts the specific instances of the experiments, which consist mostly of configuration files. In a typical demo folder<sup>3</sup>, we have:

- The URDF descriptions of the objects involved in the experiment, fixed or movable, along with the handles' position to grab them.
- The description of the robot, comprised of its own URDF file and additional information about the grippers' capabilities, such as pre-grasps positions, to improve the speed of the manipulation planning step.
- A rough initial state of the world, that is the basis for the later estimation by vision and on-board sensors.

---

<sup>3</sup>[https://github.com/agimus/agimus-demos/tree/master/talos/manipulate\\_boxes](https://github.com/agimus/agimus-demos/tree/master/talos/manipulate_boxes)

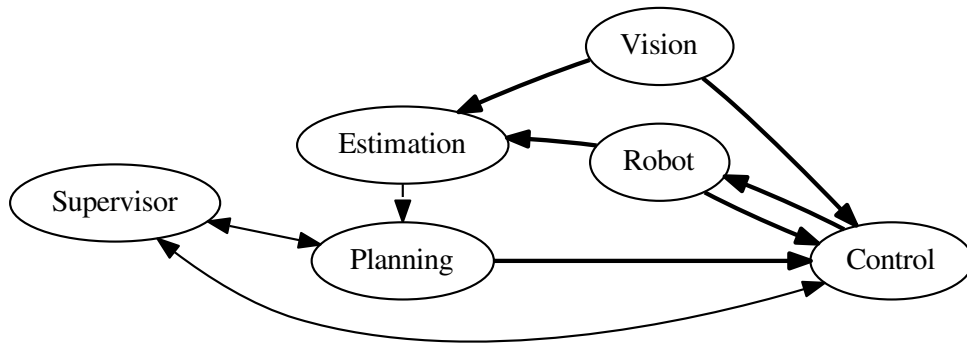


Figure 3.9: The architecture and interaction between the components of Agimus. Single arrows means that a component provides information to another. Double arrows are interacting directly, conversely to the two arrows between control and the robot, that represents an indirect communication via the robot’s low level software.

- A script describing the goal of the experiment, and, at the moment of experimentation, providing shortcuts for the motion planning and SoTs creation steps, by removing unreachable states or dead-end transitions. Those information should be computed automatically, but the goal of our contribution is not about optimisation of motion planning problems, so we left it as is, in order to abstain from increasing the complexity of the overall framework.
- Some experiment-specific files and algorithms, such as a **travelling salesman** solver for the demonstration that are presented in the chapter 5.

Most of those files are used indifferently in simulation or on the real robot, except for the physical aspect of gripping an object, which has to be simulated due to the unreliable behaviour of the physics engine of the simulator when dealing with plane-on-plane contacts.

The fig. 3.9 summarises the different part of the Agimus framework and the interactions between them.

### 3.3.7 Conclusion

In this section were presented the various pieces of software at the core of the Agimus framework. Integrating all those software, with their different means of formulating constraints or tasks, was a difficult and time-consuming part of the work of this thesis. But now that the concepts have been explained and the software presented, it is time to go onto the experiment that validated our approach.

## 3.4 Experiments

In this section is presented the demonstration that was performed to validate the generation of the controllers using the Agimus framework, and to confirm the interest in pursuing the research in this direction. First, there is a thorough introduction to the robot TALOS, accompanied by some comparisons against other humanoid

robots. After that, the course of the experiment is described, along with a typical solution provided by the framework. Finally, the downsides of this first demonstration are discussed.

### 3.4.1 The TALOS humanoid robot

The TALOS robot, fig. 3.10, has been developed by the Spanish company PAL Robotics, based on the requirements [9] provided by the Gepetto team from the LAAS-CNRS in Toulouse, France.

It is a 1.75 m humanoid robot weighing around 100 kg, with two 6 DoFs legs, a 2 DoFs waist, two 7 DoFs arms each with a 1 DoF gripper and a 2 DoFs head. It has been designed primarily for dexterous bi-handed manipulation. As such, the arms can lift up to 6 kg fully stretched and the reachable workspace in front of the robot is maximised. TALOS is also capable of walking at a mean speed of  $10 \text{ cm s}^{-1}$  during 1.5 h.

It is equipped with high-precision magnetic encoders, temperature and currents sensors for each of those articulations, coupled with torque sensors in all of them except the head's and grippers. As usual for such robots, an Inertial Measurement Unit (IMU) provides a 1 kHz filtered measure of the orientation, angular speeds and linear accelerations of the torso in which it is installed. By default, a RGB-D Orbbec Astra camera is mounted in the head. Finally, in the ankles and wrists, there are 6 axis force-torque sensors.

An EtherCAT bus [199] connects all those actuators and sensors to a control computer, running a Linux Ubuntu operating system patched with RT-Preempt, that add hard real-time capabilities to the system. All this allows the control loop to run at 2 kHz, with a theoretical bandwidth margin to go up to 5 kHz to provide the shortest reaction time necessary for the execution of dynamic movements. Because our controllers are only able to be run at 1 kHz, we interpolate the commands before sending them to the robot. The software stack is based around the well-known middleware ROS.

There exists multiple control back-ends provided either by the manufacturer or by the academic users of the robot. A second computer, also running Ubuntu, but



Figure 3.10: TALOS humanoid robot developed by PAL Robotics



without the real-time patch, handles the data coming from the high bandwidth sensors such as cameras and laser rangefinders, and the software that do not need the real-time features of the control computer. Off-the-shelf, the planning of the robot’s movements is assured by MoveIt! [200], and a simulation framework based on Gazebo [201] is provided by PAL Robotics.

Our TALOS, named Pyrène, was the first of its kind, and therefore had some shortcomings due to its novelty, especially a bad upper body calibration, which is described more in-depth in the chapter 4. The head has lately been modified to include two Intel RealSense cameras, a T265 oriented forwards, and a D435i, oriented towards the ground to detect obstacles. The main sensor in the head is an Ouster OS1-64 LiDAR, which provides a 3D visualisation of the world around the robot up to 120 m. The software stack we use is also different from the stock TALOS. During this thesis, part of the control loop was handled using the SoT, presented in section 3.3.4, while HPP, section 3.3.3, provided the motion planning, using information gathered with the vision system and treated by ViSP.

### 3.4.1.1 Position against other humanoid robots

TALOS was specified by the Gepetto team as a successor for its aging humanoid robot, HRP-2, which was in use since 2006. The requirements written by the Gepetto’s members were based on the feedbacks from the team participating in the DARPA Robotics Challenge (DRC). The DRC was created by the Defense Advanced Research Projects Agency (DARPA), an entity of the United States of America’s army. Hold between 2012 and 2015, its goal was “to develop ground robots capable of executing complex tasks in dangerous, degraded, human-engineered environment”[202]. Because the aim was to navigate inside environments adapted to the human morphology, the majority of the teams involved in the challenge used humanoid robots.



Figure 3.11: Some of the humanoid robots used in the DRC

Many participants used older robotics platforms as a basis, with updates to be able to handle the proposed tasks. HRP-2, originally conceived in 2003, became

	HRP-2 Kai [203]	DRC-HUBO+ [204]	Atlas 2016 version [205]	TALOS [9]	WALK-MAN 2018 version [206]
Height	1.71 m	1.70 m	1.50 m	1.75 m	1.915 m
Width	0.63 m	0.59 m	0.65 m	0.55 m to 0.78 m	0.815 m
Depth	0.36 m	0.35 m	0.48 m	0.330 m	0.600 m
Weight	65 kg	80 kg	75 kg	95 kg	132 kg
Total DoFs	32	32	28	32	33
Head	2 DoFs	1 DoF	1 DoF	2 DoFs	2 DoFs
Arms	2×7 DoFs	2×7 DoFs	2×6 DoFs	2×7 DoFs	2×7 DoFs
Hands	2×1 DoF	2×1 DoF	No	2×1 DoF	2×1 DoF
Waist	2 DoFs	1 DoF	3 DoFs	2 DoFs	3 DoFs
Legs	2×6 DoFs	2×6 DoFs	2×6 DoFs	2×6 DoFs	2×6 DoFs
Wheels	No	2×1 DoF	No	No	No

Table 3.1: Description of some robots used in the DRC

HRP-2 Kai, fig. 3.11a, a taller version, with an additional DoF on each arm to improve the manipulation capabilities of the robot. DRC-Hubo, fig. 3.11b, winner of the challenge, is also a taller version of the Hubo lineage. Its originality came from its ability to transform between a wheeled position, faster and safer on the ground, and a walking position, essential to climb stairs.

The DARPA also provided some teams, which were successful in the virtual parts of the challenge, with Atlas humanoid robots, fig. 3.11c, made by the company Boston Dynamics. This 1.88 m and 150 kg robot used hydraulic power instead of electrical actuation, but required to be tethered to an outside source of power.

Following the challenge, which had led to the publications of many papers giving feedbacks about design choices, many research team updated again their robots to pursue further advances, while other teams, such as Gepetto, designed new robots from the ground up, in order to benefit from the experience and the progress of key technologies, like motors' and computers'. The new Atlas, fig. 3.12a, presented in 2016, is smaller and lighter, at 82 kg for a height of 1.75 m. It lacks the hands of its predecessor, but keeps the hydraulic actuation while gaining autonomy with an on-board battery pack as power source. Those changes are consistent with the new research focus of Boston Dynamics, which have shifted toward highly dynamic motions instead of manipulation.

The robot WALK-MAN, which took part in the DRC, was also updated using feedbacks from its performance during the challenge. The new version, fig. 3.12c, presented in 2018, is both stronger and lighter than the DRC's version, thanks to the use of aluminium and titanium alloys for its frame, instead of steel. Its computational abilities were also upgraded to increase its capacity to perform actions autonomously. Indeed, the original European project was to design a robot mainly teleoperated, and increase the autonomy over time.

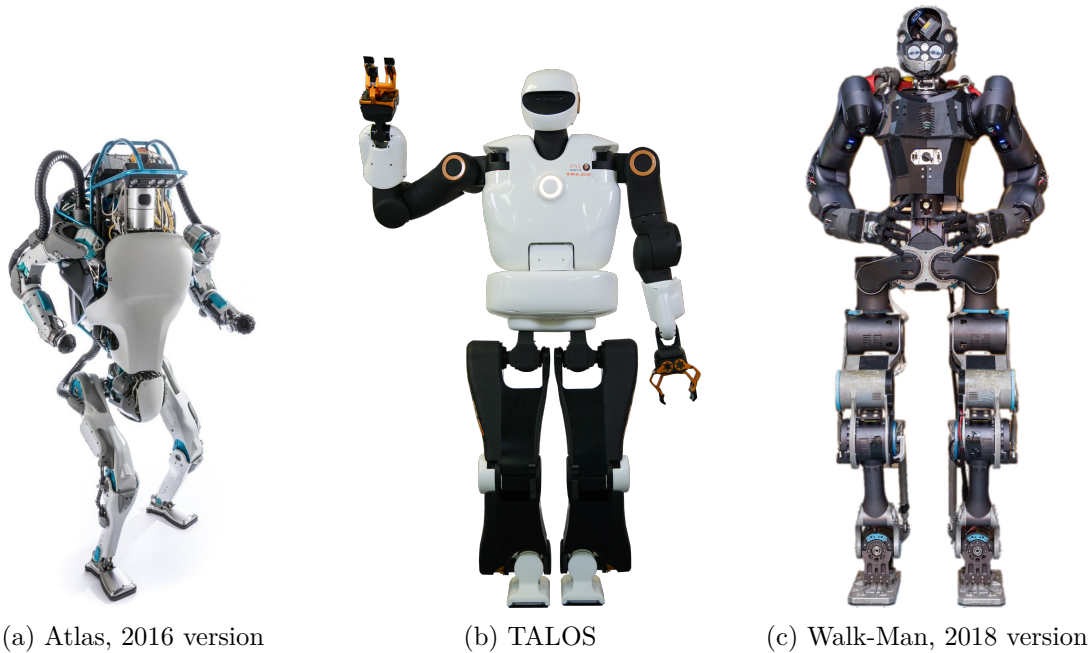


Figure 3.12: Humanoid robots in the years following the DRC

### 3.4.2 Course of the experiment

Although our work is general and applies on instances of manipulation problems with several robots and several objects, we illustrated our framework on a simpler example where a TALOS robot is requested to grab a wooden plank lying on a table, turn it upside-down, then place it back on the table.

#### Description of the setup

Before the start of our framework, the robot is positioned in its stand-up configuration, in front of a table, on which two AprilTags are fixed at known coordinates. On this table, we placed a plank of wood, also decorated with some AprilTags, on one of its largest faces.

The exact positions of the table relative to the robot is not known, and the plank of wood is deliberately randomly placed on the table in order to exhibit the interest of the estimation step, based on the localisation of the AprilTags markers. Indeed, for other experiments done by the team, the positions of both the robots and the objects manipulated were enforced, which necessitates careful measurements, taking valuable time and increasing the risk of failure if they were slightly off.

#### Inputs of the problem

In the previous section, we presented a typical demonstration folder containing the configuration files that describe the experiment to perform. Those files are the only inputs given to the framework, and their content is recalled here for clarity:

- The model of the robot and information about its grippers.
- Data about the environment: models and rough positions of the objects, static or movable, and information about their handles for the latter.

- A goal, which is here to turn the plank upside-down from its initially estimated orientation (the goal is therefore not fixed but depends on the placement of the plank during the estimation step).

### Construction of the graph of constraints

The graph of constraints of the experiment is automatically built from the information regarding the grippers and the objects' handles. The states in the graph are linked to the **grasps**, explained in section 3.2.1.

We designate those states using the gripper-handle pairs:  $((g_1, h_1), (g_2, h_2))$  where  $g_i$  and  $g_2$  are the TALOS grippers, and  $h_1, h_2$  take values in  $(H_1, H_2, H_3, H_4, \text{None})$ , and represent the handles of the plank, illustrated previously in fig. 3.2.  $h_i = \text{None}, i = 1, 2$  means that gripper  $i$  does not grasp anything.

Thus the theoretical number of states is  $5 * 5 = 25$ . However we only considered a smaller number of states, since some combinations of grasps are always in collision. It is obvious that the two grippers cannot grab the same handle at the same time, nor can they grip the rotated handle on the same side of the box. A simple pruning thus reduces the number of reachable states to 17:

- 5 for  $((g_1, \text{None}), (g_2, h_2)), h_2 \in (H_1, H_2, H_3, H_4, \text{None})$
- 3 for  $((g_1, H_1), (g_2, h_2)), h_2 \in (H_3, H_4, \text{None})$
- 3 for  $((g_1, H_2), (g_2, h_2)), h_2 \in (H_3, H_4, \text{None})$
- 3 for  $((g_1, H_3), (g_2, h_2)), h_2 \in (H_1, H_2, \text{None})$
- 3 for  $((g_1, H_4), (g_2, h_2)), h_2 \in (H_1, H_2, \text{None})$

At the time of the experiment, those combinations were detected by a human operator and integrated into the script describing the experiment, but this detection might be done automatically. The user also provides some waypoints states to guide the motion planner and decrease the planning time, see section 3.2.1. That could also be automated for such a simple object, with waypoints always positioned some 15 cm above the handle of the object. However, for more complex objects, like a drill, for example, this would require further research to reliably automatise.

### Initial state estimation

Now that the graph of constraints has been computed, the framework uses the sensors of the robot to estimate the initial state of the system. Moreover, neither the exact position of the table nor that of the plank is known before this step, whereas those information are needed by the manipulation planner. The file describing the demonstration only provides a rough estimate of where to find those objects relative to the robot expected starting position.

The estimation of those positions is done by the computer vision system of *Agimus* which detects and computes the poses of the AprilTags on the table and the plank. The relative transformation between each tag and the corresponding object is provided in the script describing the experiment. The rest of the system is the robot, which is able to compute its overall state via the measurements of its encoders in each joints of its body. The resulting estimation is then compared to the constraints of the states of the graph of constraints in order to find the one closest

to the current situation. Finally, the raw estimated state of the robot and the world around it is projected onto the selected state to provide the mathematically sound initial situation that the motion planner requires.

### **Precision about the human operator**

From the start to the end of the experiment, the robot is **nearly** autonomous. Indeed, all the controllers are running on its on-board control computer, and, in order to demonstrate the efficiency of our framework, the rest of Agimus, including the motion planner, is operating on the second on-board computer.

The "**nearly**" part comes from the supervision offered by the human operator. Its first intervention happens at the end of the estimation process, to validate that the situation seen by the robot and the initial state selected in the graph are indeed representing the real scene. Secondly, the motion planner uses a variant of the RRT algorithm, which samples random configurations. This can lead to movements achieving the goal, but with weird detours that increase the overall duration of the experiment, thus increasing the risk of something going amiss.

Therefore, the operator validates the movement proposed by the planner, or asks for further optimisation or a totally new plan. Then, they are manually ordering the robot to proceed with each step of the experiment, in order to have some time to check beforehand that the next movement will not break the system, because this robot is quite costly. These steps are not necessary for our framework to work, but the cost of a real robot is too high not to stay cautious.

### **Manipulation planning**

At this moment in the experiment, we have the initial state of the experiment, the goal state, and a graph of constraints whose connectivity will guide the manipulation planner. Provided a solution exists, the planner returns it as a sequence of elementary paths. An elementary path belongs to a single transition in the graph of constraints, and therefore will be subject to the same controller from start to end. The human operator may choose to redo this step if the complete planned path could endanger the robot.

### **Controllers generation**

Using the information contained in the graph of constraints, and the solution provided by the manipulation planner, Agimus formulates a SoT for each transition of the graph passed through during the experiment. This part of the algorithm has been seen in details in section 3.2.5 and section 3.3.6.

After those steps, the situation is as follows:

- The robot is standing still in front of the table, upon which the wooden plank lies untouched.
- The graph of constraints has been created and the initial state determined automatically via the use of the robot's sensors.
- The manipulation planner has found a solution to reach the goal.
- Agimus has formulated the controllers to execute this solution.

### 3.4.3 Presentation of a typical run

As the motion planner is based on RRT, there is no guarantee that the robot will find the same solution from one instance of the problem to another. The constraints and the intermediate waypoints, however, provide a way to structure the solution found by the robot.

A typical solution is provided in fig. 3.13, and explained thereafter. Unless stated otherwise, the controllers are using the planned motions as references.

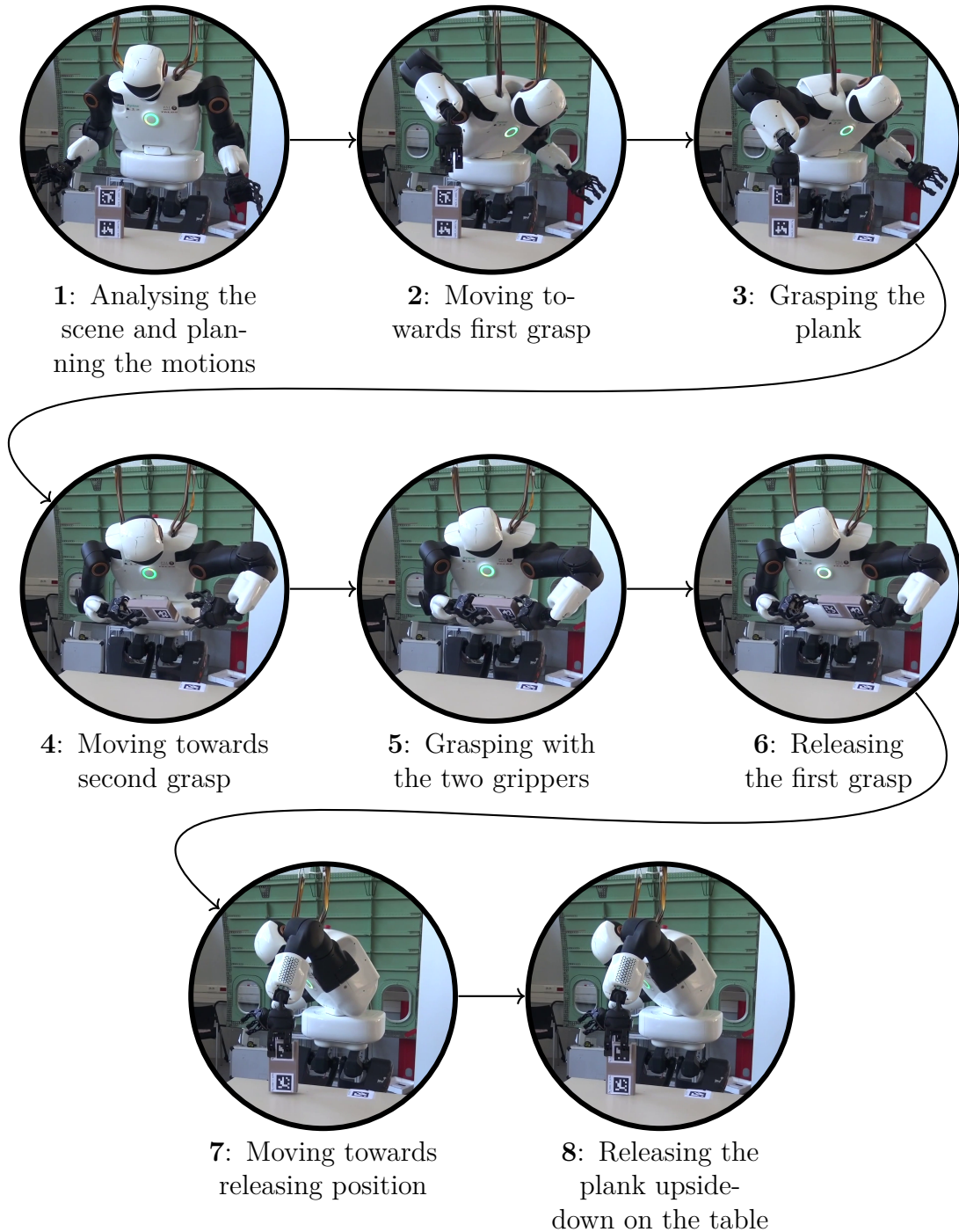


Figure 3.13: A typical run of the plank manipulation experiment

1. The robot goes to its offline-planned initial position. It is position controlled using the manufacturer software. It uses its cameras and encoders to determine the situation and then plan the motions.

**Active Agimus controllers:**

None.

2. Waypoint with the gripper above the first handle of the plank. This motion is not using visual servoing but only following blindly the plan computed at the previous step.

**Active Agimus controllers:**

- CoM position, left and right feet positions
- Right gripper position
- Posture task

3. Grasp of the plank using the right gripper.

**Active Agimus controllers:**

- CoM position, left and right feet positions
- Right gripper position
- Right gripper admittance control, to maintain a 5 N gripping force
- Posture task

4. Left gripper at the waypoint to grab the second handle of the plank.

**Active Agimus controllers:**

- CoM position, left and right feet positions
- Right gripper position
- Left gripper position, relative to the measured position of the plank
- Right gripper admittance control, to maintain a 5 N gripping force
- Posture task

5. Grasp of the plank using both grippers.

**Active Agimus controllers:**

- CoM position, left and right feet positions
- Right gripper position
- Left gripper position, relative to the measured position of the plank
- Right gripper admittance control, to maintain a 5 N gripping force
- Left gripper admittance control
- Posture task

6. Release of the plank by the right gripper. Note that the right gripper's position is now relative to the plank's position, instead of the left gripper, as in the previous steps.

**Active Agimus controllers:**

- CoM position, left and right feet positions
- Left gripper position
- Right gripper position, relative to the measured position of the plank
- Left gripper admittance control, to maintain a 5 N gripping force
- Posture task

7. Waypoint above the release position of the now bottom-up plank.

**Active Agimus controllers:**

- CoM position, left and right feet positions
- Left gripper position
- Left gripper admittance control, to maintain a 5 N gripping force
- Posture task

8. Finally, the plank is placed down on the table by the robot, upside-down, as requested.

**Active Agimus controllers:**

- CoM position, left and right feet positions
- Left gripper position
- Posture task

The robot was able to perform this experiment multiple times with a rate of success of around 80%. The failures mainly came from the sliding of the plank from the gripper, which prevented the second grasp from happening. A video is available at <https://peertube.laas.fr/videos/watch/c76b5e69-41b5-4096-847b-7df9f52da072>. In this video, it can be seen that the robot is greatly overshooting its commands. This was due to a low-level bug (forcing a non-zero speed at the goal position) on the robotic platform that could not be solved by the manufacturer before this video was due for the presentation at a conference. Since it was not impacting our experiment, we decided to proceed and explain the bug instead of trying to poorly minimise it by slowing down the robot's speed.

### 3.4.4 Discussion

The first observation that can be made is that our framework allowed the successful execution of the experiment. While it might seem that we are using a very complex machinery for a simple bi-manipulation task, let us not forget the demonstrated contributions that underlie *Agimus*. Those main additions to the pre-existing software are the estimation of the initial state of the robot and environment, and the automated generation of the controllers and the associated FSM to handle the execution of the whole experiment.

First, we have added an estimation system to the pre-existing concept of the graph of constraints. This extension of the manipulation planner is quite useful,



because until this, one had to rely upon the reliable placement of the manipulated objects. It was achieved either manually by means of measurements, or in a more industrial setup, by constraining all the manipulated parts physically. One of the goals of the joint laboratory supporting this work being the cooperation between robots and humans, the ability to handle the little chaos coming from small misplacement is necessary.

The second point, the generation of the controllers, is also important, especially for the future of our work. Indeed, other teams have been linking controllers to part of a planned configuration path, in order to execute it robustly, and using FSM to orchestrate the switching between controllers. However, at the time this work was submitted, we did not find another work performing **automated** generation of the controllers or the associated FSM. The versatility of our approach became visible when we had to add admittance control to the gripper. The formulation of the task itself was done manually, but the rest of the system, especially the generation of the controllers, stayed the same and the experiment was improved quickly.

But there were also downsides to our framework that we did not anticipate enough.

#### **3.4.4.1 Failures' detection**

In its current form the system does not track failures. For instance it happened during the first tests that in some configurations the box slid from the robot gripper. This has been partly fixed by adding rubber to the grippers to increase the friction coefficient while holding the object. A more sound solution was the addition of the admittance control to take care of the gripper's clamping force. Nevertheless, sliding still occurred in some cases.

Detecting such changes with the help of computer vision would allow the robot to re-plan the remaining movements if necessary. However, recovery from complete failure may lead to complex solutions which are not feasible. For example if the plank falls on the floor, the situation involves a more complex motion such as kneeling down. Finding this would involve computationally intensive searches for the current system, and would be very risky to execute on the real robot. In this case, it is easier to ask for the help of an operator.

#### **3.4.4.2 Limitations caused by the poor kinematic calibration of TALOS**

During the first wave of experiments, we were not having too much trouble with the robot. It was able to reach for the plank almost all the time, and only the sliding of the plank in the gripper was concerning, as mentioned in the previous paragraph. However, after a human error during an experiment, the robot had to be sent back to its manufacturer for repairs. When it came back, the gripper missed the plank nearly every-time.

We firstly thought that the calibration of the camera was now incorrect, or that the eye-to-hand parameters were reset during the repairs. A new calibration of the camera was performed using state-of-the-art software dedicated to the task. This did not improve the success rate of the experiment.

Measurements between the camera and the centre of the detected AprilTags

showed that the computed pose of the object was correct with a precision of around 1 cm. As we sometimes saw gap of nearly 10 cm between the gripper's position and the object, it was clearly not the principal cause of the observed problem of precision. Further tests were able to demonstrate that the robot was incorrectly reporting the position of its grippers relative to its floating base.

The chain of events was as such as follows:

- The camera locates the AprilTags on the object, and reports its pose relative to its local frame within a centimetre of precision.
- The pose of the object relative to the base of the robot is computed, using the values of the encoders in the chain of articulation between the base and the camera frame. This introduces small errors because the chain is short and not too badly calibrated.
- Our software uses the robot model and information coming from the torso's and arm's encoders to compute the pose of the gripper and move it toward the object. The determination of the gripper's pose is subject to large errors, due to the arm poor calibration and the length of the kinematic chain between the base of the robot and the gripper (8 joints).

In some rare situations, the robot was able to catch the object because the position of the arm induced a compensation between some errors in the gripper's pose estimation. In other situations, the errors were accumulating, leading to a final error of around 10 cm, the same order of magnitude that we observed experimentally.

Two axis were explored simultaneously to prevent this behaviour from occurring again. On one hand, the chapter 4 explains in-depth our investigation, propositions and finally the solution found to get a better calibration of the arms of our TALOS humanoid robot. On the other hand, the chapter 5 deals with the addition of visual servoing before grasping an object, in order to improve the precision of the manipulation.

## 3.5 Conclusion

In this chapter was presented the first contribution of my thesis. It is a framework named *Agimus* that assembles together a manipulation planner and a real-time controllers executable on a real humanoid robot. The major additions to those two existing software are a process of initial state estimation for the robot and its environment, and a process to automatically generate the controllers using the information available to the manipulation planner. As such, a user should be able to quickly create experiments involving the manipulation of objects that do not have to be placed precisely, providing that the robot is still able to see them.

During this chapter were presented the concepts surrounding the framework, then the software implementing them, some of which existed beforehand, and some that were made for the demonstration. Then, a demonstration performed by a real humanoid robot TALOS was described. Finally, we discussed the upsides and downsides of our approach and the teaching learnt from the experiments.

The original publication of this work is:

**Alexis Nicolin**, Joseph Mirabel, Sébastien Boria, Olivier Stasse, and Florent Lamiraux. “Agimus: A new framework for mapping manipulation motion plans to sequences of hierarchical task-based controllers”. In: *2020 IEEE/SICE International Symposium on System Integration (SII)*. IEEE. 2020, pp. 1022–1027

# Chapter 4

## Calibration

### Contents

---

4.1	Introduction . . . . .	76
4.2	Calibration using HPP's Newton-Raphson projection .	77
4.3	Calibration as several manipulator arms . . . . .	80
4.4	Whole-body elasto-geometric calibration of a TALOS robot . . . . .	82
4.5	Conclusion . . . . .	83

---

## 4.1 Introduction

In the previous chapter, we ended on a bitter note about our experiment where a TALOS robot was asked to turn a plank upside-down. Following a maintenance, the robot could no longer grab the plank properly. Before its reparations it was able to seize the plank on the table with a good success rate. But after its return, the TALOS robot was only able to catch it correctly on rare occasions. Often, the plank and the gripper were separated by a clearly visible distance, of up to 10 cm.

The first sources of error we suspected were the camera and the computer vision software tasked with detecting and estimating the poses of the AprilTags used in our experiments. We calibrated the camera’s intrinsic parameters, to no avail, then the extrinsic parameters, *i.e.* the position of the camera relative to the robot’s frame of origin. The results were worse, and coarse measurements of the distance and angles between the camera lens and the markers showed that the reported data were coherent with the reality. The vision was therefore not the cause of our issues.

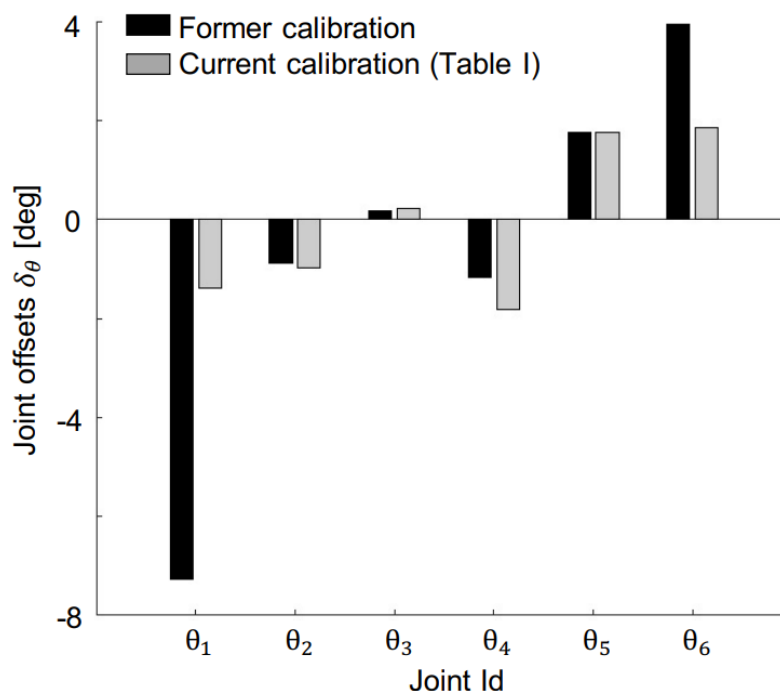


Figure 4.1: Comparison of the joints’ offsets in our TALOS right arm before and after the calibration process. Graph kindly provided by the authors of [207]

During a colleague experiment, we noticed that the first joint of the arm, at the level of the shoulder, along the vertical axis, was visibly shifted. The angle was later shown to be about  $7^\circ$  off [207], with another large offset of  $4^\circ$  for a joint of the wrist. A graph of the offsets of the joints of the right arm, before and after calibration, has been kindly provided by Bonnet et al., fig. 4.1. Those defects were overlooked during our experiments because we use a different initial configuration for the robot, where its shoulders are angled towards the front, making the misalignment less likely to be noted. Our colleagues, doing walking experiments, were aligning the shoulders with the chest of the robot, making the shift unmistakable. We also later tested the

robot by commanding the grippers to face each other. They were not aligned at all, as they should have been, meaning that the two arms had different offsets.

The origin of our troubles having been identified, it was clear that the robot needed to be calibrated in order to reconcile the computer model and the reality. The manufacturer was aware of some misalignment issues on the arms of the robot, although offsets of this magnitude rarely went unnoticed before the robots were sent back to their users. We learnt during our discussions with them that there were no definite calibration method for the arms of the robots. While the legs were assembled and adjusted with their parts held in clamps, the arms were too fragile for the procedure. So they were placed on the body and then aligned manually by a human operator.

Obviously, this approach was successful enough that we did not complain about the precision of the robot during our first round of experiments. However, such a simple procedure would not yield a calibration good enough for the drilling operations that were the final goals of the Airbus-LAAS joint laboratory ROB4FAM. We then decided to use the new Motion Capture system (MoCap) available to our laboratory team to calibrate the robot ourselves, to the required precision.

In this short chapter are presented the two approaches that I used at the start of my thesis in order to try to calibrate the TALOS humanoid robot. The first was based on the projection of the raw robot's configuration onto a HPP state, in the same way as the estimation step of Agimus, described in the section 3.2.2. The second method tried was based on the calibration of industrial manipulator arms. Those two attempts were fruitless, and this time-consuming work was passed to a more senior colleague already versed in the domain of kinematics and dynamics identification of robots' parameters. The last section of this chapter quickly describes his method, that is currently being submitted to a journal.

## 4.2 Calibration using HPP's Newton-Raphson projection

The first calibration method that was envisioned was based on the same algorithms as the Agimus framework. As a recall, during the estimation step of this framework, the robot takes the data coming from its on-board sensors in order to assess its own state and that of the surrounding world. Because no sensor is perfect, or in our case because the computer model of a robot sometimes do not reflect the reality well enough, this raw state of the world is projected onto the graph of constraints. This projection has two goals. It allows the system to figure out at which step of the demonstration it is, and therefore which steps still need completion. It is also used to obtain a mathematically sound representation of the state of the world, compatible with the manipulation planner that is used afterwards.

Our working hypothesis for this approach was the following: the bodies of the robot in the computer model are correct, and the joints' positions relative to those bodies are also accurate, but there is an offset angle for each of the revolute joints, that is incorrectly adjusted by the manufacturer. The method we devised was the following:

- Add a virtual revolute joint for each existing joint of the arm in the computer model. This virtual joint would measure the offset between the data of the real robot and its digital counterpart.
- Stick reflective markers onto the robot’s chest and grippers.
- Use a MoCap to measure the pose of the gripper relative to the chest, at several random configurations.
- Create in HPP one state for each configuration, using the commanded joint values as constraints. Those states are all contained into a larger state with parametrised constraints for the offsets we are seeking, and for the exact positions of the reflective markers fixed on the robot.
- We then project at the same time onto their respective state the recorded positions of the markers registered by the MoCap for each configuration.

By locking the joints of the robot in the model and asking the software to perform a projection, the only variables that can be used are the virtual joints that represent the offsets. We had 13 DoFs that were not locked, 7 for the arms’ offsets and  $2 \times 3$  for the positions of the markers on the chest and gripper. However, we measured only the 3 components of the relative positions between the gripper’s marker and the chest’s. Therefore, the projection method could not provide a unique solution had we used only one state and one measurement. That was why all those states were part of the larger parametrised state mentioned above. By projecting every measurement at once, the parameters, *i.e.* the offsets and the positions of the markers, would be the same for all the projections. The method was analogous to performing a least mean squared errors method.

This method required some modifications of HPP, especially for the simultaneous projections of the configurations. It was successfully tested on simulation data and the results were satisfying.

### Motion Capture system

A MoCap is a group of synchronised infrared cameras coupled with a dedicated software, that is able to accurately pinpoint the positions of reflective markers in 3 dimensions. It operates by first detecting and isolating the markers in the images, and then by triangulating their positions using multiple cameras calibrated to have overlapping fields of view. The MoCap we used was provided by Qualisys.

We can see some of the cameras in fig. 4.2, along with the TALOS humanoid robot equipped with reflective markers. The system is equipped with 20 Miquis M3 cameras that should be able to reach a 3D resolution of around 0.11 mm, up to a frequency of 340 Hz, in ideal conditions. It covers the whole experiment room of nearly  $5 \times 10$  m, over which the re-projection error was estimated by the software to be around 0.6 mm. This performance was confirmed by independent measurements performed using a laser tracker, itself accurate to 10  $\mu\text{m}$  to 50  $\mu\text{m}$ , *i.e.* an order of magnitude more precise.

Those characteristics were more than sufficient for our needs, and we limited the acquisition frequency to 100 Hz. In order to maximise the quality of the data for our measurements, each configuration was sampled for 10 s, and we kept the 1 s where the measurements were the most stable.

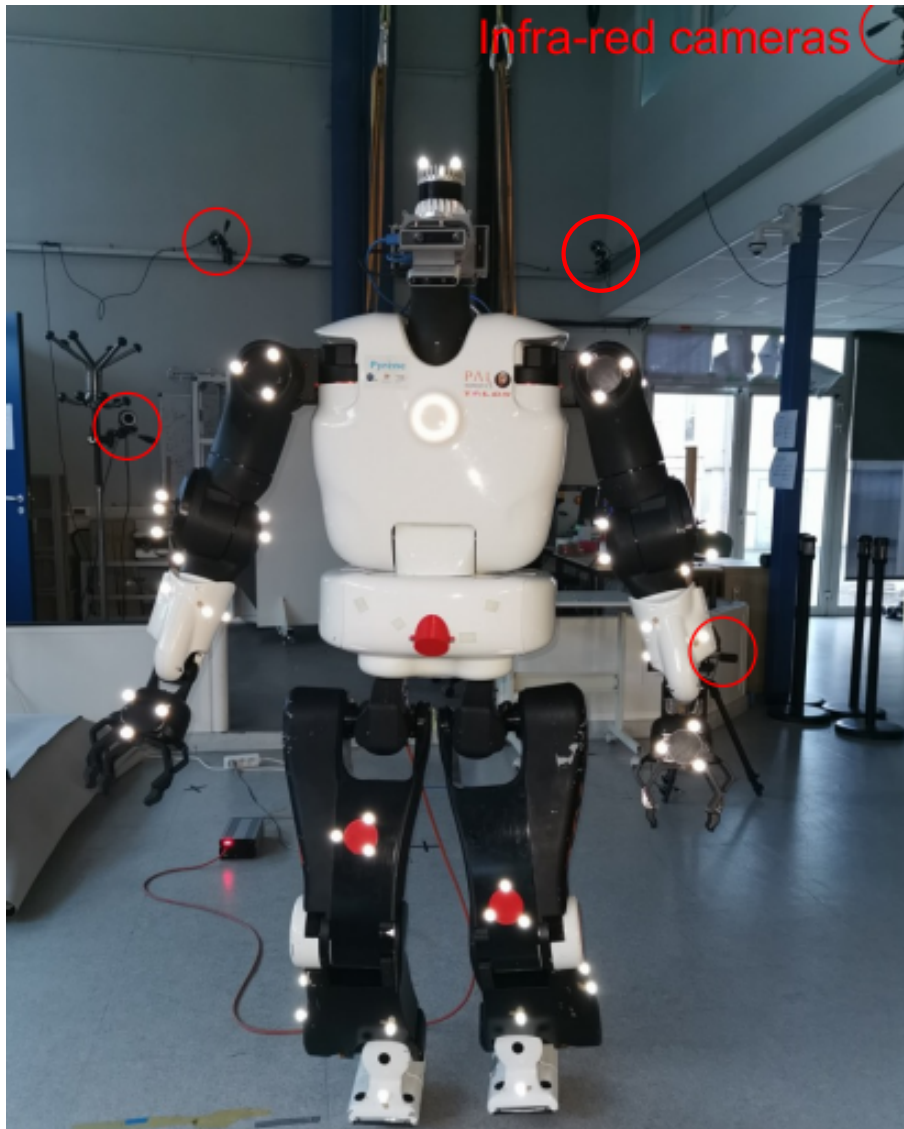


Figure 4.2: A TALOS humanoid robot equipped with IR reflective markers and surrounded by MoCap cameras

When this approach was carried out on the real humanoid robot, the results were less than convincing. The offsets computed by this method were wildly dependent on the subset of the whole dataset that we used. As we were taking subsets of more than 50 configurations randomly from a complete dataset of around 100, it should not have happened. Further investigations showed that the robot was less rigid than previously thought and documented. Therefore our hypothesis of simple offsets for the angles of the joints was not sufficient.

In order to continue in the direction of this method, we should have taken into account many other effects. Those would have needed an order of magnitude more sampled configurations to reach convergence for the calibration values, which was not tractable for HPP. A second option would have been to generate the configurations providing the most observability of those offsets, but it was outside our realm of competence. For all these reasons, this approach was abandoned for a more proven, but more naive calibration method, which is the subject of the next section.



### 4.3 Calibration as several manipulator arms

The second method aim was to be simple and effective. In the previous section, we were already trying to calibrate one arm from the chest up to the gripper, independently of the rest of the humanoid robot. Therefore, it seemed a good idea to try to implement calibration methods created for industrial manipulator arms. Indeed, each arm of TALOS is like a 7 DoFs manipulator grounded in the chest. In a similar manner, the legs are 6 DoFs serial articulated systems with the pelvis of the robot as their base.

The subject of manipulator arm calibration has been discussed for decades [208, 209] due to its importance to the industry. Industrial robots have always had a good repeatability, because their job is to repeat the same task continuously, but the accuracy was not always great. Those robots were historically programmed manually by human operators, who could see the result of their commands and compensate for the eventual errors. But this was a time-consuming process of trial and error, that was not desirable. Robots are increasingly being programmed by motion planners, or moving in reaction to real-time inputs. For those tasks to succeed, a good accuracy is necessary, and therefore a good calibration of the kinematic and sometimes dynamic parameters of the system is required.

#### Denavit-Hartenberg parameters

In robotics, the Denavit-Hartenberg parameters (DH parameters) [210] are four parameters and an associated convention to attach reference frames to the bodies of a kinematic structure. This is often used to define the pose of a body relative to its parent. While other conventions have been proposed over the years, this one and its modified version [211] are still the most cited and used to this day. In this convention, the transformation between the frames of two consecutive bodies is made of two screws.

#### Definition 9 — A screw

It is a transformation combining a rotation  $\theta$  around an axis with a translation  $d$  along the same axis. The order of those two elementary transformations does not matter, because:  $Trans_{axis}(d) \cdot Rot_{axis}(\theta) = Rot_{axis}(\theta) \cdot Trans_{axis}(d)$ .

The first screw is along the axis of the joint between the two consecutive bodies, conventionally named  $[Z]$ :

$$\begin{aligned} {}^{n-1}Z_n &= Rot_{z_{n-1}}(\theta_n) \cdot Trans_{z_{n-1}}(d_n) \\ &= \begin{bmatrix} \cos(\theta_n) & -\sin(\theta_n) & 0 & 0 \\ \sin(\theta_n) & \cos(\theta_n) & 0 & 0 \\ 0 & 0 & 1 & d_n \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \quad (4.1)$$

The second, named  $[X]$ , is along the  $\mathbf{x}$  axis of the new frame defined by the previous screw:

$$\begin{aligned} {}^{n-1}X_n &= Rot_{x_n}(\alpha_n) \cdot Trans_{x_n}(a_n) \\ &= \begin{bmatrix} 1 & 0 & 0 & a_n \\ 0 & \cos(\alpha_n) & -\sin(\alpha_n) & 0 \\ 0 & \sin(\alpha_n) & \cos(\alpha_n) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \quad (4.2)$$

It defines the new origin and the new reference frame for the next joint in the kinematic chain. The overall transformation is shown on fig. 4.3. The Denavit-Hartenberg parameters (DH parameters) [210] for each transformation are thus the four values:  $d, \theta, a, \alpha$ . Finally, for a serial robot, we have the equation:  $[T] = [Z_1][X_1][Z_2][X_2] \dots [Z_n][X_n]$  Where  $[T]$  is the transformation between the base of the robot and the  $n^{\text{th}}$  body.

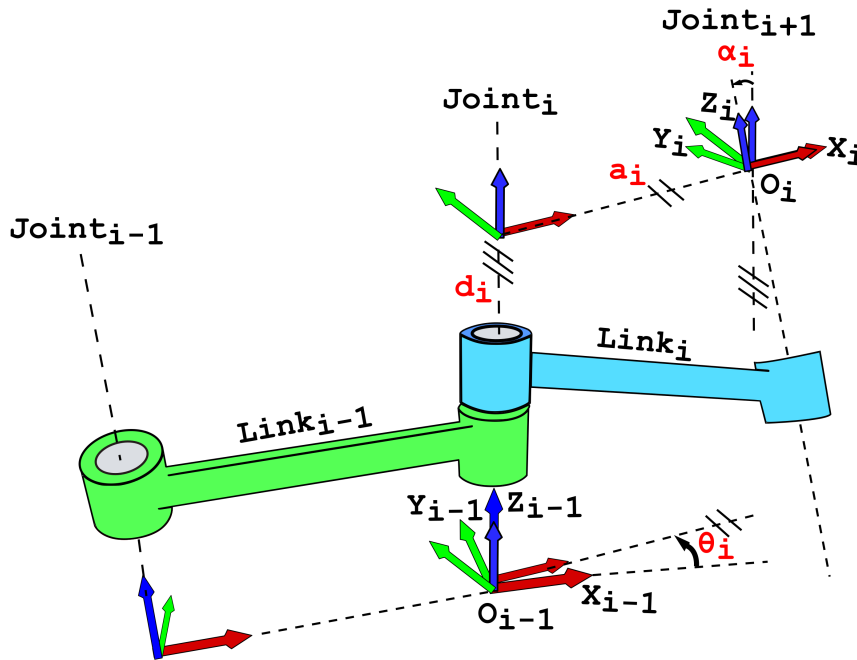
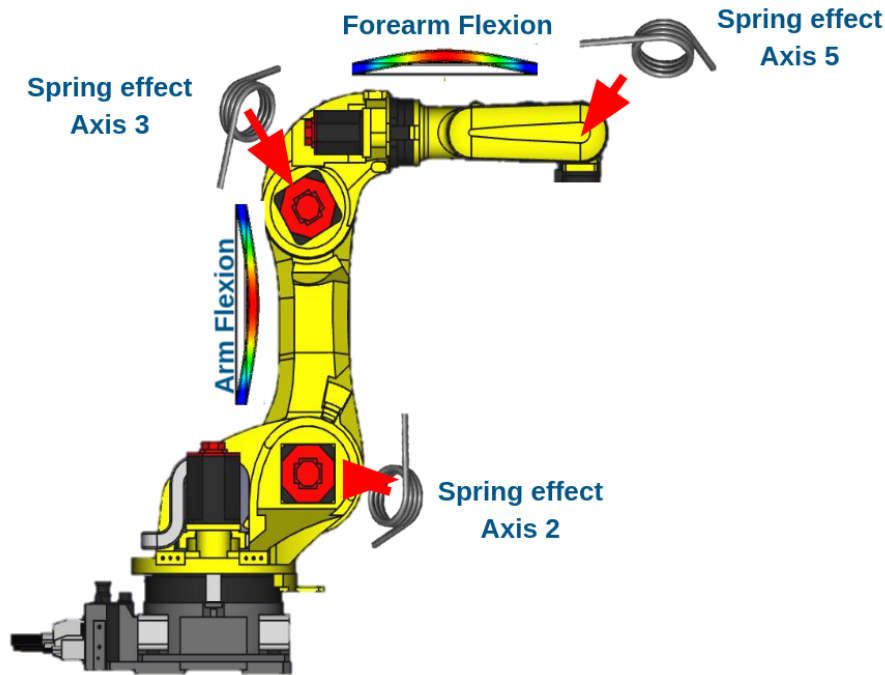


Figure 4.3: The four classical DH parameters, shown in red, to transform the frame  $i-1$  into frame  $i$

Using publicly available software dedicated to manipulator arms' calibration, we again tried to process our robot's data in order to compute those parameters for each joint. However, the flexibilities of our robot once again rendered the procedure difficult. Indeed, the software we chose was expecting a very rigid robot, like an industrial manipulator arm, and not a lightweight humanoid robot.

After this new attempt devoid of result, it was decided to pass this calibration project to a colleague whose speciality is the identification of the kinematic and dynamic parameters of robots.



©S. Boria, D. Van Damme, Airbus

Figure 4.4: The various deformations on an industrial robot.

## 4.4 Whole-body elasto-geometric calibration of a TALOS robot

This short section is not a part of my own work, but instead of other members of the team who pursued the calibration project after me. I wanted to include a short summary of their methods and results in order to bring closure to this chapter. However at the time of writing their work is awaiting reviews following a submission to IEEE Transactions on Robotics, therefore I kept the explanations short to avoid plagiarising their future publication. There are two main differences with the work reported on in the last two sections. First, the authors took into account the mechanical deformations of the robots' arms and legs in their optimisation of the kinematic parameters. Secondly, the data acquisition was not performed on random postures any more, but on fewer carefully selected configurations.

One of our main assumption was that the humanoid robot was sufficiently rigid to neglect the deformations of its joints and bodies. Indeed, while the joints are less robust than those of an industrial manipulator, the bodies constituting its limbs are also lighter than those of such manipulators. However, it seems that the use of strain wave gears in TALOS in lieu of the planetary gears of a standard industrial arm invalidates this hypothesis. The authors of the new procedure were thus taking into account the deformations induced by the own weight of the arm, or the weight of the whole robot in the case of the legs' parameters computation. The estimation of those deformations uses information provided by the torque sensors of the robot. Because those sensors are quite infrequent in today's humanoid robots, the method is not directly applicable on every robot.

The second improvement to our approach was the generation of optimal postures of the robot to gather calibration data. In the 1990s, Gautier and Khalil worked

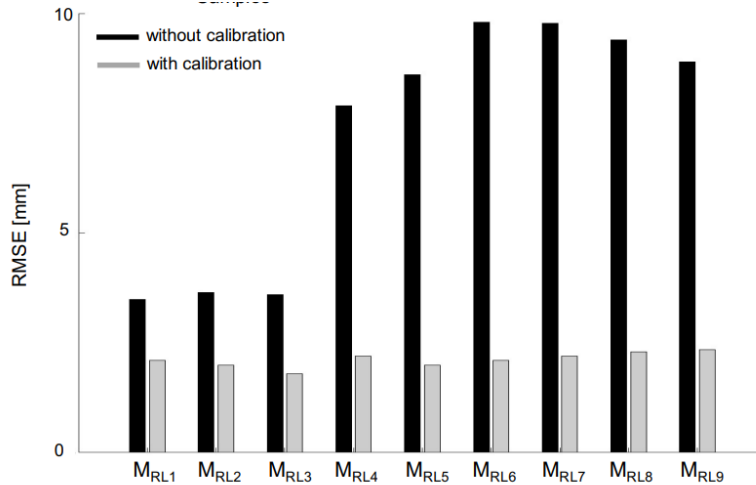


Figure 4.5: Root mean square of the tracking error between the robot’s model and the real measurements, before and after the calibration process. Graph kindly provided by the authors of [207].

on the reduction of the number of parameters needed to accurately model a real robot [212, 213]. The follow up on this work was the formulation of the best trajectory planning strategies to observe those parameters [214, 215]. Aimed with a similar goal, the authors of the new calibration procedure for humanoid robots devised a method to generate postures that improve the observability of the modelling parameters that they chose. Not only did this approach use fewer configurations, and thus took less time to perform, but it also improved the accuracy of the results. Indeed, with fewer but more important postures, there were less low-value data that could pollute the optimisation problem that computes the parameters of the kinematic chain of the robot.

The procedure was tested on our badly calibrated TALOS humanoid robot. For the experimental setup, the main difference with the approaches described previously was the increase in the number of bodies that were equipped with reflective markers. Indeed, while we used a cluster of markers for the chest and one on the end-effector, there, all bodies along the kinematic chain were tracked. Before the calibration, the position of the clusters of reflective markers were acquired, for 20 postures not used during the calibration measurements. The computer model was used to compute the theoretical positions of those clusters for those configurations. The root mean square error between the computed positions and the real data, for each limb of the robot, was used to estimate the precision of the calibration. After the calibration, a new round of data acquisition was performed and a new precision index was computed. The calibration procedure improved the accuracy of the positioning of the real robot by a factor of 3, as can be seen in fig. 4.5.

While this section was not about my work, it shows that the problem of the calibration of the humanoid robot was finally solved.

## 4.5 Conclusion

This chapter described the attempts that we made to properly calibrate our hu-

manoid robot whose arm was clearly not reporting its position correctly. Despite the absence of satisfying results, and the loss of time, it had allowed me to familiarise myself with some general knowledge around robots' kinematic and dynamic parameters. The robot was then left with its incorrect calibration until its following maintenance, while we carried on with the work about the Agimus framework. Since we wanted to eventually add visual servoing into the framework, this poor calibration was not the end of our endeavour, but clearly an important obstacle that complicated our testing and forced us to move forwards sooner than expected.

# Chapter 5

## Agimus: Visual Servoing

### Contents

---

<b>5.1</b>	<b>Introduction</b>	<b>86</b>
<b>5.2</b>	<b>Visual Servoing</b>	<b>86</b>
5.2.1	AprilTags' detection and pose estimation	87
5.2.2	Visual servoing	88
5.2.3	Visual servoing task in the hierarchical controller	90
<b>5.3</b>	<b>Experiments</b>	<b>91</b>
5.3.1	TALOS turns a plank upside down, using vision	91
5.3.2	TIAGo	93
5.3.3	Tiago performs deburring tasks	95
5.3.4	Discussion	99
<b>5.4</b>	<b>Conclusion</b>	<b>99</b>

---

## 5.1 Introduction

In the last chapter, we discussed the process of finding a calibration method adapted to our humanoid robot. This quest initially began when we found that the digital model of our robot was clearly different from the reality, especially for the arms. Indeed, we had a discrepancy of around 10 cm between the theoretical position of the robot's hand and the measurement made on the real robot. This was deleterious for our experiments, all based around manipulation of objects.

We thought that robot's calibration processes, having been discussed for half a century, were mature enough to be used on our humanoid robot, provided we made some modifications. However, we rapidly found that a research humanoid robot is much more complex and much less rigid than an industrial robotic manipulator. Therefore the mostly solved problem of robotic arm manipulators' calibration was not easily adapted to our situation.

Fortunately for us, the aim of this thesis, from the beginning, was the planning of reactive visual servoing tasks. In the chapter 3, the visual part was reduced to the initial estimation of the situation of and around the robot. However, it was hinted in the section 3.2.5 that the controller activated a few centimetres before reaching the position allowing the grasping of an object was intended to receive a visual servoing task eventually. We originally envisioned the addition of this task to be able to grab a object whose position changed between the estimation and the execution, or even maybe to grab a moving object. However, since the robot's poor calibration issue was not addressed at this point, the aim changed to simply be able to accurately reach the object.

In this chapter is presented the second part of the contribution around the Agimus framework. Since the core parts of Agimus have already been detailed in the chapter 3, only the changes linked to the addition of visual servoing are touched upon. The first section deals with the theory behind our visual servoing scheme, and its implementation in the Agimus framework. In the second section, two experiments will be presented. The first is actually the same as the one presented in chapter 3, but with the addition of a visual servoing control to counter the issues of calibration of the robot. The second is performed by another robot, TIAGo, and focused on the more practical industrial task of deburring drilled holes in an aircraft part facsimile.

## 5.2 Visual Servoing

In the chapter 3, we were already using visual information during our experiments. This was done at the beginning of the demonstration, and the results were manually approved by the operator. The planned motions were then executed blindly by the robot, which followed the planned path while ensuring its balance. The framework was however already ready for the next step, which was the addition of real-time visual servoing tasks.

Indeed, as was explained in section 3.2.5, we were generating a controller for the transition where the robot makes the final move to grab an object. This controller had a task to control the position of its gripper, but it was at the time using the

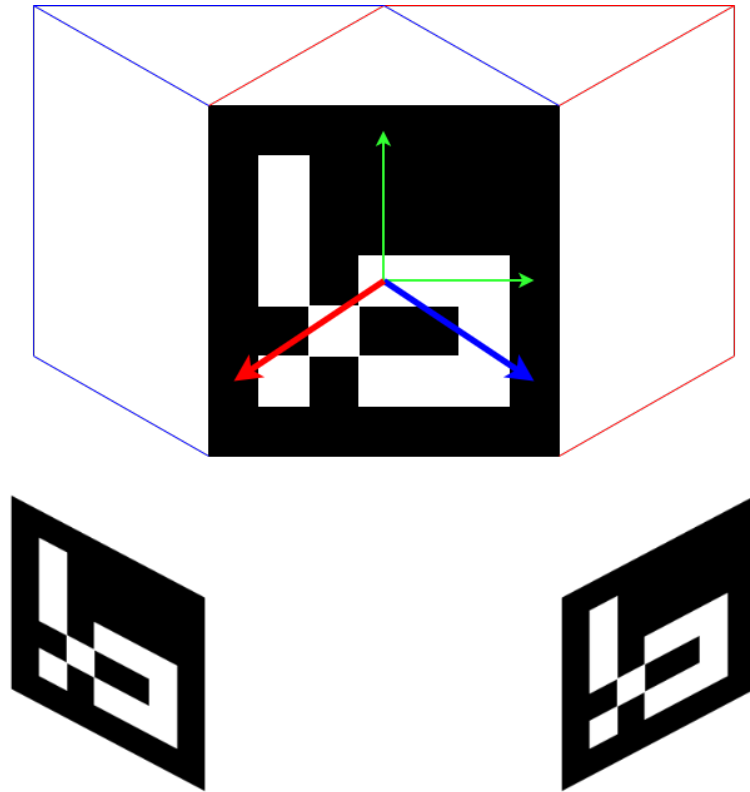


Figure 5.1: The ambiguity problem in pose estimation. The projection could come from either the blue cube or the red cube, depending on the point of view.

planned motions as reference. For the experiments of this chapter, we switched to a visual servoing scheme based on AprilTag markers placed on the objects manipulated and on the obstacles around the robot.

### 5.2.1 AprilTags' detection and pose estimation

An AprilTag is a marker made of a group of black and white pixels, enclosed in a black border, itself surrounded by white. The pixels code for a value, which allows the use and discrimination of multiple markers in the same environment because they exist in tens of different identifiers. The process of detection and identification of the AprilTags is detailed in the original publication [171]. We suppose that the detection was a success and we have a list of AprilTags, along with the coordinates of their four corners in the image frame.

The pose of each marker is then estimated using those coordinates. This is called the Perspective-4-Points problem [165] and many methods exist to solve it: RANSAC [165], ePnP [175], etc. However, since the 4 points that we use for the pose estimation are on the same plane, an ambiguity exists between two possible results for the computed point of view [173], as illustrated in fig. 5.1. In the experiments of chapter 3, this phenomenon appeared, but it was never a real issue. Indeed, the result of the estimation step was validated by a human, and the real-time control was not using visual information. Thus, we just discarded the wrong estimation, and moved the plank a little in the hope of escaping the local minima causing the



improper computing of the object's pose. But in this chapter, we are adding a real-time visual servo control scheme and therefore would appreciate to ensure the quality of the pose estimation, and above all avoid alternating between the two ambiguous results during the execution.

A solution to this issue was proposed by another member of our team in the laboratory. Our robots are equipped with cameras incorporating natively a depth sensor, accurately aligned with the colour image sensor. His solution was simple:

- The computer vision software detect all AprilTags in the image
- Their poses are estimated. Each time, two solutions exist, so a measure of the qualities of those estimation is computed simultaneously. This measure of quality is usually function of the distance between the points detected in the image and a reprojection of those points using the estimated pose. The greater the distance between detection and reprojection, the lower the quality of the estimation. If the quality of one solution is overwhelmingly greater than the other, it is chosen.
- If the qualities of both estimations are equivalent, we take the depth measurements of all the pixels contained inside the square delimiting the AprilTag. Then the mean normal of this surface is computed, and the estimation whose normal is the closest to the depth surface's normal is kept.

He eventually proposed his solution to the developers of the software ViSP that we use for our computer vision's needs, and they incorporated it into the their framework.

In fig. 5.2, one can see that the table, the robot's wrists and the object are all equipped with multiple markers. This redundancy is necessary because the robot often occlude its own field of view with its arms or grippers. Indeed, when it grabs the plank, one of the marker is almost completely hidden by the gripper, and the visual servoing scheme must rely on the second AprilTag. The addition of numerous markers can also improve the accuracy of the pose estimation. It is especially the case in the second experiment of this chapter, illustrated fig. 5.10, where the robot has to position its tool within a few millimetres of precision.

## 5.2.2 Visual servoing

In the domain of visual servo control, there exists two main configurations for the relative positions of the camera and the end-effector that is controlled. The first one, called **eye-to-hand**, has the camera fixed in the world, observing both the end-effector and the target. The second one is **eye-in-hand**, where the camera is attached to the controlled end-effector, and observing only the target. Other configurations, involving multiple cameras, or cameras moving independently of the end-effector, exist but are not used in our framework.

In our experiment, we have two simultaneous control goals that involve vision. We first need to keep the objects and the visually controlled end-effectors in the field of view of the robot, in order to be able to effectively control them. The second is to actually move the end-effector at the desired position, using visual information. Since the second task is done in front of the robot camera, there never was conflicts



Figure 5.2: TALOS robot with AprilTag markers on its wrists

between the tasks in practice.

Image-Based Visual Servoing (IBVS) has its objective written directly in terms of image coordinates. In our case, we want to be able to see all AprilTags at once from the point of view of the robot. Therefore, we compute the mean coordinates, in the image frame, of the AprilTags currently detected, and control the head of the robot, containing the camera, in order to maintain those at the centre of the image.

For the Position-Based Visual Servoing (PBVS) scheme, the pose of the object relative to the camera is estimated. The control is then expressed in real world coordinates, based on this estimation and a specified goal. Coming back to our framework, we want the gripper of the robots to approach, and eventually grab, an object. For the estimation step, we ease the burden by using the AprilTags, with the methods described in the previous section. Formulating the control using the real world coordinates allows us to change the source of the reference frame as needed.

Mathematically, we have:

$$e_{SE(3)}(T_1, T_2) = \log(T_1^{-1}T_2) \quad (5.1)$$

where  $(T_1, T_2) \in SE(3)^2$  are two poses. This log represent the screw velocity  $(v, \omega) \in \mathbb{R}^6$  that moves  $T_1$  to  $T_2$  in unit time. If  $T_1 = T_2$ , obviously the error is zero, the two poses are superposed and the task has converged. The control is applied to a gripper of the robot, therefore  $T_1 = g_1$  or  $T_1 = g_2$ .

In the chapter 3, we used the planned position of the gripper as the sole reference for  $T_2$ , and a high gain on the task to ensure the close following of this reference.

Here, we want to use the pose estimation done by computer vision as the reference  $T_2 = H_1$ , where  $H_1$  is the measured pose of the handle of the object. But the MoCap could also be used to compute the position of the object and the gripper of the robot in this control scheme.

Because of the calibration issues of our robot, our control scheme needs to take into account both the estimated pose of the object and the estimated pose of the gripper as seen by the computer vision algorithm, instead of the pose returned by the proprioception of the robot. The reference is thus modified as follows to take into account the difference between the controlled position of the gripper, based on encoders' data, and the position measured by the cameras:

$$T_2 = g_1^{-1} g_1^{measured} H_1 \quad (5.2)$$

The visual servoing task is finally formulated as:

$$e_{SE(3)}(g_1, H_1) = \log(g_1 \cdot (g_1^{-1} g_1^{measured} H_1)) = \log(g_1^{measured} H_1) \quad (5.3)$$

The gain applied to the task is lowered because we want the robot to smoothly reach its goal which should be static or at least not moving too much.

### 5.2.3 Visual servoing task in the hierarchical controller

In section 3.2.5, we mentioned the existence of control tasks for the positioning of the robot's grippers. However, at the time, the references given to the control stack were only coming from the planned motions, and not from real-time data. It was done in order to prepare the continuation of the development of Agimus in the form of visual servoing, which is described in this chapter. Therefore, the implementation of the visual servoing schemes described in the previous section was quite easy. The generation of controllers was already creating tasks for the control of the grippers, so we just had to modify those tasks to incorporate real-time visual servo control.

At the time of this publication, the task to control the gaze of the robot was not implemented. During the experiments, the robots were only following the planned paths for their head configurations, instead of using computer vision data to maintain the AprilTag's markers at the centre of their visual fields. The tests performed showed that it was largely sufficient because the perturbations of the position of the objects stayed small compared to the relatively large field of view of the robots. Therefore, adding that task into the controllers would have only impacted performance without any real benefit.

The tasks controlling the grippers of the robots, however, were of a crucial importance. Indeed, our humanoid robot was at the time unable to grab the plank of wood because of an incorrect estimation of its own grippers' poses. If the object is visible, the task relies on the computer vision information for the control, as detailed in the previous section. During the execution, the gap between the planned poses and the real poses estimated by the vision node is measured continuously. In the case of a loss of the line of sight, the planned reference is used, but shifted using the last information available.

## 5.3 Experiments

In this section, two experiments are presented to demonstrate the implementation of visual servoing tasks in the Agimus framework, and to discuss the performance of this approach. The first experiment is nearly the same as the one described in section 3.4, with the addition of visual servoing tasks to precisely control the grasping of the wooden plank. The second demonstration is done on another robot, with a differential drive base, a lifting torso and a single arm. This experiment is about performing tasks that are more appealing to our industrial partners, like drilling or deburring holes.

### 5.3.1 TALOS turns a plank upside down, using vision

At the end of chapter 3, our experiment was hindered by the mechanical inaccuracy of the robot, following a maintenance. The robot was almost always missing the wooden plank by up to 10 cm, because of erroneous offsets in some joints of its arm. Different approaches tried to correct this problem by calibrating the robot are discussed in chapter 4, but after two failures, we decided to circumvent the problem by using visual servoing.

The goal of this thesis, from the beginning, is the planning of visual servoing tasks. Our original goal was to incorporate the visual tasks in order to be able to move the plank between the estimation step and the execution step. The robot would have to update its trajectory in real-time to grab the plank at its new position. Then, we would have tried to add a feed-forward system to allow the moving of the plank during execution, with the robot able to follow it and grab it when it stopped. But for this experiment, we aimed at simply reproducing the results of the first experiment, *i.e.* grab the plank and place it back upside-down, with a superior robustness.

For this new experiment, we kept the setup, overall protocol and goal of the one described in section 3.4. The difference is the use of a visual servo control at the points of the demonstration relying on precise manipulation. Those key-points are the grabbing of the plank, the hand-over and the release of the object at the end. At those moment, instead of blindly following the planned motion based on the initial estimation of situation, the tasks described in the previous section are added to take priority in the grippers' positions control.

The visual servo tasks are added on the transition between the waypoints of the graph of constraints, before the robot grabs the plank, and state where the grasping effectively occurs. When the robot reach the waypoint, the visual servoing task is activated slowly to allow a smooth correction of the gripper's pose. Then the reference trajectory computed by the motion planner is resumed, shifted to correspond to the measured pose of the gripper and the plank at that moment.

The rest of the execution is still entirely based on the planned path. The results of the experiment were very good, and the robot was on again able to grab the plank of wood. The sliding of the plank in the gripper, which was already an infrequent issue since the addition of an admittance force controller for the grippers, was no longer a problem. Instead of relying on the perceived position of the first gripper to position the second, the controller was directly using the AprilTags on the plank

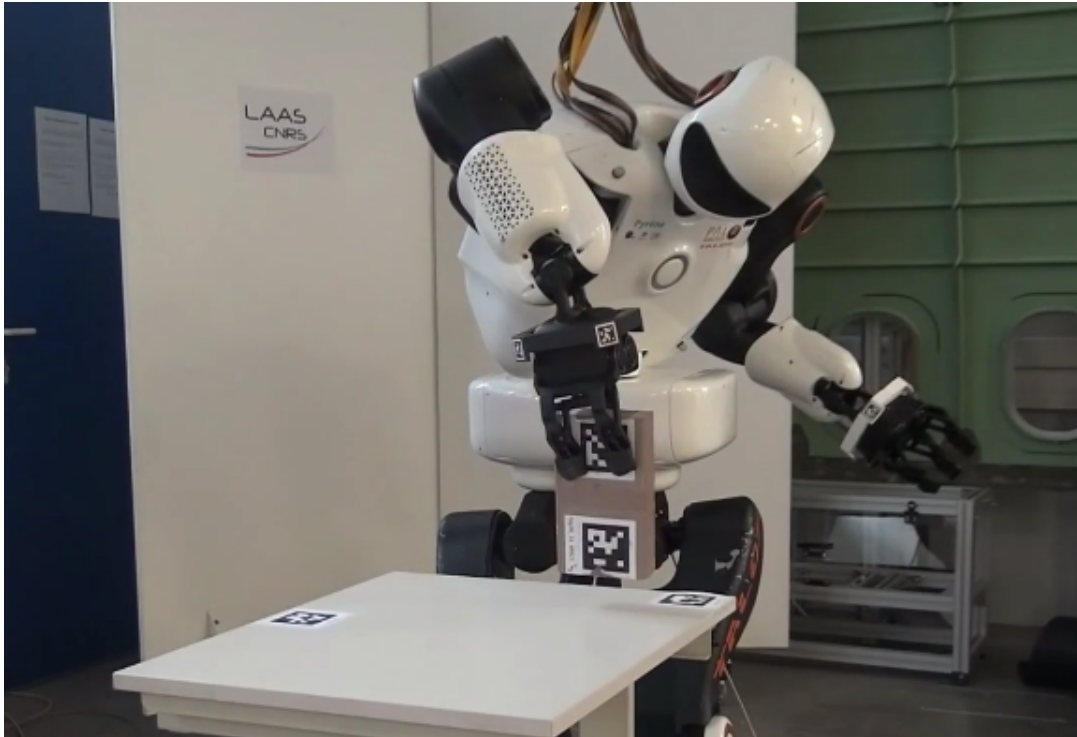
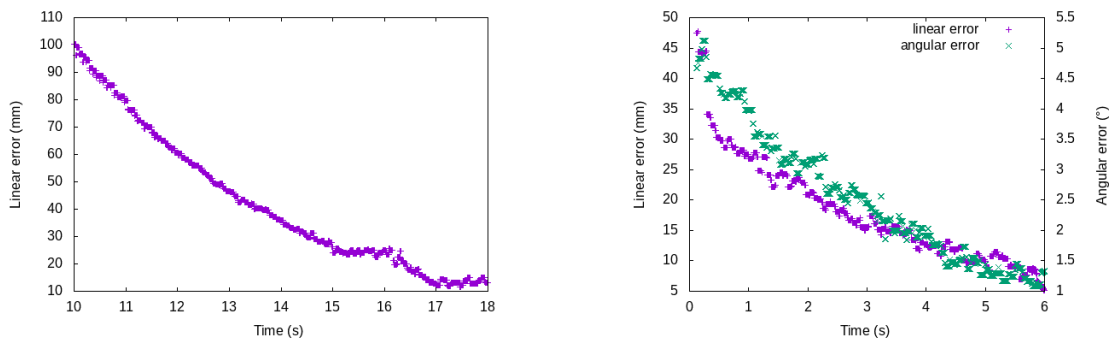


Figure 5.3: TALOS robot placing a plank on a table

to estimate where to grab the object. On fig. 5.3, we can see the robot placing the plank back on the table. We clearly see that the plank has slightly slipped in the gripper, but the control re-align it correctly before the release.



(a) Position of the gripper relative to the plank. Angular alignment is not controlled nor tracked until the gripper is less than 50 mm away from its goal.

(b) Final approach of the gripper towards the plank.

Figure 5.4: Tracking error between the gripper and the plank

On fig. 5.4a and fig. 5.4b, we can see the tracking error between the robot's gripper and the **grasp** pose it needs to reach to grab the plank. Like all the other tasks controlling the robot, we use an exponential decay for the error reduction (see section 3.2.4.1), which is clearly visible on the two figures. On the first figure, the bump around 15s in the error of the linear motion corresponds to the moment the robot resumes the following of the planned trajectory, to descend towards the plank. The original plan moves the reference in around a second, but the visual

servoing task has a very low gain because the camera refresh rate is low compared to the robot's control frequency. Therefore the tracking of this reference is slow and slightly delayed, which explains the increase of linear error.

The fig. 5.5 presents the experiment with visual servoing, in the same fashion as section 3.4. A video is available at <https://hal.laas.fr/hal-02494737>. The experiment being the same as before, the process is not described again. The only difference is that the position of the grippers are controlled by visual reference instead of planned references. The effects of the correction can be seen between the steps 2 and 3, 5 and 6, and finally 9 and 10.

### 5.3.2 TIAGo



Figure 5.6: TIAGo robot, made by PAL Robotics

TIAGo, fig. 5.6, is a mobile manipulator made by the Spanish company PAL Robotics [10], using a modular approach both for the hardware and the software. This robot is made of a mobile base with an upper body composed of a torso, on which are mounted an arm and a head. The differential drive mobile base has been designed to also work as a stand-alone robot aimed at fulfilling logistics tasks. This base can navigate autonomously thanks to a SLAM fed by a plane laser rangefinder on the front of the robot. The torso of TIAGo is actuated, allowing the total height of the robot to be adjusted between 110 cm and 145 cm.

The arm, on the front of the upper half of the torso, has 7 DoFs, 3 of which are situated on the last module, the wrist. The company capitalised on the technologies used on their first robots, fig. 5.7, namely REEM [216], a service robot, and REEM-C, a small biped humanoid, by reusing the same motorised modules for the arms of TIAGo. At the end of the wrist, there is a 6 axis force-torque sensor with a mounting plate that allows the addition of modular end-effectors. At the time of writing, the end-effector's options are multiple parallel grippers and a hand with 5 fingers. The hand option has 19 DoFs, 3 of which are actuated, and is based on the work of Catalano et al. [217]. The main idea is that the few actuated DoFs exploit synergies between the total DoFs in order to be able to grasp objects of dissimilar shapes. Finally, the head, at the top of the lifting torso, can pan and tilt. It is equipped with a RGB-D camera.

Like its brother TALOS, described in section 3.4.1, its software architecture is based on a Linux Ubuntu operating system, patched with RT-Preempt to provide hard real-time features. On top of this, the ROS middleware orchestrates the multiple components that interpret data from the sensors or control the actuators. Low-level control is implemented as `ros_control`'s plugins, enabling position, velocity or current control of the motors, while the high-level motion planning is done by MoveIt!.



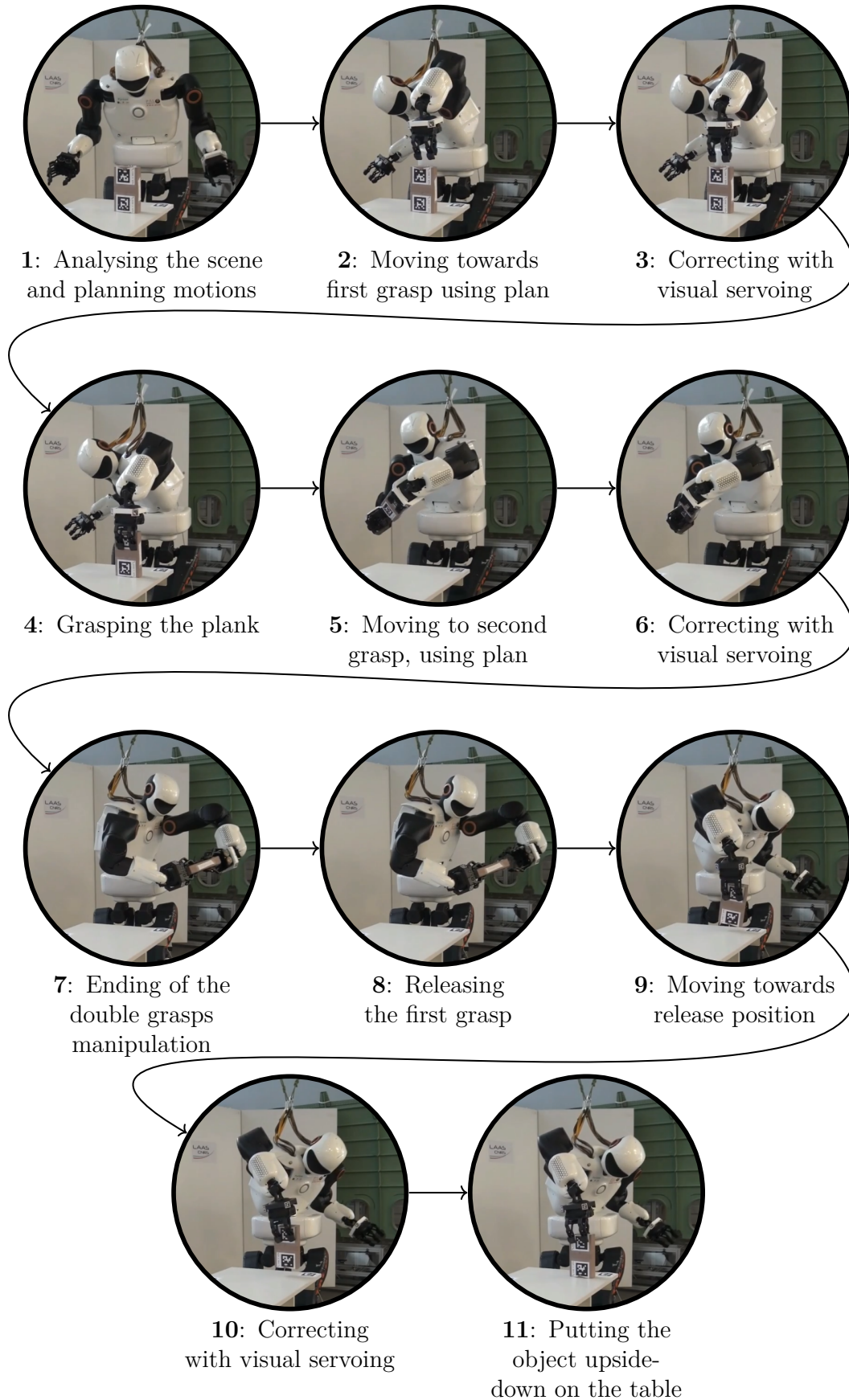


Figure 5.5: TALOS manipulates a plank with visual servoing

### 5.3.3 Tiago performs deburring tasks

The industrial goal of this thesis and more generally the joint laboratory ROB4FAM is to demonstrate that a robot is able to help human workers performing dull tasks. One of those task is the deburring (smoothing the rough edges) of holes in drilled parts. It is an essential step in manufacturing because those edges can be quite sharp and can cut not only the workers themselves, be also fasteners or pieces of equipment going through those holes.

For this demonstration, we used the TIAGo robot presented in the previous section, which is equipped with its 5 fingers hand mounted on its end-effector. This hand is used to grab a drill on which a deburring tool has been installed. Finally, we took a 3D printed mockup of a real aircraft part, fig. 5.8, quite complex and with multiple holes, to demonstrate the capabilities of the framework Agimus.

The 5 fingers hand of the robot is strong enough to hold the drill steady during the motions of the robot's arm. However, the grasping action itself has a bad repeatability, which means that the pose of the drill relative to the hand is not accurately known at the beginning of the experiment. The drill is equipped with an AprilTag marker that allows a quick and easy determination of its pose relative to the robot's camera. In the same manner, the part also has multiple markers fixed on it.

The software setup is nearly the same as the one used for TALOS in section 5.3.1. One key difference is the addition of an optional movement of the mobile base at the beginning of the demonstration, to approach the aircraft's part. The second difference is the representation of the deburring on the software side. The holes that need deburring are represented by **handles** placed of the part. A **gripper** is attached to the deburring tool inserted in the drill. The deburring task is thus performed by following trajectories between **pre-grasp** and **grasp** for these **handles**. See fig. 5.9 for an illustration of the handles and the gripper.

In the same fashion as the TALOS experiment, the typical solution of the TIAGo experiment is provided in fig. 5.5, and explained thereafter.



Figure 5.7: REEM-C (left) and REEM (right)



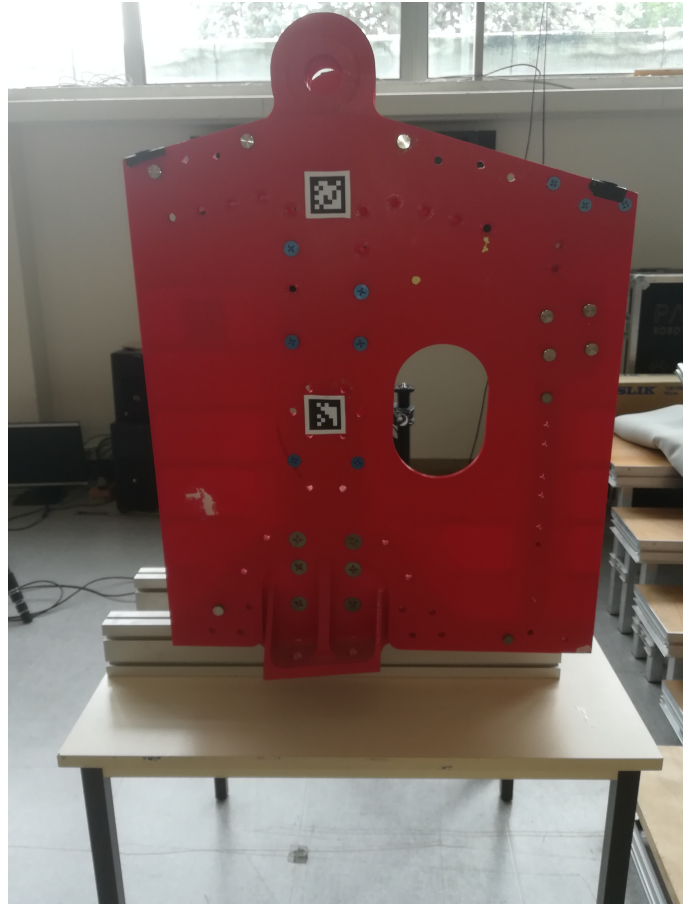


Figure 5.8: 3D printed mockup of a subcomponent of an Airbus A380 engine pylon

1. The robot autonomously goes towards the aircraft's part, which is at a known position in an already mapped room. This is done by the ROS navigation stack provided with the robot.  
**Active Agimus controllers:**  
 None.
2. It comes to its destination and uses its cameras and the AprilTags to determine the precise position of the component and moves towards it to be at reaching distance.  
**Active Agimus controllers:**  
 None.
3. TIAGo uses its cameras to locate the holes in need of deburring and plan the motions to achieve its goal. The 3D model is used to determine which holes to work on.  
**Active Agimus controllers:**  
 None.
4. It goes to its initial position, arm deployed.  
**Active Agimus controllers:**
  - Posture task

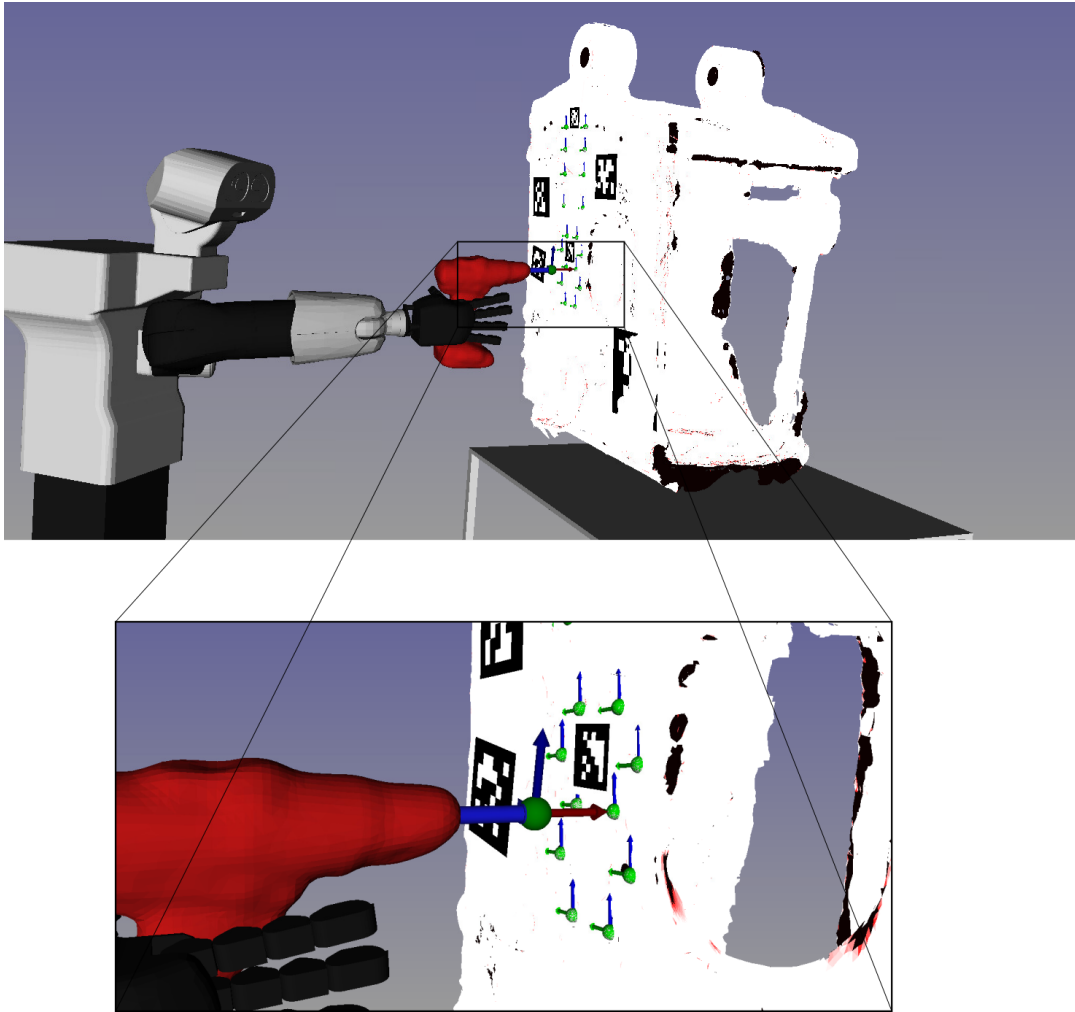


Figure 5.9: Deburring tasks are represented by virtual grasp – pre-grasp pairs. A **gripper** frame is attached to the deburring tool and **handle** frames are attached to each hole of the part. Part and drill models have been built using an RGB-D camera and 3D reconstruction software.

5. TIAGo points its arm at the first planned hole's position. Due to the low precision grabbing of the drill, we can see that it is very badly oriented.

**Active Agimus controllers:**

- Posture task

6. Visual servoing is activated to correct the position and orientation of the drill before performing the action itself.

**Active Agimus controllers:**

- Visual servo control of the drill's position relative to the hole
- Posture task

7. With the visual servoing still active, the robot performs its deburring. The contact with the part is checked by vision and also with a limitation on the force applied by the robot. This force is measured by the force-torque sensor

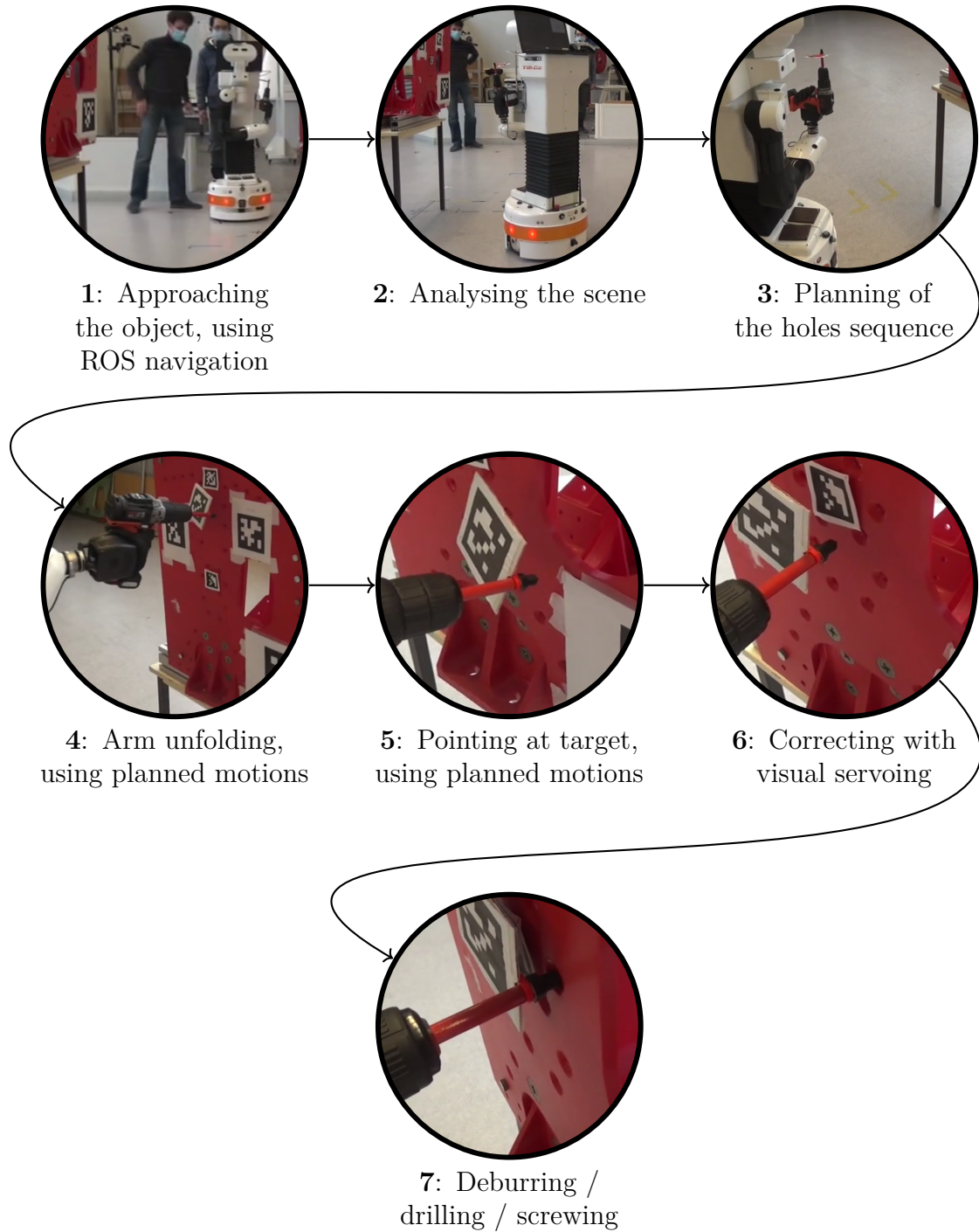


Figure 5.10: TIAGo experiment with deburring holes on an aircraft's part

in the wrist of TIAGo. In a real deburring scenario, this force should be controlled by the robot to ensure the quality of the work performed.

**Active Agimus controllers:**

- Visual servo control of the drill's position relative to the hole
- Posture task

The video at <https://peertube.laas.fr/videos/watch/6f40ea79-abcd-490e-a616-3a67bf297d93> illustrates the various steps of the method. This experiment has been successfully executed several times, including in front of our industrial partners.

### 5.3.4 Discussion

In this chapter we demonstrated that the Agimus framework can easily be adapted to other robots. It also demonstrates that the concepts of *grippers*, *handles* and *grasps*, that underlie the graph of constraints, are versatile enough to formulate other tasks than grabbing objects.

The addition of visual servoing tasks to improve the precision of the fine movements of grasping or deburring was also a success. It allowed the TALOS humanoid robot to perform its plank's manipulation experiment, section 3.4, even better than when it was calibrated correctly. Very recent tests, during a live demonstration that was unfortunately not recorded, also showcased that the visual servoing scheme is able to deal with the displacement of the part during execution.

There are however still room for improvements. Some discontinuities can be seen in the commands when switching from the execution of the planned path to the visual servo control task. The visual tasks are only active beginning at the pre-grasp states of the graph, and de-activate when coming back to this state after the deburring. Therefore, during the movements between two holes, we can clearly observe that the drill of TIAGo is not oriented correctly because no visual servoing occurs. After each task, the following paths could be modified to integrate the measured difference between the planned reference position of the drill and the real measure by vision. Thus, the drill would be near its correct orientation during the movements between holes, and this could gain some time during the demonstration. Such detail was of no importance for the sake of the demonstration, but could mean a lot when porting to industrial manipulators.

## 5.4 Conclusion

In this chapter, we presented the addition of visual servo control tasks into the Agimus framework. The original goal of this modification was to improve the precision of the manipulation motions, and allow adaptation to slight changes in the setup between the estimation step and the execution. Actually, it mainly circumvented the calibration issues of our humanoid robot, by adding control over the grippers' positions instead of relying on blind following of the planned paths.

The visual servo control features were successfully tested on two kind of experiments, with two different robots. The first experiment is the follow-up of the one in chapter 3, with added precision and robustness. The second experiment hinge toward a more industrial demonstration. A TIAGo robot, comprised of a mobile base, a lifting torso, a robotic arm manipulator and a head, was required to manipulate a drill to deburr the holes of an aircraft component. While this demonstration was performed on a robot that is far too compliant to handle real drilling or deburring tasks, it showcases the capabilities of the Agimus framework.

The work detailed in this chapter was presented in the following publication:

Joseph Mirabel, Florent Lamiraux, Thuc Long Ha, **Alexis Nicolin**, Olivier Stasse, and Sébastien Boria. “Performing manufacturing tasks with a mobile manipulator: from motion planning to sensor based motion control”. In: *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*. IEEE. 2021, pp. 159–164

# Chapter 6

## Middle Sized Drilling Robot

### Contents

---

6.1	Introduction . . . . .	102
6.2	Presentation of the robot . . . . .	102
6.3	Drilling process and its challenges . . . . .	105
6.4	Conclusion . . . . .	107

---

## 6.1 Introduction

The previous chapters dealt with the development of the Agimus framework. It is aimed at facilitating the planning and execution of manipulation tasks on various kinds of robots. In chapter 3, the framework was successfully tested on a full-sized humanoid robot, TALOS, to demonstrate the capabilities of the system on such a complex robot. The last chapter, chapter 5, dealt with the addition of visual servoing tasks into the framework, and ported the demonstration to a mobile robot equipped with an arm, TIAGo. By using a robot closer to a classical industrial manipulator arms, we wanted to show that our work was not just a research subject, but could be the beginning of a solution for our partners in the aircraft industry.

This chapter presents one robot designed by Airbus to perform drilling on the aluminium panels that are parts of a plane's fuselage. The first section explains the need for a completely new robot and presents the envisioned solution. Because of the industrial aim of this robot, only a few details can be given at this point, for confidentiality reasons. The second section deals with the drilling process and how the robot's sensors will help it perform adequately. My contributions to the project around the Middle Sized Drilling Robot (MSDR) were limited, mainly due to the pandemic. I was advising the team working on the robot in Spain, about computer vision and reactive planning operations.

## 6.2 Presentation of the robot

Robots have been used in the industry for half a century, either to increase productivity, or to replace workers in dirty, dull or dangerous tasks. Aircrafts' manufacturers, like their counterparts in the automotive industry, tried to include robots in their processes, from the parts' fabrication up to the final assembly. However, issues quickly appeared in this endeavour because of the specificities of planes compared to cars. Two key differences can explain the majority of the problems encountered. The most obvious is the size of the vehicle, while the second is the volume manufactured each year.

Planes' dimensions are in the range of 30 m to 70 m, compared to the 4 m to 5 m of a car. This is not typically a problem when the robots work on aircrafts' components, but it becomes one during assembly. Because robots have a practical size limit, coming from physical and economical constraints, there are two choices available. Either increase the number of fixed robots around the plane, or add DoFs to a few robots to allow them to move between working areas. The second solution has been tried in one of the assembly line of the A321, in Hamburg, featured Figure 6.1. Multiple robots, each equipped with a huge end-effector, were fixed onto an structure that could move along a rail. Overall, this system weighed more than 20 t per side, which was quite heavy to drill  $\varnothing 4.8$  mm holes. Another issue was that the accuracy of the system was mostly managed by the high rigidity of the structure.

The second issue originates from the software side. Industrial robots, as of today, are mostly programmed offline, using simulations of the 3D models being worked on. Then, those programs are executed blindly by the robots, with a limited, or even non-existing, ability to deal with uncertainties. Thus, they require some testing



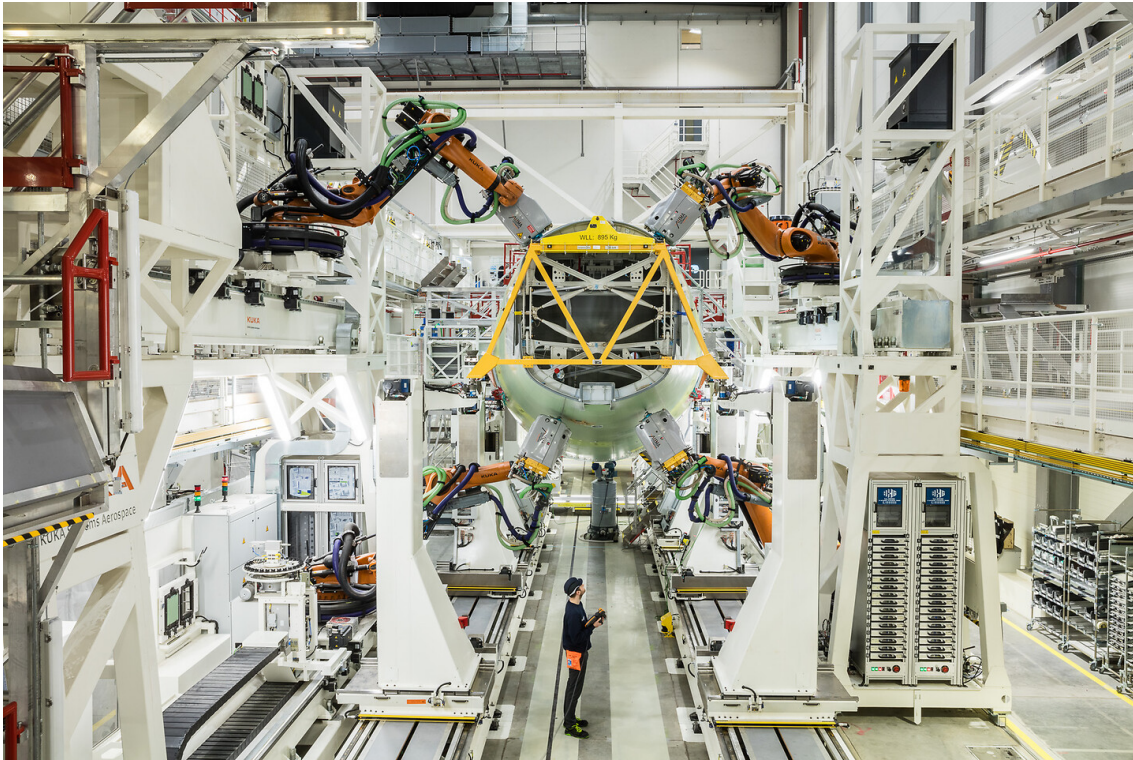


Figure 6.1: Airbus A321 automated fuselage assembly line in Hamburg Finkenwerder

before working a real aircraft's part. Eventually, some changes between the digital mock-up and the real component may require the manual insertion of offsets in the program, or a complete new offline simulation of the movements.

In the car industry, assembly lines can output hundreds of cars a day. The time required to program the robots and test the assembly lines is earned back by the volume of production increase robots allow. However, in the aircraft industry, the typical output of a final assembly line is less than **one** plane each day. Therefore, the time invested in the programming and testing of robots is difficult to get back.

Overall, the combination of the low output of the industry and the lack of robots capable of autonomous adaptation to their task on the market fuelled the need for a new in-house design for a drilling robot, the MSDR.

The MSDR has been proposed by Airbus' robotics team to integrate cost-effective robots in the plane construction workflow. The manipulator arm itself is an M-800iA/60 robot, fig. 6.2, made by the Japanese robot manufacturer Fanuc. It is a 6-axes robotic arm capable of moving a payload of 60 kg up to a distance of 2040 mm. Each of its joint is measured by two different encoders. The typical one on the motor side, plus a second one at the output of the reducer, to compensate eventual deflections or backlash. Those additional encoders allow a repeatability of 0.03 mm over the robot's workspace. Those excellent results are paired with a new calibration procedure that led to an independent and impressive measurement of the absolute accuracy in the same order of magnitude.

This robot is equipped with an end-effector designed by the robotics team to cater the needs of airframe drilling processes. Because the end-effector is a future industrial tool, I am not authorised to go into details about its exact sensors or





Figure 6.2: M-800iA/60 manipulator from Fanuc

capabilities. It is comprised of:

- A drill, mounted on a linear rail, accompanied by a suction hose and adequate lubrication system,
- Cameras, to detect previously drilled holes and use them to align the robot with the aircraft part,
- Laser range-finders, to place the end-effector perfectly orthogonally to the part,
- Force sensors, to detect the contact with the part and drill with the required force.

All those elements will work together to enable accurate drilling of the holes necessary for the assembly of the aluminium panels that make a fuselage. A photograph of the robot with an incomplete end-effector being tested can be seen fig. 6.3. One particularity of this robot is that it is designed to be either fixed to the ground or put on a movable platform to enlarge its area of work. In the next section, details will be given about the drilling process and how those sensors will help achieve this performance robustly.

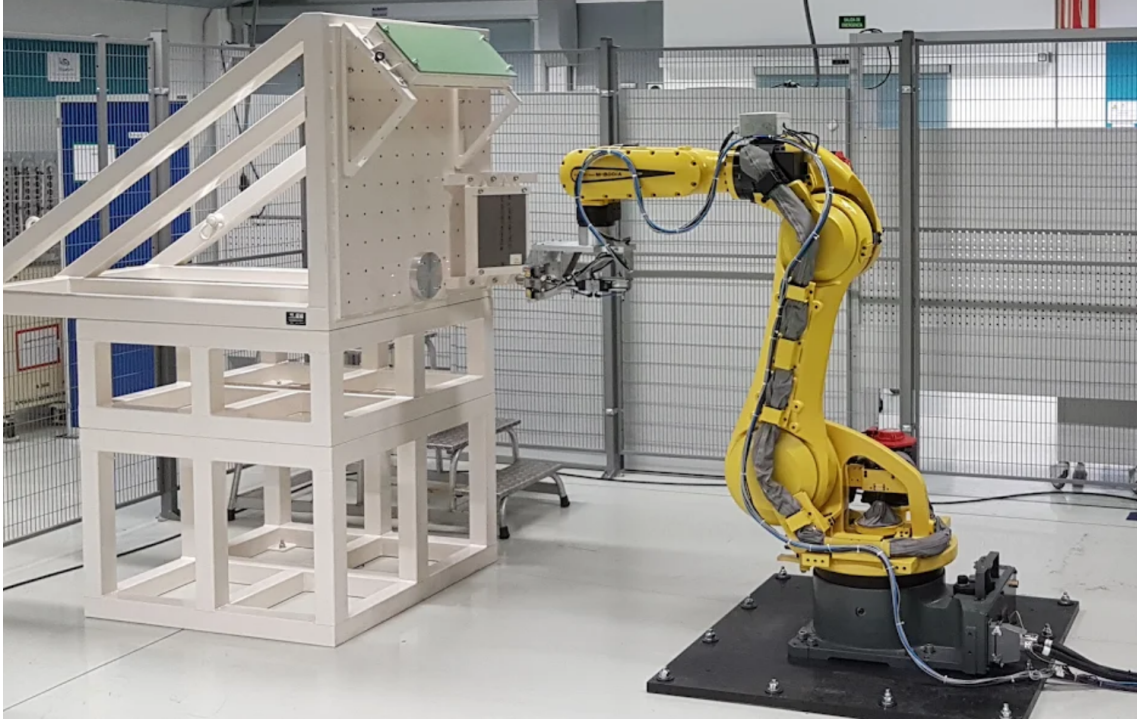


Figure 6.3: The Middle Sized Drilling Robot

### 6.3 Drilling process and its challenges

In the last section, we presented the future drilling robot made by Airbus. The next paragraphs explain the nominal process that the robot will follow to help the human operators. Then, we detail the challenges that each step of this process includes.

Before the robot operation, some of the component's holes have already been pre-drilled by a skilled human operator. The robot uses those holes as references to drill the remaining ones by itself, as illustrated in fig. 6.4. The procedure for drilling a row of holes is as follows:

1. **Move to R1** The robot's tool is moved to the first reference hole, **R1**, by a program created offline.
2. **Alignment** Position and orientation are adjusted using the cameras and laser range-finders to achieve an accuracy of 0.2mm and 0.5°. This corrected pose of the tool becomes the new reference for **R1**.
3. **Move to R2** The robot's tool is moved by the program created offline towards the second reference hole, **R2**.
4. **Alignment** Same process as step 2 but for the second reference.
5. **Interpolation** Using the corrected values for **R1** and **R2**, the positions of the holes that remain to be drilled are interpolated.
6. **Move to hole 1** The tool is moved in front of the first hole by the online program, and the orientation is again adjusted.

7. **Drilling** The hole is drilled using a control loop fed by the force sensors' data.
8. **Move to hole  $n$  and drill** Repeat of the two last steps until the end.

Whenever one of those steps cannot be performed, the robot should autonomously detect its failure and request the assistance of a human. Such failures may be the impossibility to detect the reference holes or to align the tool orthogonally with respect to the airframe.

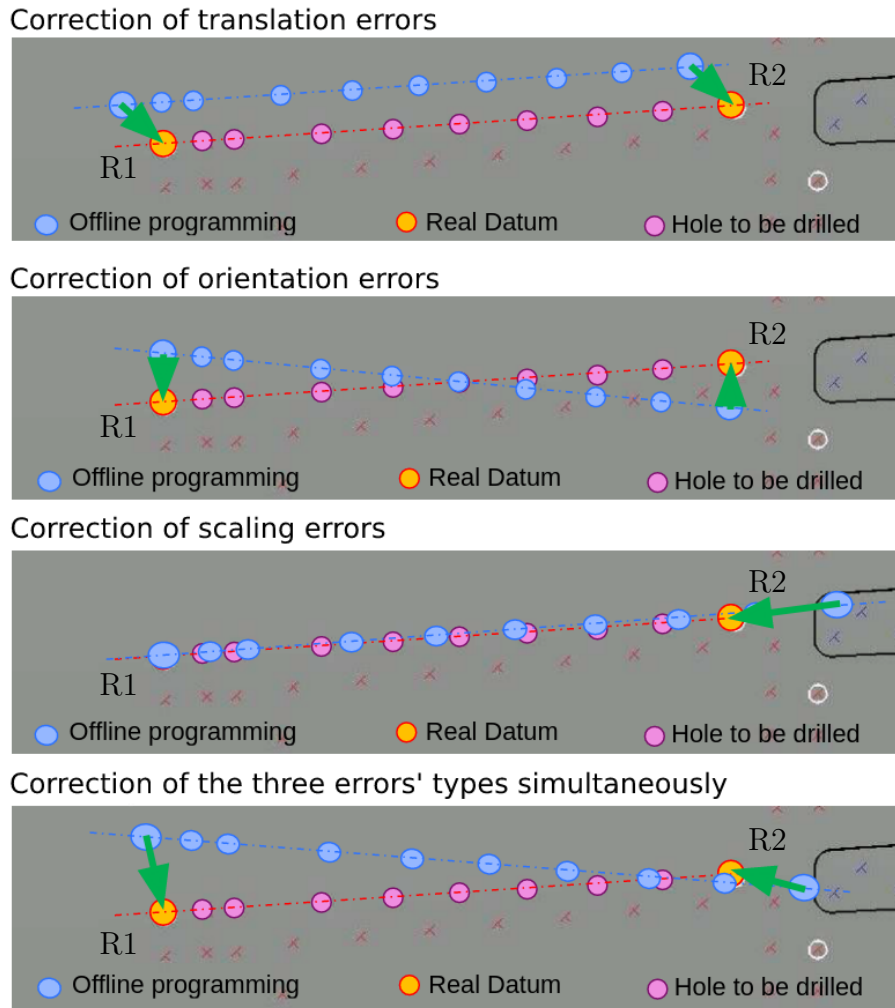


Figure 6.4: The MS DR uses reference holes to adapt its offline programming

The detection of the reference holes use the data from the cameras and computer vision. Scientifically, this part of the process is known and is not very different from what we did in chapter 5. The alignment using the laser range-finders is a matter of trigonometry. What is not simple however is to check that the lasers are not firing into already drilled holes, which would provide erroneous information. The presence of multiple sensors allows the cross-checking of the measurements to compensate for such issue.

Some other objectives are linked with the work presented in chapter 3 and chapter 5, and more broadly the goals of the ROB4FAM joint laboratory. The ultimate

target of the work in ROB4FAM and with the MSDR is the *concurrent coordination* between the robot and its human collaborators.

On the control side, a lighter concept of safety cell for the robot is already implemented. It slows down the robot when a human comes nearby in order to be able to stop if they come too close and risk an accident. The work of the laboratory was to improve this solution via passive torque controllers that would ensure the physical safety of the operators, but also the robot and the environment around.

On the planning side, which is more in line with the subject of this manuscript, we wanted the robot to be able to react and adapt its plan to changes of the environment or of the objectives. By reacting, we mean that the operator may have drilled more holes than only the references, and the robot should not take time and risk ruining the holes by re-drilling them. Instead, it should avoid it in real-time without a step of offline replanning. Replanning would be necessary in the opposite case, when references are missing or when alignment fails. In those events, at the moment the robot would ask for the help of an operator. We would like it to re-plan autonomously its mission and drill other rows of holes if it can, to limit its downtime.

In this section we presented the nominal drilling process of the MSDR. Some of the tasks involved in this process are already mastered either at the scientific or even industrial level, *e.g.* the hole detection. Other more ambitious ones are subjects that were discussed in the previous chapters, and will require further work to go from the laboratory to the factory.

## 6.4 Conclusion

In this chapter we presented the MSDR robot designed by Airbus to help its operators perform dull drilling tasks. This involved the design of a new robot, with drastically increased accuracy, by the manufacturer Fanuc. Then, the robotics team at Airbus added the end-effector tooling necessary to perform the drilling to the quality required in planes. At the time of writing, the step being worked on is the programming and testing of the robot.

My contributions to this project were limited, due to the pandemic and the multiple lock-downs. Indeed, the robot was in another country and therefore programming and testing it was not possible for me. I advised the engineers working on the robot on the subject of alignment, providing state-of-the-art articles and guidance in the implementation of algorithms. I was also involved in a parallel project about reactive planning, which the MSDR should also feature in the future. The next phase of the project should begin shortly with the implementation of the higher level concepts developed for Agimus.



# Chapter 7

## Conclusion

### Summary

The objectives of this thesis were centred around the will to improve automation and capabilities of adaptation to industrial robots. Today, these robots are programmed offline either via a modelling tool or sometimes manually by a skilled operator. Then, the robots follow this programming blindly.

Our goal was to build upon the motion planner to automatically construct a set of controllers corresponding to each step of the plan. The second step was to get from a blind following of the plan to a predominant use of the sensors of the robots, including force/torque sensors and cameras. Then, the offline plan would be overtaken by the reactive controllers in case of changes between the plan and the actual situation.

In chapter 2 was presented an overview of the existing works in the various domains related to this thesis. It covers motion planning and task planning, and then the combination of those two fields to achieve manipulation of objects and the environment by robots. This was followed by a presentation of the field of control, applied to robotics, and finally visual servo control.

Chapter 3 was the main contribution of this thesis, with the presentation of the Agimus framework. This software is built on the manipulation planner of the team, HPP, and the hierarchical control software, the SoT, to provide an automated formulation of the controllers from the planned motions. I contributed to the proposition of the general architecture of the framework, then worked mainly around the vision and estimation components. I also participated in the adaptation and debugging of the ageing control software, and finally in several tests and demonstration of our framework on the TALOS humanoid robot.

In chapter 4 we dealt with the calibration issues of the humanoid robot, and discussed several trials to improve the situation. I was involved in the implementation and testing of the first solution, based on the experienced suggestions of a colleague. I also proposed, implemented and tried the simpler and more naive solution inspired by the industrial process. However, I did not work on the third method, which was the first one to return favourable results.

Chapter 5 was about the addition of visual servo control to Agimus, to improve the precision of the manipulation motions. It also allowed small modifications of

the setup after the planning step, with the robot able to adapt in real-time to the changes. I contributed to the demonstration performed on TALOS and that was the continuity of the experiment done in chapter 3. I added markers, some of the control schemes, performed tests and measurements. This work led to the submission of a paper, that was unfortunately refused.

Finally, chapter 6 presented an industrial robot conceived by Airbus robotics team to drill holes in aluminium panels that constitute the fuselage of a plane. Due to the pandemic, the robot was delayed several months in Spain, so my contributions were centred around scientific guidance to the engineers working on the robot itself.

## Future works

In our demonstrations, the framework was successful and its main purpose, the automated generation of controllers, proved to be useful. Indeed, thanks to that, we only had to modify the grippers' controllers to add visual servoing during manipulation of object on a TALOS humanoid robot.

### Improve Agimus to rely less on the user

However, the framework is still heavily relying on the user for several features, and those are paths of improvement for the future of Agimus.

A first improvement could be a better semantic for the whole system. Indeed, at the moment, the drilling and deburring actions of TIAGo, seen in section 5.3.3 are modelled by pairs of *pre-grasp* and *grasp* states, which is not really intuitive. It could be replaced by a simpler *drill* task for the end-user, which would encapsulate the sequence of *pre-grasp* and *graps* actions. A second way to explore is the reduction of the use of visual markers and supplementary data about objects, such as their *handles* position. Both are prone to human errors when the data are manually entered in the scripts. They could be replaced eventually by a direct recognition of objects based on their 3D models, and automated extraction of the *handles* also based of the models. Those axis of development have been explored by trainees and could give good results in the long run. Finally, the framework is asking a human operator to check the planned path before execution, to prevent superfluous or even dangerous movements. We think that some measurements of the quality of a path could be automated, and that Agimus could re-plan by itself the parts with less desirable behaviour. The choice of the criteria is however difficult and may depend on the complete task itself, so a supervision by a human seems necessary for the foreseeable future.

### Detect and handle errors

At the moment, Agimus does not handle errors, and even does not detect a lot of undesirable actions. The main error checking mechanism is at the lower level of the robot's control system, and measure forces, torques and motors' states in order to prevent deterioration of the robot. The detection of errors at the task execution level is entirely done by the human supervising the experiment, which at the moment can only pause the execution to correct the problem manually.

There are two ways to improve that, one in real-time and one at the end of each sub-task. A kind of real-time checking for erroneous behaviour is already present in the form of visual servoing. Indeed, this was added to enhance the precision of the manipulation motions and to counter a bad calibration of the robot or the slipping of the object in the grippers. While this correctly handles slight variations around the plan, it cannot cope with changes such as the object falling. For such problem, it would be better for the robot to call an operator for assistance. The validation at the end of a sub-task could be similar to the estimation step of Agimus. A measurement of the robot's state and the analysis of an image of the work area would inform the framework on the success or failure of the previous task, and entail the planning of a corrective action if necessary.

### **Towards an industrial demonstration**

Agimus has been demonstrated on a humanoid robot and then on a mobile robot equipped with a manipulator arm. Those are research robots that are not present in the aeronautic industry, where all the robots are huge manipulator arms. There are still several steps before the framework can be safely used on such systems. One of those step is to try Agimus on a manipulator arm. This work has recently began with a UR10 robot, for a project linked with the ROB4FAM joint laboratory. The major envisioned difficulty when switching to industrial robot is the absence of open-source environment and access to the low level data from the sensors, or to the direct control of the motors.





# Bibliography

- [1] **Alexis Nicolin**, Joseph Mirabel, Sébastien Boria, Olivier Stasse, and Florent Lamiroux. “Agimus: A new framework for mapping manipulation motion plans to sequences of hierarchical task-based controllers”. In: *2020 IEEE/SICE International Symposium on System Integration (SII)*. IEEE. 2020, pp. 1022–1027 (cit. on pp. 7, 74).
- [2] Joseph Mirabel, Florent Lamiroux, Thuc Long Ha, **Alexis Nicolin**, Olivier Stasse, and Sébastien Boria. “Performing manufacturing tasks with a mobile manipulator: from motion planning to sensor based motion control”. In: *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*. IEEE. 2021, pp. 159–164 (cit. on pp. 7, 100).
- [3] Éric Marchand, Fabien Spindler, and François Chaumette. “ViSP for visual servoing: A generic software platform with a wide class of robot control skills”. In: *iee robotics & automation magazine* 12.4 (2005), pp. 40–52 (cit. on pp. 3, 32, 53, 132, 134).
- [4] Justin Carpentier et al. “The Pinocchio C++ library: A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives”. In: *2019 IEEE/SICE International Symposium on System Integration (SII)*. IEEE. 2019, pp. 614–619 (cit. on pp. 3, 53, 54).
- [5] Florent Lamiroux and Joseph Mirabel. *Prehensile manipulation planning: modeling, algorithms and implementation*. To appear in the IEEE Transactions on Robotics. Nov. 2020. URL: <https://hal.laas.fr/hal-02995125> (cit. on pp. 4, 38, 46, 53, 55, 131, 133).
- [6] Nicolas Mansard, Olivier Stasse, Paul Evrard, and Abderrahmane Kheddar. “A versatile generalized inverted kinematics implementation for collaborative working humanoid robots: The stack of tasks”. In: *2009 international conference on advanced robotics*. iee. 2009, pp. 1–6 (cit. on pp. 4, 31, 53, 56, 132, 134).
- [7] Claude Samson, Bernard Espiau, and Michel Le Borgne. *Robot control: the task function approach*. Oxford University Press, Inc., 1991 (cit. on pp. 4, 31, 46, 56).
- [8] Morgan Quigley et al. “ROS: An open-source robot operating system”. In: *icra workshop on open source software*. Vol. 3. 3.2. kobe, japan. 2009, p. 5 (cit. on pp. 4, 11, 134).

- [9] Olivier Stasse et al. “TALOS: A new humanoid research platform targeted for industrial applications”. In: *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*. IEEE. 2017, pp. 689–695 (cit. on pp. 4, 63, 65).
- [10] Jordi Pages, Luca Marchionni, and Francesco Ferro. “TIAGo: The modular robot that adapts to different research needs”. In: *International Workshop on Robot Modularity, IROS*. 2016 (cit. on pp. 5, 93).
- [11] Jacob T Schwartz and Micha Sharir. “On the “piano movers” problem I. The case of a two-dimensional rigid polygonal body moving amidst polygonal barriers”. In: *Communications on Pure and Applied Mathematics* 36.3 (1983), pp. 345–398 (cit. on pp. 10, 12).
- [12] Jacob T Schwartz and Micha Sharir. “On the “piano movers” problem. II. General techniques for computing topological properties of real algebraic manifolds”. In: *Advances in Applied Mathematics* 4.3 (1983), pp. 298–351 (cit. on p. 10).
- [13] Jacob T Schwartz and Micha Sharir. “On the piano movers’ problem: III. Coordinating the motion of several independent bodies: The special case of circular bodies moving amidst polygonal barriers”. In: *The International Journal of Robotics Research* 2.3 (1983), pp. 46–75 (cit. on p. 10).
- [14] Micha Sharir and Elka Ariel-Sheffi. “On the Piano Movers’ problem: IV. Various decomposable two-dimensional motion-planning problems”. In: *Communications on Pure and Applied Mathematics* 37.4 (1984), pp. 479–493 (cit. on p. 10).
- [15] Jacob T Schwartz and Micha Sharir. “On the piano movers’ problem: V. The case of a rod moving in three-dimensional space amidst polyhedral obstacles”. In: *Communications on Pure and Applied Mathematics* 37.6 (1984), pp. 815–848 (cit. on p. 10).
- [16] Markus Rickert, Arne Sieverling, and Oliver Brock. “Balancing exploration and exploitation in sampling-based motion planning”. In: *IEEE Transactions on Robotics* 30.6 (2014), pp. 1305–1317 (cit. on p. 10).
- [17] Thierry Simeon, Jean-Paul Laumond, and Florent Lamiroux. “Move3D: A generic platform for path planning”. In: *Proceedings of the 2001 IEEE International Symposium on Assembly and Task Planning (ISATP2001). Assembly and Disassembly in the Twenty-first Century. (Cat. No. 01TH8560)*. IEEE. 2001, pp. 25–30 (cit. on pp. 10, 55).
- [18] Etienne Ferre and Jean-Paul Laumond. “An iterative diffusion algorithm for part disassembly”. In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA’04. 2004*. Vol. 3. IEEE. 2004, pp. 3149–3154 (cit. on p. 10).
- [19] Jean-Paul Laumond. “Kineo CAM: a success story of motion planning algorithms”. In: *IEEE Robotics & Automation Magazine* 13.2 (2006), pp. 90–93 (cit. on p. 10).

- [20] Florent Lamiroux, Jean-Paul Laumond, Carl Van Geem, Daniel Boutonnet, and Gilbert Raust. “Trailer truck trajectory optimization: the transportation of components for the Airbus A380”. In: *IEEE robotics & automation magazine* 12.1 (2005), pp. 14–21 (cit. on p. 10).
- [21] *URDF XML specifications*. URL: <http://wiki.ros.org/urdf/XML> (cit. on p. 11).
- [22] Tomás Lozano-Pérez and Michael A Wesley. “An algorithm for planning collision-free paths among polyhedral obstacles”. In: *Communications of the ACM* 22.10 (1979), pp. 560–570 (cit. on p. 11).
- [23] Tomas Lozano-Perez. “Spatial planning: A configuration space approach”. In: *Autonomous robot vehicles*. Springer, 1990, pp. 259–271 (cit. on p. 11).
- [24] Jean-Claude Latombe. *Robot motion planning*. Vol. 124. Springer Science & Business Media, 2012 (cit. on p. 11).
- [25] Howie Choset, Kevin M Lynch, Seth Hutchinson, George A Kantor, and Wolfram Burgard. *Principles of robot motion: Theory, algorithms, and implementations*. MIT press, 2005 (cit. on p. 11).
- [26] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006 (cit. on p. 11).
- [27] Osamu Takahashi and Robert J Schilling. “Motion planning in a plane using generalized Voronoi diagrams”. In: *IEEE Transactions on robotics and automation* 5.2 (1989), pp. 143–150 (cit. on p. 12).
- [28] John Canny. *The complexity of robot motion planning*. MIT press, 1988 (cit. on p. 12).
- [29] Kamal Kant and Steven W Zucker. “Toward efficient trajectory planning: The path-velocity decomposition”. In: *The international journal of robotics research* 5.3 (1986), pp. 72–89 (cit. on p. 12).
- [30] Edsger W Dijkstra et al. “A note on two problems in connexion with graphs”. In: *Numerische mathematik* 1.1 (1959), pp. 269–271 (cit. on p. 12).
- [31] Peter E Hart, Nils J Nilsson, and Bertram Raphael. “A formal basis for the heuristic determination of minimum cost paths”. In: *IEEE transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107 (cit. on p. 12).
- [32] Oussama Khatib. “Real-time obstacle avoidance for manipulators and mobile robots”. In: *Autonomous robot vehicles*. Springer, 1986, pp. 396–404 (cit. on p. 12).
- [33] Shuzhi Sam Ge and Yun J Cui. “Dynamic motion planning for mobile robots using potential field method”. In: *Autonomous robots* 13.3 (2002), pp. 207–222 (cit. on p. 12).
- [34] Liangjun Zhang, Steven M LaValle, and Dinesh Manocha. “Global vector field computation for feedback motion planning”. In: *2009 IEEE International Conference on Robotics and Automation*. IEEE, 2009, pp. 477–482 (cit. on p. 12).
- [35] Steven M LaValle et al. “Rapidly-exploring random trees: A new tool for path planning”. In: (1998) (cit. on p. 12).

- 
- [36] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. “Probabilistic roadmaps for path planning in high-dimensional configuration spaces”. In: *IEEE transactions on Robotics and Automation* 12.4 (1996), pp. 566–580 (cit. on p. 13).
- [37] Thierry Siméon, Jean-Paul Laumond, and Carole Nissoux. “Visibility-based probabilistic roadmaps for motion planning”. In: *Advanced Robotics* 14.6 (2000), pp. 477–493 (cit. on p. 13).
- [38] Junli Gao, Weijie Ye, Jing Guo, and Zhongjuan Li. “Deep reinforcement learning for indoor mobile robot path planning”. In: *Sensors* 20.19 (2020), p. 5493 (cit. on p. 13).
- [39] Sébastien Dalibard and Jean-Paul Laumond. “Linear dimensionality reduction in random motion planning”. In: *The International Journal of Robotics Research* 30.12 (2011), pp. 1461–1476 (cit. on p. 13).
- [40] David Hsu, Tingting Jiang, John Reif, and Zheng Sun. “The bridge test for sampling narrow passages with probabilistic roadmap planners”. In: *2003 IEEE international conference on robotics and automation (cat. no. 03CH37422)*. Vol. 3. IEEE. 2003, pp. 4420–4426 (cit. on p. 13).
- [41] Dmitry Berenson, Siddhartha S Srinivasa, Dave Ferguson, and James J Kuffner. “Manipulation planning on constraint manifolds”. In: *2009 IEEE International Conference on Robotics and Automation*. IEEE. 2009, pp. 625–632 (cit. on p. 13).
- [42] Sertac Karaman and Emilio Frazzoli. “Sampling-based algorithms for optimal motion planning”. In: *The international journal of robotics research* 30.7 (2011), pp. 846–894 (cit. on p. 14).
- [43] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. “Informed RRT\*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic”. In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2014, pp. 2997–3004 (cit. on p. 14).
- [44] Lucas Janson, Edward Schmerling, Ashley Clark, and Marco Pavone. “Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions”. In: *The International journal of robotics research* 34.7 (2015), pp. 883–921 (cit. on p. 14).
- [45] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. “Batch informed trees (BIT\*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs”. In: *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2015, pp. 3067–3074 (cit. on p. 14).
- [46] Lucas Janson, Brian Ichter, and Marco Pavone. “Deterministic sampling-based motion planning: Optimality, complexity, and performance”. In: *The International Journal of Robotics Research* 37.1 (2018), pp. 46–61 (cit. on p. 14).
- [47] Oren Salzman and Dan Halperin. “Asymptotically near-optimal RRT for fast, high-quality motion planning”. In: *IEEE Transactions on Robotics* 32.3 (2016), pp. 473–483 (cit. on p. 14).

- [48] Brian Ichter, James Harrison, and Marco Pavone. “Learning sampling distributions for robot motion planning”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 7087–7094 (cit. on p. 14).
- [49] Matt Zucker et al. “Chomp: Covariant hamiltonian optimization for motion planning”. In: *The International Journal of Robotics Research* 32.9-10 (2013), pp. 1164–1193 (cit. on p. 15).
- [50] Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal. “STOMP: Stochastic trajectory optimization for motion planning”. In: *2011 IEEE international conference on robotics and automation*. IEEE. 2011, pp. 4569–4574 (cit. on p. 15).
- [51] John Schulman et al. “Motion planning with sequential convex optimization and convex collision checking”. In: *The International Journal of Robotics Research* 33.9 (2014), pp. 1251–1270 (cit. on p. 15).
- [52] John Schulman, Jonathan Ho, Alex X Lee, Ibrahim Awwal, Henry Bradlow, and Pieter Abbeel. “Finding locally optimal, collision-free trajectories with sequential convex optimization.” In: *Robotics: science and systems*. Vol. 9. 1. Citeseer. 2013, pp. 1–10 (cit. on p. 15).
- [53] Mustafa Mukadam, Jing Dong, Xinyan Yan, Frank Dellaert, and Byron Boots. “Continuous-time Gaussian process motion planning via probabilistic inference”. In: *The International Journal of Robotics Research* 37.11 (2018), pp. 1319–1340 (cit. on p. 15).
- [54] Diane Bury, Jean-Baptiste Izard, Marc Gouttefarde, and Florent Lamiraux. “Continuous tension validation for cable-driven parallel robots”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 6558–6563 (cit. on pp. 16, 55).
- [55] Mike Stilman. “Global manipulation planning in robot joint space with task constraints”. In: *IEEE Transactions on Robotics* 26.3 (2010), pp. 576–584 (cit. on pp. 18, 19).
- [56] Steven M LaValle, Jeffery H Yakey, and Lydia E Kavraki. “A probabilistic roadmap approach for systems with closed kinematic chains”. In: *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*. Vol. 3. IEEE. 1999, pp. 1671–1676 (cit. on p. 18).
- [57] Beobkyyoon Kim, Terry Taewoong Um, Chansu Suh, and Frank C Park. “Tangent bundle RRT: A randomized algorithm for constrained motion planning”. In: *Robotica* 34.1 (2016), pp. 202–225 (cit. on p. 19).
- [58] Léonard Jaillet and Josep M Porta. “Path planning under kinematic constraints by rapidly exploring manifolds”. In: *IEEE Transactions on Robotics* 29.1 (2012), pp. 105–117 (cit. on p. 19).
- [59] Ricard Bordalba, Lluís Ros, and Josep M Porta. “A randomized kinodynamic planner for closed-chain robotic systems”. In: *IEEE Transactions on Robotics* 37.1 (2020), pp. 99–115 (cit. on p. 19).
- [60] Kris K Hauser. “Fast interpolation and time-optimization on implicit contact submanifolds.” In: *Robotics: Science and systems*. Citeseer. 2013, p. 22 (cit. on p. 19).

- 
- [61] Joseph Mirabel and Florent Lamiraux. “Manipulation planning: building paths on constrained manifolds”. 2016 (cit. on p. 20).
- [62] Joseph Mirabel. “Manipulation planning for documented objects”. PhD thesis. Institut National Polytechnique De Toulouse, 2017 (cit. on p. 20).
- [63] Zachary Kingston, Mark Moll, and Lydia E Kavraki. “Exploring implicit spaces for constrained sampling-based planning”. In: *The International Journal of Robotics Research* 38.10-11 (2019), pp. 1151–1178 (cit. on p. 20).
- [64] Manuel Kudruss et al. “Optimal control for whole-body motion generation using center-of-mass dynamics for predefined multi-contact configurations”. In: *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*. IEEE. 2015, pp. 684–689 (cit. on p. 20).
- [65] Shuuji Kajita et al. “Biped walking pattern generation by using preview control of zero-moment point”. In: *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*. Vol. 2. IEEE. 2003, pp. 1620–1626 (cit. on p. 20).
- [66] Wael Suleiman, Eiichi Yoshida, Jean-Paul Laumond, and André Monin. “On humanoid motion optimization”. In: *2007 7th IEEE-RAS International Conference on Humanoid Robots*. IEEE. 2007, pp. 180–187 (cit. on p. 20).
- [67] Emanuel Todorov, Tom Erez, and Yuval Tassa. “MuJoCo: A physics engine for model-based control”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 5026–5033 (cit. on pp. 20, 32).
- [68] Sébastien Dalibard, Antonio El Khoury, Florent Lamiraux, Alireza Nakhaei, Michel Taïx, and Jean-Paul Laumond. “Dynamic walking and whole-body motion planning for humanoid robots: An integrated approach”. In: *The International Journal of Robotics Research* 32.9-10 (2013), pp. 1089–1103 (cit. on pp. 20, 32).
- [69] Nicolas Perrin, Olivier Stasse, Léo Baudouin, Florent Lamiraux, and Eiichi Yoshida. “Fast humanoid robot collision-free footstep planning using swept volume approximations”. In: *IEEE Transactions on Robotics* 28.2 (2011), pp. 427–439 (cit. on p. 21).
- [70] Robert J Griffin, Georg Wiedebach, Stephen McCrory, Sylvain Bertrand, Inho Lee, and Jerry Pratt. “Footstep planning for autonomous walking over rough terrain”. In: *2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids)*. IEEE. 2019, pp. 9–16 (cit. on p. 21).
- [71] Marco Cagnetti, Pouya Mohammadi, and Giuseppe Oriolo. “Whole-body motion planning for humanoids based on CoM movement primitives”. In: *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*. IEEE. 2015, pp. 1090–1095 (cit. on p. 21).
- [72] Paolo Ferrari, Marco Cagnetti, and Giuseppe Oriolo. “Humanoid whole-body planning for loco-manipulation tasks”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 4741–4746 (cit. on p. 21).

- [73] Steve Tonneau, Nicolas Mansard, Chonhyon Park, Dinesh Manocha, Franck Multon, and Julien Pettré. “A reachability-based planner for sequences of acyclic contacts in cluttered environments”. In: *Robotics Research*. Springer, 2018, pp. 287–303 (cit. on p. 21).
- [74] Pierre Fernbach. “Modèles réduits fiables et efficaces pour la planification et l’optimisation de mouvement des robots à pattes en environnements contraints”. PhD thesis. Université Paul Sabatier-Toulouse III, 2018 (cit. on p. 21).
- [75] Pierre Fernbach, Steve Tonneau, and Michel Taïx. “CROC: Convex resolution of centroidal dynamics trajectories to provide a feasibility criterion for the multi contact planning problem”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 1–9 (cit. on p. 21).
- [76] Justin Carpentier and Nicolas Mansard. “Multicontact locomotion of legged robots”. In: *IEEE Transactions on Robotics* 34.6 (2018), pp. 1441–1460 (cit. on p. 21).
- [77] Yu-Chi Lin, Brahayam Ponton, Ludovic Righetti, and Dmitry Berenson. “Efficient humanoid contact planning using learned centroidal dynamics prediction”. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 5280–5286 (cit. on p. 21).
- [78] Brahayam Ponton, Majid Khadiv, Avadesh Meduri, and Ludovic Righetti. “Efficient multicontact pattern generation with sequential convex approximations of the centroidal dynamics”. In: *IEEE Transactions on Robotics* (2021) (cit. on pp. 21, 32).
- [79] Allen Newell and Herbert Simon. “The logic theory machine – A complex information processing system”. In: *IRE Transactions on information theory* 2.3 (1956), pp. 61–79 (cit. on p. 22).
- [80] Allen Newell, John C Shaw, and Herbert A Simon. “Report on a general problem solving program”. In: *IFIP congress*. Vol. 256. Pittsburgh, PA. 1959, p. 64 (cit. on pp. 22, 133).
- [81] Richard E Fikes and Nils J Nilsson. “STRIPS: A new approach to the application of theorem proving to problem solving”. In: *Artificial intelligence* 2.3-4 (1971), pp. 189–208 (cit. on pp. 22, 134).
- [82] Drew McDermott et al. *PDDL - The planning domain definition language*. 1998 (cit. on p. 23).
- [83] International Conference on Automated Planning and Scheduling. *International Planning Competition*. URL: <https://www.icaps-conference.org/competitions/> (visited on 11/28/2021) (cit. on pp. 23, 134).
- [84] Maria Fox and Derek Long. “PDDL2. 1: An extension to PDDL for expressing temporal planning domains”. In: *Journal of artificial intelligence research* 20 (2003), pp. 61–124 (cit. on p. 23).
- [85] Avrim L Blum and Merrick L Furst. “Fast planning through planning graph analysis”. In: *Artificial intelligence* 90.1-2 (1997), pp. 281–300 (cit. on p. 25).



- 
- [86] Henry Kautz, David McAllester, and Bart Selman. “Encoding plans in propositional logic”. In: *KR 96* (1996), pp. 374–384 (cit. on p. 25).
- [87] Henry Kautz and Bart Selman. “Unifying SAT-based and graph-based planning”. In: *IJCAI*. Vol. 99. 1999, pp. 318–325 (cit. on p. 26).
- [88] Blai Bonet and Hector Geffner. “Planning as heuristic search: New results”. In: *European Conference on Planning*. Springer. 1999, pp. 360–372 (cit. on p. 26).
- [89] Jörg Hoffmann and Bernhard Nebel. “The FF planning system: Fast plan generation through heuristic search”. In: *Journal of Artificial Intelligence Research* 14 (2001), pp. 253–302 (cit. on p. 26).
- [90] Yixin Chen, Chih-Wei Hsu, and Benjamin W Wah. “SGPlan: Subgoal partitioning and resolution in planning”. In: *Edelkamp et al. (Edelkamp, Hoffmann, Littman, & Younes, 2004)* (2004) (cit. on p. 26).
- [91] Malte Helmert. “The fast downward planning system”. In: *Journal of Artificial Intelligence Research* 26 (2006), pp. 191–246 (cit. on p. 26).
- [92] Malte Helmert. “A Planning Heuristic Based on Causal Graph Analysis.” In: *ICAPS*. Vol. 16. 2004, pp. 161–170 (cit. on p. 26).
- [93] Tomas Lozano-Perez. “A simple motion-planning algorithm for general robot manipulators”. In: *IEEE Journal on Robotics and Automation* 3.3 (1987), pp. 224–238 (cit. on p. 27).
- [94] Gordon Wilfong. “Motion planning in the presence of movable obstacles”. In: *Annals of Mathematics and Artificial Intelligence* 3.1 (1991), pp. 131–150 (cit. on p. 27).
- [95] Rachid Alami, Thierry Simeon, and Jean-Paul Laumond. “A geometrical approach to planning manipulation tasks. The case of discrete placements and grasps”. In: *The fifth international symposium on Robotics research*. MIT Press. 1990, pp. 453–463 (cit. on pp. 27, 28).
- [96] Karim Bouyarmane, Stéphane Caron, Adrien Escande, and Abderrahmane Kheddar. “Humanoid Robotics: A Reference”. In: ed. by Ambarish Goswami and Prahlad Vadakkepat. Springer, 2018. Chap. Multi-contact Motion Planning and Control (cit. on p. 27).
- [97] Mike Stilman and James J Kuffner. “Navigation among movable obstacles: Real-time reasoning in complex environments”. In: *International Journal of Humanoid Robotics* 2.04 (2005), pp. 479–503 (cit. on p. 27).
- [98] Dennis Nieuwenhuisen, A Frank Van Der Stappen, and Mark H Overmars. “An effective framework for path planning amidst movable obstacles”. In: *Algorithmic Foundation of Robotics VII*. Springer, 2008, pp. 87–102 (cit. on p. 27).
- [99] Mike Stilman and James Kuffner. “Planning among movable obstacles with artificial constraints”. In: *The International Journal of Robotics Research* 27.11-12 (2008), pp. 1295–1307 (cit. on p. 27).

- [100] Jur Van Den Berg, Mike Stilman, James Kuffner, Ming Lin, and Dinesh Manocha. “Path planning among movable obstacles: a probabilistically complete approach”. In: *Algorithmic Foundation of Robotics VIII*. Springer, 2009, pp. 599–614 (cit. on p. 27).
- [101] Martin Levihn, Jonathan Scholz, and Mike Stilman. “Planning with movable obstacles in continuous environments with uncertain dynamics”. In: *2013 IEEE International Conference on Robotics and Automation*. IEEE. 2013, pp. 3832–3838 (cit. on p. 27).
- [102] Martin Levihn, Mike Stilman, and Henrik Christensen. “Locally optimal navigation among movable obstacles in unknown environments”. In: *2014 IEEE-RAS International Conference on Humanoid Robots*. IEEE. 2014, pp. 86–91 (cit. on p. 27).
- [103] Sébastien Dalibard, Alireza Nakhaei, Florent Lamiroux, and Jean-Paul Laumond. “Manipulation of documented objects by a walking humanoid robot”. In: *2010 10th IEEE-RAS International Conference on Humanoid Robots*. IEEE. 2010, pp. 518–523 (cit. on p. 27).
- [104] Athanasios Krontiris and Kostas E Bekris. “Dealing with difficult instances of object rearrangement.” In: *Robotics: Science and Systems*. Vol. 1123. 2015 (cit. on p. 27).
- [105] Jun Ota. “Rearrangement of multiple movable objects-integration of global and local planning methodology”. In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA’04. 2004*. Vol. 2. IEEE. 2004, pp. 1962–1967 (cit. on p. 27).
- [106] Changkyu Song and Abdeslam Boularias. “Object rearrangement with nested nonprehensile manipulation actions”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019, pp. 6578–6585 (cit. on p. 27).
- [107] Shuai D Han, Nicholas M Stiffler, Athanasios Krontiris, Kostas E Bekris, and Jingjin Yu. “Complexity results and fast methods for optimal tabletop rearrangement with overhand grasps”. In: *The International Journal of Robotics Research* 37.13-14 (2018), pp. 1775–1795 (cit. on p. 27).
- [108] Weiwei Wan, Hisashi Igawa, Kensuke Harada, Hiromu Onda, Kazuyuki Nagata, and Natsuki Yamanobe. “A regrasp planning component for object reorientation”. In: *Autonomous Robots* 43.5 (2019), pp. 1101–1115 (cit. on p. 27).
- [109] Weiwei Wan, Kensuke Harada, and Fumio Kanehiro. “Preparatory manipulation planning using automatically determined single and dual arm”. In: *IEEE Transactions on Industrial Informatics* 16.1 (2019), pp. 442–453 (cit. on p. 27).
- [110] Joshua A Haustein, Isac Arnekvist, Johannes Stork, Kaiyu Hang, and Danica Kragic. “Learning manipulation states and actions for efficient non-prehensile rearrangement planning”. In: *arXiv preprint arXiv:1901.03557* (2019) (cit. on p. 27).

- 
- [111] Jur Van Den Berg, Jack Snoeyink, Ming C Lin, and Dinesh Manocha. “Centralized path planning for multiple robots: Optimal decoupling into sequential plans.” In: *Robotics: Science and systems*. Vol. 2. 2.5. 2009, pp. 2–3 (cit. on p. 27).
- [112] Mokhtar Gharbi, Juan Cortés, and Thierry Siméon. “Roadmap composition for multi-arm systems path planning”. In: *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2009, pp. 2471–2476 (cit. on p. 28).
- [113] Kiril Solovey, Oren Salzman, and Dan Halperin. “Finding a needle in an exponential haystack: Discrete RRT for exploration of implicit roadmaps in multi-robot motion planning”. In: *Algorithmic Foundations of Robotics XI*. Springer, 2015, pp. 591–607 (cit. on p. 28).
- [114] Rahul Shome and Kostas E Bekris. “Anytime multi-arm task and motion planning for pick-and-place of individual objects via handoffs”. In: *2019 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*. IEEE. 2019, pp. 37–43 (cit. on p. 28).
- [115] Rahul Shome, Kiril Solovey, Andrew Dobson, Dan Halperin, and Kostas E Bekris. “d-RRT\*: Scalable and informed asymptotically-optimal multi-robot motion planning”. In: *Autonomous Robots* 44.3 (2020), pp. 443–467 (cit. on p. 28).
- [116] Joseph Mirabel et al. “HPP: A new software for constrained motion planning”. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, pp. 383–389 (cit. on p. 28, 55).
- [117] Siddharth Srivastava, Eugene Fang, Lorenzo Riano, Rohan Chitnis, Stuart Russell, and Pieter Abbeel. “Combined task and motion planning through an extensible planner-independent interface layer”. In: *2014 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2014, pp. 639–646 (cit. on p. 29).
- [118] Thierry Siméon, Jean-Paul Laumond, Juan Cortés, and Anis Sahbani. “Manipulation planning with probabilistic roadmaps”. In: *The International Journal of Robotics Research* 23.7-8 (2004), pp. 729–746 (cit. on p. 29).
- [119] Benoit Dacre-Wright, Jean-Paul Laumond, and Rachid Alami. “Motion planning for a robot and a movable object amidst polygonal obstacles”. In: *Proceedings 1992 IEEE International Conference on Robotics and Automation*. IEEE Computer Society. 1992, pp. 2474–2475 (cit. on p. 29).
- [120] Stephane Cambon, Rachid Alami, and Fabien Gravot. “A hybrid approach to intricate motion, manipulation and task planning”. In: *The International Journal of Robotics Research* 28.1 (2009), pp. 104–126 (cit. on p. 29).
- [121] Caelan Reed Garrett, Tomas Lozano-Perez, and Leslie Pack Kaelbling. “FFRob: Leveraging symbolic planning for efficient task and motion planning”. In: *The International Journal of Robotics Research* 37.1 (2018), pp. 104–136 (cit. on p. 30).

- [122] Neil T Dantam, Zachary K Kingston, Swarat Chaudhuri, and Lydia E Kavraki. “An incremental constraint-based framework for task and motion planning”. In: *The International Journal of Robotics Research* 37.10 (2018), pp. 1134–1151 (cit. on p. 30).
- [123] andrew M Wells, Neil T Dantam, Anshumali Shrivastava, and Lydia E Kavraki. “Learning feasibility for task and motion planning in tabletop environments”. In: *IEEE robotics and automation letters* 4.2 (2019), pp. 1255–1262 (cit. on p. 30).
- [124] Beomjoon Kim, Zi Wang, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. “Learning to guide task and motion planning using score-space representation”. In: *The International Journal of Robotics Research* 38.7 (2019), pp. 793–812 (cit. on p. 30).
- [125] Michael Görner, Robert Haschke, Helge Ritter, and Jianwei Zhang. “MoveIt! task constructor for task-level motion planning”. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 190–196 (cit. on p. 30).
- [126] Samer Alfayad, Fethi B Ouezdou, Faycal Namoun, and Gordon Gheng. “High performance integrated electro-hydraulic actuator for robotics. Part I: Principle, prototype design and first experiments”. In: *Sensors and Actuators A: Physical* 169.1 (2011), pp. 115–123 (cit. on p. 31).
- [127] Samer Alfayad, Fethi B Ouezdou, Faycal Namoun, and Gordon Gheng. “High performance integrated electro-hydraulic actuator for robotics. Part II: Theoretical modelling, simulation, control & comparison with real measurements”. In: *Sensors and Actuators A: Physical* 169.1 (2011), pp. 124–132 (cit. on p. 31).
- [128] Felix Grimmering et al. “An open torque-controlled modular robot architecture for legged locomotion research”. In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 3650–3657 (cit. on p. 31).
- [129] andrea Del Prete, Nicolas Mansard, Oscar E Ramos, Olivier Stasse, and Francesco Nori. “Implementing torque control with high-ratio gear boxes and without joint-torque sensors”. In: *International Journal of Humanoid Robotics* 13.01 (2016) (cit. on p. 31).
- [130] Noélie Ramuzat, Florent Forget, Vincent Bonnet, M Gautier, S Boria, and Olivier Stasse. “Actuator model, identification and differential dynamic programming for a talos humanoid robot”. In: *2020 European Control Conference (ECC)*. IEEE. 2020, pp. 724–730 (cit. on p. 31).
- [131] Richard P Paul. *Robot manipulators: Mathematics, programming, and control: The computer control of robot manipulators*. Richard Paul, 1981 (cit. on p. 31).
- [132] Oussama Khatib. “A unified approach for motion and force control of robot manipulators: The operational space formulation”. In: *IEEE Journal on Robotics and Automation* 3.1 (1987), pp. 43–53 (cit. on p. 31).
- [133] Yoshihiko Nakamura and Hideo Hanafusa. “Optimal redundancy control of robot manipulators”. In: *The International Journal of Robotics Research* 6.1 (1987), pp. 32–42 (cit. on pp. 31, 56).

- 
- [134] Bruno Siciliano and J-JE Slotine. “A general framework for managing multiple tasks in highly redundant robotic systems”. In: *Fifth International Conference on Advanced Robotics’ Robots in Unstructured Environments*. IEEE. 1991, pp. 1211–1216 (cit. on p. 31).
- [135] Karim Bouyarmane and Abderrahmane Kheddar. “On weight-prioritized multitask control of humanoid robots”. In: *IEEE Transactions on Automatic Control* 63.6 (2017), pp. 1632–1647 (cit. on p. 31).
- [136] Philip E Gill, Walter Murray, and Michael A Saunders. “SNOPT: An SQP algorithm for large-scale constrained optimization”. In: *SIAM review* 47.1 (2005), pp. 99–131 (cit. on p. 32).
- [137] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006 (cit. on p. 32).
- [138] Philip E Gill, Walter Murray, and Margaret H Wright. *Practical optimization*. SIAM, 2019 (cit. on p. 32).
- [139] Michael JD Powell. “A fast algorithm for nonlinearly constrained optimization calculations”. In: *Numerical analysis*. Springer, 1978, pp. 144–157 (cit. on p. 32).
- [140] Robert J Vanderbei and David F Shanno. “An interior-point algorithm for nonconvex nonlinear programming”. In: *Computational Optimization and Applications* 13.1 (1999), pp. 231–252 (cit. on p. 32).
- [141] Richard H Byrd, Mary E Hribar, and Jorge Nocedal. “An interior point algorithm for large-scale nonlinear programming”. In: *SIAM Journal on Optimization* 9.4 (1999), pp. 877–900 (cit. on p. 32).
- [142] Adrien Escande, Nicolas Mansard, and Pierre-Brice Wieber. “Hierarchical quadratic programming: Fast online humanoid-robot motion generation”. In: *The International Journal of Robotics Research* 33.7 (2014), pp. 1006–1028 (cit. on pp. 32, 48).
- [143] Ixchel G Ramirez-Alpizar et al. “Motion generation for pulling a fire hose by a humanoid robot”. In: *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*. IEEE. 2016, pp. 1016–1021 (cit. on p. 32).
- [144] Mingxing Liu, Yang Tan, and Vincent Padois. “Generalized hierarchical control”. In: *Autonomous Robots* 40.1 (2016), pp. 17–31 (cit. on p. 32).
- [145] Hongkai Dai, andrés Valenzuela, and Russ Tedrake. “Whole-body motion planning with centroidal dynamics and full kinematics”. In: *2014 IEEE-RAS International Conference on Humanoid Robots*. IEEE. 2014, pp. 295–302 (cit. on p. 32).
- [146] Maximilien Naveau, Manuel Kudruss, Olivier Stasse, Christian Kirches, Katja Mombaur, and Philippe Souères. “A reactive walking pattern generator based on nonlinear model predictive control”. In: *IEEE Robotics and Automation Letters* 2.1 (2016), pp. 10–17 (cit. on p. 32).
- [147] Carlos Mastalli et al. “Crocodyl: An efficient and versatile framework for multi-contact optimal control”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 2536–2542 (cit. on pp. 32, 55, 56).

- [148] Carlos Mastalli et al. *Crocoddyl: A fast and flexible optimal control library for robot control under contact sequence*. 2019. URL: <https://github.com/loco-3d/crocoddyl/wikis/home> (cit. on pp. 32, 55, 56).
- [149] He Li, Robert J Frei, and Patrick M Wensing. “Model hierarchy predictive control of robotic systems”. In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 3373–3380 (cit. on p. 32).
- [150] Wael Suleiman, Eiichi Yoshida, Fumio Kanehiro, Jean-Paul Laumond, and André Monin. “On human motion imitation by humanoid robot”. In: *2008 IEEE International conference on robotics and automation*. IEEE. 2008, pp. 2697–2704 (cit. on p. 32).
- [151] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel Van De Panne. “Deepmimic: Example-guided deep reinforcement learning of physics-based character skills”. In: *ACM Transactions on Graphics (TOG)* 37.4 (2018), pp. 1–14 (cit. on p. 32).
- [152] Kevin Bergamin, Simon Clavet, Daniel Holden, and James Richard Forbes. “DReCon: data-driven responsive control of physics-based characters”. In: *ACM Transactions On Graphics (TOG)* 38.6 (2019), pp. 1–11 (cit. on p. 32).
- [153] Nicolas Heess et al. “Emergence of locomotion behaviours in rich environments”. In: *arXiv preprint arXiv:1707.02286* (2017) (cit. on p. 32).
- [154] Xue Bin Peng, Glen Berseth, KangKang Yin, and Michiel Van De Panne. “Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning”. In: *ACM Transactions on Graphics (TOG)* 36.4 (2017), pp. 1–13 (cit. on p. 32).
- [155] Ilge Akkaya et al. “Solving Rubik’s cube with a robot hand”. In: *arXiv preprint arXiv:1910.07113* (2019) (cit. on p. 32).
- [156] François Chaumette and Seth Hutchinson. “Visual servo control. I. Basic approaches”. In: *IEEE Robotics & Automation Magazine* 13.4 (2006), pp. 82–90 (cit. on p. 32).
- [157] Seth Hutchinson, Gregory D Hager, and Peter I Corke. “A tutorial on visual servo control”. In: *IEEE transactions on robotics and automation* 12.5 (1996), pp. 651–670 (cit. on p. 32).
- [158] François Chaumette and Seth Hutchinson. “Visual servo control. II. Advanced approaches”. In: *IEEE Robotics & Automation Magazine* 14.1 (2007), pp. 109–118 (cit. on p. 32).
- [159] Roger Tsai. “A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses”. In: *IEEE Journal on Robotics and Automation* 3.4 (1987), pp. 323–344 (cit. on p. 33).
- [160] Zhengyou Zhang. “A flexible new technique for camera calibration”. In: *IEEE Transactions on pattern analysis and machine intelligence* 22.11 (2000), pp. 1330–1334 (cit. on p. 33).
- [161] Peter Sturm and Srikumar Ramalingam. *Camera models and fundamental concepts used in geometric computer vision*. Now Publishers Inc, 2011 (cit. on p. 33).

- [162] John Canny. “A computational approach to edge detection”. In: *IEEE Transactions on pattern analysis and machine intelligence* 6 (1986), pp. 679–698 (cit. on p. 33).
- [163] Chris Harris, Mike Stephens, et al. “A combined corner and edge detector”. In: *Alvey vision conference*. Vol. 15. Citeseer. 1988 (cit. on p. 33).
- [164] Jianbo Shi et al. “Good features to track”. In: *1994 Proceedings of IEEE conference on computer vision and pattern recognition*. IEEE. 1994, pp. 593–600 (cit. on p. 33).
- [165] Martin A Fischler and Robert C Bolles. “Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography”. In: *Communications of the ACM* 24.6 (1981), pp. 381–395 (cit. on pp. 33, 87).
- [166] Daniel F DeMenthon and Larry S Davis. “Model-based object pose in 25 lines of code”. In: *International journal of computer vision* 15.1-2 (1995), pp. 123–141 (cit. on p. 33).
- [167] Ulrich Neumann and Youngkwan Cho. “A self-tracking augmented reality system”. In: *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*. 1996, pp. 109–115 (cit. on p. 33).
- [168] Hirokazu Kato and Mark Billinghurst. “Marker tracking and HMD calibration for a video-based augmented reality conferencing system”. In: *Proceedings 2nd IEEE and ACM International Workshop on Augmented Reality (IWAR'99)*. IEEE. 1999, pp. 85–94 (cit. on p. 33).
- [169] Jun Rekimoto. “Matrix: A realtime object identification and registration method for augmented reality”. In: *Proceedings. 3rd Asia Pacific Computer Human Interaction (Cat. No. 98EX110)*. IEEE. 1998, pp. 63–68 (cit. on p. 33).
- [170] Edwin Olson. “AprilTag: A robust and flexible visual fiducial system”. In: *2011 IEEE International Conference on Robotics and Automation*. IEEE. 2011, pp. 3400–3407 (cit. on p. 33).
- [171] John Wang and Edwin Olson. “AprilTag 2: Efficient and robust fiducial detection”. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, pp. 4193–4198 (cit. on pp. 33, 42, 87, 131).
- [172] Radu Horaud, Bernard Conio, Olivier Le Boulleux, and Bernard Lacolle. “An analytic solution for the perspective 4-point problem”. In: *Computer Vision, Graphics, and Image Processing* 47.1 (1989), pp. 33–44 (cit. on p. 34).
- [173] Denis Oberkampf, Daniel F DeMenthon, and Larry S Davis. “Iterative pose estimation using coplanar feature points”. In: *Computer Vision and Image Understanding* 63.3 (1996), pp. 495–511 (cit. on pp. 34, 87).
- [174] Long Quan and Zhongdan Lan. “Linear n-point camera pose determination”. In: *IEEE Transactions on pattern analysis and machine intelligence* 21.8 (1999), pp. 774–780 (cit. on p. 34).
- [175] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. “EPnP: An accurate  $O(n)$  solution to the PnP problem”. In: *International journal of computer vision* 81.2 (2009), p. 155 (cit. on pp. 34, 87).

- [176] Shiqi Li, Chi Xu, and Ming Xie. “A robust  $O(n)$  solution to the perspective-n-point problem”. In: *IEEE transactions on pattern analysis and machine intelligence* 34.7 (2012), pp. 1444–1450 (cit. on p. 34).
- [177] Nikolaos P Papanikolopoulos, Pradeep K Khosla, and Takeo Kanade. “Visual tracking of a moving target by a camera mounted on a robot: A combination of control and vision”. In: *IEEE transactions on robotics and automation* 9.1 (1993), pp. 14–35 (cit. on p. 34).
- [178] Ezio Malis, Francois Chaumette, and Sylvie Boudet. “2 1/2 D visual servoing”. In: *IEEE Transactions on Robotics and Automation* 15.2 (1999), pp. 238–250 (cit. on p. 34).
- [179] Bernard Espiau, François Chaumette, and Patrick Rives. “A new approach to visual servoing in robotics”. In: *IEEE Transactions on Robotics and Automation* 8.3 (1992), pp. 313–326 (cit. on p. 34).
- [180] Jonas Koenemann et al. “Whole-body model-predictive control applied to the HRP-2 humanoid”. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2015, pp. 3346–3351 (cit. on p. 34).
- [181] Viviane Cadenat, Philippe Souères, and Michel Courdesses. “An hybrid control for avoiding obstacles during a vision-based tracking task”. In: *1999 European Control Conference (ECC)*. IEEE. 1999, pp. 1114–1119 (cit. on p. 34).
- [182] Viviane Cadenat, Ricardo Swain, Philippe Soueres, and Michel Devy. “A controller to perform a visually guided tracking task in a cluttered environment”. In: *Proceedings 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human and Environment Friendly Robots with High Intelligence and Emotional Quotients (Cat. No. 99CH36289)*. Vol. 2. IEEE. 1999, pp. 775–780 (cit. on p. 34).
- [183] Jian Chen, Warren E Dixon, M Dawson, and Michael McIntyre. “Homography-based visual servo tracking control of a wheeled mobile robot”. In: *IEEE Transactions on Robotics* 22.2 (2006), pp. 406–415 (cit. on p. 34).
- [184] Jean-Bernard Hayet, Frédéric Lerasle, and Michel Devy. “A visual landmark framework for indoor mobile robot navigation”. In: *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*. Vol. 4. IEEE. 2002, pp. 3942–3947 (cit. on p. 34).
- [185] Jean-Bernard Hayet, Claudia Esteves, Gustavo Arechavaleta, Olivier Stasse, and Eiichi Yoshida. “Humanoid locomotion planning for visually guided tasks”. In: *International Journal of Humanoid Robotics* 9.02 (2012) (cit. on p. 34).
- [186] Moslem Kazemi, Kamal Gupta, and Mehran Mehrandezh. “Path-planning for visual servoing: A review and issues”. In: *Visual Servoing via Advanced Numerical Methods* (2010), pp. 189–207 (cit. on p. 34).
- [187] Fereshteh Sadeghi, Alexander Toshev, Eric Jang, and Sergey Levine. “Sim2real viewpoint invariant visual servoing by recurrent control”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4691–4699 (cit. on p. 34).



- 
- [188] Noé G Aldana-Murillo, Luis Sandoval, Jean-Bernard Hayet, Claudia Esteves, and Hector M Becerra. “Coupling humanoid walking pattern generation and visual constraint feedback for pose-regulation and visual path-following”. In: *Robotics and Autonomous Systems* 128 (2020), p. 103497 (cit. on p. 34).
- [189] Joseph Mirabel and Florent Lamiroux. “Manipulation planning: Addressing the crossed foliation issue”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 4032–4037 (cit. on pp. 38, 44).
- [190] Joseph Mirabel and Florent Lamiroux. “Handling implicit and explicit constraints in manipulation planning”. In: *robotics: science and systems 2018*. 2018, 9p (cit. on p. 43).
- [191] Mylène Campana, Florent Lamiroux, and Jean-Paul Laumond. “A gradient-based path optimization method for motion planning”. In: *advanced robotics* 30.17-18 (2016), pp. 1126–1144 (cit. on p. 46).
- [192] Tsuneo Yoshikawa. “Manipulability of robotic mechanisms”. In: *The international journal of Robotics Research* 4.2 (1985), pp. 3–9 (cit. on p. 49).
- [193] Sachin Chitta et al. “Ros\_control: A generic and simple control framework for ros”. In: *the journal of open source software* 2.20 (2017), pp. 456–456 (cit. on pp. 53, 57).
- [194] Roy Featherstone. *Rigid body dynamics algorithms*. Springer, 2014 (cit. on p. 54).
- [195] Jia Pan, Sachin Chitta, and Dinesh Manocha. “FCL: A general purpose library for collision and proximity queries”. In: *2012 IEEE International Conference on Robotics and Automation*. IEEE. 2012, pp. 3859–3866 (cit. on pp. 54, 133).
- [196] Justin Carpentier and Nicolas Mansard. “Analytical derivatives of rigid body dynamics algorithms”. In: *Robotics: Science and systems (RSS 2018)*. 2018 (cit. on p. 54).
- [197] *ROS Transform Library 2*. URL: <http://wiki.ros.org/tf2> (cit. on pp. 59, 134).
- [198] *Common Object Request Broker Architecture*. URL: <https://www.omg.org/spec/CORBA/> (cit. on p. 59).
- [199] International Electrotechnical Commission et al. “Industrial communication networks—fieldbus specifications—part 3–12: data-link layer service definition—part 4–12: datalink layer protocol specification—type 12 elements”. In: *iec, dec* 61.1 (2007), pp. 58–53 (cit. on p. 63).
- [200] David Coleman, Ioan Sucan, Sachin Chitta, and Nikolaus Correll. “Reducing the barrier to entry of complex robotic software: A MoveIt! case study”. In: *arXiv preprint arXiv:1404.3785* (2014) (cit. on p. 64).
- [201] Nathan Koenig and Andrew Howard. “Design and use paradigms for gazebo, an open-source multi-robot simulator”. In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No. 04ch37566)*. Vol. 3. IEEE. 2004, pp. 2149–2154 (cit. on p. 64).

- [202] *Archive of the DARPA Robotics Challenge (DRC) website*. URL: [https://web.archive.org/web/20130120060850/http://www.darpa.mil/our\\_work/tto/programs/darpa\\_robotics\\_challenge.aspx](https://web.archive.org/web/20130120060850/http://www.darpa.mil/our_work/tto/programs/darpa_robotics_challenge.aspx) (cit. on p. 64).
- [203] Kenji Kaneko et al. “Humanoid robot HRP-2Kai—improvement of HRP-2 towards disaster response tasks”. In: *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*. IEEE, 2015, pp. 132–139 (cit. on p. 65).
- [204] Jeongsoo Lim et al. “Robot system of DRC-Hubo+ and control strategy of team kaist in darpa robotics challenge finals”. In: *the darpa robotics challenge finals: humanoid robots to the rescue*. Springer, 2018, pp. 27–69 (cit. on p. 65).
- [205] *Boston Dynamics’ Atlas robot*. URL: <https://www.bostondynamics.com/atlas> (cit. on p. 65).
- [206] *Description of the new WALK-MAN robot*. URL: <https://spectrum.ieee.org/new-version-of-walkman-is-slimmer-quicker-better-at-quenching-your-flames> (cit. on p. 65).
- [207] Vincent Bonnet, Joseph Mirabel, David Daney, Florent Lamiroux, Maxime Gautier, and Olivier Stasse. *Practical whole-body elasto-geometric calibration of a humanoid robot. Application to the Talos robot*. Submitted to the IEEE Transactions on Robotics. Oct. 2021 (cit. on pp. 76, 83).
- [208] Pradeep K Khosla and Takeo Kanade. “Parameter identification of robot dynamics”. In: *1985 24th IEEE conference on decision and control*. IEEE, 1985, pp. 1754–1760 (cit. on p. 80).
- [209] Donna E Whitney, CA Lozinski, and Johnathan M Rourke. “Industrial robot forward calibration method and results”. In: (1986) (cit. on p. 80).
- [210] Jacques Denavit and Richard Scheunemann Hartenberg. “A kinematic notation for lower-pair mechanisms based on matrices”. In: *Journal of Applied Mechanics* 23 (1955), pp. 215–221 (cit. on pp. 80, 81, 133).
- [211] Wisama Khalil and J.F. Kleinfinger. “A new geometric notation for open and closed-loop robots”. In: vol. 3. May 1986, pp. 1174–1179. DOI: 10.1109/ROBOT.1986.1087552 (cit. on p. 80).
- [212] Maxime Gautier and Wisama Khalil. “On the identification of the inertial parameters of robots”. In: *Proceedings of the 27th IEEE Conference on Decision and Control*. Vol. 3. IEEE Austin, 1988, pp. 2264–2269 (cit. on p. 83).
- [213] Maxime Gautier and Wisama Khalil. “Direct calculation of minimum set of inertial parameters of serial robots”. In: *IEEE Transactions on robotics and Automation* 6.3 (1990), pp. 368–373 (cit. on p. 83).
- [214] Maxime Gautier. “Numerical calculation of the base inertial parameters of robots”. In: *Journal of robotic systems* 8.4 (1991), pp. 485–506 (cit. on p. 83).
- [215] Jan Swevers, Chris Ganseman, D Bilgin Tukel, Joris De Schutter, and Hendrik Van Brussel. “Optimal robot excitation and identification”. In: *IEEE transactions on robotics and automation* 13.5 (1997), pp. 730–740 (cit. on p. 83).

- [216] Luca Marchionni, Jordi Pages, Jordi Adell, Jose Rafael Capriles, and Hilario Tomé. “Reem service robot: How may i help you?” In: *International Work-Conference on the Interplay Between Natural and Artificial Computation*. Springer. 2013, pp. 121–130 (cit. on p. 93).
- [217] Manuel G Catalano, Giorgio Grioli, Edoardo Farnioli, Alessandro Serio, Cristina Piazza, and Antonio Bicchi. “Adaptive synergies for the design and control of the Pisa/IIT SoftHand”. In: *The International Journal of Robotics Research* 33.5 (2014), pp. 768–782 (cit. on p. 93).

# Glossary

**Agimus** A planning and execution framework developed by the LAAS-CNRS. It build upon the concept of the graph of constraints presented in HPP to combine this manipulation planner with a automated generation of SoT controllers to ensure the good execution of the plan.

**AprilTag** A type of 2D visual fiducial markers, mainly used in augmented reality and robotics. They can easily be detected by computer vision software, and their position and orientation can be precisely computed. (See [171])

**Configuration** a set of values — one for each DoF — necessary and sufficient to define the position of every point of a system, composed of a robot and, if applicable, movable objects.

**Configuration Space** the set of all the configurations achievable by the system.

$\mathcal{CS}_{\text{free}}$  the subset of  $\mathcal{CS}$  devoid of collision;  $\mathcal{CS}_{\text{free}} = \mathcal{CS} \setminus \mathcal{CS}_{\text{obs}}$ . This is the subset where we plan the robot's trajectories.

$\mathcal{CS}_{\text{obs}}$  the subset of  $\mathcal{CS}$  where at least one body of the system collides with the environment or another body of the system.

**Humanoid Path Planner** a C++ Software Development Kit implementing path planning for kinematic chains in environments cluttered with obstacles. The platform implements an original way of modeling manipulation planning through a constraint graph that represents the numerical constraints that define the manipulation problem. (See [5])

**Middle Sized Drilling Robot** a drilling robot designed by Airbus Operations to help in the process of aircrafts' assembly, which requires tens of thousands of holes. It is based on a Fanuc M-800*i*A/60 robot, equipped with a custom end-effector containing a drill, cameras to detect reference holes, laser sensors for precise alignment, and force sensors to ensure adequate drilling force and prevent slippage.

**ROB4FAM** Robots For the Future of Aircrafts Manufacturing: a joint laboratory between Airbus and the LAAS-CNRS aimed at developping software technologies to adapt the robots to the aircraft industry

**Stack-of-Tasks** a C++ Software Development Kit implementing a control architecture for redundant robots and more specifically for humanoid robots. The

framework is flexible enough to implement hierarchical control and weighted control. (See [6])

**Visual Servoing Platform** a modular cross platform library that allows prototyping and developing applications using visual tracking and visual servoing technics. ViSP is able to compute control laws that can be applied to robotic systems. It provides a set of visual features that can be tracked using real time image processing or computer vision algorithms. (See [3])

# Acronyms

**ABA** Articulated Body Algorithm

**BIT\*** Batch Informed Trees

**CHOMP** Covariant Hamiltonian Optimisation for Motion Planning

**CoM** Center of Mass

**CRBA** Composite Rigid Body Algorithm

*CS Glossary:* Configuration Space

**CSP** Constraint Satisfaction Problem

**DARPA** Defense Advanced Research Projects Agency

**DDP** Differential Dynamic Programming

**DH parameters** Denavit-Hartenberg parameters (See [210])

**DoF** Degree of Freedom

**DRC** DARPA Robotics Challenge

**FCL** Flexible Collision Library (See [195])

**FF** Fast Forward algorithm

**FMT\*** Fast Marching Tree

**FSM** Finite State Machine

**GIK** Generalised Inverted Kinematics

**GPMP** Gaussian Process Motion Planner

**GPS** General Problem Solver (See [80])

**HPP** *Glossary:* Humanoid Path Planner (See [5])

**IBVS** Image-Based Visual Servoing

**IMU** Inertial Measurement Unit

- IPC** International Planning Competition (See [83])
- LAAS-CNRS** Laboratory for Analysis and Architecture of Systems
- MHPC** Model Hierarchy Predictive Control
- MoCap** Motion Capture system
- MSDR** *Glossary:* Middle Sized Drilling Robot
- NAMO** Navigation Among Movable Obstacles
- NR** Newton-Raphson
- OMPL** Open Motion Planning Library
- PBVS** Position-Based Visual Servoing
- PDDL** Planning Domain Definition Language
- PRM** Probabilistic Road-Maps
- PRM\*** Optimal PRM
- RGD** Randomised Gradient Descent
- RHP** Recursive Hermite Projection
- RNEA** Recursive Newton-Euler Algorithm
- ROS** Robot Operating System (See [8])
- RRT** Rapidly-exploring Random Tree
- RRT\*** Optimal RRT
- SAT** Propositional Satisfiability Problem
- SLAM** Simultaneous Localisation And Mapping
- SoT** *Glossary:* Stack-of-Tasks (See [6])
- STOMP** Stochastic Trajectory Optimisation for Motion Planning
- STRIPS** Stanford Research Institute Problem Solver (See [81])
- TF2** ROS transform library 2 (See [197])
- TS** Tangent-space Sampling
- URDF** Unified Robot Description Format
- ViSP** *Glossary:* Visual Servoing Platform (See [3])
- V-PRM** Visibility-PRM
- ZMP** Zero Moment Point

## Résumé

La robotique est de plus en plus présente dans le cadre industriel. À l'origine programmés manuellement par leurs opérateurs, les robots d'aujourd'hui utilisent de plus en plus des trajectoires calculées par des programmes simulant l'ensemble de l'environnement de travail. Des erreurs de modélisation, voire d'exécution, rendent cependant obligatoire des périodes d'essais pour s'assurer que le robot s'acquittera correctement de sa tâche.

Cette thèse a pour objectif de relier le concept de contrainte que le planificateur de mouvements prend en compte avec les logiciels de commande en temps réel du robot, qui gère le bon déroulement des tâches. La commande doit ainsi prendre en compte les données provenant des multiples capteurs du robot pour adapter les trajectoires planifiées à la réalité et s'assurer de la réalisation de ses objectifs. Cette thèse s'effectue dans le cadre d'une convention CIFRE entre le laboratoire du LAAS-CNRS et l'entreprise Airbus Operations. Elle fait par ailleurs partie de ROB4FAM, un laboratoire commun entre ces deux entités, qui a pour but l'étude du futur de la robotique appliquée à la construction aéronautique.

La contribution majeure de cette thèse est la génération automatique de ces logiciels de commande à partir des contraintes exprimées à l'étape de planification. Jusqu'à présent, ce type de correspondance était plutôt établi manuellement. Parmi les différents types de commande existants, un accent tout particulier a été mis sur la commande par asservissement visuel. En effet, les caméras sont des capteurs à la fois abordables et capables de générer des données d'une grande richesse, mais surtout les ordinateurs modernes sont désormais en mesure de traiter efficacement ce volume d'information en temps réel.

Enfin, les logiciels de commande que nous présentons dans ce manuscrit utilisent une hiérarchie stricte de commandes. Cela signifie que le robot cherchera à assurer la réalisation du premier objectif, généralement lié à la préservation de son intégrité, avant de prendre en compte les suivants. Grâce à cette architecture, le robot peut s'adapter à des changements mineurs de ses objectifs tout en garantissant la sécurité des opérateurs, de son environnement de travail et de sa structure même.

Les concepts et logiciels développés au cours de cette thèse ont été mis en œuvre sur un robot humanoïde, TALOS, puis sur un robot mobile à roue équipé d'un bras manipulateur, TIAGo. Ils ont été capables d'accomplir leurs tâches malgré le déplacement des objets qu'ils allaient manipuler entre l'étape de planification des mouvements et celle d'exécution du programme.



## Abstract

Robots are increasingly present in the industrial environment. Originally programmed manually by their operators, today's robots are progressively using paths calculated by programs that simulate the entire work environment. However, modelling or even execution errors require testing periods to ensure that the robot will perform its task correctly.

This thesis intends to link the concept of constraint that the motion planner takes into account with the robot's real-time control software, which manages the correct execution of tasks. The control must take into consideration the data from the robot's multiple sensors to adapt the planned paths to reality and ensure that its objectives are achieved. This thesis is being carried out within the framework of a CIFRE agreement between the LAAS-CNRS laboratory and the company Airbus Operations. It is also part of ROB4FAM, a common laboratory between those two entities, which aims at studying the future of robotics applied to aeronautical construction.

The main contribution of this thesis is the automatic generation of such control software based on the constraints expressed at the planning stage. Until now, this type of correspondence was mostly established manually. Among the different types of control schemes available, special emphasis has been placed on visual servo control. This is due to the fact that cameras are affordable sensors capable of generating a considerable amount of data. But above all modern computers are now able to process this volume of information efficiently in real-time.

Finally, the control software we present in this manuscript uses a strict hierarchy of commands. This means that the robot will seek to ensure the achievement of the first objective, usually related to the preservation of its integrity, before considering the following ones. With this architecture, the robot can adapt to minor changes in its objectives while ensuring the safety of the operators, its working environment and its structure itself.

The concepts and software developed during this thesis were implemented on a humanoid robot, TALOS, and then on a mobile wheeled robot equipped with a manipulator arm, TIAGo. They were able to accomplish their tasks, despite the displacement of the objects they were going to manipulate between the movement planning stage and the program execution stage.