Trajectory planning and control for robot manipulations

Ran Zhao

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)*

Présentée et soutenue le *24/09/2015* **par :**

Ran ZHAO

Trajectory Planning and Control for Robot Manipulations

JURY

Ahmed RAHMANI	Maître de conférences	Rapporteur
Andre CROSNIER	Professeur d'Université	Rapporteur
Christian BES	Professeur d'Université	Examinateur
Micaël MICHELIN	Ingénieur de recherche et développement	Examinateur
Rachid ALAMI	Directeur de Recherche CNRS	Examinateur
Daniel SIDOBRE	Maître de conférences	Directeur de Thèse

École doctorale et spécialité :
EDSYS : Robotique 4200046
Unité de Recherche :
Laboratoire d'Analyse et d'Architecture des Systèmes (UMR 8001)
Directeur(s) de Thèse :
Daniel SIDOBRE
Rapporteurs :
Ahmed RAHMANI et *Andre CROSNIER*

UNIVERSITÉ TOULOUSE III - PAUL SABATIER
ÉCOLE DOCTORALE SYSTÈMES

THÈSE

en vue de l'obtention du

Doctorat de l'Université de Toulouse
délivré par l'Université Toulouse III Paul Sabatier

Spécialité: Computer Science and Robotics

Trajectory Planning and Control for Robot Manipulation

Ran ZHAO

Préparée au Laboratoire d'Analyse et d'Architecture des Systèmes
sous la direction de M. Daniel SIDOBRE

Jury

M. Ahmed RAHMANI	Rapporteur
M. Andre CROSNIER	Rapporteur
M. Christian BES	Examinateur
M. Micaël MICHELIN	Examinateur
M. Rachid ALAMI	Examinateur
M. Daniel SIDOBRE	Directeur de Thèse

Abstract

In order to perform a large variety of tasks in interaction with human or in human environments, a robot needs to guarantee safety and comfort for humans. In this context, the robot shall adapt its behavior and react to the environment changes and human activities. The robots based on learning or motion planning are not able to adapt fast enough, so we propose to use a trajectory controller as an intermediate control layer in the software structure. This intermediate layer exchanges information with the low level controller and the high level planner.

The proposed trajectory controller, based on the concept of Online Trajectory Generation (OTG), allows real time computation of trajectories and easy communication with the different components, including path planner, trajectory generator, collision checker and controller.

To avoid the replan of an entire trajectory when reacting to a human behaviour change, the controller must allow deforming locally a trajectory or accelerate/decelerate by modifying the time function. The trajectory controller must also accept to switch from an initial trajectory to a new trajectory to follow. Cubic polynomial functions are used to describe trajectories, they provide smoothness, flexibility and computational simplicity. Moreover, to satisfy the objective of aesthetics, smoothing algorithm are proposed to produce human-like motions.

This work, conducted as part of the ANR project ICARO, has been integrated and validated on the KUKA LWR robot platform of LAAS-CNRS.

Keywords: Robotics, Trajectory planning, Trajectory control, Humn-Robot Interaction, Manipulation

Résumé

Comme les robots effectuent de plus en plus de tâches en interaction avec l'homme ou dans un environnement humain, ils doivent assurer la sécurité et le confort des hommes. Dans ce contexte, le robot doit adapter son comportement et agir en fonction des évolutions de l'environnement et des activités humaines. Les robots développés sur la base de l'apprentissage ou d'un planificateur de mouvement ne sont pas en mesure de réagir assez rapidement, c'est pourquoi nous proposons d'introduire un contrôleur de trajectoire intermédiaire dans l'architecture logicielle entre le contrôleur bas niveau et le planificateur de plus haut niveau

Le contrôleur de trajectoire que nous proposons est basé sur le concept de générateur de trajectoire en ligne (OTG), il permet de calculer des trajectoires en temps réel et facilite la communication entre les différents éléments, en particulier le planificateur de chemin, le générateur de trajectoire, le détecteur de collision et le contrôleur.

Pour éviter de replanifier toute une trajectoire en réaction à un changement induit par un humain, notre contrôleur autorise la déformation locale de la trajectoire et la modification de la loi d'évolution pour accélérer ou décélérer le mouvement. Le contrôleur de trajectoire peut également commuter de la trajectoire initiale vers une nouvelle trajectoire. Les fonctions polynomiales cubiques que nous utilisons pour décrire les trajectoires fournissent des mouvements souples et de la flexibilité sans nécessiter de calculs complexes. De plus, les algorithmes de lissage que nous proposons permettent de produire des mouvements esthétiques ressemblants à ceux des humains.

Ce travail, mené dans le cadre du projet ANR ICARO, a été intégré et validé avec les robots KUKA LWR de la plate-forme robotique du LAAS-CNRS.

Mots clés: Robotique, planification de trajectoires, contrôle de trajectoire, interaction homme-robot, manipulation.

Acknowledgement

The thesis at *LAAS-CNRS* has been three years of precious experience. During these years, I have learned so much. Therefore, I would like to express my gratitude, without trying to make a complete list of the people who have helped me during these years.

Firstly, I would like to express my sincere gratitude to Prof. Daniel Sidobre and Prof. Rachid Alami for giving me this opportunity to work in a prestigious team and in the promising area of robotics. I would like to thank my supervisor, Prof. Daniel Sidobre, for the continuous support of my Ph.D study and related research, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my Ph.D study.

Besides, I would like to thank Mathieu Herrb and Anthony Mallet for their support and time. They gave me a lot of useful suggestions both on the hardware and the software of the system.

Then I would like to say thanks to the other members of the group who helped me a lot in the past three years. Many thanks to the project partners for their friendly collaboration. Also I thank my friends in the following institution: UPS, INSA, and SUPAERO, with whom I have spent the breaks from work on sports and inspiring discussion on everything.

Last but not the least, I would like to thank my family: my parents and my sister for supporting me spiritually throughout writing this thesis and my life in general. A Special thanks to my wife, Sang Rui, for her accompany, encouragement and great support during my Ph.D study.

Contents

List of Figures

1 Introduction

1.1 Introduction

Interactive robots are now beginning to joint assembly and production lines. They complement the previous generation of robots, which were designed for performing operations quickly, repeatedly and accurately. They have now a long heritage in the manufacturing industry for operating in large numbers in relatively static environments. The development of traditional robotics in industry is confronted to the difficulty of programming and to the cost of safe guard designed to separate humans and robots. Due to the multiple advantages over a human operator, the arm manipulators, for example, have been used in various kinds of industrial applications, such as pick-and-place operations, welding, machining, painting, etc. The introduction of interactive robots simplifies the design of the production lines and the programming of robots.

Arm manipulator control for industrial applications has now reached a good level of maturity. Many solutions have been proposed for specific uses. However, all these applications are confined to structured and safe spaces where no human-robot interactions occur. With the growth of robotic presence in the human community, the need of intuitiveness in the human-robot interaction has become inevitable. The human-robot coexistence with various degrees of restriction can be found in many areas, specifically in the industry services. There exist many tasks in the industrial scenarios where the robots (robotic arms) work together with the humans. Therefore, it is necessary for the robots to interact smoothly and

intuitively to solve the task effectively, successfully and safely.

If robots could coexist with human workers, the robots could carry out monotonous and repetitive tasks with accuracy and at high-speed. Human workers could use their skills to do more complex tasks, such as assembly and preparation, post-processing tasks for the robots. This compensation of each other's disadvantages holds promise for cooperation between human workers and robots. The concept of closeness is to be taken in its full meaning, robots and humans share the workspace but also share goals in terms of task achievement. Figure C.1 illustrates a scenario where a human operator cooperates with a KUKA LWR arm manipulator to achieve an assembly task.

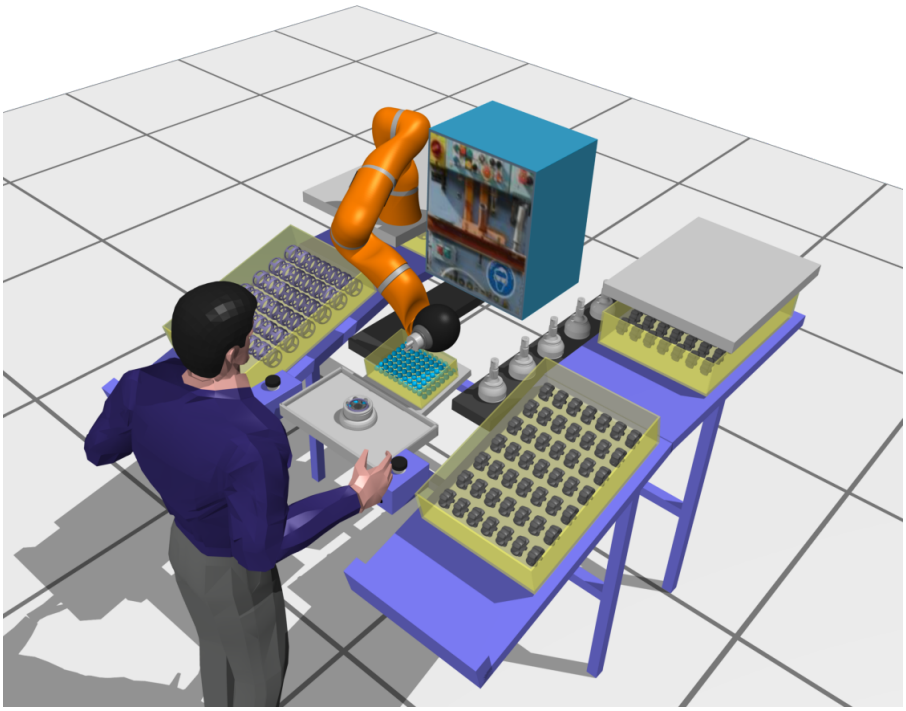


Figure 1.1: The robot manipulator shares the workspace with a human operator. They cooperate to assemble a Rzeppa Joint. Figure comes from the ANR Project *ICARO*

To realize robots working alongside humans in environments such as the home, at work, a hospital or any public place, the robots have to be safe, reliable and easy to use. In practice, the robot must guarantee the safety of humans when they share the same workspace, it is important to make the robot to avoid hurting humans during the operation even when an unexpected or not detected event occurs. The human's comfort is another important aspect in robotic applications. In the context of human robot interaction, the robot should not cause excessive stress and discomfort to the human for extended periods of time [Lasota 14].

The two important approaches to define a robotic task are learning and motion planning. The learning approach uses different techniques to record a human-demonstrated motion or a skill, plans a task from these data and improves the execution of the task by learning,

such as *Reinforcement Learning (RL)* [Kober 13] and *Learning from Demonstration (LfD)* [**?**]. These approaches often represent the learned actions as trajectories. However, there are always unforeseen events in a dynamic environment. In this case the learning approaches are becoming less effective to adapt the produced trajectory to the context in real time, e.g. to grasp a moving object, or to avoid an obstacle.

The other way to define the robotic task is motion planning. Most of the motion planners define the motion by paths, which will be followed up by the robot. Unfortunately, the robot controller can only trace a small subset of these paths, but fails to follow up the majority of them efficiently and precisely. A better approach is to define motions more precisely using trajectories, which define the position as a function of time. We propose to introduce an intermediate level of control between the motion planner and the low level controller provided by the robot manufacturers. When the task is changing, e.g. by the presence of humans, planning new trajectories is necessary. In this case, a feedback from the controller describing its state and a trajectory describing the future state could be helpful to adapt the planned trajectory to the robot's predicted state at the end of the next planning loop.

Moreover, motion planning is usually done off-line, especially when the trajectory generation processes are computationally expensive. However, to perform a large variety of tasks autonomously and reactively, a robot must propose a flexible trajectory controller that can generate and control the trajectories in real time. To react to environment changes, the trajectory generation must be done online. Meanwhile, the robot needs to guarantee the human safety and the absence of collision. So the model for trajectory must allow fast computation and easy communication between the different components, including path planner, trajectory generator, collision checker and controller. To avoid replanning of an entire trajectory, the model must allow deforming locally a path or a trajectory. The trajectory controller must also accept to switch from an initial trajectory to follow to a new one.

1.2 Motivation

In a near future, robots are going to share workbench with humans, they even will work on the same workpiece together. Thus, the role of industrial robots is becoming a tool that engineers and technicians can interact with. Historically, robots used in industrial service are designed and programmed for relatively static environments. Anything unanticipated in the robot's environment is essentially invisible, as the robot feedback is really limited (joint sensors for position and eventually torques). These primitive sensory capacities in most cases necessitate running robots in 'work cells', free from people and other changing elements. Once programmed, it is expected that the work environment the robot interacts with remain within a very narrow range of variance. Thus the robots are isolated in a physical as well as sensorial sense, little different from any number of dangerous, automated factory machines.

The motivation of this work can be expressed by an analogy: imagine that two human operators need to finish an assembly task together on a fabrication line. One of the operator $\mathscr{A}$ has to focus on the main construction task, while the other operator $\mathscr{B}$ is expected to prepare work sites, collect and deliver tools just as they are needed, or stabilize components during assembly. For industries or enterprises, it might be a lack of competitiveness in the modern business with expensive labor. So it is necessary to extend the capacity of the robots to enable the possibility for a robot to work with human and to assist humans. This thesis sets out to develop a robot that work side-by-side with human, i.e. the robot will assist the operator in the whole task. We propose to build a more flexible controllers that are needed to switch between different sensors and control laws, and thus to build a more reactive system. A part of the solution is to use trajectories to exchange information among the motion planner, the collision checker, the vision/force systems and the low level controller.

1.3 Research Objectives

The objective of this work is to build more reactive robots by introducing a trajectory controller as an intermediate layer in the software structure in the context of human-robot interaction. Thus the controller should provide a method of generating smooth and time-optimal trajectories in real time. It also must be able to react to the environment changes. Therefore, the trajectory generation algorithm must be computationally simple. The trajectories must take into account the physical limitations of the robot, that is, not only the velocity limitation but also acceleration and jerk limitations.

This work aims to produce a trajectory controller that can not only accept any trajectory produced by a path planner, but also can approximate any types of trajectories. It will enlarge the type of paths that the robot can follow, which makes the robot competent in more complex tasks. The proposed trajectory controller, based on the concept of Online Trajectory Generation (OTG), allows real time computation of trajectories and easy communication with the different components, including path planner, trajectory generator, collision checker and controller. The controller can also allow deforming locally a trajectory and accept to switch from an initial trajectory to a new trajectory to follow.

Moreover, the controller can accelerate/decelerate the robot on the main trajectory by changing the time function $s(t)$. Imagine for example that a fast movement could cause some people anxiety when the robot is close to them. In this situation, the controller can still execute the task but only at a reduced speed. The time-scaling schemes can also be used to avoid dynamic obstacle while not changing the path of the robot.

A further objective is to apply this technique to sensor-based control. When the robot is able to build the dynamic model of the manipulated object, to track the movement of objects and human body parts, and to detect special events, it needs to finish the tasks while reacting to the movement or events. Based on the work of previous colleagues, we

propose a trajectory controller to achieve reactive manipulations. The controller integrates information from multiple sources, such as vision systems and force sensors, and use online trajectory generation as the central algorithm.

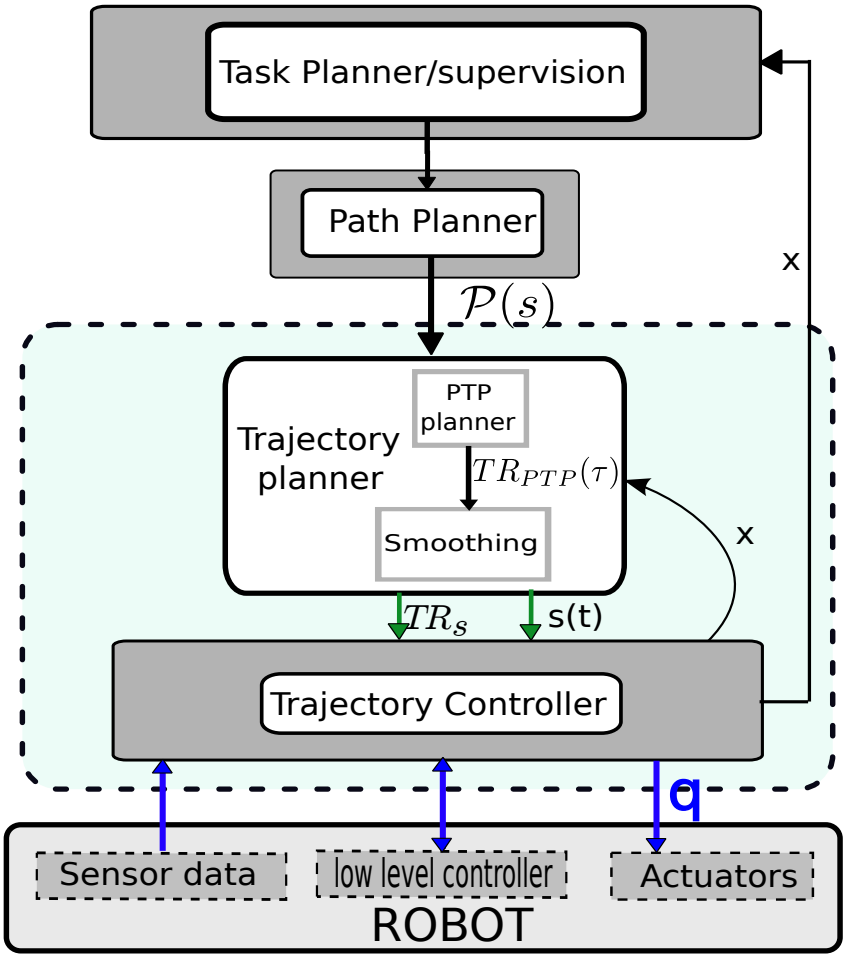


Figure 1.2: Trajectory Controller as an intermediate layer in the software architecture, the servo system on robot is fast, while task planning and path planning are slow.

Figure C.2 illustrates the intermediate level in the software structure. The controller integrates information from other modules in the system, including geometrical reasoning and human aware motion planning. From the sensor information, the trajectory planner can generate non-linear time-scaling functions or replan new trajectories for reacting to the environment change. One advantage of our algorithm is to be more reactive to dynamically changing environments. A second advantage is the simplicity of the use of the controller.

1.4 Outline of this Manuscript

Following this introduction, this dissertation begins with the presentation of background and literature review on motion control, focusing on learning approach, path planning, tra-

jectory generation and reactive trajectory control. Chapter 3 presents the trajectory generation problem, including straight-line trajectory generation between waypoints, smooth near time-optimal trajectory generation. All these algorithms produce jerk-bounded motions. In Chapter 4, we present the trajectory approximation with polynomial functions. Chapter 5 focuses on the reactive trajectory controller, with the concept of control primitives and how it is used in human robot interactions. Following the three chapters, we give the discussion and conclusion, as well as the recommendation for future works in chapter 6. Because each chapter deals with a different problem, experimental results are given at the end of each chapter.

1.5 Publication, Software Development, and Research Projects

Publication during the thesis:

- RAN ZHAO, DANIEL SIDOBRE, Trajectory Smoothing using Jerk Bounded Shortcuts for Service Manipulator Robots, *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Hamburg, Germany, Sept. 28 – Oct. 02, 2015 (Accepted)*

- RAN ZHAO, DANIEL SIDOBRE AND WUWEI HE, Online Via-points Trajectory Generation for Reactive Manipulations, *2014 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM), besançon, France, 2014*

- WUWEI HE, DANIEL SIDOBRE AND RAN ZHAO, A Reactive Trajectory Controller for Object Manipulation in Human Robot Interaction, *The 10th International Conference on Informatics in Control, Automation and Robotics (ICINCO), Reykjavik, Iceland, 2013*

- WUWEI HE, DANIEL SIDOBRE AND RAN ZHAO, A Reactive Controller Based on Online Trajectory Generation for Object Manipulation, *Lecture Notes in Electrical Engineering, 159-176, Springer International Publishing*

- DANIEL SIDOBRE, RAN ZHAO, Trajectoires et contrôle des robots manipulateurs interactifs, *Journées Nationales de la Robotique Humanoïde, Nante, France, Jun. 2015*

- RAN ZHAO, DANIEL SIDOBRE, Online Via-Points Trajectory Generation for Robot Manipulations, Congrès des Doctorants EDSYS,2014

This work was partially reported in the *ICARO* project, which aims to developed tools to improve and to simplify the interaction between industrial robots on one side and humans and the environment on the other side. The author contributed to the development of several

software running on the robot *kuka LWR*, and maintenance of the robots. The author participated actively to the development of software and to the final integration for a research project:

- Project *ICARO*[1], see Figure C.1. *ICARO* is a collaborative research project aiming to develop tools to improve and simplify interaction between industrial robots and humans and their environment. ICARO was funded by the program CONTINT of the ANR agency from 2011 to 2014.

The author participated in the development of several softwares:

- *softMotion-libs*: A C++ library for online trajectory generation. It can be tested in the MORSE, the Modular OpenRobots Simulation Engine[2].
- *lwri-genom*: A GenoM3 module for the trajectory generation of the KUKA LWR arm.
- *lwrc-genom*: A GenoM3 module for the trajectory control of the KUKA LWR arm.
- *coldman-genom*: A GenoM3 module for the collision detection in robot manipulation.
- *pr2-softMotion*: Genom interface for pr2-soft-controllers

Most of the softwares listed above are accessible in *robotpkg*[3].

[1] http://icaro-anr.fr/
[2] http://www.openrobots.org/wiki/morse/
[3] http://robotpkg.openrobots.org/

2

Related Work and Background

2.1 Introduction

Service robots are becoming used to work in environments like home, hospitals and schools in the presence of humans. Therefore this expansion raises challenges on many aspects. Since the tasks for the robot to realize will not be predefined, they are planned for the new situations that the robots have to handle. Clearly, purely motion control with predefined trajectories will be not suitable for these situations. The robot should acquire the ability to react to the changing environment. Before presenting the main background of our developments, several aspects need to be discussed and be compared to the state of the art: architecture for human-robot interaction, trajectory generation and control including online adaptation and monitoring. In this chapter, we will give a brief review of some foundation of the service robotic. Then we will survey the planning-based and learning-based techniques that are used for trajectory generation, followed by background material that motivates this research.

2.2 Software Architecture for Human Robot Interaction

The robots capable of doing HRI must realize several tasks in parallel to manage various information sources and complete tasks of different levels. Figure 2.1 shows the proposed architecture where each component is implemented as a GENOM module. GENOM

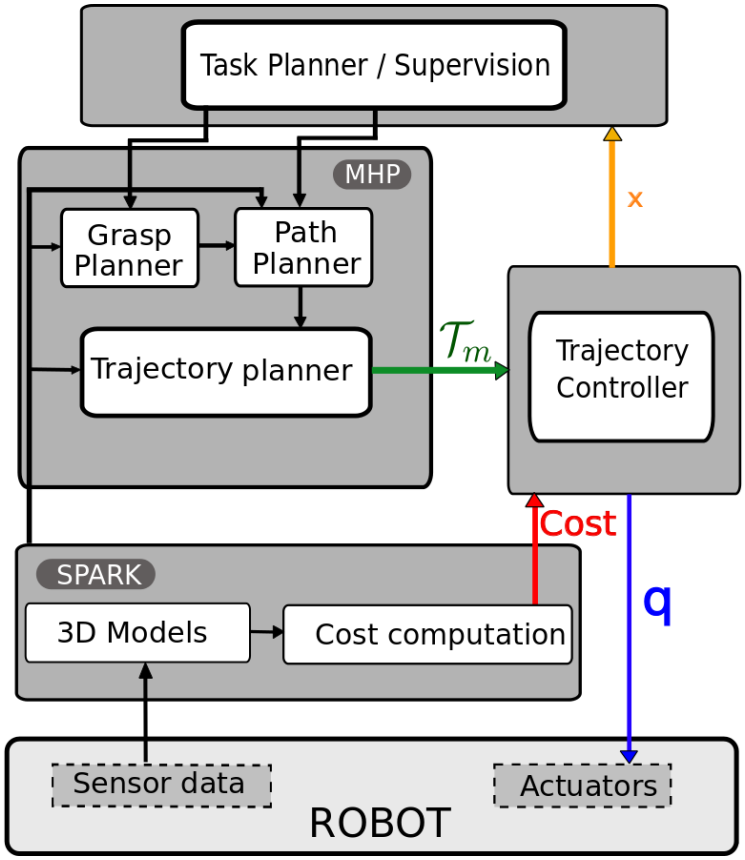


Figure 2.1: Software Architecture of the robot for HRI manipulation. $\mathcal{T}_m$ is the main trajectory calculated initially by MHP. The controller takes also costs from SPARK, which maintains a module of the environment. The controller sends control signals in joint (q in the figure) to the robot arm controller, and during the execution, the controller returns the state of the controller (s in the figure) to the supervisor

[Mallet 10a, Mallet 11] is a development environment for complex real time embedded software.

At the top level, a task planner and supervisor plan tasks as the output, such as cleaning the table, bring an object to a person and then supervises the execution. The module SPARK (**Spa**tial **R**easoning and **K**nowledge) maintains a 3D model of the whole environment, including objects, robots, posture and position of humans [Sisbot 07b]. It provides also the related geometrical reasoning on the 3D models, such as evaluating the collision risk between the robot parts and between the robot and its environment. An important element regarding SPARK is it produces cost maps, which describe a space distribution relatively to geometrical properties like human accessibility. The software for perception, from which SPARK updates the 3D model of the environment, is omitted here for simplicity. The module runs at a frequency of 20Hz, limited mainly by the complexity of the 3D vision and of the detection of human.

MHP (**M**otion in **H**uman **P**resence) is another important module that integrates path

and grasp planner. RRT (Rapidly exploring Random Tree)[LaValle 01b] and its variants [Mainprice 10] are used by the path planner. The paths could be described in Cartesian or joint spaces depending on the task type. The output path of the planner is defined as a broken line. From this path, an output trajectory is computed to take the time into account. MHP calculates a new trajectory each time the task planner defines a new task or when the supervisor decides that a new trajectory is needed to react to the changes of the environment.

When the motion adaptation is achieved by path replanning, the robot would switch between planning and execution, producing slow reactions and movements because of the time needed for the complex path planner. Furthermore, if object or human moves during the execution of a trajectory planned by the module MHP, the task will fail and so a new task or a new path needs to be planned. The human counterpart often finds the movement of the robot unnatural and so not intuitive to interact with.

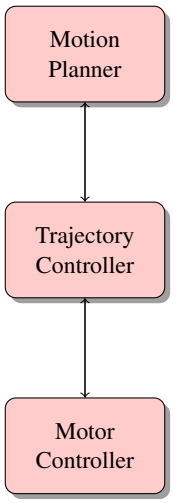


Figure 2.2: Trajectory Controller is introduced as an intermediate layer in the software architecture, between the low level and fast motor controller and the high level planner, which is slow.

2.3 Trajectory Generation

Trajectory generation computes the time evolution of a motion for the robot. Trajectories can be defined in joint space or in Cartesian space. They are then directly provided as the input for the controller. Trajectories are important because they enable the system to ensure:

- feasibility: the motion can be verified to respect the dynamic constraints of lower-level controllers.
- safety and comfort: the trajectories can limit velocity, acceleration and jerk, which are directly related to the safety and comfort for humans.

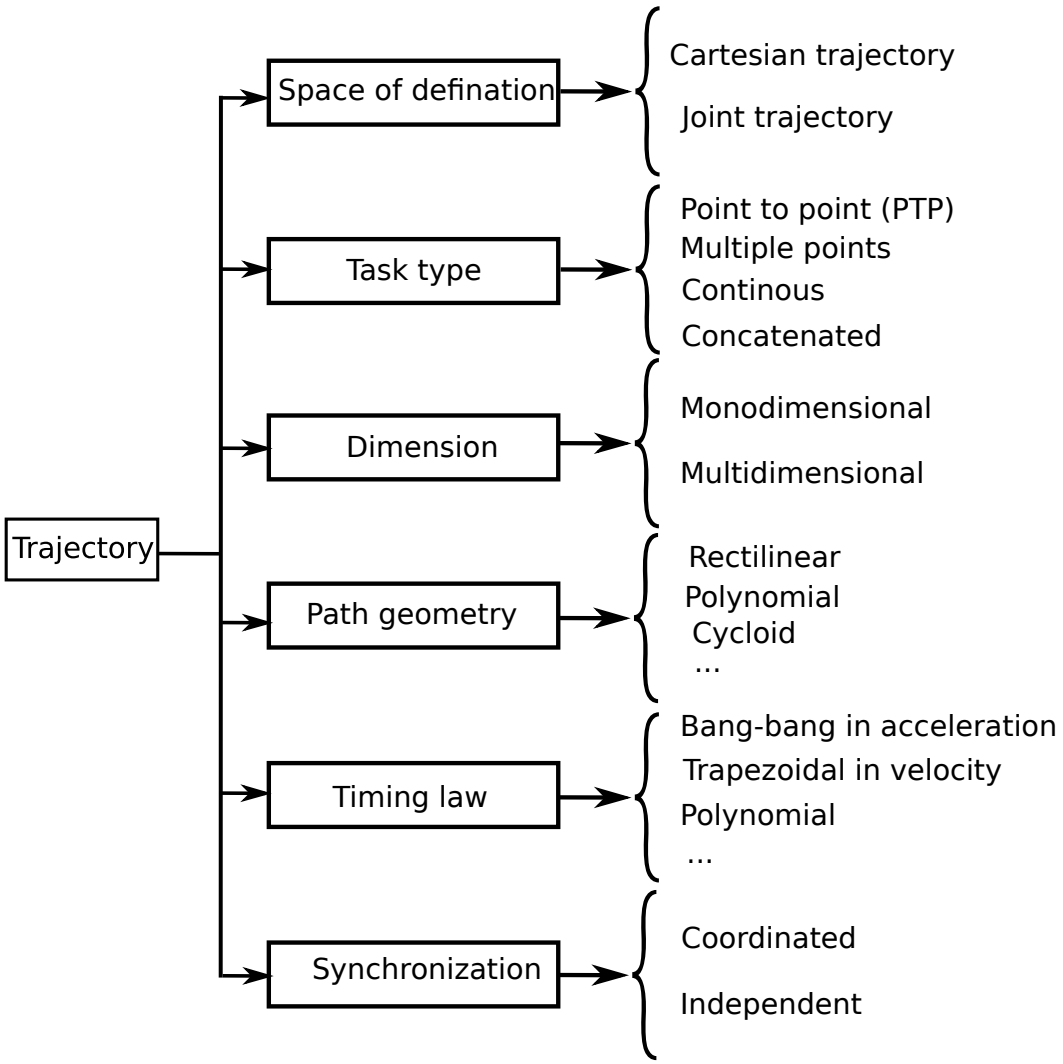


Figure 2.3: Categories of trajectories [Siciliano 08b]

- optimization: optimization can integrate both geometry and time.
- flexibility: trajectories allow to define a lot of tools to adapt and transform them.

2.3.1 Trajectory Types

Siciliano proposed classifications to verify trajectories [Siciliano 08b] (Figure 2.3). From spatial point of view, trajectories can be planned in joint space or Cartesian space. Joint space trajectories have several advantages:

- Trajectories planned in joint space can be used directly to control low-level motors without need to compute inverse kinematics.
- No need to deal with the redundant joints or singularities of multi-DOF manipulators.

- The dynamic constraints, like maximum acceleration on joints, can be considered while generating the trajectories. On the contrary, these constraints should be tested after inverse kinematic for Cartesian space trajectories.

Cartesian space trajectories are directly related to task properties, allowing a more direct visualization of the generated path. Generally, they produce more natural motions, which are more acceptable results for people. For a simple motion like hand over an object, a trajectory planned in Cartesian space can produce a straight-line movement, which is not easy to guarantee while planning in joint space. The advantage of Cartesian space planning is also evident for task constraints. For example, when the robot needs to manipulate a cup of tea without spilling it out.

From another point of view, a motion can be completed by the choice of different timing laws $s(t)$ along the same path $\mathscr{P}$.

$$\mathscr{T}(t) = \mathscr{P}(s(t)) \tag{2.1}$$

where $\mathscr{T}(t)$ is the trajectory. If $s(t) = t$, the path is parameterized by the natural time. The timing law is chosen based on task specifications (stop in a point, move at constant velocity, and so on). It also may consider optimality criteria (min transfer time, min energy, ...). Constraints are imposed by actuator capabilities (max torque, max velocity, ...) and/or by the task (e.g., max acceleration on payload).

Trajectories might also be classified as coordinated trajectories or independent trajectories, according to the synchronization property. For coordinated trajectories the motions of all joints (or of all Cartesian components) start and end at the same instant. It is equivalent that all joints have the same time law. While independent trajectories are timed independently according to the requested displacement and robot capabilities. This kind of trajectory exists mostly only in joint space.

From another point of view, Biagiotti [Biagiotti 08] proposed a classification based on the dimension and task type, as shown in Figure 2.4. *Mono-dimensional* trajectories correspond to trajectories for systems of only one degree of freedom (DOF), while *Multi-dimensional* for more than one DOF. Compared to the case of mono-dimension, the difficulty for multi-dimensional trajectories is the synchronization of the different axis. A point-to-point trajectories simply link two points, while a multi-points trajectories pass through all points in the middle. Trajectory approximation and interpolation are usually used when we get movement data from a set of measured locations at known times. In general, trajectory interpolation of locations defines the velocity.

These trajectories are usually long-term trajectories as there is some distance between each waypoints provided by a motion planner. A trajectory can also connect two different trajectories. When the robot is moving along a trajectory and receiving a request to switch to another trajectory, generation of a connection trajectory can allow the robot to joint up smoothly the new trajectory. In this case, the trajectory links two robot states of non-null

velocities and/or accelerations.

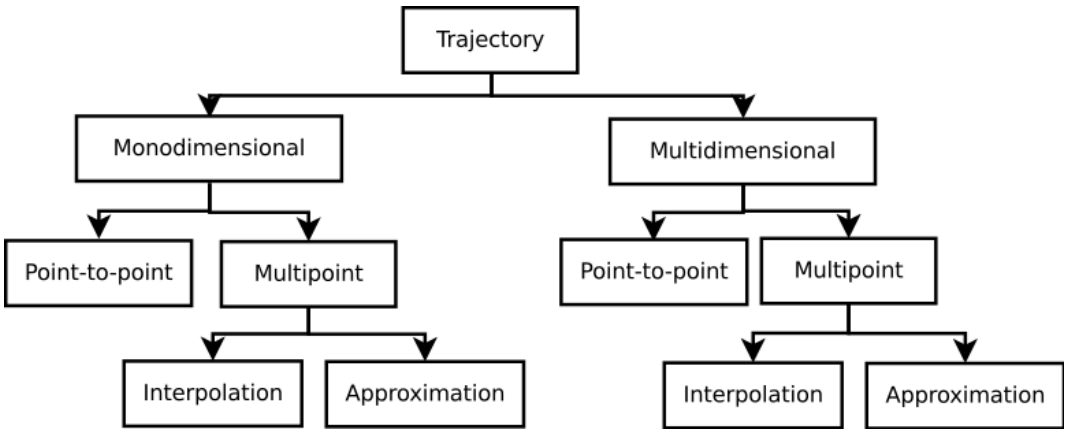


Figure 2.4: Classification of trajectories based on dimension and task type [Biagiotti 08]

2.3.2 Trajectory Generation Algorithms

Trajectory generation for manipulators have been discussed in numerous books and papers, among which readers can find [Brady 82], [Khalil 99] and [Biagiotti 08]. Kroger, in his book [Kroger 10b], gives a detailed review on on-line and off-line trajectory generation. He proposes a classification based on the complexity of the generation related to the number of constraints, as in table 2.1 [Kroger 10b]. From any situation, Kröger proposes to reach a goal defined by constraints (position, velocity, acceleration, jerk, ...) while respecting bounds (V_{max}, A_{max}, J_{max}, D_{max}, ...). V_{max} is the maximum velocity, A_{max} is the maximum acceleration, J_{max} is the maximum jerk, D_{max} is the maximum derivative of jerk. The definitions presented in this table are used in this document.

2.3.2.1 Robot Motion Representation

We firstly consider the kinematics of a rigid body in a 3D space. Considering a reference frame, F_w, which can be defined as an origin O_w and an orthogonal basis (X_w, Y_w, Z_w). A rigid body B is localized in 3D space by a frame F_B, as shown by Figure 2.5.

Translations and rotations shall be used to represent the relation between these two frames. For the translation, Cartesian coordinates are used, but for rotations, several choices are available:

- Euler rotations
- Rotation matrices
- Quaternions
- Euler axis and angle

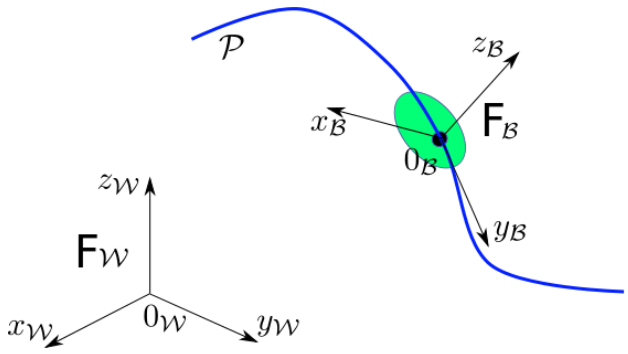


Figure 2.5: Rigid body localization in 3D space

The homogeneous transformation matrix is often used to represent the relative displacement between two frames in computer graphics and robotics because it allows common operations such as translation, rotation, and scaling to be implemented as matrix operations. The displacement from frame F_W to frame F_B can be written as:

$$T_{F_W}^{F_B} = \begin{bmatrix} & & & x \\ & R_{3\times3} & & y \\ & & & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

In which, $R_{3\times3}$ is the rotation matrix and $[x,y,z]^T$ represents the translation. Given a point b which is localized in local frame F_B:

$$^{B}P_b = [x_b\ y_b\ z_b\ 1]^T$$

And $T_{F_B}^{F_W}$ represents the transformation matrix between frame F_B and F_W, then the position of point b in frame F_W is given as:

$$^{W}P_b = T_{F_W}^{F_B\,B}P_b$$

If a third frame is given as F_D, the composition of transformation matrix is also directly given as:

$$T_{F_W}^{F_D} = T_{F_W}^{F_B}T_{F_B}^{F_D}$$

Other representations used in this thesis are quaternion and Euler axis and angle, the details of which is given in Appendix. Readers can also refer to the literature such as the book of Siciliano [Siciliano 08a] for more comparison and discussion on different types of representations for rotations and translations.

2.3.2.2 Online Trajectory Generation (OTG)

A large work is relative to *off-line* trajectory generation in the literature. A general overview of basic off-line trajectory planning concepts is presented in the textbook of Khalil and Dombre [Khalil 99]. Kahn and Roth [M.E 71] belong to the pioneers in the field of time-optimal trajectory planning. They used methods of optimal, linear control theory and achieved a near-time-optimal solution for linearized manipulators. The work of Brady [Brady 82] introduces several techniques of trajectory planning. In later works, the manipulator dynamics were taken into account [Bobrow 85], and jerk-limited trajectories were applied [Kyriakopoulos 88]. The concept of Lambrechts et al. [Lambrechts 04] produces very smooth fourth-order trajectories but is also limited to a known initial state of motion and to one DOF. As the thesis tries to achieve reactive robot control, only *on-line* generation is suitable for the trajectory control level.

We first introduce the concept of the Online Trajectory Generation. A first OTG algorithm is able to calculate a trajectory at any time instant, which makes the system transfer from the current state to a target state. Figure 2.6 illustrates the input and output values of OTG algorithm. The trajectory generator is capable of generating trajectories in one time cycle. The usage of an online trajectory generation algorithm may have two main reasons:

- The first reason is that the trajectory can be adapted in order to improve the path accuracy. [Dahl 90] proposed to use one-dimensional parameterized acceleration profiles along the path in joint space instead of adapted splines. [Cao 94, Cao 98] used cubic splines to generate smooth paths in joint space with time-optimal trajectories. In this work, a cost function was used to define an optimization problem considering the execution time and the smoothness. [Constantinescu 00] suggested a further improvement of the approach of [Shiller 94] by leading to a limitation of the jerk in joint space, considering the limitation of the derivative of actuator force/torques. [Macfarlane 03] presented a jerk-bounded, near time-optimal, one-dimensional trajectory planner that uses quintic splines, which are computed online. Owen published a work on online trajectory planning [Owen 04]. Here, an off-line planned trajectory was adapted online to maintain the desired path. The work of Kim in [Kim 07] took robot dynamics into account.

- The other one is the robotic system must react to unforeseen events based on the sensor singles when the robot works in an unknown and dynamic environment. [Castain 84] proposed a transition window technique to perform transitions between two different path segments. [Liu 02a] presented a one-dimensional method that computes linear acceleration progressions online by parameterizing the classic seven-segment acceleration profile. [Ahn 04] used sixth-order polynomials to represent trajectories, which is named Arbitrary States POlynomial-like Trajectory (ASPOT). In [Chwa 05], Chwa presented an advanced visual servo control system using an online trajectory planner

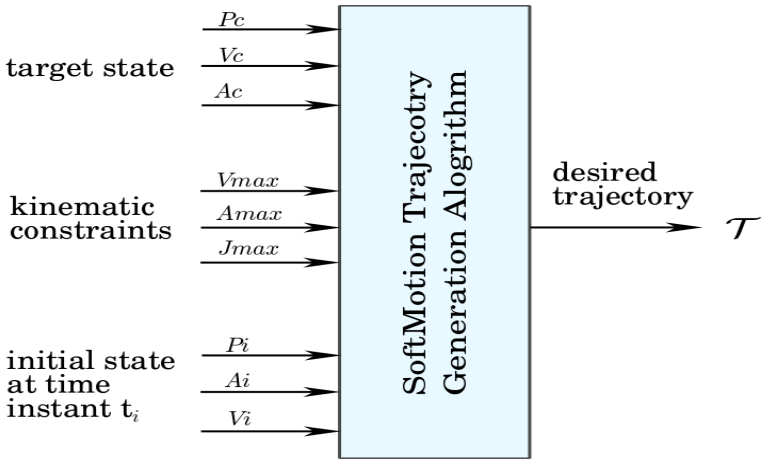


Figure 2.6: Input and output values of the Online trajectory generation algorithm. *P*: Positions, *V*: Velocities, *A*: Accelerations

considering the system dynamics of a two-link planar robot. An algorithm proposed in [Haschke 08a] is able to generate jerk-limited trajectories from arbitrary state with zero velocity. Broquère proposed in [Broquere 08] an online trajectory planner for an arbitrary numbers of independent DOFs. This approach is strongly related to a part of this thesis.

Some approaches to build a controller capable of controlling a complete manipulation tasks are based on Online Trajectory Generation. More results on trajectory generation for robot control can be found in [Liu 02b], [Haschke 08b], and [Kröger 06]. Kröger proposed an algorithm to generate type IV trajectories in table 2.1. Broquère et al. ([Broquere 08]) proposed type V trajectories, with arbitrary final velocity and acceleration. The difference is important when the controller needs to generate trajectories to join points with arbitrary velocity and acceleration. This is the case to follow a trajectory using an external sensor.

2.3.3 Planning-based Trajectory Generation

We present firstly basic notions for planning in presence of geometrical constraints for Human Robot Interaction. Then we discuss the motion of a rigid body in space, before we introduce the motion planning techniques and control laws for the robot manipulators.

2.3.3.1 Geometrical Constraints in HRI

The presence of humans in the workspace of a robot imposes new constraints for the motion planning and the control for navigation and manipulation. This field has been intensively studied especially at *LAAS-CNRS*. The more important constraints are relative to the se-

	$V_F = 0$ $A_F = 0$ $J_F = 0$	$V_F \in \mathbb{R}$ $A_F = 0$ $J_F = 0$	$V_F \in \mathbb{R}$ $A_F \in \mathbb{R}$ $J_F = 0$	$V_F \in \mathbb{R}$ $A_F \in \mathbb{R}$ $J_F \in \mathbb{R}$
$A^{max} \in \mathbb{R}$	Type I	Type II	-	-
$A_{max} \in \mathbb{R}$ $J_{max} \in \mathbb{R}$	Type III	Type IV	Type V	-
$A_{max} \in \mathbb{R}$ $J_{max} \in \mathbb{R}$ $D_{max} \in \mathbb{R}$	Type VI	Type VII	Type VIII	Type IX

Table 2.1: Different types for on-line trajectory generation. V_F: final velocity, A_F: final acceleration, J_F: final jerk.

curity, the visibility and the comfort of human counterpart, two of these constraints are illustrated in Figure 2.7.

For robot motion, the workspace could be associated with many cost maps, each computed for a type of constraint. The first costmap is computed mainly to guarantee the safety and security of people at motion planning and robot control level by considering the distance to dangers. In this case, only distances are taken into consideration. This constraint keeps the robot far from the head of a person to prevent possible dangerous collision between the robot and the person. The theory from [Hall 63] shows that the sensation of fear is generated when the threshold of intimate space is passed by other people, causing insecurity sentiments. For this reason, the cost near a person is high while tends towards zero when the distance becomes high.

The second constraint is called visibility, this is to limit firstly the surprise effect to a person while robot is moving nearby, secondly, a person feels less surprised when the robot is moving in the visible zone, and feels more comfortable and safe [Sisbot 07a]. For example, when the robot hand over an object, this constraints can verify that the person is paying attention to the object exchange.

Other constraints are also used, which can be found in [Sisbot 07a] and related publications. For example, while planning a point in space to exchange an object, this point should be reachable by the person, computed by the length of his arm and the possibility to produce a comfortable posture for the person. A cost of comfort is also computed for every human posture [Yang 04].

When all the cost maps are computed, they are combined in the global cost $c(h,x)$:

$$c(h,x) = \sum_{i=1}^{N} w_i c_i(h,x)$$

in which h is the posture of the human and x represent the three-dimensional position in

which the cost maps are computed. w_i is the weight. This combined cost map is used during the motion planning. In this thesis, we proposed to use it to modulate the velocity at the trajectory control level.

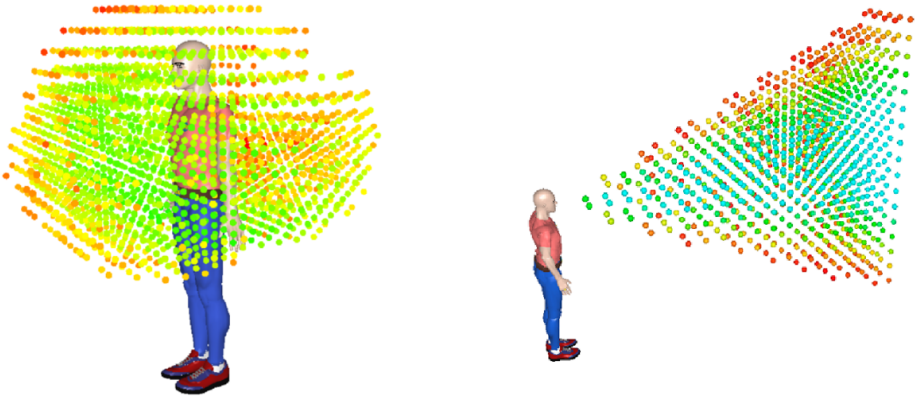


Figure 2.7: Left: 3D reachability map for a human. Green points have low cost, meaning that they are easier for the human to reach, while the red ones, having high cost, are difficult to reach. One application is that when the robot plans to give an object to a human, an exchange point must be chosen in the green zone. Right: 3D visibility map. Based on visibility cost, the controller can suspend the execution if the human is not looking at the robot.

2.3.3.2 Path Planning

Through this thesis, the robot is assumed to operate in a three-dimensional space ($\mathbb{R}^3$), called the work space ($\mathscr{W}$). This space contains many obstacles, which are rigid bodies, written as $\mathscr{WO}_i$, i means it is the i^{th} obstacle. And the free space is then $\mathscr{W}_{free} = \mathscr{W} \setminus \bigcup_i \mathscr{WO}_i$, where $\setminus$ is the set subtraction operator. Motion planning can be performed in working space or in configuration space $\mathscr{Q}$, called C-space (Figure 2.8). C-Space is the set of all robot configurations. Joint space is often used as C-space for manipulators. Configurations are often written as q [1]. The obstacles in the configuration space correspond to configurations where the robot is in collision with an obstacle in the workspace.

A path is a continuous curve from the initial configuration (q_I in Figure. 2.8) to the final configuration (q_F in Figure. 2.8). It can be defined in the configuration space or in the workspace (planning in Cartesian space). Path is different from trajectory such that trajectories are functions of time while paths are functions of a parameter. Given a parameter $u \in [u_{min}, u_{max}]$, often chosen such that $u \in [0,1]$, a path in configuration space is defined as a curve $\mathscr{P}$ such that:

$$\mathscr{P} : [0,1] \rightarrow \mathscr{Q} \;\; where \;\; \mathscr{P}(0) = q_I, \mathscr{P}(1) = q_F \;\; and \;\; \mathscr{P}(u) \in \mathscr{Q}_{free}, \forall u \in [0,1] \qquad (2.3)$$

[1] It should be noticed that q is also used as to represent quaternions.

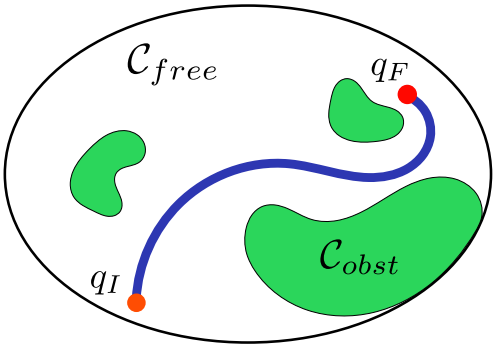


Figure 2.8: Configuration Space and a planned path in C space from q_I to q_F.

Path planning has been one of the essential problems to solve in robotics. Among numerous papers and books, chapter V of *Handbook of Robotics*, by Kavraki and La Valle [Kavraki 08] provides an introduction to this domain, and the book of LaValle [LaValle 06] provides numerous methods. For this work, we selected the large class of planners that provide the output path in the form of via-points. Figure 2.9 shows an example of the result of a path planning, which gives a series of waypoints in the configuration space, linking points q_I and q_F.

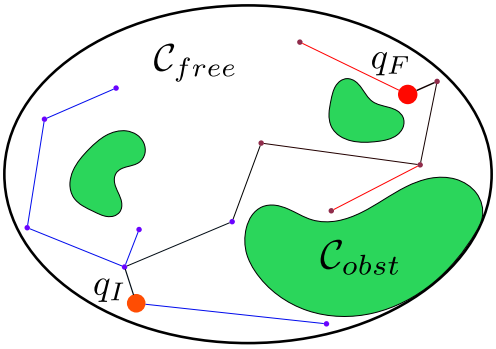


Figure 2.9: Results of path planning (by diffusion) as a series of points in the configuration space. The resulting path is in black.

When the robot shares the workspace with humans, the path planner must take into account the costs of HRI constraints. We perform this planning with the T-RRT method [Jaillet 10] which takes advantage of the performance of two methods. First, it benefits from the exploratory strength of RRT-like planners [LaValle 01a] resulting from their expansion bias toward large Voronoi regions of the space. Additionally, it integrates features of stochastic optimization methods, which apply transition tests to accept or reject potential states. It makes the search follow valleys and saddle points of the cost-space in order to compute low-cost solution paths (Figure. 2.10). This planning process leads to solution paths with low value of integral cost regarding the costmap landscape induced by the cost function.

To smooth the broken line obtained, the shortcut method [Berchtold 94] or the path per-

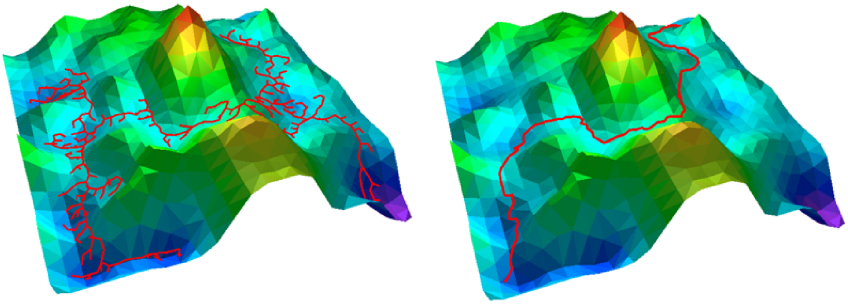


Figure 2.10: T-RRT constructed on a 2D costmap (left). The transition test favors the exploration of low-cost regions, resulting in good-quality paths (right).

turbation variant described in [Mainprice 11] are usually employed. In the latter method, a path $\mathscr{P}(s)$ (with $s \in \mathbb{R}^+$) is iteratively deformed by moving a configuration $q_{perturb}$ randomly selected on the path in a direction determined by a random sample q_{rand}. This process creates a deviation from the current path, the new segment replaces the current segment if it has a lower cost. Collision checking and kinematic constraints verification are performed after cost comparison because of the longer computing time.

The path $\mathscr{P}(s)$ computed with the human-aware path planner consists of a set of via-points that correspond to robot configurations. Via-points are connected by local paths (straight line segments). Additional via-points can be inserted along long path segments to enable the path to be better deformed by the path perturbation method. Thus each local path is cut into a set of smaller local paths of maximal length l_{max}.

2.3.4 Learning-based Trajectory Generation

Here we present a brief overview of methods for motion generation based on observing and repeating sample motion trajectories. Unlike the planning and control methods we presented so far, imitation learning methods make no assumption that a cost function specifying desired motions exists. Rather, the challenge is to learn approximate models (e.g. with machine learning) of this cost using the available data that can be used to generate motion.

2.3.4.1 Direct Policy Learning

Direct Policy Learning (DPL) is one of the fundamental approaches for imitation learning, see [Pomerleau 91, Argall 09]. Sometimes it is also called "behavior cloning", because it is a straightforward approach that tries to repeat observed motions.

A standard way to describe a robot trajectory is $\{x_t, u_t\}_t^T = 0$, where x_t represents the robot state at time t and u_t is the control signal, e.g. the rate of change $\dot{x}_t$. DPL tries to find a policy $\pi : x_t \longmapsto u_t$ from these observations. Different assumptions can be made for the choice of x, u and π [Calinon 07], with refinements like data transformations and

active learning. Given a parameterization of the policy, DPL essentially corresponds to a regression problem, e.g. with loss:

$$E_{dpl} = \sum_{t=0}^{T} \|\pi(x_t) - u_t\|^2 \tag{2.4}$$

where $\|.\|^2$ demotes the squared L_2 norm. Minimizing E_{dpl} finds a policy close in the least squares sense to the demonstrations. The above loss can be extended to multiple demonstration trajectories by averaging over them.

Howard et. al [Howard 09] introduces an interesting alternative loss for DPL:

$$E_{dpl} = \sum_{t=0}^{T} (\|u_t\| - \pi(x_t)^T u_t / \|u_t\|)^2 \tag{2.5}$$

This loss penalizes the discrepancy between the projection of the policy $\pi(x_t)$ on u_t and the true control u_t . This article shows that in some problem domains this loss leads to better behavior than the standard least squares loss.

When the state and control spaces are high dimensional, DPL has a disadvantage: Generalization is an issue and would essentially require the data to cover all possible situations.

2.3.4.2 Markov Decision Process and Reinforcement Learning

A Markov Decision Process (MDP) is a graphical model involving world states (e.g. robot position) and actions a (e.g. go left). It is a popular formalism for multiple learning problems, including Reinforcement Learning (RL), see [Russell 09]. The MDP is defined by the following probabilities, taken for reference from [Jetchev 11]:

- world's initial state distribution $P(s_0)$
- world's transition probabilities $P(s_{t+1}|a_t, s_t)$
- world's reward probabilities $P(r_t|a_t, s_t)$ and $R(a,s) := E\{|a,s\}$
- agent's policy $\pi(a_t|s_t) = P(a_0|s_0;\pi)$

The value (expected discounted return) of policy π when started in state s with discounting factor $\gamma \in [0,1]$ is defined as:

$$V^{\pi}(s) = E_{\pi}\{r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots |s_0\} \tag{2.6}$$

One way to do reinforcement learning in a MDP is to iterate the Bellman optimality equation until convergence of the value function - Value Iteration algorithm [Bellman 03]:

$$V^*(s) = \max_a [R(a,s) + \gamma \sum_{s'} P(s'|a,s) V^*(s'))] \tag{2.7}$$

The optimal policy given the optimal value function is simply the policy maximizing the immediate reward and expected future rewards:

$$\pi^*(s) = arg\max_a [R(a,s) + \gamma \sum_{s'} P(s'|a,s)V^*(s'))] \tag{2.8}$$

The value of a state $V(s)$ is a more global indicator of desired states than the immediate reward of an action $R(a,s)$. The values $V(s)$ provides a gradient towards the desired states-going in direction of increasing $V(s)$ is the desired behavior of the robot system.

2.3.4.3 Inverse Optimal Control

Reinforcement learning or planning in general (e.g. within a MDP framework) tries to generate motions maximizing some reward. Learning (e.g. with Value Iteration) requires constant feedback in the form of rewards for the states and action of the agent. However, in many tasks the reward is not analytically defined and there is no way to access it from the environment accurately. It is then up to the human expert to design a reward leading the robot to the desired behavior. There is an algorithm that can effectively learn the desired behavior and a policy for it just by observing example motions. Inverse Optimal Control (IOC), also known as Inverse Reinforcement Learning (IRL), aims to limit the reward feedback requirement and human expertise required to design behavior for a task. IOC learns a proper reward function only on the basis of data, see [Ratliff 06]. Let's assume that policies π give rise to expected feature counts $\varpi(\pi)$ of feature vectors ϕ: i.e. what feature we will see if this policy is executed. A weight vector ω such that the behavior demonstrated by the expert π^* has higher expected reward (negative costs) than any other policy is learned by minimizing a loss:

$$\min |\omega|^2 \tag{2.9}$$

$$s.t.\ \forall \pi\ \omega^T \varpi(\pi^*) > \omega^T \varpi(\pi) + \mathscr{L}(\pi^*, \pi) \tag{2.10}$$

The term $\omega^T \varpi(\pi)$ defines an expected reward, linear in the features. The scalable margin $\mathscr{L}$ penalizes those policies that deviate more from the optimal behavior of π^*. The above loss can be minimized with a max margin formulation. Efficient methods are required to find the π that violates the constraints the most and add it as new constraint. Once the reward model is learned, another module is required to generate motions maximizing the reward, e.g. [Ratliff 06] uses an A^* planner to find a path to a target with minimal costs.

Learning a policy based on estimated costs is much more flexible than DPL, and a simple cost function can lead to complex optimal policies. IOC can often generalize well to new situations, because states with low costs create a task manifold, a whole space of desired robot positions good for the task. In some domains it is much easier to learn a mapping from state to cost than to learn a mapping from state to action. The latter is a

more complex and higher dimensional problem, especially when considering actions in high dimensional continuous spaces such as robot control.

2.3.4.4 Discriminative Learning

Discriminative learning provides a common framework for many learning problems, including structured output regression. Popular approaches include large margin models [Tsochantaridis 05] and energy based models using neural networks [Lecun 06]. Data is given in the form of pairs of input and output values $\{xi, yi\}$. As in standard discriminative approaches (e.g., structured output learning), the energy or cost $f(x_i, y_i; \omega)$ provides a discriminative function such that the true output should get the lowest energy from the model f:

$$y_i = \arg\min_{y \in \mathsf{y}} f(x_i, y) \tag{2.11}$$

Training the parameter vector ω of the model f is done by minimizing a loss over the dataset. The loss should have the property that f is penalized whenever the true answer yi has higher energy than the false answer with lowest energy which is at least distance r away:

$$\overline{y_i} = \arg\min_{y \in \mathsf{y} \| y - y_i \| > r} f(x_i, y) \tag{2.12}$$

Finding the most offending answer $\overline{y_i}$ is very often a complicated inference problem in itself.

2.4 Robot Motion Control

Firstly, trajectory generation based approaches were developed. In [Buttazzo 94], results from visual system pass firstly through a low-pass filter. The object movement is modeled as a trajectory with constant acceleration, based on which, catching position and time is estimated. Then a quintic trajectory is calculated to catch the object, before being sent to a PID controller. The maximum values of acceleration and velocity are not checked when the trajectory is planned, so the robot gives up when the object moves too fast and the maximum velocity or acceleration exceeds the capacity of the servo controller. In [Gosselin 93], inverse kinematic functions are studied, catching a moving object is implemented as one application, a quintic trajectory is used for the robot manipulator to joint the closest point on the predicted object movement trajectory. The systems used in those works are all quite simple and no human is present in the workspace. A more recent work can be found in [Kröger 12b], in which a controller for visual servoing based on Online Trajectory Generation (OTG) is presented. The results are promising.

Secondly, the research area of visual servoing provides also numerous results, a survey

of which were presented by Chaumette and Hutchinson [Chaumentte 06], [Chaumentte 07] and a comparison of different methods can be found in [Farrokh 11]. Classical visual servoing methods produce rather robust results and stability and robustness can be studied rigorously, but they are difficult to integrate with a path planner, and could have difficulties when the initial and final positions are distant.

Another approach to achieve reactive movements is through Learning from Demonstration (LfD). In [Calinon 04] and [Vakanski 12], points in the demonstrated trajectory are clustered, then a Hidden Markov Model(HMM) is built. Classification and reproduction of the trajectories are then based on the HMM. A survey for this approach is proposed in [Argall 09]. Although LfD can produce the whole movement for objects manipulation, many problems may arise in a HRI context as LfD demands large set of data to learn, and the learned control policies may have problem to cope with a dynamic and unpredictable environment where a service robot works.

The controller must be capable of dealing with various data in HRI context. Compared to methods mentioned above, approaches based on OTG have the following advantages:

- The integration with a path planner is easy and allows to comply with kinematic limits like the one given by human safety and comfort.

- The path to grasp a complex moving object is defined in the object frame, making sure that the grasping movement is collision free.

- The trajectory based method allows to create a simple standard interface for different visual and servo systems, so easy plug-in modules can be created.

2.5 Trajectory Control

Reactive controller for object manipulation is a research topic that is part of the fundamentals of robotic manipulation. Considering that we have a robotic system with multiple degrees of freedom (DOFs) and a mobile base. This system is equipped with one or more sensors delivering digital and/or analog sensor signals. No matter of question, sensor integration and sensor-based control belong to the very basics in robotics. Nonetheless, there is still one important question in the robot control level: If we consider a robot in an arbitrary state of motion, how can we control the robot if we want the robot to react instantaneously to unforeseen sensor events? This is comparable to many scenarios in the human daily life (e.g. if a human touch a very hot surface, he/she will reacts immediately by pulling his/her hand away) and in industries (e.g. during an assembly task, an operator grasps a workpiece when he sees that the workpiece arrives in front of him on a conveyor belt).

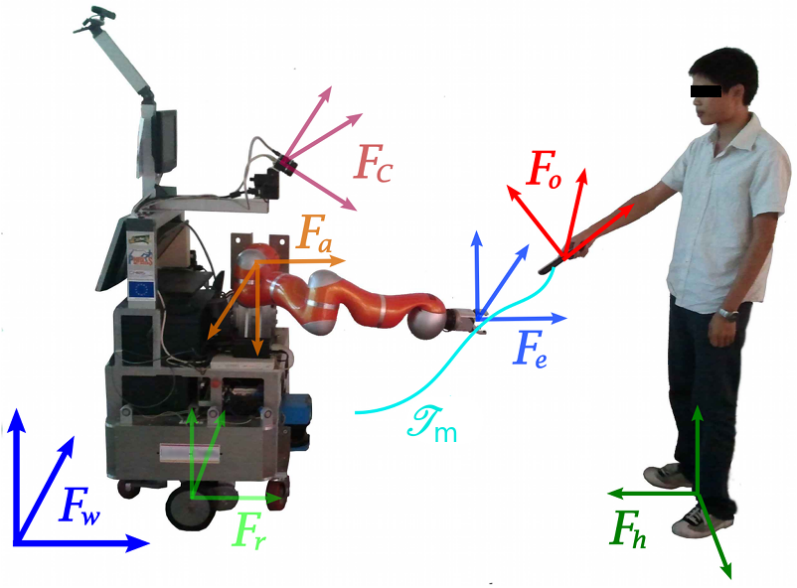


Figure 2.11: Frames for object exchange manipulation: F_w: world frame; F_r: robot frame; F_a: robot base frame; F_c: camera frame; F_e: end effector frame; F_o: object frame; F_h: human frame. The trajectory $\mathcal{T}_m$ realizing a manipulation should be successfully controlled in different task frames. Figure from [He 15].

2.5.1 Control primitives

In HRI, the robot does various tasks like picking up an object, giving an object to human, taking an object from the human. For each task, He [He 13a] proposed first to plan a path to achieve it and then to transform the path into a trajectory. The controller designed here takes directly this trajectory as input and segments it based on the sensor information.

Figure 2.11 shows the basic frames needed to define a task. The trajectory $\mathcal{T}_m$ defines the move that allows the robot to do the task of grasping an object handed by the human.

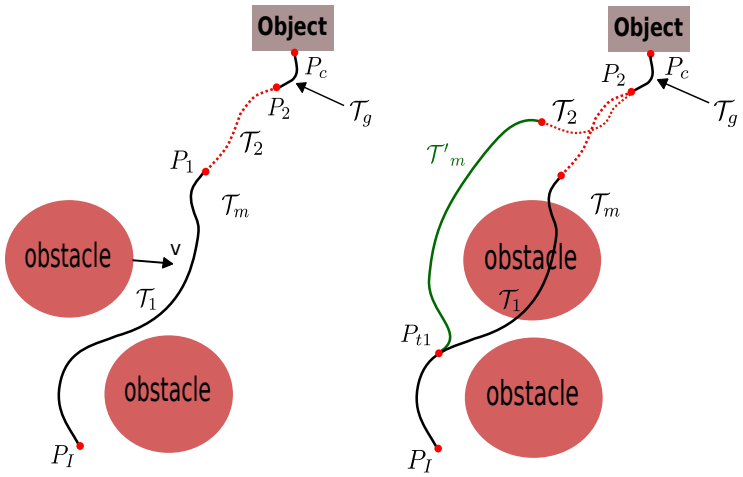


Figure 2.12: Left: trajectories of the control primitives. Right: trajectory switching for the controller due to the movement of an obstacle.

Based on the cost values associated with each point of the trajectory, the trajectory is divided into segments associated with a control strategy. The 3D cost maps used are of different types: collision risk map calculated based on the minimum distance between the trajectory and the obstacles; visibility and reachability map of a human [Sisbot 11] and safety and comfort 3D map of a human, section 2.3.3.1 presented two examples of cost maps. For example, when the risk of collision with the robot base is high, the trajectory can be controlled in the robot frame. Similarly, in the case where the human is handing an object to the robot, the grasping must be controlled in the object frame. [Sidobre 12] details other aspects of the use of cost maps to plan manipulation tasks.

To simplify the presentation, in the reminder of the document we focus on the manipulation tasks where a human hands over an object to the robot. During the manipulations, the human moves and the different frames defining the task move accordingly. Based on the change of cost values, [He 15] divide the trajectory $\mathscr{T}_m$ in Figure 2.11 into three segments, as illustrated in the configuration space in the left part of Figure 2.12. In the figure, the points connecting the trajectory segments are depicted by red dots. The first segment $\mathscr{T}_1$, which is defined in the robot frame, has a high risk of auto-collision. When human or object moves, the cost value of collision risk stays the same. Segment $\mathscr{T}_2$ has a lower collision cost value, so modifying the trajectory inside this zone does not introduce high collision risk. The end part, segment for grasping movement $\mathscr{T}_g$, has a high collision cost value. To ensure the grasping succeeds without collision this segment of trajectory should be controlled in the moving object frame.

We name *task frame* the frame in which the trajectory must be controlled. We define a *control primitive* $\mathscr{C}\mathscr{P}$ by the combination of five elements: a segment of trajectory, a cost function, a task frame, a control mode, and a stop condition.

$$\mathscr{C}\mathscr{P}(t) = (\mathscr{T}_{seg}(t),\ c(t),\ \mathscr{F},\ \mathscr{O},\ \mathscr{S})^T \tag{2.13}$$

In which, $\mathscr{T}_{seg}(t)$ is the trajectory segment, $c(t)$ is the cost value, provided by SPARK and associated with the trajectory which is monitored during the execution of a control primitive, $\mathscr{F}$ is the task frame, $\mathscr{O}$ is the control mode which we will define in next section, and $\mathscr{S}$ is the stop condition of the control primitive. For example, the grasping movement includes five elements: the trajectory segment $\mathscr{T}_g$, the high collision risk cost value $C(t)$, the task frame $\mathscr{F}_o$, the control mode as trajectory tracking, and the stop condition $\mathscr{S}$ as a predefined threshold for the distance between the robot end effector and the end point of $\mathscr{T}_g$. In the literature, Manipulation Primitives or Skill Primitives are often the concept for the intermediate level between planning and control and have been discussed in numerous works, as in [Kröger 11].

Using the definition of control primitives ($\mathscr{C}\mathscr{P}(t)$) and Motion Condition: $M(t) = (X(t), V(t), A(t))$, or written as M_t. The initial trajectory $\mathscr{T}_m$ is segmented into a series of $\mathscr{C}\mathscr{P}(t)$. The cost values $c(t)$ are used during the segmentation, they are also monitored

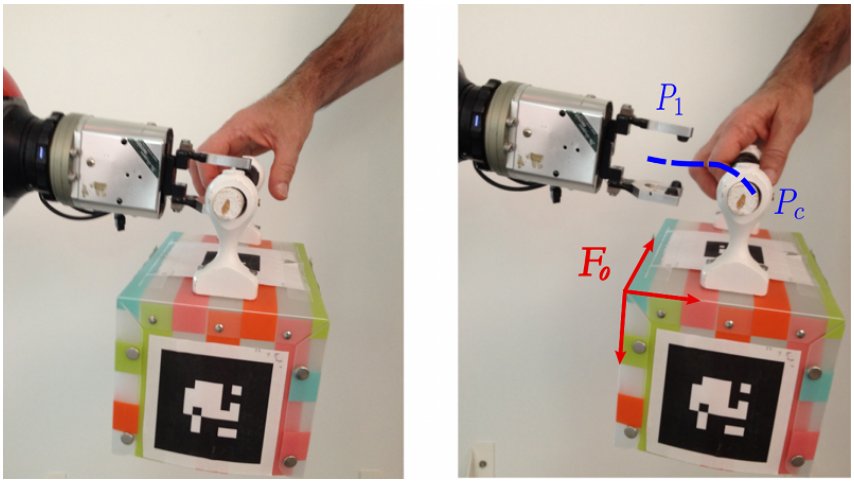


Figure 2.13: A simple case of grasp. Left: a planned grasp defines contact points between the end effector and the object. Right: To finish the grasping, the manipulator must follow the blue trajectory P_1 - P_c, and then close the gripper. This movement must be controlled in the object frame F_o.

by the controller during execution of a control primitive. The collision checker integrates data from vision, human perception and encoder of the robot. It prevents collision risk by slowing down or suspending the task execution. With all the data and the current Motion Condition M_t of the robot, different control modes can compute Motion Condition for the next control cycle, which are the input for the robot servo system.

Figure 2.13 shows the last control primitive of *grasping an object*. It is similar to the end part, $\mathscr{T}_g$, of the trajectory in Figure 2.11. The grasp position, the contact points and the final trajectory are planned by the grasp planner. More details on the grasp planner are given in [Bounab 08] and [Saut 12]. When the object moves, the object frame F_o and the path of the trajectory move at the same time. So to avoid collision, the trajectory of these control primitives must be controlled in the object frame F_o.

2.5.2 Reactive Trajectory Controller

At the control level, a task is defined by a series of control primitives, each defined by a quintuplet. The first level of the proposed trajectory controller is a state machine, which monitors the execution, controls the succession of the control modes, and manages the collision risk and other special situations. Target tracking and trajectory tracking are parts of the control modes presented after the state machine.

2.5.2.1 Execution Monitoring

A state machine controls the switching between the different control modes associated with each control primitive and monitors the execution. Due to human presence, the robot en-

vironment is moving and the control task must be adapted accordingly. The state machine can also suspend or stop the control of a control primitive like depicted in Figure 2.14.

- Suspend Events: When the visual system fails or the target becomes unavailable, or because of some specific human activities based on the monitoring of cost value $\mathscr{C}(t)$, the trajectory controller should suspend the task.

- Stop Events: Whatever the control mode chosen, unpredictable collisions can occur and they must stop the robot. Our controller uses two modules to detect these situations.

  The first is a geometric collision checker based on results from Larsen et al.[Larsen 99]. It updates a 3D model of the workspace of the robot, and runs at the same frequency as the trajectory controller. This checker is geometric based, and can stop the robot when a collision between the robot and the environment is predicted.

  De Luca [De Luca 08] proposed to monitor the external torques. The method was designed to detect unexpected physical collision between the robot and the obstacles. The fast detection of collision is realized using the momentum-based method reported in the paper, which does not require any external sensing. This monitor provides a security guarantee for Human Robot Interaction context. With the implementation of the torque monitor on the robot, the robot automatically stops when collision occurs.

- Slow Down On Trajectory: Based on the input cost function, the controller can slow down on the main trajectory by changing the time function $s(t)$. Imagine that a fast movement could cause some people anxiety when the robot is close to them, for example. We propose to use the geometric models of the robot and models of the human, updated at each iteration during the execution to ensure the safety and comfort of humans. We choose to take into account the weighted average *cost* of the security and visibility constraints introduced in chapter II. The method to adapt the motion law is the same as the one presented in the previous section. The costs are high when the distance human-robot is short or when the robot is outside the field of view of the human, the cost taken into account is $cost_{inv} \in [0,1]$ such that:

$$cost_{inv}(k\Delta T) = 1 - cost(k\Delta T) \tag{2.14}$$

  The cost $cost_{inv}$ is then smoothed on-line, using methods presented in section 3.2.2.5.

Each elementary controller based on online trajectory controller is implemented with a simple state machine inside.

2.5.2.2 Trajectory Control Mode

[He 13b] also proposed several control modes for the trajectory controller:

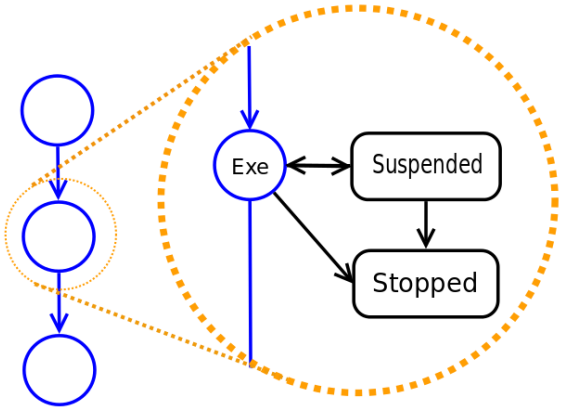


Figure 2.14: In the left, each circle represents the controller of a control primitive. The system can suspend or stop the execution of a control primitive.

- Target tracking. If we suppose the robot is in an area without risk of collision, the system can track the end point of the trajectory. In this case, the controller generates iteratively a trajectory to reach the end point and send the first step of this trajectory to a low-level controller. In the special case where the controller does target tracking with visual system, it does visual servoing.

- Trajectory tracking in task frame. Once the robot reaches a point, which is close to the object to be grasped, it starts the grasping movement. The object can still move, but as the robot is in the high cost zone, it should track the grasp trajectory in the task frame.

- Path re-planning and trajectory switching. During the execution, a path can be re-planned, for example when an obstacle blocks the robot's trajectory. A new trajectory is computed by the planner and given to the controller that switches to the new trajectory.

The controller integrates various information from high-level software, such as SPARK (Spatial Reasoning and Knowledge Module) and MHP (Motion in Human Presence), which were presented previously in this chapter.

2.6 Conclusion

This chapter presented firstly the software architecture for Human Robot Interaction and included the literature review of the topic of this thesis and some presentation of the related works at LAAS-CNRS. Then the planning based motion generation and learning motion

generation were surveyed. This thesis has been a part of the effort to develop a reactive robot manipulator. For this reason, a controller in the trajectory level is essential. So we compared both the trajectory controller based on planning and learning, and concluded that in the HRI context, learning approaches are not fast enough to cope with a dynamic and unpredictable environment. However, a controller based on Online Trajectory Generation can integrate various information from high-level software, visual and servo systems, and the low-level robot controller.

3

Methodology: Trajectory Generation

3.1 Introduction

Introducing a trajectory generator inside the controller provides the robot with a description of the near future of its movements. These trajectories allow the robot to anticipate different events and notably collisions. This anticipation gives a strong advantage to the high level layers of a hierarchical controller, which can in particular better ensure the safety of the interacting humans.

In this chapter, we present tools to define and manipulate trajectories with the aim of building robot multi-layer controller. Trajectory generators and trajectory approximations define two types of tools. Tools for multi-dimensional trajectories are more complexes and some tools developed for approximation are useful for generation. We also propose to take advantage of the time evolution defining a trajectory to improve the smoothing of trajectories compared with the classical approach that smooth firstly the path before generating the time evolution.

Autonomous robots that interact with humans or operate in human environments must have the capability to quickly generate safe and natural looking motion. So far, it has been a challenge to simultaneously satisfy the three objectives of speed, safety, and aesthetics for high-DOF robots performing complex tasks in unstructured environments. Sample-based planners (e.g., probabilistic roadmaps (PRMs)[Kavraki 96], rapidly-exploring random trees(RRTs)[LaValle 01b], etc.) are widely used to plan collision-free paths for high-

DOF robots. These planners are often fast, but they produce jerky and unnatural paths, which are, for example, defined by via-points as a broken lines. The following of these paths by a robot is not possible without stopping the robot at every vertex along the path. This is slow and looks unnatural, so smoothing is often performed before execution.

The rest of the chapter is organized as follows. Firstly, we give the basic concept of converting paths to trajectories in the Section 3.2. The polynomial line trajectory generation will be discussed in Section 3.3. After this, two different algorithms to smooth the generated straight-line trajectory are presented in Section C.2.4 and Section C.2.5, respectively.

3.2 A Trajectory Model

A *Path* denotes the locus of points in the joint space, or in the operational space, which the manipulator has to follow in the execution of the assigned motion. A path is then a pure geometric description of motion. The goal of trajectory generation if to generate the reference inputs to the motion control system which ensures that the manipulator executes the planned trajectory.

Broquère proposed the Soft Motion Trajectory planner to generate a trajectory from a path[Broquere 08, Broquère 10, Broquère 11]. The path can come from a RRT path planner or its variants, presented in chapter II. This section is the results of previous work at *LAAS*, and has been reported in [Sidobre 12]. The author of this document has participated in some development and the test of the software. Research in robotics is often a cooperative work, and the content of this section, although not part of the scientific contribution of the author, is included because it is a key to understand this thesis.

3.2.1 Basic Concepts of the Trajectory Generation

3.2.1.1 Trajectory Model

Trajectories are time functions defined in geometrical spaces, essentially Cartesian space and joint space. The rotations can be described using different coordinates system: quaternion, vector and angle etc. The books from Biagiotti [Biagiotti 08] on one hand and the one from Kröger [Kröger 10a] on the other hand summarize background trajectory material.

Given a system whose position is defined by a set of coordinate X if the coordinates are in Cartesian space or Q if the coordinates are in joint space, a trajectory $\mathscr{T}$ is a function of time defined as:

$$\mathscr{T} : [t_I, t_F] \longrightarrow \mathbb{R}^n \tag{3.1}$$

$$t \longmapsto \mathscr{T}(t) = X(t) \quad \text{or} \quad Q(t) \tag{3.2}$$

Where $\mathscr{T}(t) = Q(t) = ({}_1Q(t), {}_2Q(t), \cdots, {}_nQ(t))^T$ for joint motions or $\mathscr{T}(t) = X(t) = ({}_1X(t), {}_2X(t), \ldots, {}_nX(t))^T$ in Cartesian space. The trajectory is defined from the time interval $[t_I, t_F]$

to $\mathbb{R}^n$ where n is the dimension of the motion space. The $\mathscr{T}(t)$ function can be a direct function of time or the composition $\mathscr{C}(s(t))$ of a function giving the path $\mathscr{C}(s)$ and a function $s(t)$ describing the time evolution along this path.

At first glance the latter offer more possibilities as the time evolution is independent of the geometrical path and so the two elements can be modified independently. Unfortunately, this approach is limited by the difficulty to integrate the derivative of the path to obtain the curvilinear abscissa. Without this parameterization, the function $s(t)$ doesn't give the tangential velocity and the kinematic of the motion is difficult to manipulate and interpret. So, as the former has a simpler expression, it provides simpler solutions to define and manipulate trajectories.

A trajectory $\mathscr{T}(t)$ defined from t_I to t_F can be defined by a series of trajectories defined between intermediate points. Given, t_u which satisfies $t_I < t_u < t_F$, an equivalent representation of $\mathscr{T}(t)$ is defined by the series of two trajectories T_1 and T_2 defined respectively by:

$$\begin{array}{ll} \mathscr{T}_1 : [t_I, t_u] \longrightarrow \mathbb{R}^n & \mathscr{T}_2 : [t_u, t_F] \longrightarrow \mathbb{R}^n \\ t \longmapsto \mathscr{T}_1(t) = \mathscr{T}(t) & t \longmapsto \mathscr{T}_2(t) = \mathscr{T}(t) \end{array} \tag{3.3}$$

Similarly a trajectory can be defined by a series of sub-trajectories if some continuity criteria specified for the trajectory and its derivative are verified. Generally this criterion is defined as a differentiability class C^k with $k \geq 2$. The possible choices to define trajectory functions are very large, but as we intend to compute motions in real time, we need a simple solution like polynomial functions. As we need C^2 functions, we choose polynomial function of third degree and name this trajectories Soft Motion trajectories. Using a long series of polynomial function, trajectories following very complex path can be defined. It is also possible to approximate or interpolate a set of points to define Soft Motions trajectories. In the sequel, we firstly present Soft Motion trajectories and then a set of consistent trajectory generator to solve robotic problems.

3.2.1.2 Motion Condition

For the discussion of the next sections, we define a Motion Condition $M(t)$ as a triplet associating the position, velocity and acceleration at time t along the trajectory:

$$M(t) = (X(t),\, V(t),\, A(t)) \quad \text{in Cartesian space} \tag{3.4}$$

$$= (Q(t),\, \dot{Q}(t),\, \ddot{Q}(t)) \quad \text{in joint space} \tag{3.5}$$

We define the function *getMotionCond* from $[t_I, t_F] \subset \mathbb{R}$ to $\mathbb{R}^3$ to compute the motion condition $M(t)$ from a trajectory $\mathscr{T}$ and a time t:

$$M(t) = getMotionCond(t, \mathscr{T}) \tag{3.6}$$

3.2.1.3 Series of 3rd degree polynomial trajectories

A trajectory $\mathscr{T}(t)$ is represented by a combination of n series of cubic polynomial curves. The trajectory ${}_j\mathscr{T}(t)$ corresponds to the evolution of the j axis and is composed of N cubic polynomial segments (curves) (Figure 3.1). We consider that all the axes have the same number of segments otherwise they can be divided.

Functions ${}_jJ_k(t)$, ${}_jA_k(t)$, ${}_jV_k(t)$, ${}_jX_k(t)$ respectively represent the jerk, acceleration, velocity and position evolution over the segment k for the axis j. t_I is the initial time of the trajectory and t_F the final one.

A segment is defined by the Eq. (3.7) and depends on its duration T_k and on five parameters:

- the initial time t_{lk},
- the initial conditions (3 parameters: ${}_jA_k(t_{lk})$, ${}_jV_k(t_{lk})$, ${}_jX_k(t_{lk})$),
- the jerk value ${}_jJ_k$

$\forall t \in [t_{lk}, t_{lk}+T_k]$:

$$ {}_jX_k(t) = \frac{{}_jJ_k}{6}(t-t_{lk})^3 + \frac{{}_jA_k(t_{lk})}{2}(t-t_{lk})^2 + {}_jV_k(t_{lk})(t-t_{lk}) + {}_jX_k(t_{lk}) \tag{3.7} $$

where ${}_jJ_k$, ${}_jA_k(t_{lk})$, ${}_jV_k(t_{lk})$, ${}_jX_k(t_{lk})$ and t_{lk} are constant $\in \mathbb{R}$.

The initial Motion Conditions of the trajectory ${}_j\mathscr{T}(t)$ are ${}_jM_I = ({}_jX_I, {}_jV_I, {}_jA_I)$:

$$ \begin{aligned} {}_jX_1(t_I) &= {}_jX_I \\ {}_jV_1(t_I) &= {}_jV_I \\ {}_jA_1(t_I) &= {}_jA_I \end{aligned} \tag{3.8} $$

and the final conditions ${}_jM_F = ({}_jA_F, {}_jV_F, {}_jX_F)$:

$$ \begin{aligned} {}_jX_N(t_F) &= {}_jX_F \\ {}_jV_N(t_F) &= {}_jV_F \\ {}_jA_N(t_F) &= {}_jA_F \end{aligned} \tag{3.9} $$

where $t_F - t_I = \sum_{i=1}^{K} T_i$.

The multidimensional trajectory is then a composition of trajectories as:

$$ \mathscr{T}(t) = [{}_1\mathscr{T}(t)\ {}_2\mathscr{T}(t)\ \dots\ {}_n\mathscr{T}(t)]^T \tag{3.10} $$

where n is the number of axis.

We define Soft Motion trajectories as series of 3^{rd} degree polynomial trajectories. Such a trajectory is composed of a vector of one-dimensional trajectories. Without loss of generality, we suppose that all ${}_jX(t)$ or all ${}_jQ(t)$, $0 \leq j < n$ (n is the dimension of the space) share the same time intervals and that $t_I = 0$.

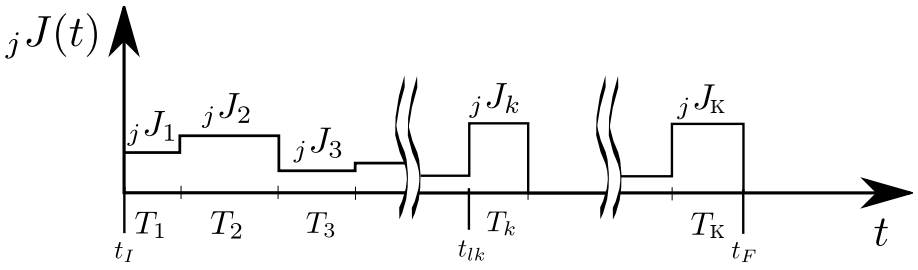


Figure 3.1: The jerk evolution for the j axis of the $\mathscr{T}(t)$ trajectory.

A one dimensioned trajectory ${}_jX(t)$ can be defined by its initial motion conditions (${}_jX(0) = {}_jX_I$, ${}_jV(0) = {}_jV_I$ and ${}_jA(0) = {}_jA_I$) and K elementary trajectories ${}_jX_i(t)$ defined by the jerk ${}_jJ_i$ and the duration T_i where $1 \leq i \leq K$ and $\sum_{i=1}^{K} T_i = t_F - t_I$. By integration we can define the acceleration ${}_jA_i(t)$, the velocity ${}_jV_i(t)$ and then the position ${}_jX_i(t)$.

Assuming $k \leq K$ is such that $\sum_{i=1}^{k-1} T_i \leq t < \sum_{i=1}^{k} T_i$, the trajectory ${}_jX(t)$ and its derivative are defined by :

$$ {}_jJ(t) = {}_jJ_k \tag{3.11} $$

$$ {}_jA(t) = {}_jJ_k \left(t - \sum_{i=1}^{k-1} T_i \right) + \sum_{l=1}^{k-1} {}_jJ_l \, T_l + {}_jA_I \tag{3.12} $$

$$ \begin{aligned} {}_jV(t) = {} & \frac{{}_jJ_k}{2} \left(t - \sum_{i=1}^{k-1} T_i \right)^2 + \sum_{l=1}^{k-1} {}_jJ_l \, T_l \left(t - \sum_{i=1}^{l} T_i \right) \\ & + \sum_{l=1}^{k-1} \frac{{}_jJ_l \, T_l^2}{2} + {}_jA_I t + {}_jV_I \end{aligned} \tag{3.13} $$

$$ \begin{aligned} {}_jX(t) = {} & \frac{{}_jJ_k}{6} \left(t - \sum_{i=1}^{k-1} T_i \right)^3 + \sum_{l=1}^{k-1} \frac{{}_jJ_l \, T_l}{2} \left(t - \sum_{i=1}^{l} T_i \right)^2 \\ & + \sum_{l=1}^{k-1} \frac{{}_jJ_l \, T_l^2}{2} \left(t - \sum_{i=1}^{l} T_i \right) + \sum_{l=1}^{k-1} \frac{{}_jJ_l \, T_l^3}{6} \\ & + \frac{{}_jA_I}{2} t^2 + {}_jV_I t + {}_jX_I \end{aligned} \tag{3.14} $$

This general expression of the trajectories and their derivatives can be used directly to control a arm, for example, but it is not easy to obtain directly. So we will now describe different generators to build these trajectories.

From the K couples $({}_jJ_k, T_k)$ and the initial conditions (3.8) of the trajectory ${}_j\mathscr{T}(t)$ we can compute the Motion Condition along the j axis at a given time with (3.12), (3.13) and

(3.14). In order to simplify the notation, the j index representing the axis will be omitted most of the time.

3.2.2 One Dimensional Point to Point Trajectory Generation

3.2.2.1 The Kinematic Constraints

The trajectory generation method is based on constraints satisfaction for velocity, acceleration and jerk. Each constraint is supposed constant along the planned motion. In the multidimensional case, each axis can have different constraints. We also suppose that the constraints are symmetrical:

$$\begin{aligned} {}_jJ_{min} &= -{}_jJ_{max} \\ {}_jA_{min} &= -{}_jA_{max} \\ {}_jV_{min} &= -{}_jV_{max}. \end{aligned} \tag{3.15}$$

Hence, the jerk, acceleration and velocity must respect:

$$\begin{aligned} |{}_jJ(t)| &\leq {}_jJ_{max} \\ |{}_jA(t)| &\leq {}_jA_{max} \\ |{}_jV(t)| &\leq {}_jV_{max}. \end{aligned} \tag{3.16}$$

In the context of human robot interaction, safety and comfort should be concerned in any applications. In the literature, a lot of works considered the human's safety and/or human's comfort in a human-robot interaction task, either for service robotics or for industrial robotics [Alami 06, Meisner 08, Arai 10, Sisbot 07b, Sisbot 10, Haddadin 08, Tonietti 05, Rybski 12, Mainprice 13, Nikolaidis 13]. The first, and most obvious, is physical safety. To maintain physical safety, all unwanted human-robot contact must be prevented, and if contact is required by the task at hand or is inevitable for another reason, the forces exerted by the robot on the human must fall below limits that could cause discomfort, large velocity or injury

The second, and often overlooked aspect is comfort. In the context of HRI, the robot should not cause excessive stress and discomfort to the human for extended periods of time. Therefore, the separation distance, the end effector speed, the advance notice of robot motion and the human's field of vision should be all considered in the trajectory planning level.

3.2.2.2 The Canonical Point-to-Point Case and its Kinematic Constraints

In the basic case a motion between two points where initial and final kinematic conditions are null, the Figure 3.2 represents the optimal point-to-point motion (according to the im-

posed kinematic constraints). This point-to-point motion is composed of seven segments of cubic polynomial functions at most [Broquere 08].

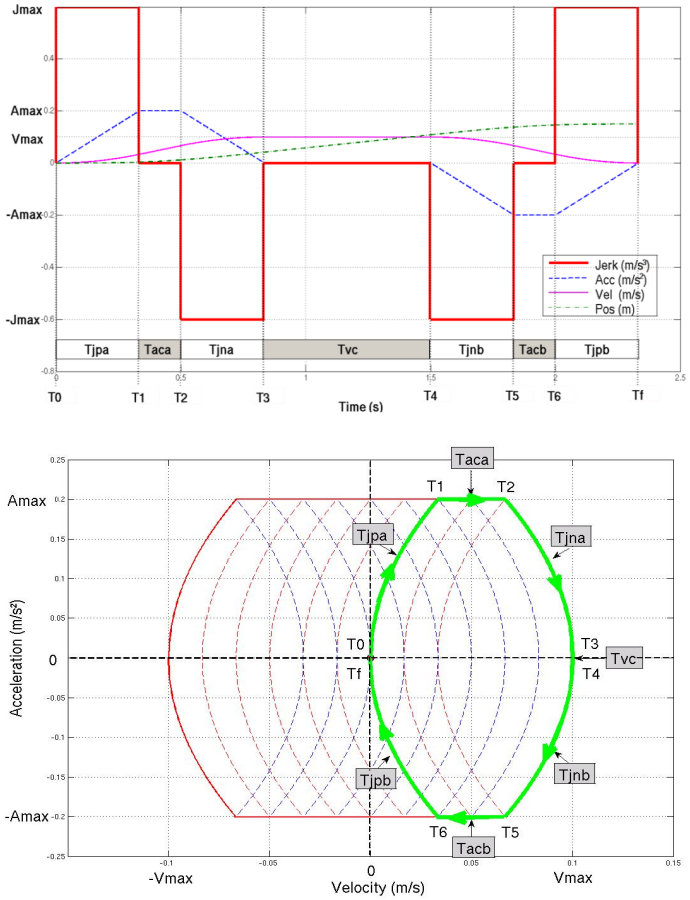


Figure 3.2: Jerk, acceleration, speed and position curves and motion in the acceleration-velocity frame for a single axis.

In the multidimensional case each axis has also seven cubic polynomial segments at most. Computation details can be found in [Broquère 11].

3.2.2.3 The Minimal Time Motion Between Two Non-null Kinematic Conditions

From the canonical point-to-point case we extend the monodimensional algorithm to compute minimal time motion between two non-null kinematic states (non-null acceleration and velocity). An overview of this algorithm is presented in [Broquere 08] and the details in [Broquère 11]. This kind of motion is composed of a set of elementary motions saturated in jerk, acceleration or velocity. The number of elementary motions is also seven at most. For the multidimensional case, [Broquère 11] proposes a solution to synchronize the axis motions.

3.2.2.4 The Time Imposed Motion Between Two Non-null Kinematic Conditions: the *3-Segment Method*

[Broquère 10] previously presented a method for computing a motion with an imposed duration. This method does not bound the jerk, acceleration nor velocity. It uses three cubic polynomial curves to define such a motion. This simple definition provides a solution to compute analytically the motion.

3.2.2.5 Smoothing an Input Function

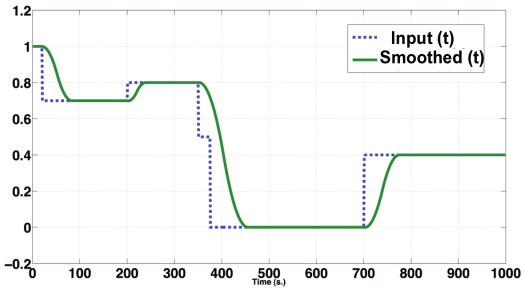


Figure 3.3: Example of the smoothing of a set function.

Broquère proposed a method in [Broquere 08] to compute online a smooth movement from an input defined by acceleration and velocity. At each update of the set function, a move is computed from the current state of the system. This move is bounded by the kinematic constraints (J_{max}, A_{max} and V_{max}). Under these kinematic constraints, the minimal time motion is defined by the critical movement associated to the critical length dc [Broquere 08].

Thus, in order to allow a mono-dimensional system to reach its set value in minimal time, the critical movement is computed at each iteration. An example of a smoothed signal is plotted in the Figure 3.3. The blue dotted curve is the input and the green curve is the smoothed velocity. The method acts like a filter for the acceleration.

3.3 Trajectory Generation From a Given Path

In this section, we introduce how to convert a path, defined in the form of several waypoints, to a trajectory that can be directly followed by the robot. The input is the path $\mathscr{P}$ computed by the path planner, such as a RRT planner or a PRM planner.

The first step is to calculate a trajectory passing through all the nodes of the path $\mathscr{P}$. This trajectory, which we call $\mathscr{T}_{ptp}$ consists of point-to-point movement (Sect. 3.2.2.2) and therefore includes stop motions at each configuration defining a node.

In literature there are several examples of bounded jerk trajectory planning methods also known as “Trajectory with Double S Velocity Profile[Biagiotti 08]”. Several algo-

rithm for planning trajectories with bounded jerk was proposed [Broquère 10] [Gerelli 10] [Haschke 08b]. But, none of the already proposed methods manage correctly the shape of the trajectory and the inter-axis phase synchronisation for multi-DOFs movements. They manage to synchronize the trajectory for multi-DOFs robots through time scaling techniques. This technique pre-calculates the time on each DOF and readjusts the kinematics limits ($J_{max}, A_{max}, V_{max}$) in the basis of the slowest variable. However, trajectories are only guaranteed to have the same duration and not the real synchronization, since motion variables do not complete the same percentage of the trajectory at a given instant of time. In other words, this only succeeds to synchronize the time along the whole motion, but not to manage to synchronize the times on each 3^{rd} degree polynomial segment.

Our proposed method called On-line Phase-Synchronized Bounded Jerk Trajectory allows us to overcome all these issues, assuring phase synchronization, low computational time for real-time purposes, no need of iterative/optimization processes and no collisions with mechanical stops. It is one of the contributions of this thesis. In the following sections, we take the joint space trajectory generation as an example to detail the approach.

3.3.1 Phase-Synchronized Trajectory

3.3.1.1 Trajectory Generation and Synchronization

Let $\mathscr{C} = \mathbb{R}^n$ denote the n-dimensional configuration space, q_0 is the initial position and $q_f \in \mathscr{C}$ is the target joint value. Once the final joint value $q_f \in \mathscr{C}$ has been determined, the trajectory is generated by imposing that all the first three time derivatives of joint variables have to be limited, according to the bounded jerk planning definition:

$$\begin{aligned} |\dot{q}| &\leq V_{max} \\ |\ddot{q}| &\leq A_{max} \\ |\dddot{q}| &\leq J_{max} \end{aligned} \tag{3.17}$$

where J_{max} is a user-design parameter representing the maximum allowed jerk, that should be fixed according to measurements performed on human subject movements. All constraints $[q_0, q_f, V_{max}, A_{max}, J_{max}]$ are imposed respectively by maximum joint admissible excursions and maximum kinematic values of motor performance according to the human state.

Different solutions were proposed to generate monoaxial trajectories, but for the multiaxial case, generating trajectories with the same duration is not enough and a continuous phase synchronization is necessary to define the shape of the path. For example, to generate a straight line the ratio between the velocities of each axes must be constant.

Our trajectory planner takes as input the values $[q_0, q_f, V_{max}, A_{max}, J_{max}]$ and generates for each joint a bang–bang jerk law $\dddot{q}(t)$. The jerk law is then integrated three times to

obtain the trajectory $q(t)$ to follow.

Definition 1. *Phase synchronization is the synchronization in position, velocity, acceleration and jerk spaces. It means that, given any instant of time, all variables must complete the same percentage of their trajectories. In a n dimensional space, it verifies the following law:*

$$\frac{{}_iq(t)-{}_iq(t_I)}{{}_jq(t)-{}_jq(t_I)}=\frac{{}_iq(t)-{}_iq_0}{{}_jq(t)-{}_jq_0}=\lambda \quad \forall i,j\in[1,n],t\in[t_0,t_f] \tag{3.18}$$

Where λ is a constant, q_0 is the initial position. To compute the factor λ simply, we use the initial joint value and the final joint value which are given as conditions:

$$\lambda=\frac{{}_iq(t_f)-{}_iq_0}{{}_jq(t_f)-{}_jq_0}=\frac{{}_iq_f-{}_iq_0}{{}_jq_f-{}_jq_0} \tag{3.19}$$

Applying the equation above on each axis, we can get a set of λ. Since our trajectory planner generate jerk profile first and then make the integration, it is necessary to develop the relationship between the jerk profile of each joint and the constant λ. For sake of simplicity, we consider a two-DOFs robot manipulator, then we have the following property:

Property 1. *Given that the following initial conditions are verified*

$$\begin{aligned} {}_1\dot{q}(t_0)={}_2\dot{q}(t_0)=0 \quad {}_1\dot{q}(t_f)={}_2\dot{q}(t_f)=0 \\ {}_1\ddot{q}(t_0)={}_2\ddot{q}(t_0)=0 \quad {}_1\ddot{q}(t_f)={}_2\ddot{q}(t_f)=0 \end{aligned}$$

and that the following kinematic constraints hold all along the trajectory

$$\frac{{}_1\dot{q}_{max}}{{}_2\dot{q}_{max}}=\frac{{}_1\ddot{q}_{max}}{{}_2\ddot{q}_{max}}=\frac{{}_1\dddot{q}_{max}}{{}_2\dddot{q}_{max}}=\frac{{}_1J_{max}}{{}_2J_{max}}=\lambda \tag{3.20}$$

then, the synchronization condition given by Eq. C.2 is also satisfied along the trajectory. Moreover, we will have the following equalities along the trajectory as well:

$$\frac{{}_1\dot{q}(t)}{{}_2\dot{q}(t)}=\frac{{}_1\ddot{q}(t)}{{}_2\ddot{q}(t)}=\lambda \quad \forall t\in[t_0,t_f] \tag{3.21}$$

Proof 1. *For the sake of clarity only two joint variables will be taken into account. Since we use 3^{rd} polynomial functions to represent the trajectory, therefore we have the following*

equalities in any time interval $t \in [t_0, t_f]$*:*

$$
\begin{aligned}
{}_iq(t) &= {}_i\dddot{q}(t_0)\frac{(t-t_0)^3}{6} + {}_i\ddot{q}(t_0)\frac{(t-t_0)^2}{2} + {}_i\dot{q}(t_0)(t-t_0) + {}_iq(t_0) \\
{}_i\dot{q}(t) &= {}_i\dddot{q}(t_0)\frac{(t-t_0)^2}{2} + {}_i\ddot{q}(t_0)(t-t_0) + {}_i\dot{q}(t_0) \\
{}_i\ddot{q}(t) &= {}_i\dddot{q}(t_0)(t-t_0) + {}_i\ddot{q}(t_0)
\end{aligned}
$$

where $i \in [1,2]$*. If the initial and final conditions (the velocity and acceleration) are null, and* λ *satisfies the Eq. 3.20, at any time instant* t*, the accelerations of each joint are :*

$$
\begin{aligned}
{}_1\ddot{q}(t) &= {}_1\dddot{q}(t_0)(t-t_0) + {}_1\ddot{q}(t_0) = {}_1J_{max}(t-t_0) && (3.22) \\
{}_2\ddot{q}(t) &= {}_2\dddot{q}(t_0)(t-t_0) + {}_2\ddot{q}(t_0) = {}_2J_{max}(t-t_0) \\
&= \frac{1}{\lambda}{}_1J_{max}(t-t_0) = \frac{1}{\lambda}{}_1\ddot{q}(t) && (3.23)
\end{aligned}
$$

while the velocity returns to the following representations respectively:

$$
\begin{aligned}
{}_1\dot{q}(t) &= {}_1J_{max}\frac{(t-t_0)^2}{2} + {}_1\ddot{q}(t_0)(t-t_0) && (3.24) \\
{}_2\dot{q}(t) &= {}_2J_{max}\frac{(t-t_0)^2}{2} + {}_2\ddot{q}(t_0)(t-t_0) = {}_2J_{max}\frac{(t-t_0)^2}{2} + \frac{1}{\lambda}{}_1\ddot{q}(t_0)(t-t_0) \\
&= \frac{1}{\lambda}({}_1J_{max}\frac{(t-t_0)^2}{2} + {}_1\ddot{q}(t_0)(t-t_0)) = \frac{1}{\lambda}{}_1\dot{q}(t) && (3.25)
\end{aligned}
$$

Then, the Eq. 3.21 of Property 1 is proved. To verify if the trajectory is phase-synchronized or not when the kinematic constraints satisfy Eq. 3.20, we need to represent the relationship between the joint value $q(t)$ *and the maximum jerk* J_{max} *for each axis.*

$$
\begin{aligned}
{}_1q(t) &= {}_1J_{max}\frac{(t-t_0)^3}{6} + {}_1q_0 \\
{}_2q(t) &= {}_2J_{max}\frac{(t-t_0)^3}{6} + {}_2q_0
\end{aligned}
$$

Thus, we can get the following equality with the transformation of the above equations:

$$
\frac{{}_1q(t) - {}_1q_0}{{}_2q(t) - {}_2q_0} = \frac{{}_1J_{max}}{{}_2J_{max}} = \lambda \qquad (3.26)
$$

which is the same as the Eq. C.2. Therefore, we can get the conclusion that the trajectory is phase-synchronized.

For the n-dimensional problem, it can be as well easily approved that the property holds.

Property 1 gives the solution to extend the algorithm to the case n-DOFs.

- Firstly, we compute the final time for each dimension. Considering the largest motion

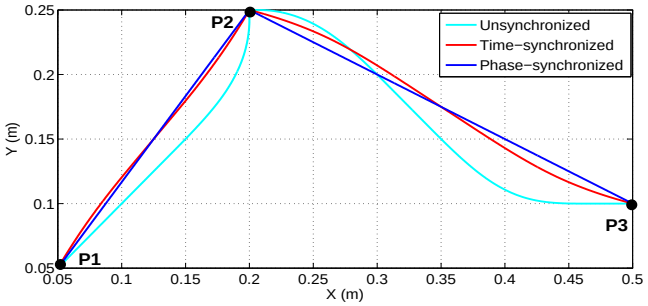


Figure 3.4: Way points motion: time synchronized, phase-synchronized and without synchronization

time T_{max}, we readjust the other dimension motions to this time.

$$T_{max} = maxf({}_jq_0, {}_jq_f, V_{max}, A_{max}, J_{max}) \quad j \in [1,n] \tag{3.27}$$

- Time adjusting is done by decreasing linearly $J_{max}, A_{max}, V_{max}$. According to Property 1, we should determine the index of joint which has the longest execution time. We use m to represent this joint. Thus, the kinematic constraints of other joints can be adjusted as follow:

$$\begin{aligned}
{}_jV_{max} &= \lambda_j V_{max} = \frac{{}_jq_0 - {}_jq_f}{{}_mq_0 - {}_mq_f} V_{max} \\
{}_jA_{max} &= \lambda_j A_{max} = \frac{{}_jq_0 - {}_jq_f}{{}_mq_0 - {}_mq_f} A_{max} \\
{}_jJ_{max} &= \lambda_j J_{max} = \frac{{}_jq_0 - {}_jq_f}{{}_mq_0 - {}_mq_f} J_{max}
\end{aligned}$$

For each segment of the trajectory, one of the velocity, acceleration, or jerk functions of the n initial joints is saturated, while the others are inside their validity domain. In other words, the motion consumes minimum time for one direction. At other directions, the motions are conditioned by the minimum one. Repeating this strategy for each straight segment, we build the time optimal trajectory $\mathscr{T}_{ptp}$ that stops at each waypoint.

3.3.1.2 Comparison of synchronization methods

In this paragraph, the performance of the time-synchronized algorithm is compared with the synchronization achieved by the proposed method. Figure 3.4 illustrates the path of a simple via-points motion in 2D space. P_1, P_2, P_3 are waypoints with zero velocities and accelerations. We generated the time-synchronized, phase-synchronized and un-synchronized

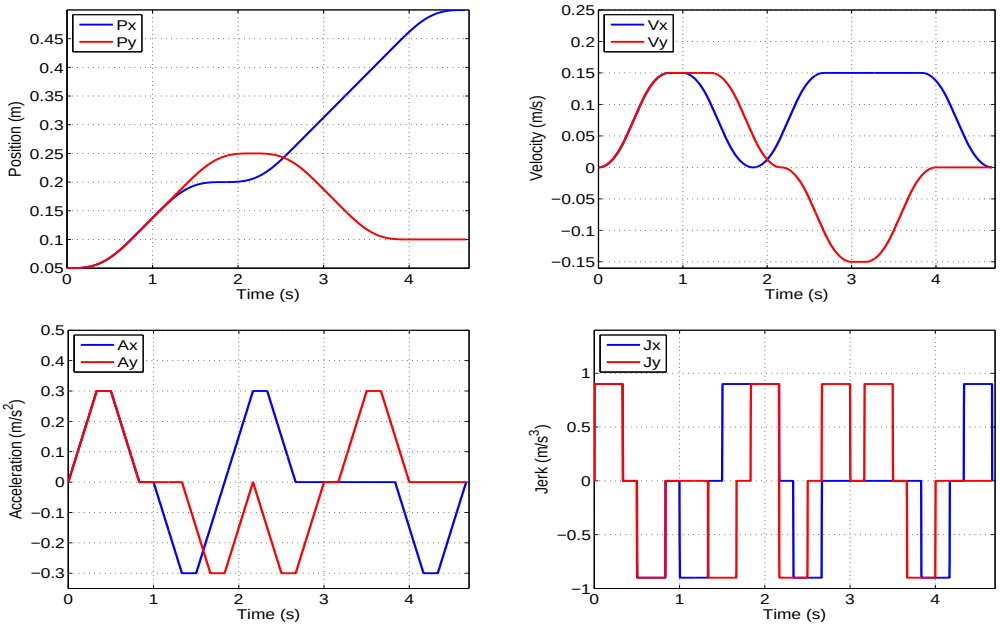


Figure 3.5: Position, velocity, acceleration and jerk profile of unsynchronized 2D via-points motion.

trajectories with the same kinematic constraints:

$$\begin{aligned} |J_{max}| \leq \quad & 0.9 m/s^3 \\ |A_{max}| \leq \quad & 0.3 m/s^2 \\ |V_{max}| \leq \quad & 0.9 m/s \end{aligned}$$

The difference among these three trajectories are clearly depicted from Figure 3.5 to Figure 3.7. Phase-synchronized trajectories are very important for many real-world applications. For instance, phase-synchronized trajectories make sense when we want to modulate time w.r.t. cost values, which means, all axis stay at the same phase and slow/accelerate at the same time. In our case the shape of the curve is not defined, so we have to synchronize the initial and final state of motion and to define an acceptable curve.

3.4 Smooth Trajectory Generation

The polygonal-line trajectory $\mathscr{T}_{ptp}$ obtained in the previous paragraph is feasible, but it is not satisfactory because the velocity varies greatly at each via-point to stop the motion. These stops can be avoided by allowing the trajectory to deviate slightly from the via-points by rounding the edges to smoothly travel near the point while changing the direction without stopping.

Without loss of generality we consider three adjacent points (P_{i-1}, P_i, P_{i+1}) and the

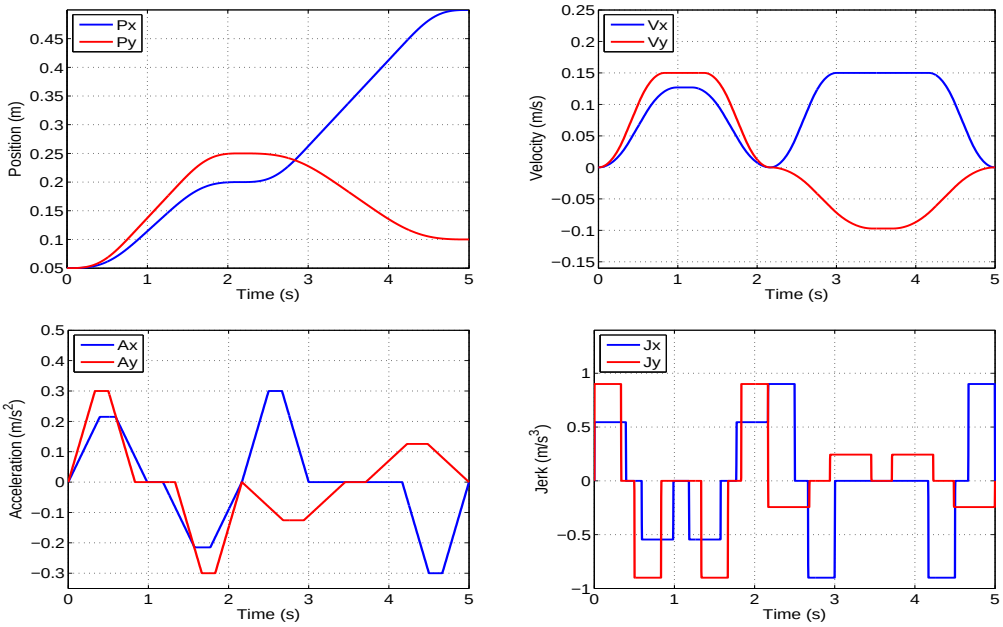


Figure 3.6: Position, velocity, acceleration and jerk profile of time-synchronized 2D via-points motion.

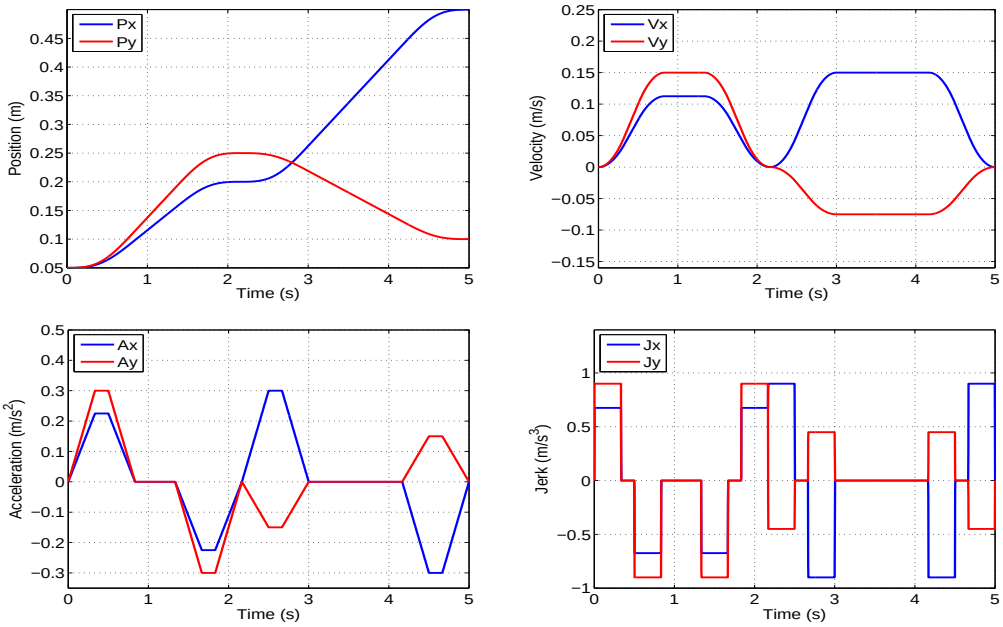


Figure 3.7: Position, velocity, acceleration and jerk profile of phase-synchronized 2D via-points motion.

smoothing at the intermediate via-point P_i. Firstly, the two straight-line trajectories $\mathscr{T}_{P_{i-1}P_i}$ and $\mathscr{T}_{P_iP_{i+1}}$ are computed respectively. Then we choose two points (P_{start}, P_{end}) based on the distance d_{start} and d_{end} to the point P_i on the trajectory. These two points define the

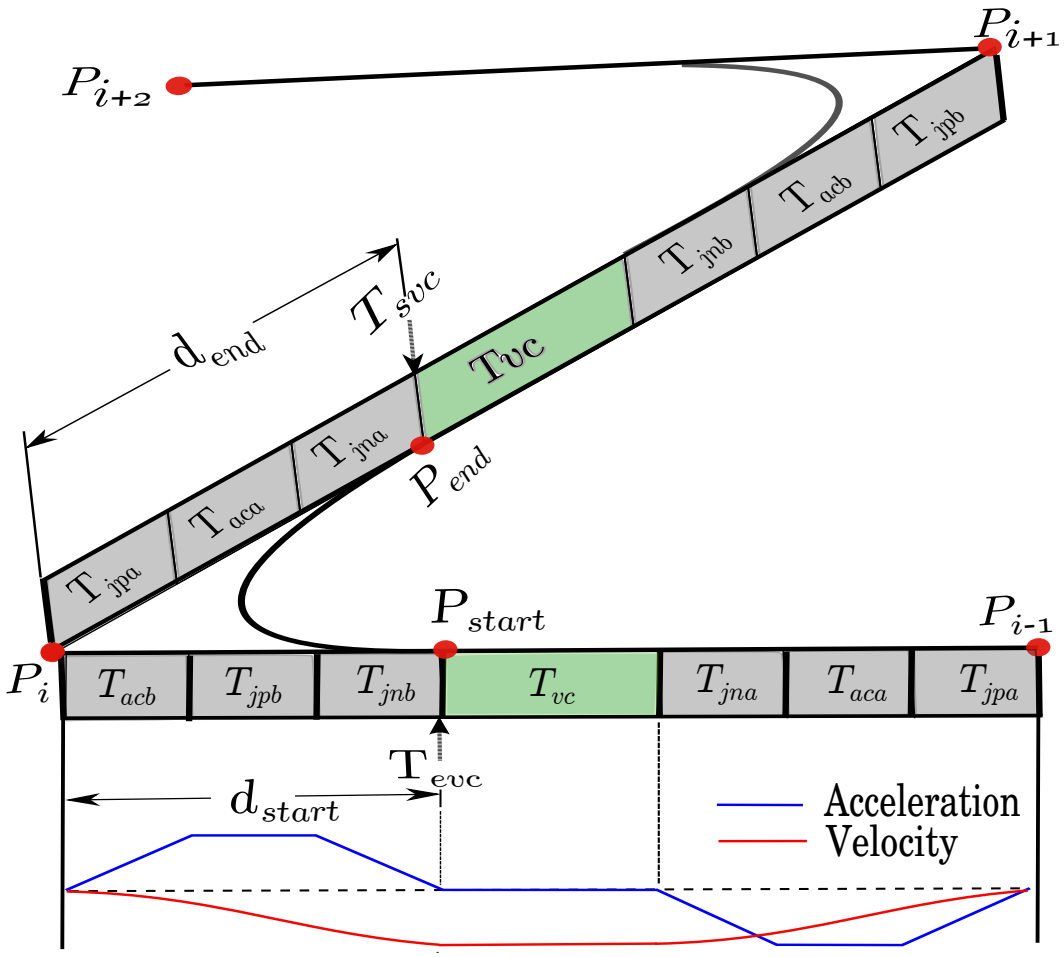


Figure 3.8: Influence of the start and end points for the smooth area

smoothed area, which is parameterised by the two distances. The possible choices of these two points are infinite, and the resulting trajectories can vary greatly due to the different choices, see Figure C.3.

But, since we wish an optimal time for the motion, a first simple idea is to keep the maximum velocity segment and smooth the area where the trajectory is not travelled at constant speed. Thence we choose the two time instants T_{evc}, which is the end of the velocity constant segment of $\mathscr{T}_{P_{i-1}P_i}$, and T_{svc}, which is the start of the velocity constant segment of $\mathscr{T}_{P_iP_{i+1}}$, as shown in Figure C.3.

In this case, the points P_{start} and P_{end} are associated with the two motion conditions: $M_{start} = (X_{evc},\ V_{evc},\ A_{evc})$ and $M_{end} = (X_{svc},\ V_{svc},\ A_{svc})$. As these points are part of the velocity constant segments, the accelerations A_{evc} and A_{svc} are zero. Then we compute the minimum time trajectory between P_{start} and P_{end} independently on each of the two axes using the 7 segments method, proposed by Broquère in [Broquere 08]. The optimal time is the longest one and the problem becomes to find a motion along the other $n-1$ axes that last the same time.

Since each joint variable is assumed to be independent, the minimum execution time T_{imp} between M_{start} and M_{end} is determined by the slowest single-joint trajectory. This thesis defines a function f to compute the optimal time for an axis using the 7 segments method. From the initial and final motion conditions (M_{start} and M_{end}) and kinematics bounds (J_{max}, A_{max} and V_{max}) for the axis, the function f computes the transit time as $f(M_{start}, M_{end}, J_{max}, A_{max}, V_{max})$. Thus,

$$T_{imp} = \max_{j \in [1,n]} (f({}_jM_{start}, {}_jM_{end}, {}_jJ_{max}, {}_jA_{max}, {}_jV_{max})) \tag{3.28}$$

T_{imp} defines the time necessary for the motion along the associated axis. For each of the other axes, the interpolation problem becomes to build a motion with predefined time between two motions conditions (${}_jM_{start}$ and ${}_jM_{end}$). A large variety of solutions exist and each one defines a different smoothing. We propose three methods by computing different parameters for the trajectory.

3.4.1 Three-Segment Interpolants

If we consider motion conditions M_{start} and M_{end} defined by a starting instant t_I and an ending instant t_F, the starting and ending situations to be connected are: (X_I, V_I, A_I) and (X_F, V_F, A_F). An interesting solution to connect this portion of trajectories is to define a sequence of three trajectory segments with constant jerk that bring the motion from the initial situation to the final one within time T_{imp}. We choose three segments because we need a small number of segments and there is not always a solution with one or two segments.

The system to be solved is then defined by 13 constraints: the initial and final situations (6 constraints), the continuity in position velocity and acceleration for the two switching situations and time. Each segment of a trajectory is defined by four parameters and time. If we fix the three durations $T_1 = T_2 = T_3 = \frac{T_{imp}}{3}$, we obtain a system with 13 parameters where only the three jerks are unknown[Broquère 10]. As the final control system is periodic with period T, the time $T_{imp}/3$ must be a multiple of the period T, and in this study, T_{imp} is chosen to be a multiple of $3T$.

3.4.2 Three-Segment Interpolants With Bounded Jerk

The three-segment interpolants solves the problem of trajectory generation with fixed duration for each segment. As the time T_{imp} is longer than the minimum time necessary for joining the two motion conditions, we can hope that a solution defined by 3 segments exists. However, it cannot be guaranteed that the computed jerks are always bounded. Here, we introduce a variant three-segment method with two defined jerk.

As for the three-segment method, the system is defined by 13 constraints. With this variant method, however, we fix the jerks on the first and third segments as $|J_1| = |J_3|$, which have the value bounded within the kinematic constraints. Then, the unknown parameters in

the system are J_2 and the three time durations. Thus we obtain a system of four equations with four parameters (J_2, T_1, T_2 and T_3):

$$A_F = J_3 T_3 + A_2 \tag{3.29}$$

$$V_F = J_3 \frac{T_3^2}{2} + A_2 T_3 + V_2 \tag{3.30}$$

$$X_F = J_3 \frac{T_3^3}{6} + A_2 \frac{T_3^2}{2} + V_2 T_3 + X_2 \tag{3.31}$$

$$T_{imp} = T_1 + T_2 + T_3 \tag{3.32}$$

where

$$\begin{aligned}
A_2 =& J_2 T_2 + J_1 T_1 + A_I \\
V_2 =& J_2 \frac{T_2^2}{2} + (J_1 T_1 + A_I) T_2 + J_1 \frac{T_1^2}{2} + A_I T_1 + V_I \\
X_2 =& J_2 \frac{T_2^3}{6} + (J_1 T_1 + A_I) \frac{T_2^2}{2} + (J_1 \frac{T_1^2}{2} + A_I T_1 + V_I) T_2 \\
& + J_1 \frac{T_1^3}{6} + A_I \frac{T_1^2}{2} + V_I T_1 + X_I
\end{aligned}$$

To choose the values of jerks on each dimension, we resort to the velocities V_I and V_F. The jerks are fixed by $J_1 = -J_3 = J_{max}$ when $V_I - V_F > 0$, and by $J_1 = -J_3 = -J_{max}$ when $V_I - V_F < 0$. If $V_I - V_F = 0$, we compare the values of A_I and A_F instead.

Algorithme 1 : All-Bounded Interpolants Generation

Require: Motion conditions: M_{start}, M_{end}; number of DOFs: n; Kinematic constraints;
Ensure: Jerk-Bounded, Acceleration-Bounded, Velocity-Bounded Interpolants

1: Compute T_{imp}: $T_{imp} = \max_{j\in[1,n]}(f({}_jM_{start}, {}_jM_{end}, {}_jJ_{max}, {}_jA_{max}, {}_jV_{max}))$
2: **for** $j = 1$ to n **do**
3: Compute the time-optimal interpolants between M_{start} and M_{end}, then get the execution time T_j
4: Get $\mathcal{N}_j$ and the execution time on each segment $T_j^{\mathcal{N}}$
5: **if** $\mathcal{N}_j = 0$ **then**
6: No motion on this joint, maintain the time-optimal interpolants
7: **else**
8: Enlarge $T_j^{\mathcal{N}}$ by $T_j^{\mathcal{N}} = T_j^{\mathcal{N}} + \frac{T_{imp} - T_j}{\mathcal{N}_j}$
9: Compute the new Jerk on each segment $J_j^{\mathcal{N}}$
10: **end if**
11: Generate the interpolants with $J_j^{\mathcal{N}}$ and $T_j^{\mathcal{N}}$
12: **end for**

3.4.3 Jerk, Acceleration, Velocity-Bounded Interpolants

Now we derive the all-bounded trajectory given a fixed duration T_{imp}. The method in section C.2.4.2 can directly bound the jerk, but have to readjust the jerk values by a predefined resolution to bound the velocity and acceleration. As we detect the longest execution time T_{imp} by computing the time-optimal trajectory on each joint, the jerk is saturated and the acceleration and velocity may be saturated, depending on different cases. Thus, we can extend the duration of all joints (except the one with the longest duration) to T_{imp} by unsaturated interpolants while maintaining the number of segments $\mathcal{N}_j$ on each joint. We name it a *Slowing Down Motion*. Algorithm 1 shows the generation of all-bounded interpolants.

However, an exception exists when the angle α (as shown in Figure 3.9) formed by the 3 points is quite small ($\alpha < \alpha_{lim}$, α_{lim} depends on the kinematic constraints), the previous solutions can not work any more because they will get either jerks much larger than the kinematic constraints or negative time. In this case, for $d \leq d_{start}$ the optimal time solution is to stop at the via-points. So we propose to increase the distance by $d'_{start} = d_{start} + d'$ to smooth the corner.

3.4.4 Managing the Error

From a user point of view, managing the position error at via-points can be more important than defining the distance d where smoothing begins. The trajectory error can be defined as the largest distance between the initial straight-line trajectory and the smoothed one. It represents how much the smooth trajectory deviates from the via-points. This approach is useful, for example, if the path planner provides some tube around the path where the motion is free of collision.

3.4.4.1 Error Definition

Considering the case of three adjacent points P_{i-1}, P_i and P_{i+1}, see Figure 3.9, $\overrightarrow{\mathbf{n}_i}$ being the normal unit vectors to the straight line $P_{i-1}P_i$, and $\mathscr{T}_{si}(t)$ the smoothed trajectory. Then the error $\mathscr{E}$ can be defined as:

$$\begin{aligned} \mathscr{E}(t) &= \min([\mathscr{T}_{si}(t) - P_i] \cdot \overrightarrow{\mathbf{n}_i}, [\mathscr{T}_{si}(t) - P_{i+1}] \cdot \overrightarrow{\mathbf{n}_{i+1}}) \\ \mathscr{E} &= \max_{t \in [t_I, t_F]} (\mathscr{E}(t)) \end{aligned} \tag{3.33}$$

To compute this error, we introduce another parameter $\mathscr{E}_v$, which is represented by the minimum distance between the vertex P_i and the trajectory $\mathscr{T}_{si}(t)$:

$$\mathscr{E}_v = \min_{t \in [t_I, t_F]} d(P_i, \mathscr{T}_{si}(t)) \tag{3.34}$$

where d is the Euclidean distance. As the start and end points of the blend we choose

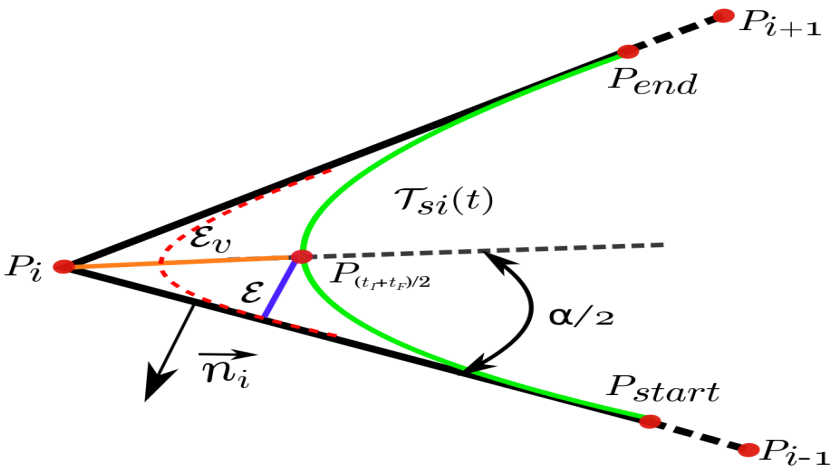


Figure 3.9: Error between the smoothed trajectory and pre-planed path

locate at the symmetric segments on each straight-line trajectory, the error $\mathscr{E}_v$ happens at the bisector of the angle α formed by the 3 adjacent points, Therefore,

$$\mathscr{E}_v = d(P_i, \mathscr{T}_{\frac{t_I+t_E}{2}}) \tag{3.35}$$

$$\mathscr{E} = \mathscr{E}_v * \sin\frac{\alpha}{2} \tag{3.36}$$

The advantage of this expression is that the error is easily computed. In the real time applications, error computation is usually time consuming while this method helps to avoid such problems.

3.4.4.2 Comply with Maximum Error

We suppose that the task planner provides a maximum tolerable error D. A simple possibility is to change the distance d to make the error $\mathscr{E} \leq D$, see the red dashed trajectory shown in Figure 3.9. We introduce a parameter δ, for a general case, it is defined as:

$$\delta = \frac{d(P_{start}, P_i)}{d(P_{Tevc}, P_i)} \tag{3.37}$$

Thus $\delta \in [0,1]$. But when $\alpha < \alpha_{lim}$, P_{start} locates at the segment before P_{Tevc}, so in this case:

$$\delta = \frac{d(P_{start}, P_i)}{d(P_{Tevc}, P_i)} - 1 \tag{3.38}$$

Suppose P^i_{end} is the end point of the previous blend and P^{i+1}_{start} is the start point of the next one. To avoid the overlapping, the distance d of new points must satisfy:

$$d^i_{end} + d^{i+1}_{start} \leq d(P_i, P_{i+1}) \tag{3.39}$$

Then we can compute in a loop and get the δ_{max} that makes $\mathscr{E} \leq D$ and also satisfies 3.39.

3.5 Comparison With B-Spline Trajectory Smoothing

Many techniques have been proposed in the literature to generate smooth trajectories using the B-Splines. [Cao 97] addressed constrained time-efficient and smooth cubic spline trajectory generation for industrial robots.[Pan 12] presented a trajectory computation algorithm to smooth piecewise linear collision-free trajectories computed by sample-based motion planners. This approach uses cubic B-splines to generate trajectories which are $\mathscr{C}^2$ almost everywhere, except on a few isolated points. [xiu Kong 13] employed the cubic B-spline to construct the curve of the square of pseudo-velocity as smooth constraint of the transformed convex optimization model, thus to plan smooth and near time-optimal trajectory for robot manipulators. [Tanaka 12] proposed a smooth trajectory generation method by minimizing the jerk by using quintic splines. While seven-degree B-splines are exploited to generate smooth joint trajectories with continuous velocity, acceleration and jerk in [Wu 09].

Compared with the B-Spline methods, the main advantage of our smoothing algorithms are:

1. We use a concept of local trajectory planning. Only the portion of trajectory that is near the waypoints are considered to smooth. Therefore, it is adequate to perform the collision check on the local smoothed trajectories. While the Spline approach is a global idea of trajectory smoothing. The occurrence of collision must be done along the whole path.

2. It is difficult to check the maximum kinematic variables (such as $J_{max}, A_{max}, V_{max}$) for the B-Spline trajectories. In contrast, the jerk, acceleration and velocity are computed directly in the construction of the trajectory with our method, so the maximum values are quite easy to obtain. Moreover, it is also easy to satisfy kinematic motion bounds.

3.6 Shortcutting Smoothing

3.6.1 Shortcutting Algorithms

The smooth trajectory generated in the previous section can avoid the halts at each waypoint, thus shortening the execution time. This method is useful when the smoothed trajectory must remain close to the initial path defined the via-points. However, to obtain more natural looking trajectories, other solutions must be developed.

To produce more human-like robot motions, we introduce a variant of the shortcutting method commonly used in robotics and graphic animation. First of all, the method uses some heuristic to choose two points along the initial trajectory. Here, we simply apply a

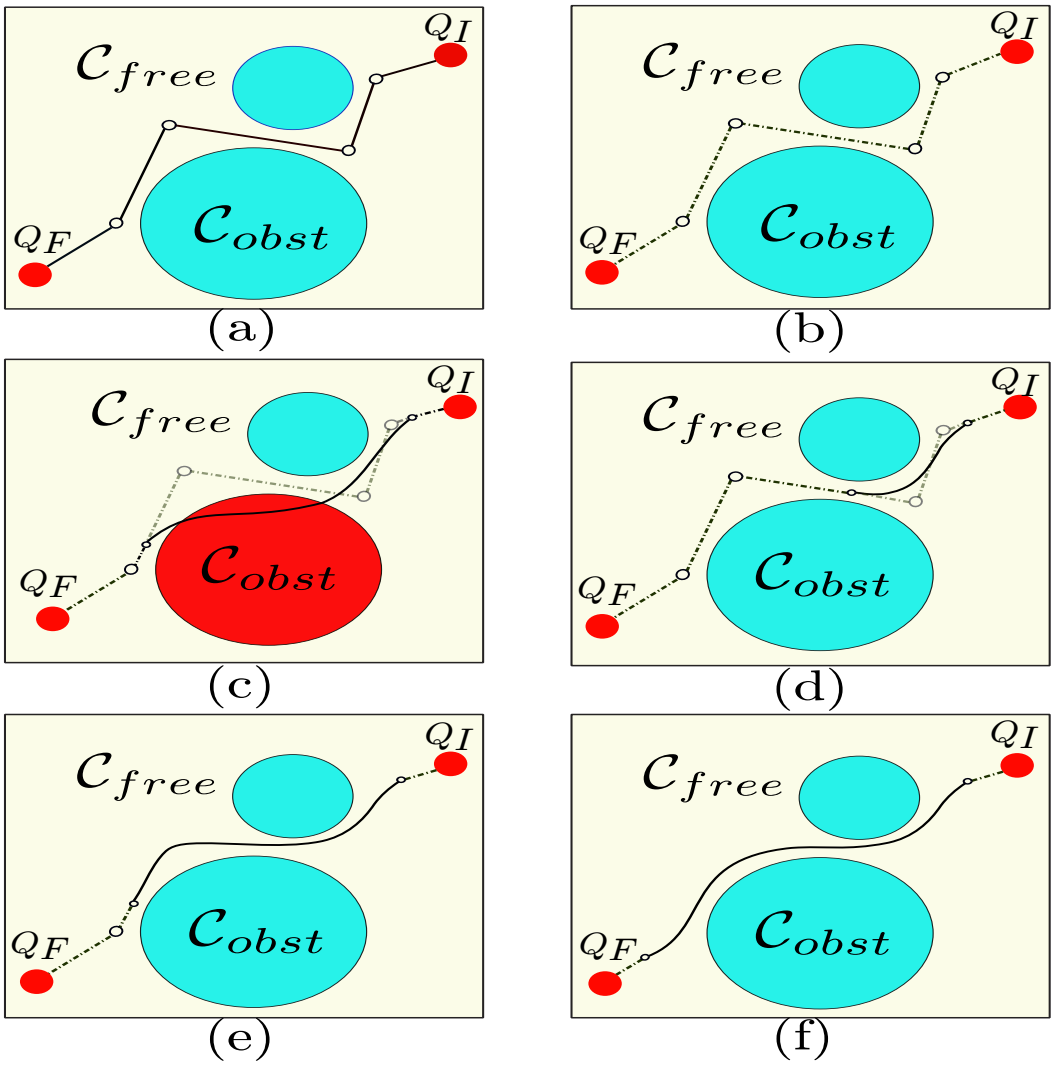


Figure 3.10: Smooth algorithm. (a) A jerky path as a list of waypoints. (b) Converting into a trajectory that halts at each waypoints. (c) Performing a shortcut that fails in collision check. (d),(e) Two more successful shortcuts (f) The final trajectory.

uniform random function to select the two points. In a second stage we generate a trajectory segment between the two motions conditions associated to the points. Then, if the generated segment passes the collision-checking test, the new segment replaces the portion of the initial trajectory. Figure C.4 illustrates the smoothing algorithm that performs four iterations of shortcutting on a polygonal path. The collision check fails during the first shortcut, and after two more attempts, a feasible trajectory is generated.

The algorithm 2 describes the shortcutting method. In a first stage, we suppose that a motion planner produces a polygonal collision-free path. From this polygonal path, we build a feasible trajectory composed of piecewise straight-lines that are bounded in jerk, acceleration and velocity. Then the trajectory is smoothed iteratively. The construction of the shortcutting trajectory uses the smooth trajectory generation approaches presented in

Algorithme 2 : Shortcutting algorithm

Require: a path as a list of waypoints, iteration count N
Ensure: a smoothed collision-free trajectory
1: Plan a time-optimal trajectory $\mathscr{T}_{ptp}$ that stops at each waypoints;
2: Initialize the smooth trajectory $\mathscr{T}_{smooth}$ →clear(), $\mathscr{T}_{smooth} = \mathscr{T}_{ptp}$
3: **for** $iteration = 0$ to N **do**
4: Pick t_I and t_F randomly from $[0, t_F]$
5: $M_{start} = getMotionState(\mathscr{T}_{smooth}, t_I)$=$(Q_I, V_I, A_I)$
$M_{end} = getMotionState(\mathscr{T}_{smooth}, t_F)$=$(Q_F, V_F, A_F)$
6: $\mathscr{T}_{sc} = ComputeShortcutTraj(M_{start}, M_{end})$
7: **if** $\mathscr{T}_{sc} \rightarrow$ CollisionFree() **then**
8: Replace $\mathscr{T}_{M_{start}M_{end}}$ by $\mathscr{T}_{sc}$ in $\mathscr{T}_{smooth}$
9: **end if**
10: **end for**
11: **return** $\mathscr{T}_{smooth}$

Section C.2.4.

The output of the algorithm is a smooth trajectory that respects the collision and kinematic motion bounds (velocity, acceleration and jerk).

3.6.2 Trajectory Collision Checking

Collision checking is a basic operation in any robot motion planning and smoothing algorithm. This operation is commonly realized by discretizing the curve at a predefined constant resolution ε and statically testing each sampled configuration. However, this approach is inexact and cannot detect any collision that occurs. If ε is too small, the collision checker will be unnecessarily slow. On the other hand, choosing ε too large might result in missing some obstacles.

In order to overcome this problem, we use an alternative local trajectory-checking algorithm based on the divide and conquer algorithm [Schwarzer 04] and the concept of bubbles of free configuration space, introduced in [Quinlan 95]. It is an exact collision checking by attempting to cover the path with collision-free neighborhoods. This algorithm recursively splits the path in two sub-paths, and then calculates bubbles of free space around a configuration and therefore can guarantee the collision-free status of a trajectory segment by overlapping these bubbles along each segment (Figure 3.11(a)). The bubble $\mathscr{B}(Q)$ at the current configuration Q is an upper bound computed using a distance d_{obst}, where d_{obst} defines the minimum distance between the robot in configuration Q and the obstacles. For the robots with n revolute joints, the bubble will be diamond shaped [Quinlan 95]:

$$\mathscr{B}(Q) = \left\{ X \in \mathscr{C} : \sum_{i=1}^{n} r_i |X_i - Q_i| \leq d_{obst} \right\} \tag{3.40}$$

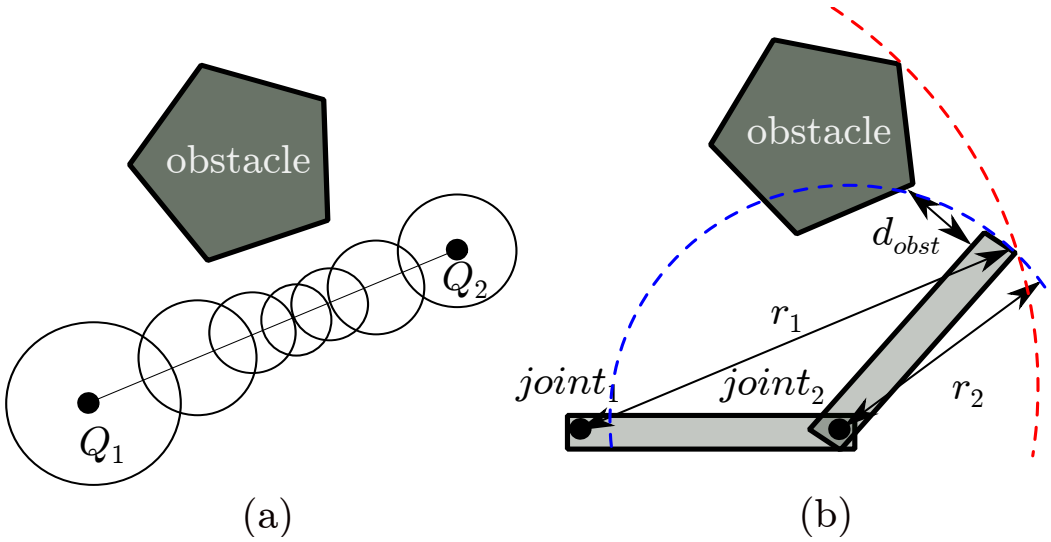


Figure 3.11: (a) A collision-free $\mathscr{C}$-space path covered by free bubbles. (b) 2D robot manipulator showing the maximum distance r_1, r_2 and the minimum obstacle distance d_{obst}. The circle at the axis of joint 1 of radius r_1 (the red dashed line) contains the entire manipulator. The circle at joint 2 of radius r_2 (the blue dashed line) contains link 2.

where r_i is the radius of the cylinder that is centered along the axis of the i-th joint and contains all the subsequent links of the manipulator. Figure 3.11(b) gives an example of a two-degree-of-freedom planar robot showing the maximum distance of each joint and the minimum obstacle distance d_{obst}. Algorithm 3 shows the overall pseudo-code of exact collision check.

Algorithme 3 : Exact collision check

Require: A trajectory $\mathscr{T}$ in time interval $[t_I, t_F]$
Compute the free bubbles $\mathscr{B}(Q_I)$ and $\mathscr{B}(Q_F)$
if OVERLAP($B(Q_I)$, $B(Q_F)$) **then**
  return collision-free
else
  Bisect the trajectory at $\frac{t_I+t_F}{2}$
  repeat
    Recurse on the two halves
  until $collisionDetected$ or $B(Q) \leq threshold$
  return collision or collision-free
end if

Given a trajectory segment $\{\mathscr{T}(t)|t_I \leq t \leq t_F\}$, the algorithm computes the free bubbles $\mathscr{B}(Q_I)$ and $\mathscr{B}(Q_F)$ for initial and final configurations Q_I and Q_F, respectively. If the bubbles overlap, the algorithm terminates and reports a collision-free trajectory. Otherwise, the segment is bisected at $\frac{t_I+t_F}{2}$ and the algorithm recurses on the two halves. We break the recursion when a collision is detected, or the largest segment, uncovered by bubbles,

becomes smaller than the predefined threshold.

As this method calculates a lower bound for the free bubble radius based on the minimum obstacle distance, the radius tends to get very small at a configuration with a low obstacle distance. Numerous distance and collision calculations are required in this situation, which slows down the collision check procedure. The trajectories that pass away from the obstacles are preferable, allowing to choose the minimum radius reasonably large.

3.6.3 Online Shortcutting

To avoid waiting for smoothing it is possible to interleave trajectory smoothing with execution. Shortcutting algorithms can be conveniently implemented in an on-line technique that executes and smooth the trajectory in parallel, which makes choosing a termination criterion unnecessary. In this technique, shortcuts are generated at random only from the trajectory after the current time, plus some small padding. The pseudo-code is as follow:

Algorithme 4 : Online Shortcutting algorithm

Require: A path as a list of waypoints to be executed and smoothed,
$padTime$, a constant $\approx$ a bound on the time it takes to perform one shortcut,
a timer function $Time()$
Ensure: A smoothed collision-free trajectory
1: Plan a time-optimal trajectory $\mathscr{T}_{ptp}$ that stops at each waypoints
2: Initialize the smooth trajectory $\mathscr{T}_{smooth}$ →clear(), $\mathscr{T}_{smooth} = \mathscr{T}_{ptp}$
3: Execute the trajectory $\mathscr{T}_{smooth}$
M = getMotionCond($Time()$, $\mathscr{T}_{smooth}$), move towards to state M
4: **while** $Time() < \mathscr{T}_{smooth}$.GetDuration() **do**
5: t_1 = Rand($Time() + padTime$, $\mathscr{T}_{smooth}$.GetDuration())
6: t_2 = Rand($Time() + padTime$, $\mathscr{T}_{smooth}$.GetDuration())
7: M_1 = getMotionCond(t_1, $\mathscr{T}_{smooth}$)
8: M_2 = getMotionCond(t_2, $\mathscr{T}_{smooth}$)
9: $\mathscr{T}_{sc}$ = ComputeShortCut(M_1, M_2)
10: **if** ($Time() > t_1$) **then**
11: The shortcut may have taken too long, if so, throw it out. Continue
12: **else**
13: Replace $\mathscr{T}_{M_1M_2}$ by $\mathscr{T}_{sc}$ in $\mathscr{T}_{smooth}$
14: **end if**
15: **end while**
16: **return** $\mathscr{T}_{smooth}$

For slightly better performance, the sampling strategy for picking points along the trajectory can be tuned. For example, increasing the probability to choose the initial time t_I close to the current execution time seems to be effective.

3.7 Simulation and Experimental Results

3.7.1 Smoothing Trajectory From a Given Path

3.7.1.1 Simulation

We set up a simulation environment with a collision checker[1], a module that reads the description of a robot (kinematic tree and list of bodies) and its environment and can check collision between any two bodies. The same module is used in real-time during the robot motion control for collision detection. In each time cycle, the collision checker checks the collision and each time a collision is detected, the system will replan the trajectory. The algorithm is shown in algorithm 5. In this simulation, we set $\delta = 1$ to manage the error between the via-points and the smoothed trajectory because we aim at a time optimal motion.

Algorithme 5 : Reactive motion to environment changes

1: Plan a point to point trajectory $\mathscr{T}_{P_iP_f}$
2: **if** motion is not completed **then**
3: Execute the trajectory and check collision risk
4: **while** a collision is detected in 1 second at time t **do**
5: Get the motion condition $M(\mathscr{T}_{P_iP_f}, t+0.5)$, mark this point as P_c that will encounter a collision in 0.5 second
6: Choose a via-point P_m
7: Compute a via-points trajectory with P_c, P_m and P_f
8: Update the trajectory
9: **end while**
10: Execute the new trajectory
11: **end if**

Fig.3.12 shows the movement of the robot arm from an initial position P_i to a final position P_f with an obstacle. The robot successfully reaches the target position after two local trajectory deformations. It shall be noted that the robot is reactive, and the same trajectory can be used for a moving obstacle.

Figure 3.13 presents the travelled trajectory along the Z axis. The position curve shows how the trajectory deviates from the pre-planed one (straight line from the initial position to the final position) due to the predicted collision risk. In the first four segments, the executed trajectory coincides with the pre-planed one. At the collision prediction point 1, the system switches to a new trajectory. The smooth velocity profile shows that the trajectory transition is instantaneous and continuous.

The computation for this Cartesian space trajectory with 6 via-points requires an average execution time of 1.2 ms (time only spent in the trajectory computation, without collision detection) on a Intel Core$^{(TM)}$2 Quad CPU 2.66GHz machine. It is fast enough for real time

[1] *http://robotpkg.openrobots.org/robotpkg/graphics/coldman-genom*

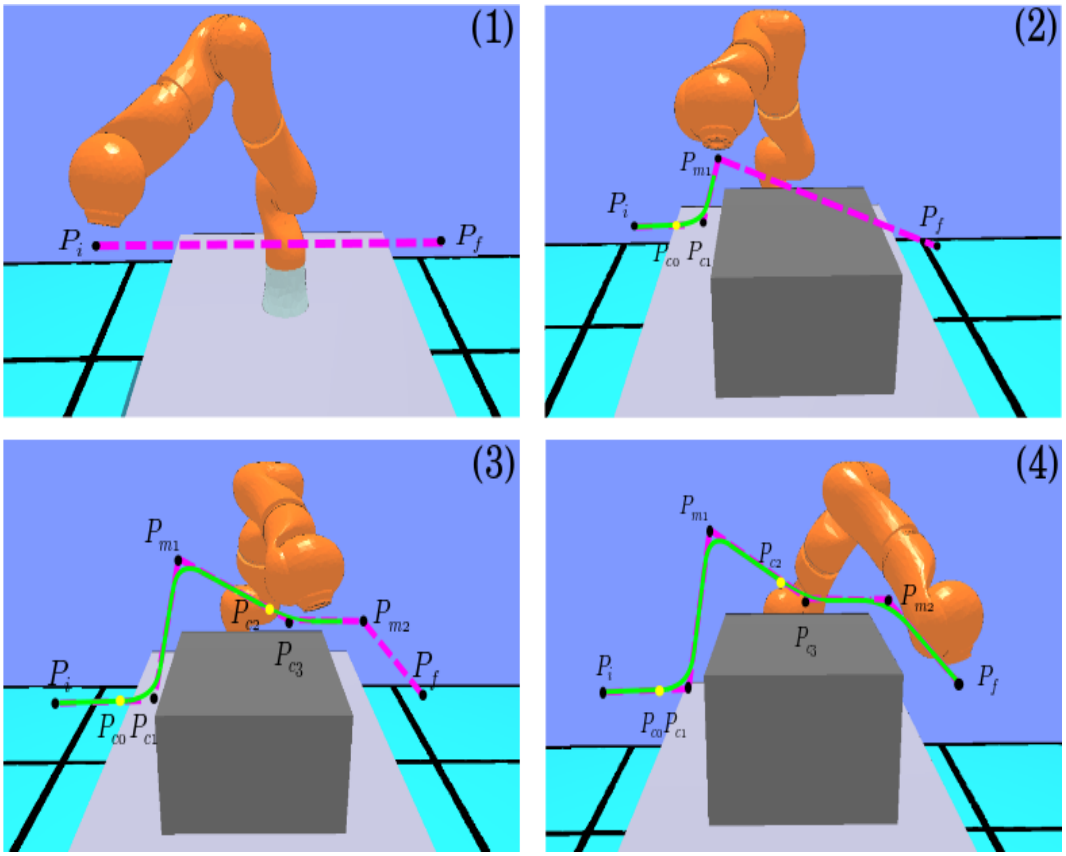


Figure 3.12: A simulated move from an initial position to a final position with a static obstacle. (1) The purple dashed line is the first trajectory computed. (2) The robot detects an obstacle and plan a new trajectory. Note that the purple trajectory is in collision. (3) A new trajectory that avoid collision. (4) The complete trajectory realized in green. The green solid line is the real path, which the robot follows. By adding two waypoints, the robot reaches the target position without collision and path replanning at the high level.

applications, such as visual servoing or reactions to other sensor events.

3.7.1.2 Robot Experiments

We also implemented the via-points trajectory generator on a KUKA light-weight robot IV [GmbH 08], which was controlled through the Fast Research Interface[Schreiber 10]. The software control is developed using Open Robots tools: GenoM3[Mallet 10a]. The sampling time is fixed to 10 ms. The *Pose* of the manipulator's end effector is defined by seven independent coordinates named Operational Coordinates. It is composed of a position vector $\mathbf{P} = [x, y, z]^T$ and by a quaternion $\mathbf{Q} = [n, \mathbf{q}]^T$, where $\mathbf{q} = [i, j, k]^T$. They give the position and the orientation of the final body in the reference frame. The linear and angular end-effector motion bounds are given in table 3.1.

Figure 3.14 illustrates a motion of the robot end effector. $\delta = 1.0$ means the smoothed trajectory starts at the end of maximum velocity segment. While $\delta = 0.8$ means the start

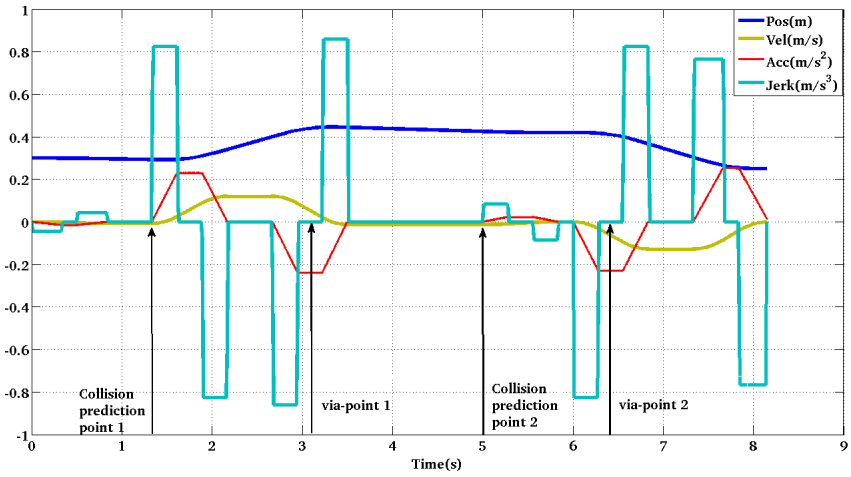


Figure 3.13: The position, velocity, acceleration and jerk of online generated via-points trajectory on Z axis

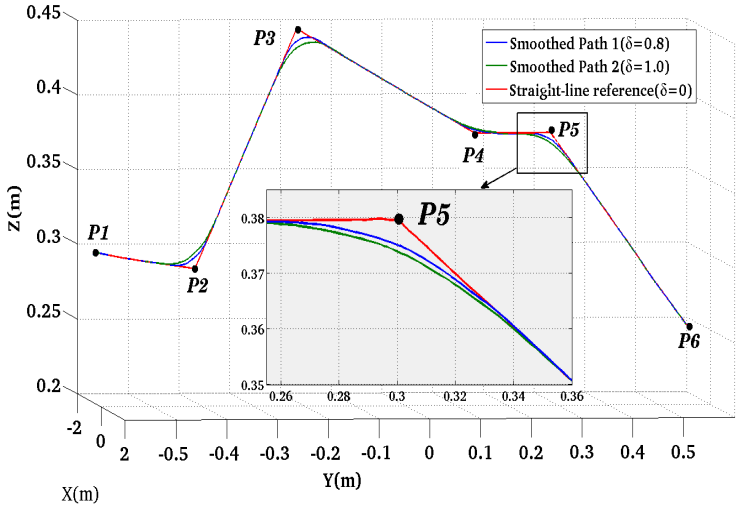


Figure 3.14: Paths of the robot end effector with different errors

points are closer to the intermediate points, which results a smaller error. The straight-line path is a path with zero error, which can be represented by defining $\delta = 0$.

3.7.2 Shortcut Smoothing Method

3.7.2.1 Simulation

We performed the simulation of a reaching task for a KUKA LWR IV robot arm (Figure 3.15). A total of 10 initial paths were generated with the same start and end configurations using a sample-based planner. Then these paths were converted into trajectories using both generation of phase-synchronized trajectory (presented in this chapter) and our smooth algorithm. Figure 3.15 shows that the smoothed path of the end-effector during the reaching

Table 3.1: Robot motion is limited in jerk, acceleration, and velocity

	Jmax	Amax	Vmax
Linear limits	0.900 m/s^3	0.300 m/s^2	0.150 m/s
Angular limits	0.600 rad/s^3	0.200 rad/s^2	0.100 rad/s

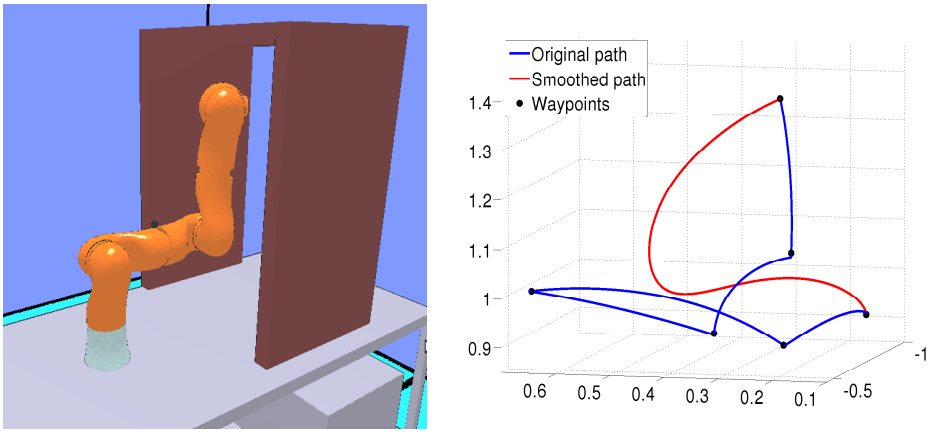


Figure 3.15: Left: A manipulator reaches under a shelf on a table from the zero position. Right: The blue curve depicts the original end effector path. The red curve depicts the smoothed path after 200 random shortcuts.

task is much shorter and more natural looking compared to the unprocessed path directly given by the motion planner. Figure 3.16 illustrates that the execution time is largely reduced by 36.77% on average after 200 shortcutting iterations.

The computation for these smooth trajectories consumes an average time of 5.2 s with 200 iterations on an Intel Core$^{(TM)}$2 Quad CPU 2.66GHz machine. Because of our analytical construction, the necessary time to build the shortcuts is negligible. The smoothing time is overwhelmingly dominated by collision checking time(4.6 s on average). As expected, the collision checker runs the most slowly when the robot passes under the shelf in this experiment.

3.7.2.2 Robot Experiments

The smoothing algorithm was also applied to a real KUKA light-weight robot IV. We used a real-world industrial scenario for evaluation. Figure 3.18 shows the planning setup. This setup will be detailed in Chapter V. The control software was also developed with GenoM3[Mallet 10b] with sampling time fixed to 10 ms.

The maximum joint velocity is decided by the physical properties of the motor. The numerical values of the velocity constraint for each joint were cited from [GmbH 08]. Thus,

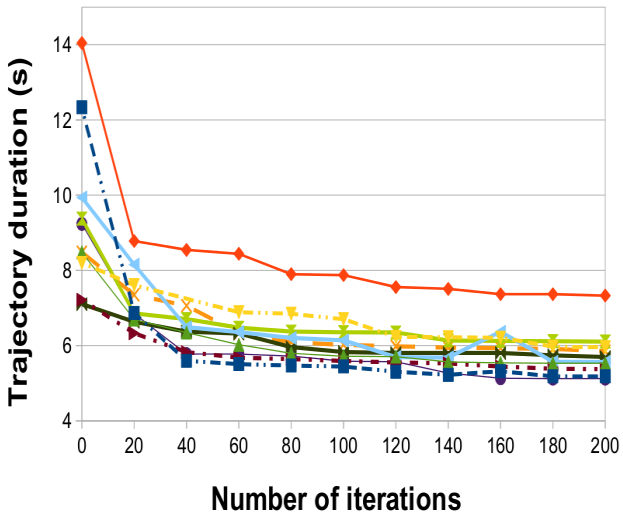


Figure 3.16: Progression of the smoothing illustrated by the duration of the trajectory travelling for 10 initial paths for the same reaching task. The trajectories are relative to the task presented in Figure 3.15

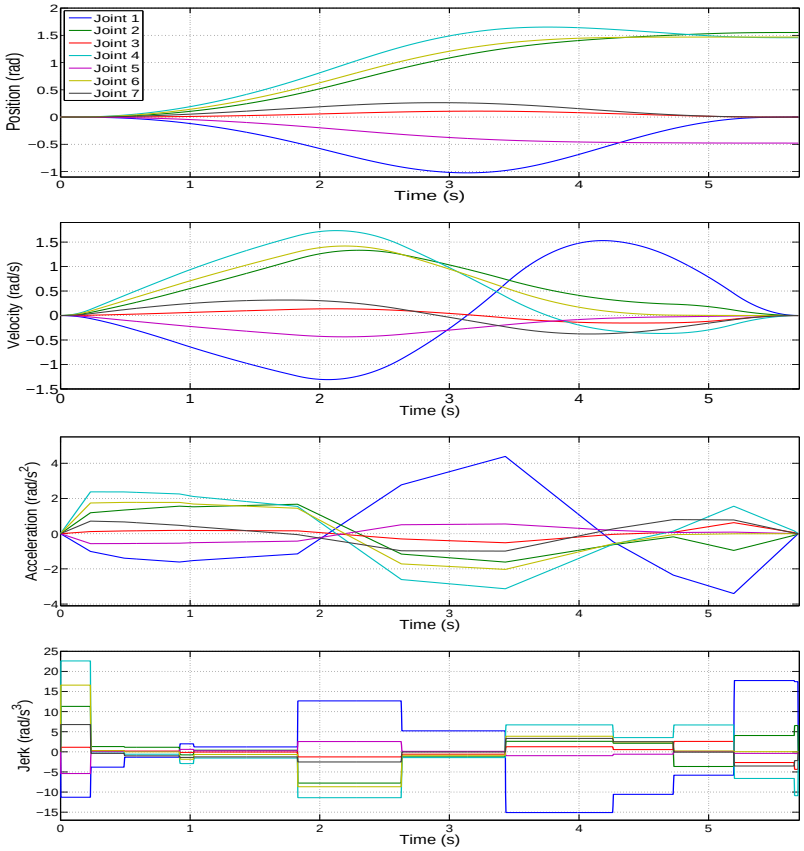


Figure 3.17: The position, velocity, acceleration and jerk profile of the calculated trajectory in the robot reaching task

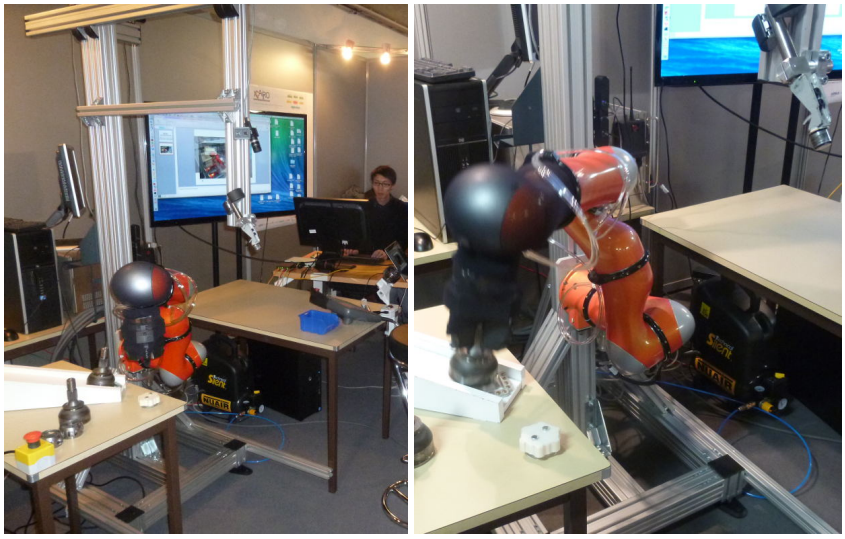


Figure 3.18: The planning setup of the ICARO industrial scenario. Left: a global view of the setup. Right: the start configuration.

V_{max} can be defined by a vector :

$$\begin{aligned} V_{max} &= [v_1 \quad v_2 \quad v_3 \quad v_4 \quad v_5 \quad v_6 \quad v_7]^T \\ &= [1.75 \quad 1.92 \quad 1.75 \quad 2.26 \quad 2.26 \quad 3.14 \quad 3.14]^T \quad rad/s \end{aligned} \tag{3.41}$$

Then, the kinematic motion bounds are defined in Table 3.2. Therefore, the numeric values

Table 3.2: Robot motion is limited in jerk, acceleration, and velocity

Jerk	Acceleration	Velocity
5*A_{max} rad/s^3	2.5*V_{max} rad/s^2	V_{max} rad/s

of the kinematic constraints are:

$$A_{max} = [4.38 \quad 4.80 \quad 4.38 \quad 5.65 \quad 5.65 \quad 7.85 \quad 7.85]^T \quad rad/s^2 \tag{3.42}$$

$$J_{max} = [21.9 \quad 24.0 \quad 21.9 \quad 28.3 \quad 38.3 \quad 39.3 \quad 39.3]^T \quad rad/s^3 \tag{3.43}$$

In the timing experiments, 50 iterations of shortcutting were finished with a computation time of 1.5 s and a reduced execution time of 2.8 s on average. Figure 3.17 illustrates the position, velocity, acceleration and jerk profile on each joint. Compared with the numeric bounds, results show that all kinematic variables were well bounded during the construction of the smooth trajectory.

3.8 Conclusions

In this chapter, we presented a solution to generate synchronized trajectories from a given path. We presented also two fast trajectory smoothing algorithms based on third-degree polynomial functions for high-DOF robot manipulators. The main contributions are:

1. The proposition of a simple and fast algorithm that operate in the configuration/velocity/acceleration state space.

2. We make an analytical derivation of time-optimal, velocity-bounded, acceleration-bounded and jerk-bounded trajectories that interpolate between endpoints with specified velocity and acceleration. For a single joint, the time-optimal interpolant can be derived in the closed form. We interpolate multiple joints by detecting the joint with the longest execution time, and then interpolate the remaining joints by finding the jerk-bounded, acceleration-bounded and velocity-bounded interpolants with fixed duration.

We developed also an algorithm to generate locally smooth trajectories from path defined by via-points while limiting the distance to the initial path. The main benefit is the possibility to guarantee that the trajectory stay inside a tube. This is particularly interesting when the path planner can define a tube free of collision along the path.

The last proposition is a global trajectory smoothing based on a well-known shortcutting heuristic for path. Smoothing directly the trajectory appears more efficient than smoothing the initial path. In addition to the computation time reduction, the trajectory can be smoothed in parallel with its execution.

Simulation and experimental results show that the use of third-degree polynomial functions to describe the trajectories provides efficient tools to obtain smooth robot moves.

4

Polynomial Trajectory Approximation

4.1 Introduction

Our objective is to build robot controllers simple and capable to realize and control a large variety of task and, in particular, human/robot interactive tasks. As we have seen, most of the tasks can be described by a trajectory and a control primitive. As the mathematical possibilities to imagine functions to depict trajectories is without any limit and the obtained model not necessary compatible with real time computation, approximation tools are necessary to transform these trajectories in a type that the controller can accept as input.

The objective of this chapter is to choose one or a small set of trajectory models and build tools to approximate any type of trajectories using these models.

As the tasks we wish to achieve are complex, the controller must provide necessary properties like allowing switching between input trajectory, control laws and sensors data. Trajectories of class C^2 are necessary to ensure human safety and comfort. As the system must be reactive, we need a fast and simple model. The trajectories are also used to exchange motion model between the different layers of the robot controller. Given this context, building an efficient trajectory controller is important likewise the choice of a good model for the trajectories. Generally polynomial functions are considered as the simplest one, so in this chapter we compare 3^{rd}, 4^{th} and 5^{th} degrees polynomial functions to approximate any trajectories.

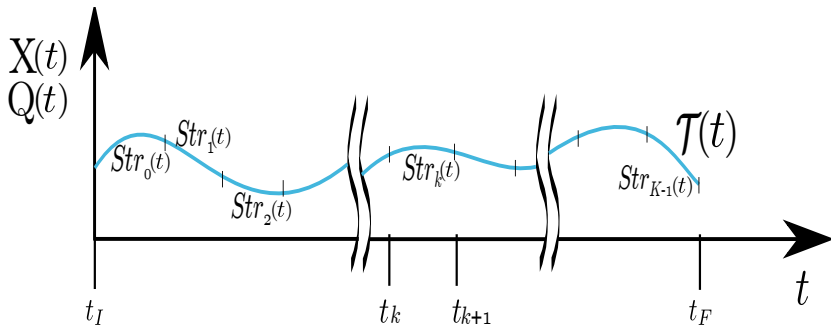


Figure 4.1: A trajectory is composed of K polynomial segments, t_I and t_F are the initial time and final time of the trajectory.

4.2 Polynomial trajectory approximations

4.2.1 Problem Formulation

We firstly give here a general definition of the problem. A trajectory $\mathscr{T}(t)$ to be approximated can be defined by a large set of mathematical functions. In order to simplify the controller we need to define a small set of functions. Considering that a trajectory $\mathscr{T}(t)$ is to be approximated over the period $t \in [T_I, T_F]$, where T_I is the initial time and T_F is the final time. For the sake of simplicity, the initial and final states of $\mathscr{T}(t)$ are defined as $M(T_I) = (X_0, A_0, V_0)$ and $M(T_F) = (X_F, A_F, V_F)$, respectively. We suppose that the initial and approximated trajectory is at least of class C^2. It is because higher-order initial trajectories can avoid large errors during the approximation, while high-order approximated trajectories have good properties for the robot controller. To build a simpler robot, our objective is to manipulate all type of trajectories. Thus the aim of approximation is to find out a series of segments of k-degree polynomial function between $M(T_I)$ and $M(T_F)$ within the period $T_{imp} = T_F - T_I$.

A trajectory $\mathscr{T}(t)$ can be represented by a series of trajectories defined between intermediate points. As shown in Figure 4.1, a trajectory $\mathscr{T}(t)$ is composed of K segments of trajectories. Thus, an equivalent representation of $\mathscr{T}(t)$ is defined as:

$$\mathscr{T}(t) = \sum_{k=0}^{K-1} Str_k(t) \quad t \in [t_I, t_F] \tag{4.1}$$

Equation 4.1 is the prerequisite to use our approximation approaches. The internal representation of the interpolated sub-trajectories requires time-continuous polynomials. One important property is the continuity class of the trajectory $\mathscr{C}^m$ ($m \geq 2$ in this paper).

4.2.2 Approximation Possibilities

Regarding the target of C^2 functions, we need polynomial function with degree higher than 2. An imposed time motion between two points involves seven constraints: three initial

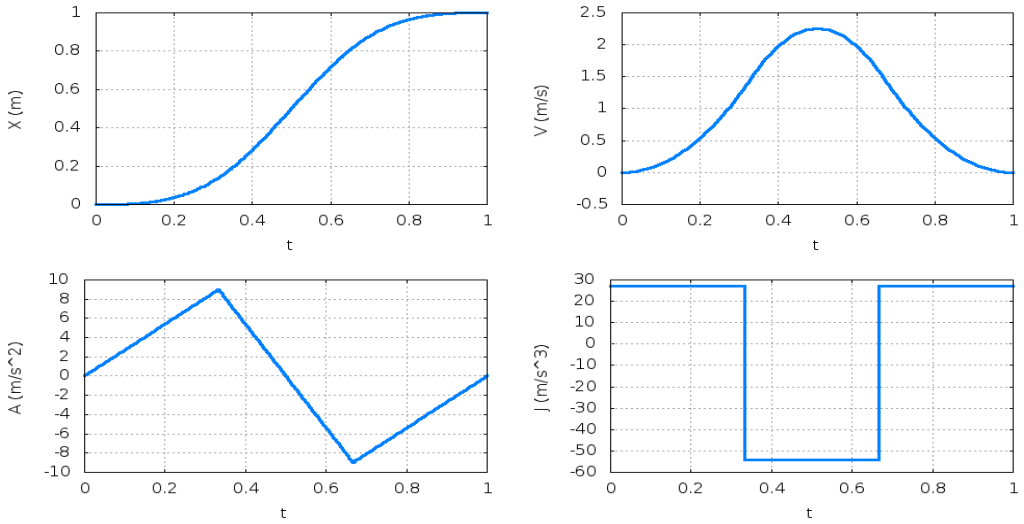


Figure 4.2: 3^{rd} degree polynomial interpolants with fixed time: the position, velocity, acceleration and jerk profiles.

conditions (X_0, A_0, V_0), three final conditions (X_F, A_F, V_F) and the imposed time T_{imp}. Thus we propose to compare 3^{rd}, 4^{th} and 5^{th} degree functions to approximate a given trajectory.

Moreover, each C^2 continuity between two segments introduces 3 constraints, so the approximation needs at least 3 trajectory segments of cubic polynomials[Broquère 10]. Each cubic segment is defined by 5 parameters: J, A, V, X_0 and T_I.

Cubic functions:

$$X(t) = \frac{J}{6}(t-T_I)^3 + \frac{A}{2}(t-T_I)^2 + V(t-T_I) + X_0 \tag{4.2}$$

A quartic polynomial segment is defined by 6 parameters, so 2 quartic segments are at least required to represent the motion. S is the snap.

Quartic functions:

$$X(t) = \frac{S}{24}(t-T_I)^4 + \frac{J}{6}(t-T_I)^3 + \frac{A}{2}(t-T_I)^2 + V(t-T_I) + X_0 \tag{4.3}$$

A single 5^{th} degree polynomial function is already characterized by 7 parameters, so we can just use one segment to define the motion. C is the first derivative of the snap. *Quintic functions*:

$$X(t) = \frac{C}{120}(t-T_I)^5 + \frac{S}{24}(t-T_I)^4 + \frac{J}{6}(t-T_I)^3 + \frac{A}{2}(t-T_I)^2 + V(t-T_I) + X_0 \tag{4.4}$$

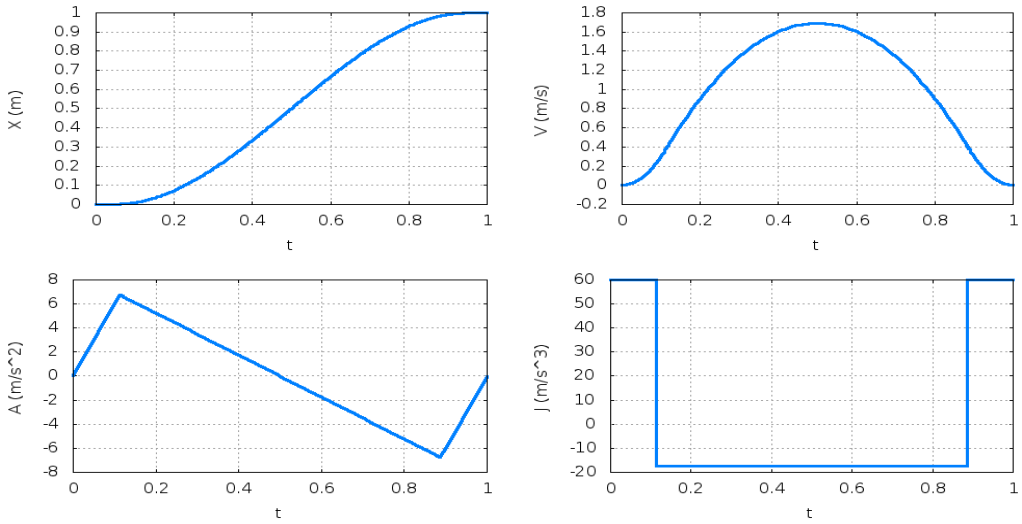


Figure 4.3: 3^{rd} degree polynomial interpolants with fixed jerk

4.2.3 3^{rd} Degree Polynomial Functions

4.2.3.1 3-Segment cubic functions with fixed time

This approach called 3-segment method is proposed by Broquère in [Broquère 10], it defines a sequence of three trajectory segments with constant jerks. Broquère fixes the three durations $T_1 = T_2 = T_3 = \frac{T_{imp}}{3}$ and obtain a system with 13 parameters where only the three jerks are unknown. Figure 4.2 shows an example of profile for position, velocity, acceleration and jerk relative to a sub-trajectory of one dimension. The initial and finial conditions are set as $M(T_I) = (0,0,0)$ and $M(T_F) = (1,0,0)$ respectively and the duration as $T_{imp} = 1s$.

4.2.3.2 3-Segment Cubics with Fixed Jerk

The 3-segment method solves the fixed time trajectory generation problem with fixed duration for each segment. However, it cannot guarantee that the computed jerk is always bounded. Here we introduce a variant of the three segments method with defined jerk.

As for the three segments method, the system is also defined by 13 constraints. But here we fix the jerk on the first and third segment $|J_1| = |J_3| \leqslant J_{max}$, where J_{max} is the bound for the jerk. Thus the unknown parameters of the system are J_2 and the three time durations linked by the whole duration of the trajectory T_{imp} ($T_{imp} = T_1 + T_2 + T_3$).

To choose the value of jerks, we take the velocity V_I and V_F into account. The jerks are fixed by $J_1 = J_3 = -J_{max}$ when $V_I > V_F$, while $J_1 = J_3 = J_{max}$ when $V_I \leqslant V_F$. Figure 4.3 illustrates a trajectory with the same situation as in Figure 4.2. The main difference is the jerk value $J_1 = J_3 = J_{max} = 60\ m/s^3$.

What to be noticed is, there are infinite choices of the jerk value. The reason why we set J_1 and J_3 as J_{max} is that we would get a longer period with a smaller J_2 ($T_2 = 0.78s$, $J_2 =$

$-18m/s^3$ in this case). So in the second segment, the acceleration will increase/decrease slower. From the movement's point of view, the motion becomes much smoother. However, if any computed time is negative or smaller than the machine cycle time, effort should be made to adjust the jerk.

4.2.4 4^{th} Degree Polynomial Functions

As discussed in section C.3.2, we need two segments of 4^{th} degree functions to define a motion and the equation C.10 is associated with 5 unknowns:

S_1 : the snap of the first segment;

S_2 : the snap of the second segment;

T_1 : the duration of the first segment;

J_0 : the initial jerk at the time instant T_I;

J_1 : the jerk at the time instant T_1;

and 3 constraints :

$$X(T_F) = X_F \tag{4.5}$$

$$V(T_F) = \frac{d(X(t))}{dt}|_{t=T_F} = V_F \tag{4.6}$$

$$A(T_F) = \frac{d^2(X(t))}{dt^2}|_{t=T_F} = A_F \tag{4.7}$$

To solve the problem, we need two more constraints. To choose these constraints, we take into account the different profile of time and jerk, with which the whole trajectory can be constructed. Therefore the trajectory is represented by different formats of 4^{th} degree functions, depending on the types of constraints.

4.2.4.1 Constraint type I

In a first instance, we introduce one parameter J_F, which is the final jerk at time T_F. In type I we have a continuous jerk at time T_1 and set the same initial and final jerks. So that the two constraints are denoted as:

$$J_1{}^+ = J_1{}^-$$
$$J_0 = J_F$$

Figure 4.4 shows the trajectory between the two same states as in Figure 4.2.

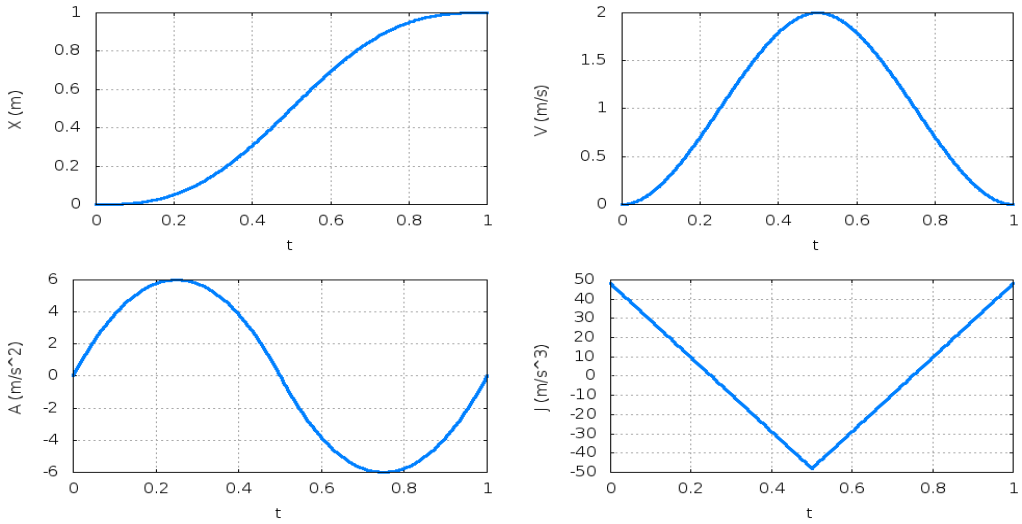


Figure 4.4: Constraint type I of 4^{th} degree polynomial interpolants

4.2.4.2 Constraint type II

Like for type I, we maintain the jerk continuity at T_1, but we make an equivalent duration on each segment, which means:

$$J_1{}^+ = J_1{}^-$$
$$T_1 = \frac{T_{imp}}{2}$$

With these 2 constraints, it is possible to solve the 3 polynomial equations 4.5 to 4.7 and get the result as following:

$$S_1 = \frac{192X_0 - 192X_F}{Timp^4} + \frac{108V_0 + 84V_F}{Timp^3} + \frac{22A_0 - 10A_F}{Timp^2} \tag{4.8}$$
$$S_2 = \frac{192X_F - 192X_0}{Timp^4} - \frac{84V_0 + 108V_F}{Timp^3} + \frac{22A_F - 10A_0}{Timp^2} \tag{4.9}$$
$$J_2 = -\frac{48X_0 + 18X_F}{Timp^3} - \frac{30V_0 + 18V_F}{Timp^2} + \frac{2A_F - 8A_0}{Timp} \tag{4.10}$$

To show the difference with the next constraint type, we choose a new final state $M(T_F) = (1,1,0)$, which is no more symmetrical in velocity. As clearly shown in Figure 4.5, the trajectory is divided into 2 segments with the same time. The initial jerk $J_0 = 30$ m/s^3 and the final jerk $J_F = 18$ m/s^3 are different. The jerk profile is continuous at time $T_1 = 0.5$ s.

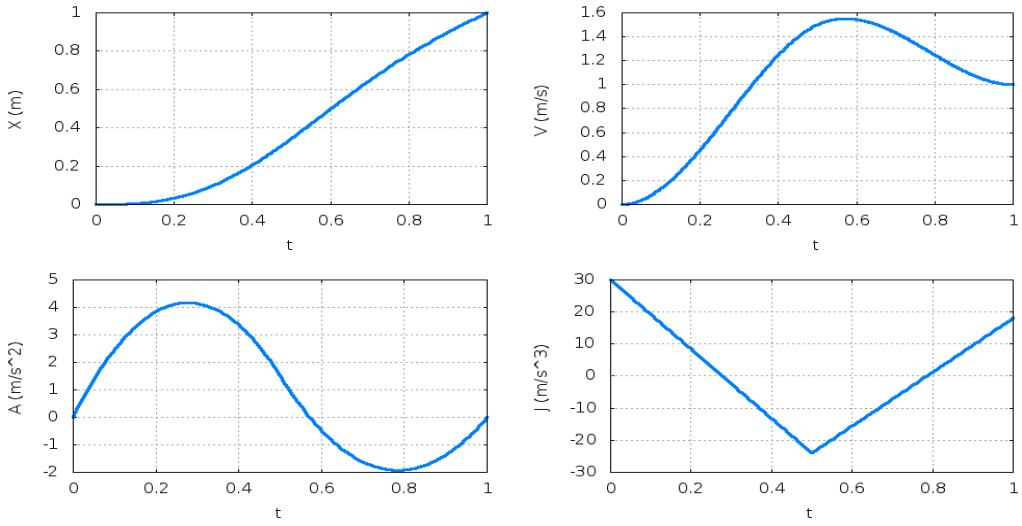


Figure 4.5: Constraint type II of 4^{th} degree polynomial interpolants

4.2.4.3 Constraint type III

The third type discards the jerk continuity while keeping the same time on each segment and same initial and final jerk:

$$J_0 = J_F$$
$$T_1 = \frac{T_{imp}}{2}$$

Similarly as for constraints type I and II, we solve the associated system as:

$$S_1 = \frac{192X_0 - 192X_F}{Timp^4} + \frac{72V_0 + 120V_F}{Timp^3} + \frac{4A_0 - 28A_F}{Timp^2} \tag{4.11}$$
$$S_2 = \frac{192X_F - 192X_0}{Timp^4} - \frac{120V_0 + 72V_F}{Timp^3} + \frac{4A_F - 28A_0}{Timp^2} \tag{4.12}$$
$$J_2 = -\frac{48X_0 + 48X_F}{Timp^3} - \frac{24V_0 + 24V_F}{Timp^2} + \frac{5A_F - 5A_0}{Timp} \tag{4.13}$$

An example of trajectory which connects the states defined in section 4.2.4.2 for type II is shown in Figure 4.6. Because of jerk discontinuity at time T_1 ($J_1{}^- = -12m/s^3$, $J_1{}^+ = -36m/s^3$), the trajectory has the same initial and final jerks ($J_0 = J_F = 24m/s^3$) with different snaps ($S_1 = -72m/s^4$, $S_2 = 120m/s^4$) in the same time interval ($T_1 = T_2 = 0.5\ s$).

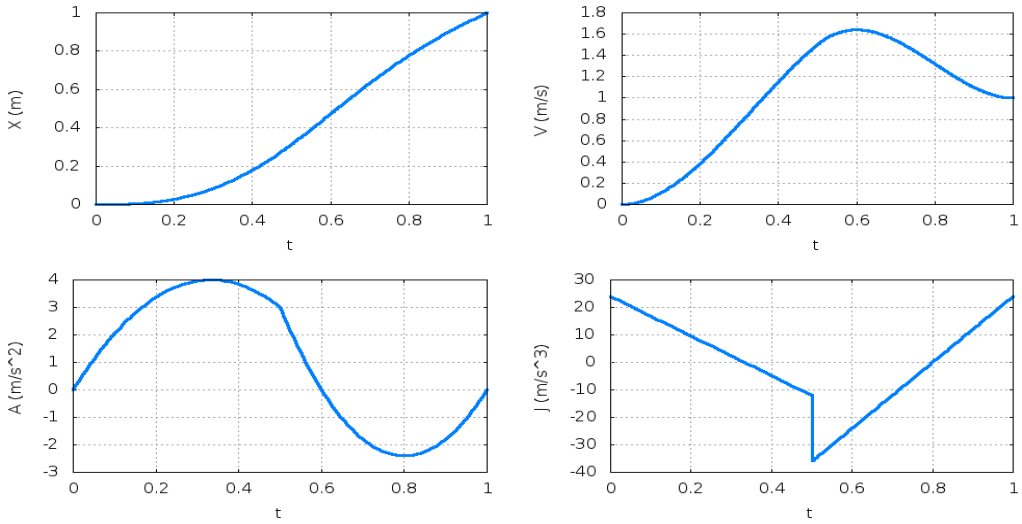


Figure 4.6: Constraint type III of 4^{th} degree polynomial interpolants

4.2.5 5^{th} Degree Polynomial Functions

Using a 5^{th} degree polynomials segment to define the motion (see Equation C.11), the solution is obtained directly from the six constraints:

$$\begin{aligned} X(T_I) &= X_0 \qquad X(T_F) = X_F \\ V(T_I) &= V_0 \qquad V(T_F) = V_F \\ A(T_I) &= A_0 \qquad A(T_F) = A_F \end{aligned}$$

Figure 4.7 illustrates one trajectory segment from state $M(0) = (0,0,0)$ to $M(1) = (1,0,0)$. The jerk profile is a parabola with the maximum value $J_m = 60m/s^3$. In this case, we have no additional parameters to bound the jerk.

4.3 Comparisons of different approximations

In this section, we propose to approximate some trajectories with the previous models. Firstly, we present some characteristics and show the results of an approximation. Then we compare the result of these approximations for some trajectories proposed in [Broquere 08]. These trajectories are defined by a path and a motion law.

4.3.1 Characteristics Definition

4.3.1.1 Trajectory error

An important characteristic of the approximation is the maximum error between the two trajectories, the approximated trajectory $\mathscr{T}(t)$ and the original trajectory $\mathscr{T}_{in(t)}$. From sev-

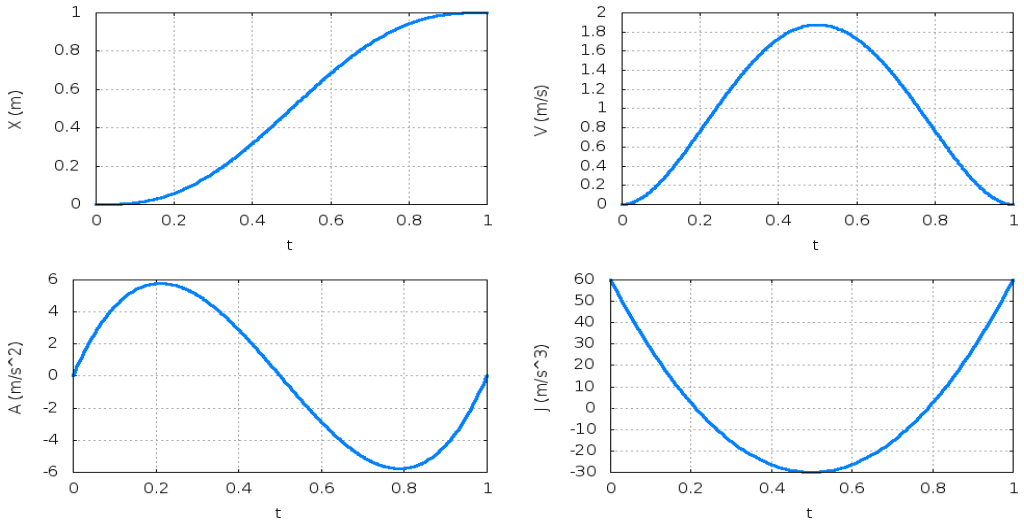


Figure 4.7: One continuous interpolant with 5^{th} degree polynomial

eral types of distances for comparison of trajectories, we choose the Hausdorff distance and the synchronous Euclidean distance and errors between velocities. The Hausdorff distance is defined as:

$$d_{Haus} = \max(\sup_{t_{in}\in[t_I,t_F]} \inf_{t\in[t_I,t_F]} d(\mathscr{T}_{in}(t_{in}), \mathscr{T}(t)), \tag{4.14}$$

$$\sup_{t\in[t_I,t_F]} \inf_{t_{in}\in[t_I,t_F]} d(\mathscr{T}(t), \mathscr{T}_{in}(t_{in})) \tag{4.15}$$

The synchronous Euclidean distance between the approximated trajectory $\mathscr{T}(t)$ and the original trajectory $\mathscr{T}_{in(t)}$ is defined as:

$$d_{SE} = \max_{t\in[t_I,t_F]} \sqrt{\sum_{j=1}^{n} ({}_j\mathscr{T}(t) - {}_j\mathscr{T}_{in}(t))^2} \tag{4.16}$$

The synchronous Euclidean distance between velocities is defined as:

$$d_{SEV} = \max_{t\in[t_I,t_F]} \sqrt{\sum_{j=1}^{n} \left(\frac{d_j\mathscr{T}(t)}{dt} - \frac{d_j\mathscr{T}_{in}(t)}{dt} \right)^2} \tag{4.17}$$

4.3.1.2 Number of segments

The total number of segments, N_{seg}, depends on the length of the intervals to approximate a function. It is an important parameter because it influences the trajectory error and the computation time. The N_{seg} can be provided by the user or computed according to the definition of a maximum error. To compare among the approaches on different paths, we set each approximation with the same maximum error and thus compute the N_{seg}, which

ensures that the trajectory error is smaller than the predefined limit.

The computation of N_{seg} for linear, circular and sinusoidal trajectories using 3^{rd} degree polynomials is illustrated in [Broquère 11]. For trajectories with irregular shapes and trajectories approximated with 4^{th} or 5^{th} degree functions, we propose an algorithm to compute N_{seg} that can limit the maximum error.

Algorithme 6 : Computation of N_{seg}

1: $N_{seg} = 1$; given a maximum error ε
2: Compute the trajectory and error d_{SE}
3: **while** $d_{SE} > \varepsilon$ **do**
4: $N_{seg} = N_{seg} + 1$
5: Compute the new trajectory and error d_{SE}
6: **end while**
7: return N_{seg}

4.3.2 Error of approximation for a trajectory

We suppose now that $\mathscr{T}_{in}(t)$ is bounded respectively in jerk, acceleration and velocity by J_{max}, A_{max} and V_{max}. We show in this paragraph that a relation exists between the error of approximation, the time T_{imp} and the bound J_{max}.

Let $\mathscr{V}_{in}(t)$ and $\mathscr{A}_{in}(t)$ denote respectively the velocity and acceleration of $\mathscr{T}_{in}(t)$. In a first time, we examine the case where the trajectory $\mathscr{T}_{in}(t)$ to approximate satisfies:

$$\mathscr{T}_{in}(t_I) = \mathscr{T}_{in}(t_F) = 0 \tag{4.18}$$

$$\mathscr{V}_{in}(t_I) = \mathscr{V}_{in}(t_F) = 0 \tag{4.19}$$

$$\mathscr{A}_{in}(t_I) = \mathscr{A}_{in}(t_F) = 0 \tag{4.20}$$

One can easily verify that these initial and final conditions gives three null jerks.

The trajectory to approximate $\mathscr{T}_{in}$ that gives the maximum error is symmetric. As the trajectory to approximate $\mathscr{T}_{in}$ is kinematically bounded and due to the symmetry, the maximum error between the two trajectories is at the middle of the trajectory. Likewise, the maximum error is obtained for a saturated function. For a short trajectory, the acceleration is not saturated and the more difficult function to approximate is defined by the four segments trajectory:

$$T_1 = T_4 = T_{imp} * \frac{2-\sqrt{2}}{4} \tag{4.21}$$

$$T_2 = T_3 = T_{imp} * \frac{\sqrt{2}}{4} \tag{4.22}$$

and the jerks are $J_1 = J_4 = J_{max}$, $J_2 = J_3 = -J_{max}$.

The maximum error between the two trajectories is then:

$$\varepsilon = \frac{\sqrt{2}-1}{48*\sqrt{2}} \approx 0.0061 * J_{max} * T_{imp}^3 \tag{4.23}$$

General case

Suppose $\mathscr{T}(t)$ is the approximation by the 3 segments method of the trajectory $\mathscr{T}_{in}(t)$between t_I and t_F.

We can write the $\mathscr{T}_{in}(t)$ trajectory as $\mathscr{T}_{in}(t) = \mathscr{T}(t) + (\mathscr{T}_{in}(t) - \mathscr{T}(t))$

By design the trajectory $\mathscr{T}_0(t) = \mathscr{T}_{in}(t) - \mathscr{T}(t)$ verifies the conditions 4.18, 4.19, and 4.20.

So the approximation error of $\mathscr{T}_0(t)$ on $[T_I, T_F]$ by a trajectory composed of three segments of cubic polynomial trajectory is less than $0.0061 \times T_{imp}^3 * (2 * J_{max})$.

As $\mathscr{T}(t)$ is approximated without error, $\mathscr{T}_{in}(t)$ that is the sum $\mathscr{T}(t) + \mathscr{T}_0(t)$ can be approximated with an error less than: $0.0061 * T_{imp}^3 * (2 * J_{max})$.

This result is extremely interesting as it gives the length of the time interval to approximate a function while insuring the approximation error is smaller than a defined threshold for J_{max}. What to be noticed is, a special case may happen where the jerk equals to zero in a segment. In this case, the error of this segment is zero, which means this is an exact approximation.

4.3.3 Example of a circular trajectory:

To approximate a trajectory following a circle of radius R at constant speed ωR, we can compute the maximum time interval T_{imp} to approximate the circle with a maximum error of ε.

The trajectory of the motion is defined by:

$$X_x(t) = R * cos(\omega t) \tag{4.24}$$

$$X_Y(t) = R * sin(\omega t) \tag{4.25}$$

and the jerk by:

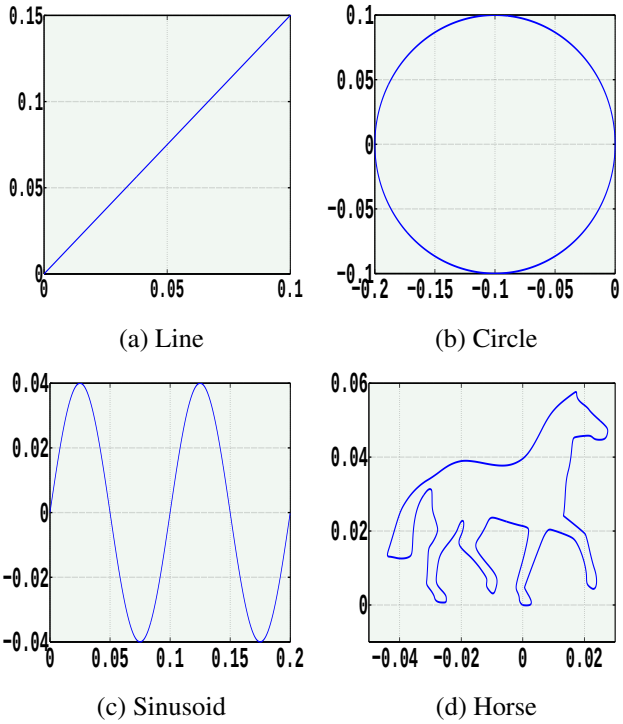
$$J_X(t) = \omega^3 R * sin(\omega t) \tag{4.26}$$

$$J_Y(t) = -\omega^3 R * cos(\omega t) \tag{4.27}$$

So the constant jerk is $\omega^3 R$ and the maximum time interval is then

$$T = \frac{\varepsilon\sqrt{3}}{0.0061 * 2 * J} = \frac{\varepsilon\sqrt{3}}{0.0061 * 2 * \omega^3 R} \tag{4.28}$$

For a mobile completing a turn in one second about a circle of radius $R = 0.1\,m$ with a



(a) Line (b) Circle

(c) Sinusoid (d) Horse

Figure 4.8: Four path samples to be approximated (unit: m)

maximum error of $\varepsilon = 10^{-6}\,m$, T is $T = 0.0149\ s$ corresponding to 68 points (6 points for an error of $\varepsilon = 10^{-3}\,m$). This result can be used directly when radius of curvature is known and the path is traversed at constant speed.

4.3.4 Comparison Demonstration

We build 4 trajectories (Figure 4.8) to evaluate our approximation procedures. These trajectories are defined from a path and a motion law: $\mathscr{T}(t) = P(u(t))$. $u(t)$ is the motion law, which is defined by a classical time-optimal 7 segment functions that is bounded in jerk, velocity and acceleration[Broquere 08]. The first three paths are linear, circular and sinusoidal, respectively, and are associated with the motion law, while the last path is defined by inkscape[1] and corresponds to a series of Bézier curves associated with the motion law. The motion law is defined as: $J_{max} = 0.9\ m/s^3$, $A_{max} = 0.3\ m/s^2$, $V_{max} = 0.04\ m/s$. To determine N_{seg}, we define the approximation accuracy with a maximum $d_{SE} = 0.001$ mm. Figure 4.9 presents the comparison between the six methods: Cubic I is the 3-Segment method with fixed time (section 4.2.3.1), Cubic II is the 3-Segment method with fixed time (section 4.2.3.2), Quartic I-III represent the 4^{th} degree polynomials with constraints type I-III. In each case, we compute all the characteristics mentioned in the last paragraph. We conclude from Figure 4.9 that:

[1] http://inkscape.org/

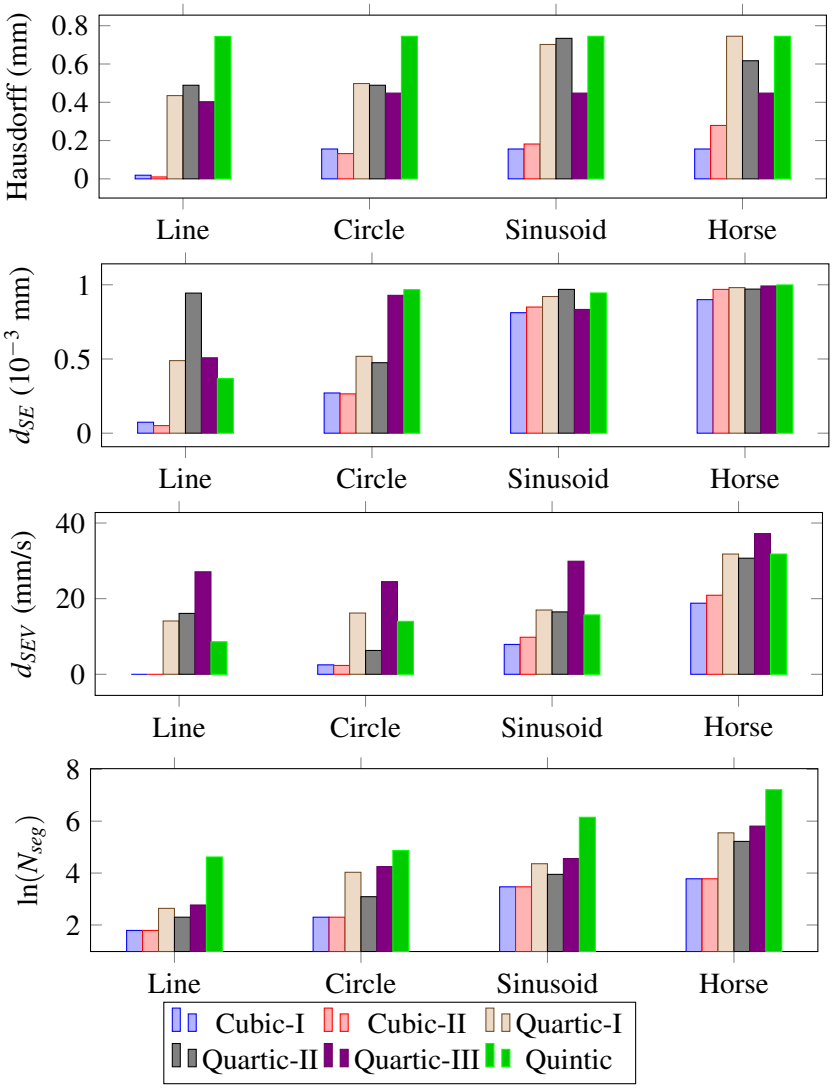


Figure 4.9: Comparison of characteristics for different approximations

1. Increasing the degree of the polynomial does not improve the quality of the approximation. With line as an example, 6, 14, 10, 16, and 100 segments are needed in approximation with cubic, Quartic I, Quartic II, Quartic III and Quintic functions, respectively. Likewise, the computation time of cubic approximation is shorter compared with other high-degree functions. The computation time is linked with the number of segments used to approximate the trajectory. The more the segments are used, the larger the computation time is.

2. The difference between Cubic-I and Cubic-II depends on the type of the trajectory. When the acceleration is high (small-radius curve), the first is more appropriate and vice versa.

3. As the 3^{rd}, 4^{th}, and 5^{th} degree approximations use 3, 2 and 1 segment, respectively,

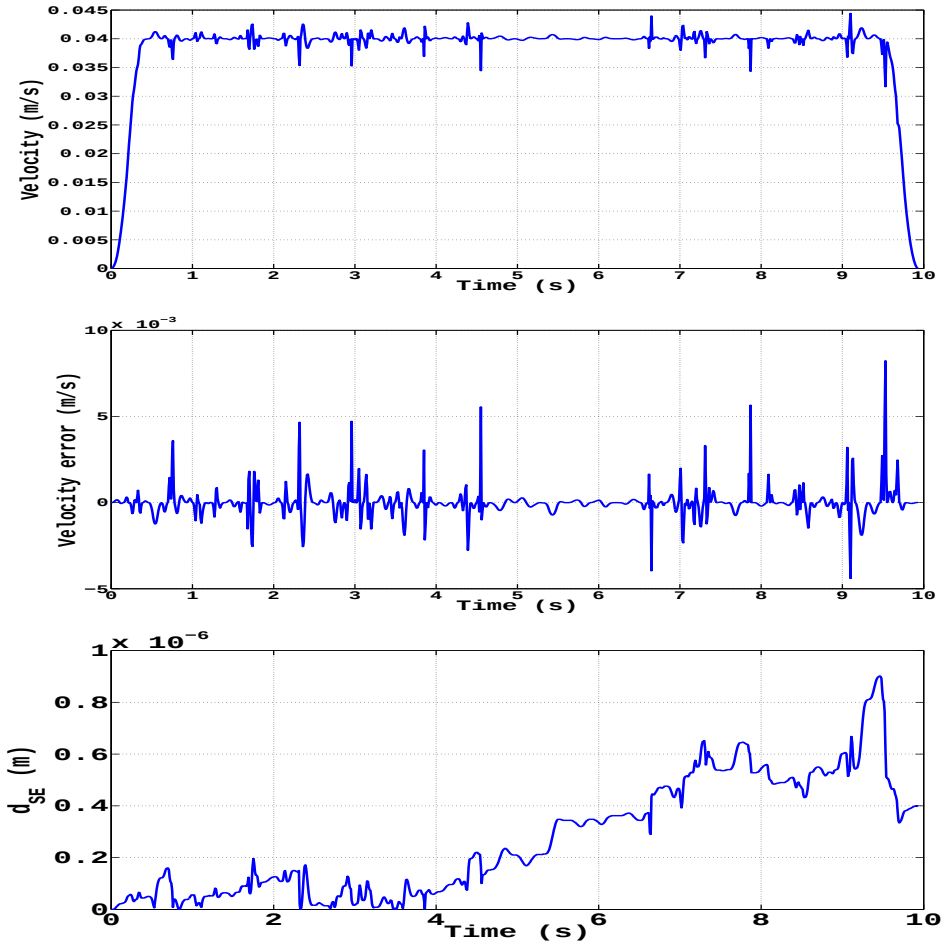


Figure 4.10: Top: computed motion law for the horse path; Middle: velocity error between desired and computed motion law; Bottom: trajectory error between the given trajectory and the approximated one.

a larger number of parameters are needed to define the approximated segment. However, this difference does not explain all the required segments for the 5th-degree case .

4.4 Experimental Results

To illustrate the implementation of approximation approaches based on SoftMotion trajectories [Broquere 08], we tested these approaches with a 7-DOF KUKA LWR-IV arm[2], which was controlled through the Fast Research Interface. The software control was developed using Open Robots tools: GenoM[Mallet 10b].

[2]http://www.kuka.com/en/company/group

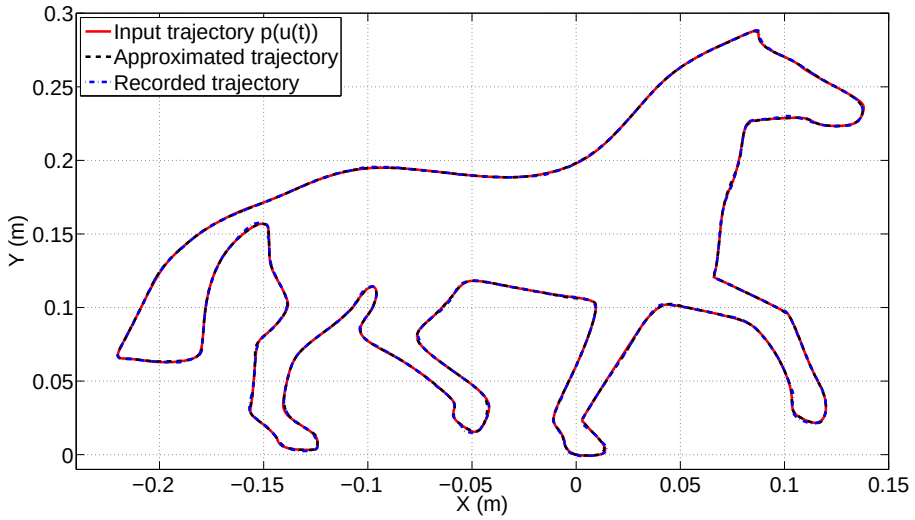


Figure 4.11: Theoretical path, computed trajectory and the one recorded while the LWR arm is executing the computed horse trajectory.

4.4.1 Control Level

The configuration of a seven joints arm manipulator is defined by a vector θ of seven independent *joint coordinates* which correspond to the angle of the articulations.

$$\theta = [q_1 \quad q_2 \quad q_3 \quad q_4 \quad q_5 \quad q_6 \quad q_7]^T$$

The *Pose* of the manipulator's end effector is defined by seven independent coordinates named *Operational Coordinates*. It is composed of a position vector $\mathbf{P} = [x, y, z]^T$ and a quaternion $\mathbf{Q} = [n, \mathbf{q}]^T$, where $\mathbf{q} = [i, j, k]^T$. They give the position and the orientation of the final body in the reference frame. Whitney [Whitney 69] gives the relation between joint velocities and Cartesian velocities:

$$\dot{\theta} = \mathbf{J}^{-1}[\mathbf{V} \quad \Omega]^T \tag{4.29}$$

where $\mathbf{V}$ and Ω represent the linear and angular velocities of the robot's end effector. And $\mathbf{J}$ is the Jacobian matrix.

To guarantee the tracking in presence of singularity, we have selected the damped least squares method for inverse kinematics:

$$\mathbf{J}^{-1} \simeq \mathbf{J}^T(\mathbf{J}^T\mathbf{J} + \lambda\mathbf{I})^{-1}$$

It is known that in the proximity of a singularity, the joint velocity references exceed the limits ($\theta \to \infty$). To avoid this problem, we propose to limit the velocity reference by weighting the velocities in function of the largest exceeding.

The KUKA arm was used to draw the horse shown in Figure 4.8d. This curve is chosen because it is composed of lines and curves. This approach is commonly used for industrial robots. As the 4^{th} and 5^{th} degree polynomial functions produce large errors, the horse trajectory was approximated using the method introduced in section 4.2.3.2. The desired motion law is a simple point-to-point soft motion computed by our soft motion planner [Broquere 08]. The linear end-effector motion bounds are set as: $J_{max} = 0.9m/s^3, A_{max} = 0.3m/s^2, V_{max} = 0.04m/s$. Figure 4.10 illustrates the computed velocity law and the two error curves. Figure 4.11 shows the input path, the approximated path and the path recorded during the robot execution. These two figures depict that the computed and executed trajectory coincides well with the given one with a quite small trajectory error. Result shows that validity of our trajectory generator.

4.5 Conclusions

In this chapter, we discussed the trajectory approximation problem and the possibility to built a simple trajectory generator using trajectory approximation. First, we proposed three approximation possibilities, 3^{rd}, 4^{th}, 5^{th} degree polynomial functions, to represent trajectories. For each trajectory type, different parameters are computed during the trajectory constructions. The overall approach can be summarized as follows:

- Expressing a given trajectory as a summation of K segments of the same duration
- Defining each segment as a polynomial function of degree k with some unknown coefficients
- Finding these unknown parameters based on the constraints of the motion (e.g., velocity, acceleration, jerk limits, etc.) while minimizing some performance criteria (e.g., trajectory error, number of segments, etc.)
- Once the unknown parameters of the approximated trajectory have been computed (e.g., end effector velocities), they are mapped to joint velocities by using the inverse of the Jacobian.

Then we compared 3^{rd}, 4^{th}, 5^{th} polynomial functions in approximation of any given geometrical trajectory by defining different characteristics, such as the trajectory error and number of segments. As a result, the 3^{rd} degree polynomial trajectories show high performance and simplicity for robot applications, which encourages us to develop tools to manipulate these trajectories for construction of robot controllers and planners.

5

Reactive Trajectory Controller

5.1 Introduction

In this chapter, we present the application of the trajectory controller in an industrial context, provided by the partners of the ANR ICARO[1] project. This setting is ideal for trajectory control because a worker is at the center of the assembly cell and the assembly to realize needs force control and real time obstacle avoidance. The trajectory controller enables the robot to realize the complete industrial assembly task while respecting human's safety and other HRI specifications. This interactive assembly task demands integration of different elements like geometrical reasoning, position and external force monitoring and control, 3D vision and human robot collaboration. The control system presented in this chapter is associated with the trajectory segmentation. This chapter presents firstly the *ICARO* project and the assembly setting, then it details the solution developed around the trajectory controller.

5.2 Introduction of *ICARO* project

ICARO is the abbreviation of "Industrial Cooperative Assistant Robotics". This project aims the development of tools in order to improve and to simplify the interaction between indus-

[1] *http://www.agence-nationale-recherche.fr/?Project=ANR-10-CORD-0025*

trial robots on one side and humans and the environment on the other side. The project also aims to produce tools build around a middleware software architecture securing the interoperability of these tools. *ICARO* distinguishes from other projects aiming service robots by the implementation of industrial use cases, by the association of research laboratories and SMEs ready to commercialize the project's results, by the importance granted to the safety standards and by the participation of a team working in ergonomics. The ergonomic aspects are important in order to deal throughout the project with all the aspects related to humans. ICARO is a highly collaborative project, which associates seven partners:

- LAAS-CNRS[2]: the two teams, Robotics and InteractionS (RIS) and Robotics, Action and Perception (RAP), are involved in this project. LAAS integrated one of the demonstration and developed tools for HRI around vision and trajectory control.
- The CRTD [3]: Le Centre de Recherche sur le Travail et le Développement. The CRTD worked on the ergonomics and the social impact of the proposed solution.
- EADS[4]: Airbus group. EADS provided an example of application in aeronautic industry and evaluated the proposed solution in this context.
- Tecnalia[5]: Tecnalia proposed an architecture for the integration developed around the middleware ROS. They also worked on the demonstration of the results.
- SIEMENS[6]: KINEO-CAM acquired by SIEMENS is a specialist of industrial collision checking and path planning.
- PSA[7]: Same for EADS, but for automotive industry. PSA provided also parts to be assembled and tools associated.
- LIRMM[8]: Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier. It provides tools for the force/torque and vision monitoring and control.

Today, inside the production plants the assembly stations are either manual or robotized. Protection grids systematically split the space between the robots and the humans and strongly limit the design of the assembly line. In particular, they occupy a large area and restrict the movement of people and vehicles. Recent evolutions of technology as well as of the safety standards allow a new production paradigm where humans carry out complex activities in collaboration with robots that execute the actions easy to automate, dangerous or non-ergonomic. Human and robots sharing a collaborative workspace is expected, either in

[2] *https://www.laas.fr/*
[3] *http://tof-ms.cnam.fr/*
[4] *http://www.airbusgroup.com*
[5] *http://www.tecnalia.com/*
[6] *http://www.siemens-home.fr/*
[7] *http://www.sochaux.psa.fr/*
[8] *http://www.lirmm.fr/*

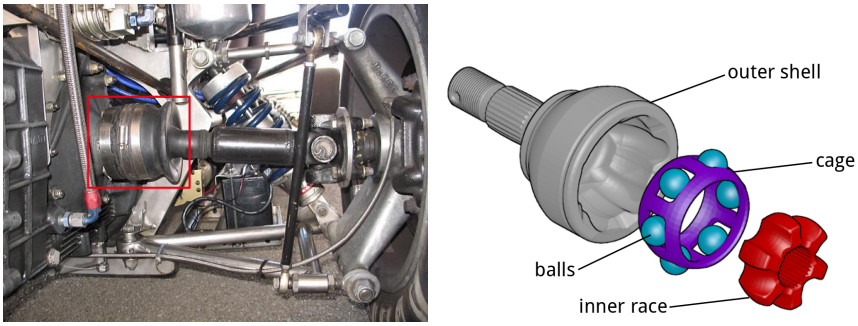


Figure 5.1: Left: Joint usage in a car. Right: Mechanical structure of a joint

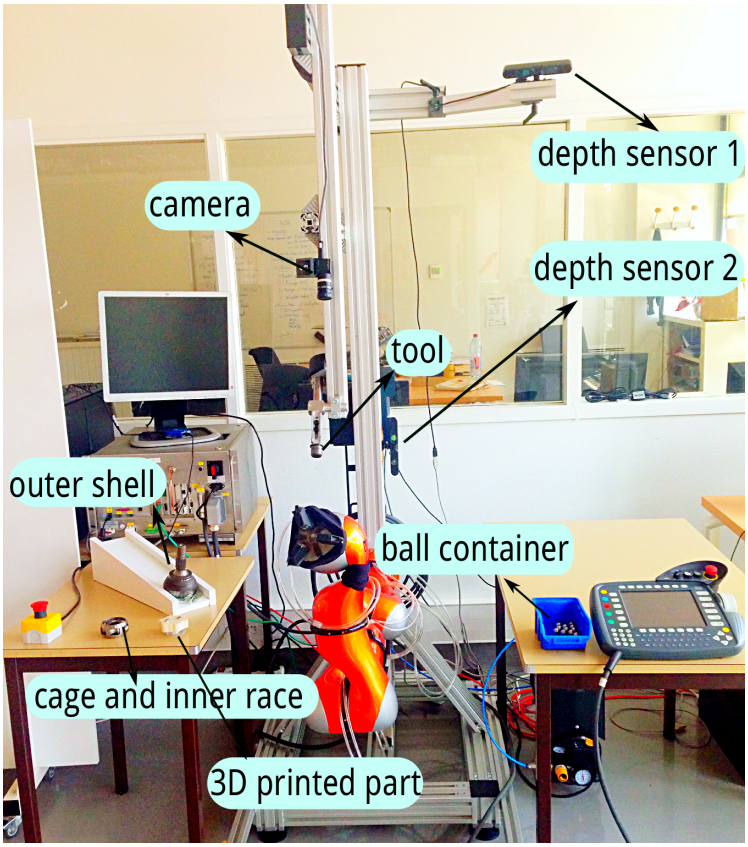


Figure 5.2: The *ICARO* setup at *LAAS-CNRS*

co-action or co-operation. An improvement of the global performance of the socio-technical system through the added value of human work in terms of ergonomics, through a better use of specific capabilities of each and through a simpler automation is expected.

So the goal of the *ICARO* project is to produce:

- Tools for intuitive robot programming including reactive planning of trajectories through fusion of real and virtual data as well as tools for robot programming through points learned by manual guidance and completed by process knowledge.

- Tools for perception of the environment linked to 3D models allowing faster, richer and more reliable interactions with the environment and the operators
- An open source software architecture enabling the interoperability of the tools developed as well as their dissemination. As ROS (the Robot Operating System)[9] middleware is now an industrial standard, it allows to transfer new human-robot interaction methods for complex manipulation applications towards the industry.
- A methodology of task attribution among humans and robots based on the physical, cognitive and ergonomic analyses as well as the production constraints.

After a long discussion, the partners chose to work on the assembly of the RZEPPA joint. These joints are widely used in industries, especially in the automotive industry. Figure 5.1 shows an example of a RZEPPA joint used in a car between the gearbox and the wheels. The mechanical structure of a joint is also illustrated in this figure. It includes 4 parts: an outer shell, a cage, 6 balls and an inner race.

The project built up an automated setting, presented in Figure 5.2. The objective of this setup is to validate the concepts developed in a real assembly task. In this task, the robot and human operator share the same workspace. The robot supports to human workers by collecting, delivering parts just as needed, by positioning of the workpiece and stabilizing components during assembly. The safety of the operator must be guaranteed when he/she cooperates with the robot manipulator, hence the robot has to be reactive. A first depth sensor is used to track the gesture of the operator. A second depth sensor is employed to update the environment model, such as the human position and the obstacle information. Collision Detector (KCD) from the SIEMENS Company provides fast and reliable collision detection, based on minimal distance computations.

Figure 5.3 illustrates the software architecture. The *ICARO* project uses a ROS architecture. In such architecture, all the models are combined in a URDF[10] file. Each component in the software structure is a ROS node[11]. The KCD module is synchronized with the OctoMap module [Hornung 13]. The OctoMap is updated at 30Hz with the point clouds acquired by an Xtion PRO LIVE RGB-D[12] camera. In order to limit the apparition of sparse outliers, this project uses a statistical filter [Rusu 11] available in the PCL library [PCL 10]. For all points, the statistical filter measures the distance between every point and its closest N neighbours. These data are fitted to a Gaussian. A threshold is set in function of the mean and variance of the Gaussian. For every point for which the mean distance to its neighbours is higher than the threshold, is considered as an outlier and is removed. The path planner uses the standard OMPL [Şucan 12] module. Each one of these nodes is implemented using a robust state machine fully driven by the ROS actionlib protocol allowing multi process

[9] *http://www.ros.org/*
[10] *http://wiki.ros.org/urdf*
[11] *http://wiki.ros.org/Nodes*
[12] *https://www.asus.com/us/Multimedia/Xtion_PRO_LIVE/*

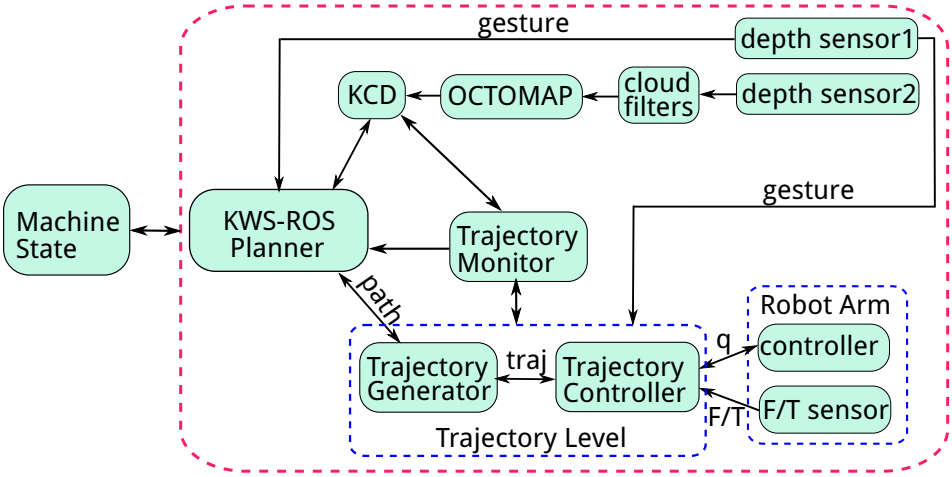


Figure 5.3: The software architecture of *ICARO* project

communication. Both nodes update the robot position by a ROS topic and provide convenient topics to add, move, remove, attach and detach geometric parts on the fly. This allows covering a wide variety of scenarios from simple pick and place tasks to complex human collaboration tasks.

We are going to present the different elements of this architecture in the rest of this chapter. We will detail the role of our controller in each sub-task. Then, the experimental results will be presented as well.

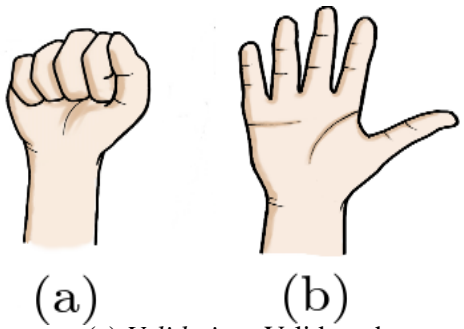


Figure 5.4: Two different gestures. (a) *Validation*: Validate the current operation, go to the next task of the whole procedure. (b) Rebut: Reject the current operation.

5.3 Applications in the *ICARO* Project

5.3.1 Gesture Tracking

The objective of this module is to use gestures analysis to replace the usage of buttons on the control panel. The advantage of using gestures is that it makes the human operator interact with the robot easily. This module use OpenNI to create a nodelet graph to transform raw

data from the device driver into point clouds, disparity images, and other products suitable for processing and visualization. Here two gestures are applied, shown in Figure 5.4. A *Validation* gesture can accept the current action and switch to the next step. For instance, when the operator is ready and the robot is at the initial position, a *Validation* gesture asks the robot to beginning the first task. On the contrary, the Rebut gesture tells the system that there is something wrong in the current operation, so that the system will halt at the moment, or execute the action corresponding to the *Rebut* gesture. For example, when the robot shows the outer shell so that the human inspects it, the *Rebut* gesture asks the robot to place the outer shell into the garbage.

What to be noticed is, the depth sensor can recognise two gestures and generate there three commands:

- Command 0: represents the *Validation* gesture;
- Command 1: represents the *Rebut* gesture;
- Command 2: the Kinect doesn't see the operator.

Command 2 is generated if the operator goes too far away from the workbench or leaves the useful area of the depth sensor. For example, if the human operator leaves temporarily for something unexpected, the robot receives immediately Command 2 and stops the current movement. The state machine memorizes the current state. When the operator come back, a *Validation* gesture trigger the recovery of the task.

The trajectory controller reacts to Command 0 and Command 1 by transforming different paths to trajectories. It reacts also to Command 2 by slowing down the motion and changing the trajectory timing law to $s(t) = 0$. Figure 5.5 illustrate a case where the trajectory controller pause and recover the robot motion by receiving different commands.

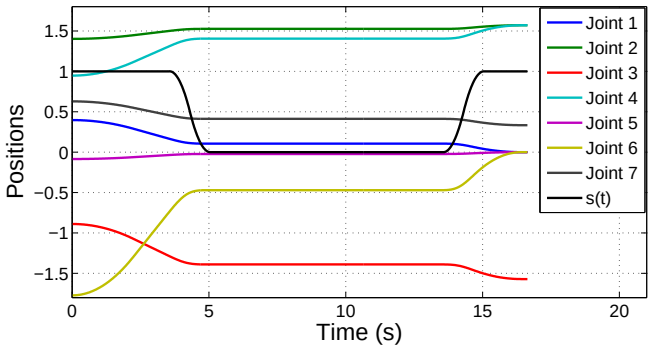


Figure 5.5: The trajectory received the Command 2 and stopped the current motion at around $t = 3.8s$. At the instant $t = 13.8s$, the *Validation* gesture was sent to trajectory controller so that it recover the previous movement.

It should be noticed that we cannot set the time law $s(t)$ to zero directly, because it will stop the robot immediately and cause a huge force/torque on the robot joints. It may damage

the hardware of the robot and it also doesn't satisfy the comfort of human. Therefore, the time law $s(t)$ must be changed smoothly. In this work, we use the concept of the Optimal Motion to describe the time variation. The Optimal Motion is a motion with jerk, acceleration and velocity constraints successively saturated [Herrera 05]. We use the velocity of an Optimal Motion to represent the time variaiton. In this case, we are able to control the time change by defining the maximum jerk J_{max}, acceleration A_{max} and velocity V_{max}. In this study, we assumed that a robot should not increase its speed to react to environment changes because it could cause some people anxiety. Thus $s(t)_{max} = 1$.

5.3.2 Reactive Planning

Reactive path planning is mainly performed by the controlling node which basically interacts with the path planning node and some critical nodes executing the global task, here grouped in the *ICARO* stack[13], such as the trajectory execution node. The reactive strategy is depicted in Figure 5.6.

As part of the ICARO project, the SIEMENS company developed a reactive path planning package over the ROS middleware. This kws-ros-interface package (hereafter kws-ros), is built on top of the SIEMENS software component. SIEMENS addresses all aspects of motion processes including collision-free automated path planning. It includes the most common motion types, such as joint motion, and features an advanced algorithm for detecting collisions along a trajectory, which is both fast and exact, regardless of kinematical complexity. SIEMENS developed a Trajectory Monitor to supervise the current robot motion and to check if future positions are in a collision state. From the task to realize and the collision states, the Trajectory Monitor requests the path planning to produce a path, which is transformed in a trajectory by the Trajectory Generator. As soon as a new trajectory is computed, the trajectory controller switches to the new one.

5.3.3 Positioning of the outer shell

The outer shell needs to be firstly aligned so that its tracks will be aligned with the insertion motion. We printed in 3D a part with a complementary shape of the internal shape of the outer shell and we fasten it on a table. Simply inserting an outer shell on the part align it, as shown in Figure 5.7. The orientation procedure is:

- Down: The robot starts at the position above the printed part and moves down until it contact the part. The contact can be detected by a change in the vertical force.

[13] Packages in ROS are organized into ROS stacks. Whereas the goal of packages is to create minimal collections of code for easy reuse, the goal of stacks is to simplify the process of code sharing. Stacks are the primary mechanism in ROS for distributing software. Each stack has an associated version and can declare dependencies on other stacks. These dependencies also declare a version number, which provides greater stability in development.

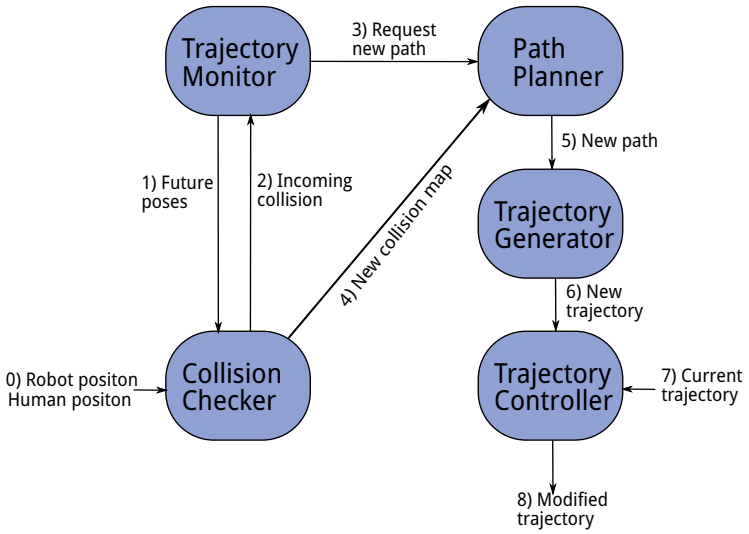


Figure 5.6: Reactive planning structure. The input and output of each component is detailed in this figure. Trajectory Monitor supervises the current robot motion and checks if future positions are in a collision state. In case of a coming collision, it requests a new path to the path planner. The new path is directly sent to the trajectory generator and is converted into a new trajectory. The trajectory controller merges the current trajectory with the new one to obtain a smooth transition while avoiding the obstacles.

- Rotation: After the contact, the robot stops and starts maintaining a vertical force until the outer shell goes down and the force decrease.
- Down: Continue going down to a fixed position.
- Release: Open the gripper to release the joint, which finish its alignment by gravity.
- Back Rotation: Rotate back to the aligned orientation.
- Grasp: Re-grasp the outer shell.
- Up: Go up to the initial position.

Figure 5.8 and Figure 5.9 shows the experimental results. F_z is the force along the Z axis. In Figure 5.8 we can see that, at $t = 10.5\ s$, the robot started to go down. At around $t = 12\ s$, F_z increased, which means the outer shell has contacted with 3D printed part. Then the trajectory controller switched to the rotation movement to find the aligned position. The vertical force was maintained at 23 N during this motion. From $t = 14\ s$ to $t = 16\ s$, the force F_z decreased smoothly to the initial value. It means the outer shell has aligned with the 3D printed part. Then the robot continued to go down until it contacted the table at around $t = 19\ s$. Gripper was opened to release the joint and the last joint of the robot was rotated back to the aligned orientation. Once the aligned task was successfully done, the robot re-grasped the outer shell and the trajectory controller switched to the up motion and

executed it at last. Figure 5.9 depicts the forces along the X and Y axis and the torques on each axis during the whole procedure.

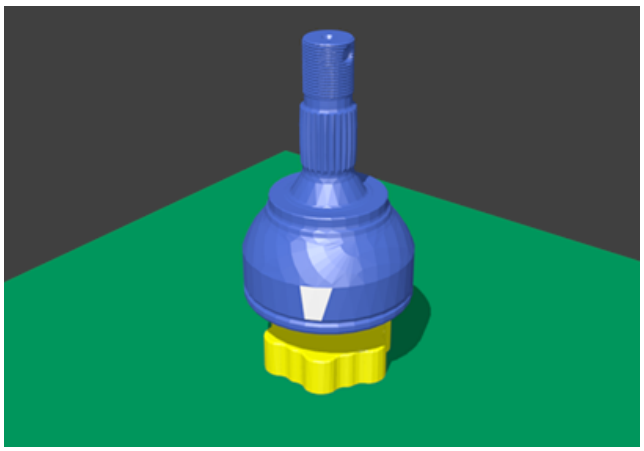


Figure 5.7: Aligning the outer shell using the force detection and a 3D printed part.

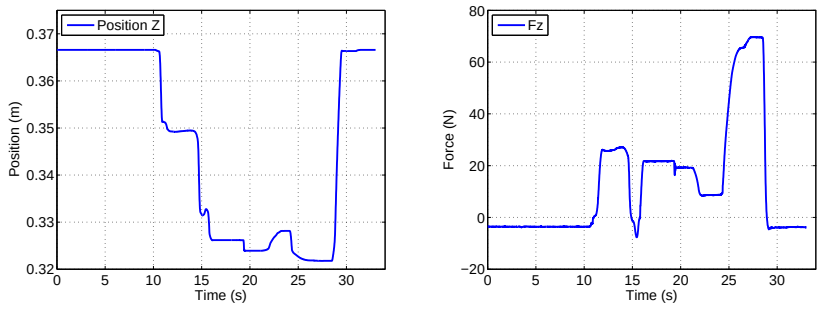


Figure 5.8: The position and force of the end-effector along the Z axis during the positioning task.

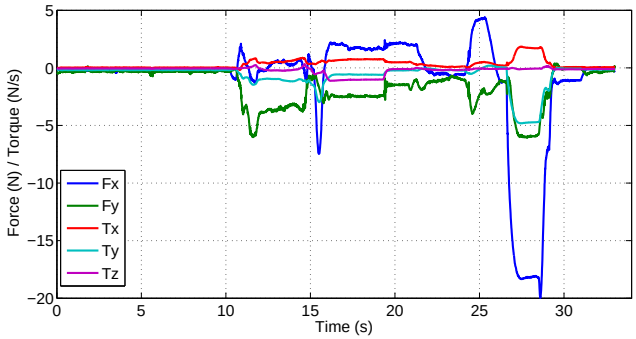


Figure 5.9: Force F_x, F_y, Torque T_x, T_y, T_z during the task. Measures are provided by the FRI interface

5.3.4 Ball Insertion Task

Inserting the six balls for the RZEPPA joint is a repeated and complex task, During the insertion, the human operator has to apply large force and musculoskeletal disorders might exist, so it is reasonable to employ robots in this job. Figure 5.10 shows a human operator inserting balls into the RZEPPA joint with a mechanical tool. Firstly, the human fixes the outer shell on the workbench and then places the cage and inner race inside the outer shell. In the third step, the operator inserts one ball inside the outer shell to align the cage and inner race with the outer shell. At last, the operator uses the tool to insert the other balls. Figure 5.11 illustrates the ball insertion procedure. We take only the case of inserting one ball, it is divided into 4 parts:

- Initial position: In this position, the cage and inner race are strictly aligned with the outer shell.
- Open position: The outer shell rotates by some angle to make a gap between the cage and the outer shell.
- Insertion Position: The operator places the ball in the gap.
- Closed Position: The outer shell rotates back to be closed. The ball is successfully inserted.

Figure 5.10: A human operator inserts balls into the RZEPPA joint. In picture (d), the human uses one hand to open the joint and the other hand to insert the ball.

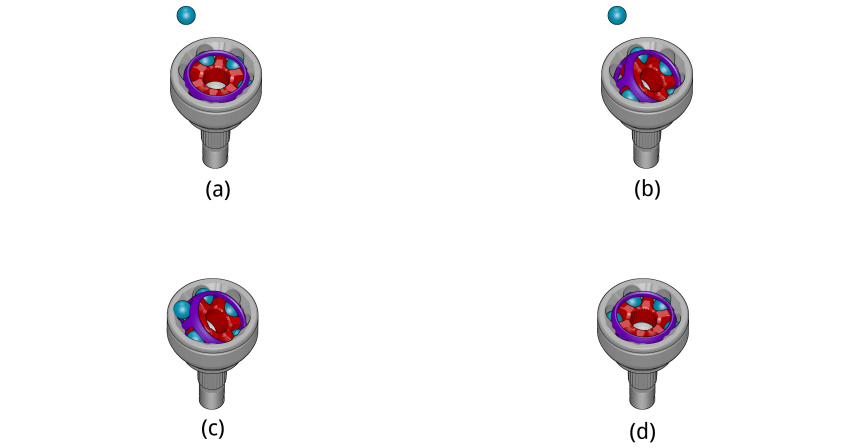


Figure 5.11: Sub-task of the insertion task of one ball. (a): Initial position. (b): Open position. (c): Insertion position. (d): Closed position.

This task is complex for robot because we must consider the force and torque produced during the insertion. The robot must be able to react to force changes and cannot be rigid. Thus here we introduce a trajectory based impedance controller. Impedance control has been broadly used in the context of human-robot interaction in the previous works. [Ikeura 95] has investigated the human characteristics and has shown that the damping parameters in the impedance model are the predominant coefficients that allow setting the acceleration/deceleration features in the context of PHRI (Physical Human Robot Interaction). [Duchaine 09] used a fixed virtual damping that can lead to an inefficient co-manipulation. [Ikeura 02] adjusted on-line the damping parameters, in an optimal manner by minimizing a selected cost function. An online adjustment of this coefficient based on a real-time estimation of the human arm stiffness was also proposed in [Tsumugiwa 02].

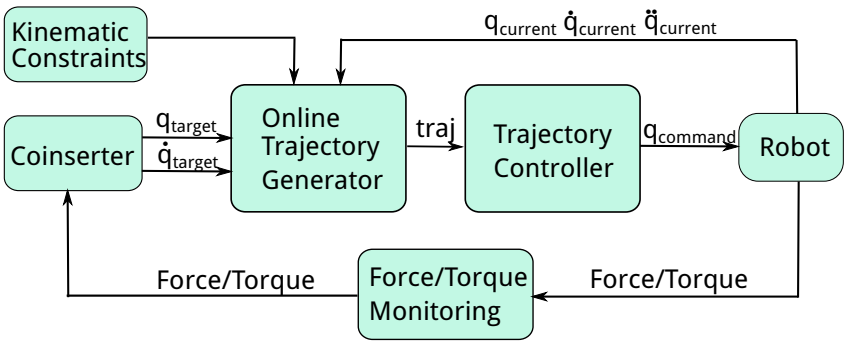


Figure 5.12: The structure of the Online Trajectory Generator based impedance controller.

In this work, we use a method that consists of an event controlled online trajectory generator associated to a control structure allowing a good tracking of the generated trajectory with a desired impedance property. The structure is illustrated in Figure 5.12. The force

applied to the end-effector is the only physically exchanged signal showing the robot how to move. *Coinserter* is a ROS node developed by the *ICARO* partner *LIRMM*. This node monitors the force and torque of the end-effector and computes the next target position and velocity for each joint. This force gives the information about the desired displacement direction of the end-effector. The *Coinserter* receives the Cartesian wrench provided by the force/torque sensor implemented in the end-effector, so an inverse kinematics process is required in this node to produce joint motions. In this case, the robot is still in position control mode, the target positions q_{target} and velocities $\dot{q}_{target}$ are generated in accordance with the monitored force and torque if they are greater than a given threshold. Then the online trajectory generator produces trajectories from the current state ($q_{current}$, $\dot{q}_{current}$,$\ddot{q}_{current}$) to the target one. As the motion planner doesn't provide the target accelerations, we set the $\ddot{q}_{target} = 0$ to compute the transition trajectories for simplicity.

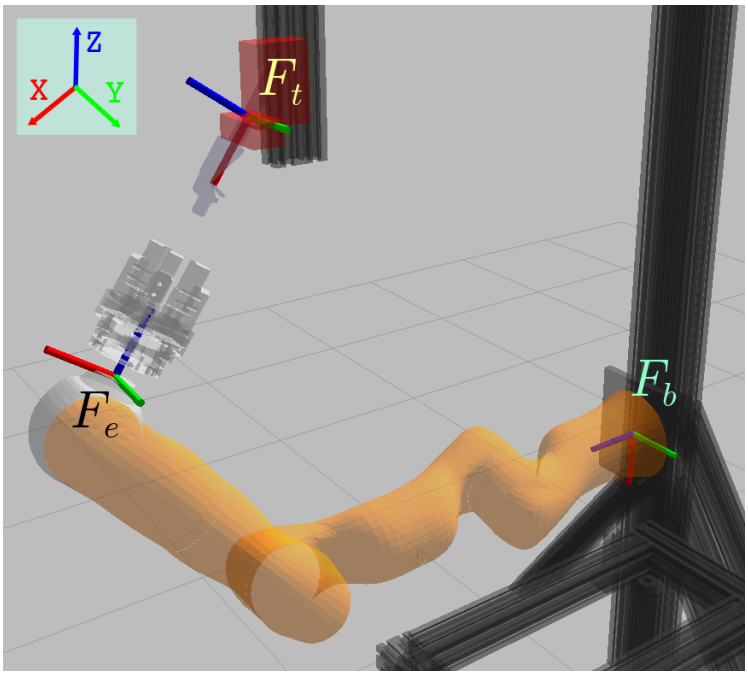


Figure 5.13: The frames of the *ICARO* setup. F_t: the tool frame; F_e: the end-effector frame; F_b: the robot base frame.

What to be noticed is, we apply an impedance controller using the wrench applied on the effector but expressed in the tool frame F_t. The different frames are depicted in Figure 5.13. To change the wrench H from a end-effector frame F_e (provided by FRI) to frame F_t, we apply:

$$^{t}H_e = {}^{t}W_e \, {}^{e}H_e \tag{5.1}$$

where $^{t}H_e$ is the Cartesian wrench expressed in the tool frame, $^{e}H_e$ represents the wrench in the end-effector frame, which is directly get from the force/torque sensor in the robot. $^{t}W_e$

is the 6x6 transform matrix:

$$ {}^{t}W_{e} = \begin{bmatrix} {}^{t}R_{e} & 0 \\ [{}^{t}T_{e}] \times {}^{t}R_{e} & {}^{t}R_{e} \end{bmatrix} \tag{5.2} $$

where ${}^{t}R_{e}$ is the rotation matrix from the end-effector frame to the tool frame. To transform the wrench to the tool frame by simple rotation, we firstly transform it into the robot base frame F_b. We simply apply the rotation from F_b to F_e (${}^{b}R_{e}$), then make the rotation from F_e to F_t:

$$ {}^{t}R_{e} = {}^{t}R_{b}\,{}^{b}R_{e} \tag{5.3} $$

with ${}^{t}R_{b}$ is the 3x3 rotation matrix from the robot base frame to the tool frame. According to Figure 5.13, F_t rotates by an angle θ about the Y axis, therefore the rotation matrix can be represented as:

$$ {}^{t}R_{b} = \begin{bmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix} \tag{5.4} $$

At last, we put this rotation matrix in W (Equation 5.2) and derive ${}^{t}H_{e}$ in Equation 5.1, which is the one used by the impedance controller.

Figure 5.14 depicts the ball insertion procedure with *ICARO* setup. The frequency of $coinserter$ is 50 HZ while our trajectory controller runs at 100 HZ. Thus our trajectory controller has 0.02 seconds to compute the trajectory from the current state $M_{current}$ to the target state M_{target} and to execute it. Here we apply the Three-Segment method With Bounded Jerk, which is presented in Chapter III to compute this short-term trajectory. The imposed time is $T_{imp} = 0.02\ s$. Thus trajectories can be built by the function:

$$ \mathscr{T} = ComputeTraj(M_{current}, M_{target}, q_{limits}, T_{imp}) \tag{5.5} $$

Where q_{limits} is the kinematic bounds of each joint.

Figure 5.15 illustrates the experimental results. As the insertion procedures for each ball is quite the same, for simplicity, we inserted only two balls in this experiment. The force along the axis F_Z, the position P_z of the end-effector, the received target joint values and the real joint positions were recorded respectively. From the top and middle figures we can see that the movement of the end-effector is consistent with the force exerted on it. while the figure on the bottom takes the position of joint 1 for example, showing that the trajectory controller can generate the switching trajectories and send the commands to the robot motors in real time.

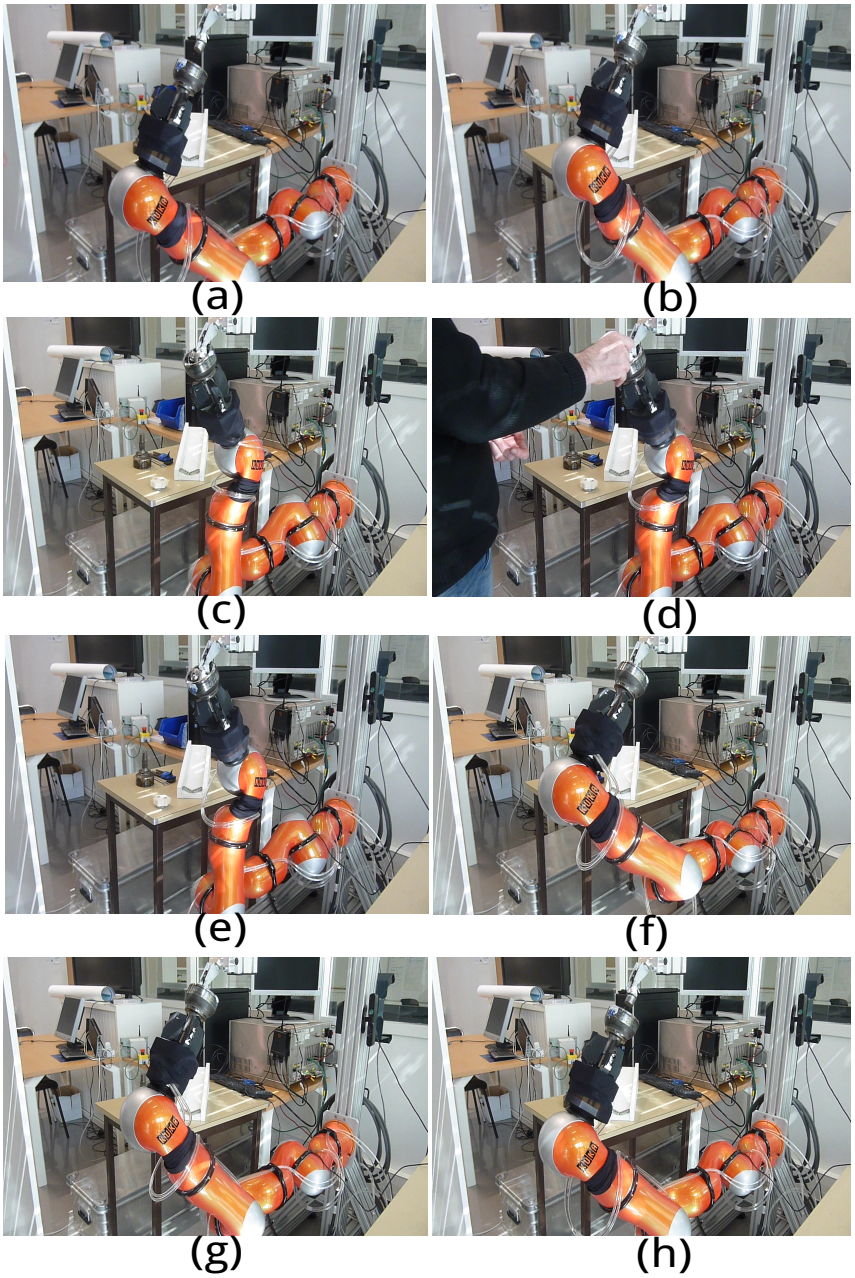


Figure 5.14: Real experiment of the ball insertion task with *ICARO* setup. (a): Initial position. (b): The outer shell is aligned with the tool. (c): The robot moves to go to open position. (d): The operator places a ball in the gap. (e): The outer shell can hold the ball without the help of human. (f): The robot goes back to close the outer shell. It moves a bigger angle than step (c) to guarantee that the ball is fully enfolded by the outer shell. (g): The robot moves again to the aligned position and is ready to rotate for inserting the next ball. (h): Once all 6 balls are inserted successfully, the robot returns to the initial position.

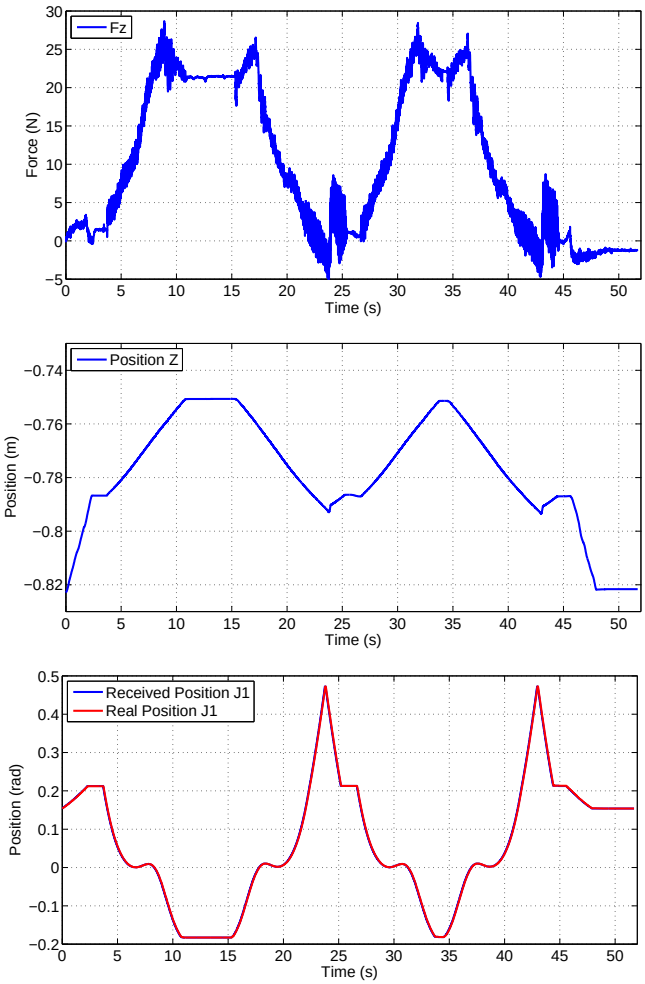


Figure 5.15: Top: the force along the Z axis on the end-effector. Middle: the position of the end-effector along the Z axis. Bottom: the received position and real position of joint 1.

5.3.5 Hands Monitoring

During the ball insertion task presented in the previous subsection, the human's hands are close to the outer shell, where the security risk exists, shown in Figure 5.16. The fingers of the operator may be stuck in the gap between the out shell and the cage. Therefore, for security reasons, the movement of human's hands must be monitored during the ball insertion procedure. A camera is used to follow the operator's action, as shown in Figure 5.17. The radiuses of the different regions are defined from the distances to the center of the outer shell. In the *danger* region, the robot is expected to keep static even it gets a *Validation* gesture to move to the next step. In other words, the safety has a higher priority than the gestures.

The integration of the monitoring of hands with the trajectory controller is done in a similar way to the gesture tracking. We define two states, *hand near* and *hand far*, which represent the danger and the safety areas. The commands are directly sent to the trajectory

Figure 5.16: The hand of the human gets close to the outer shell when inserting a ball. If the robot moves unexpectedly at this moment, the human fingers may be injured.

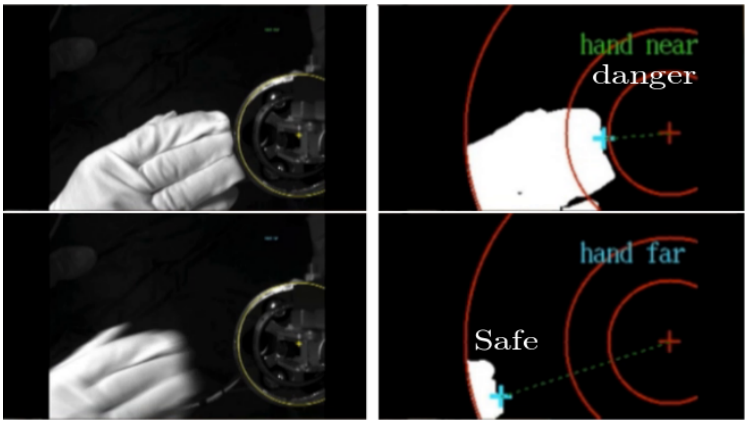


Figure 5.17: Hands monitoring for safety

controller. According to the different states, the robot executes the corresponding action, such as continuing or stopping the task.

5.4 Conclusion

A reactive trajectory controller has been presented with some results relative to an industrial project. The first results presented illustrate the versatility of the controllers based on online trajectory generation. In the example shown here, the controller can be integrated with different components, such as path planner, collision checker and robot controller. It is able to switch between trajectories and suspend/recover the control task during the time the human moves.

The trajectory controller proposed uses an online trajectory generator to build a trajectory to join up the trajectory to follow. It is very simple to use and implement and gives an

efficient solution to follow trajectories and track moving objects in the HRI context. More precisely, it can adapt kinematic limits to the changing state of the scene and switch between trajectories and control modes.

We also extend the trajectory controller to an OTG-based impedance controller. This approach mainly lies on the implementation of a specific event-based online trajectory generator, which produces trajectories to track when a force is exerted on the end-effector. The proposed approach permitted to address the issue of maintaining the immobility of the robot configuration when there is no interaction force.

The challenge is now to extend this type of trajectory controller and the concept of control primitives to manage forces, to handle events based on force sensing and to control dual arm manipulators.

6 Conclusion and perspectives

6.1 Conclusion

This work focuses on the development of online trajectory generation (OTG) for trajectory control. The objective is to build robot easier to control, in particular when the robot interacts with humans in an dynamic environment. Taking into account the time through the definition of trajectories seems to complicate the control, but as we have seen some difficult operations become simpler:

- Smoothing path: Introducing robot and task constraints to generate trajectories directly gives smooth trajectories.
- Switching between trajectories: Using the concept of online trajectory generation and trajectory control primitives, the system can switch between the current trajectory and the new one, simply by computing a connection trajectory in real time.
- Reactivity: The trajectory controller can easily switch between control modes and input trajectories, which makes the robotic system reactive to deal with unforeseen sensor events.

In a real industrial application, the high-level software on the robot plan the interaction tasks and the motion to accomplish the tasks. The main contribution of this thesis is to implement the methodologies to provide the robot with the ability to react to the sensory

information and events: mainly the visual tracking and force events. Trajectory planning and control is proposed as the center of this thesis. Some simulation and experimental results show how OTG-based trajectory control can deal with unforeseen sensor events.

6.1.1 Trajectory Generation

This work proposes new solutions to generate smooth trajectories from paths. In particular we proposed methodologies to produce trajectories from path defined by via-points. One interesting result is the possibility to produce a smoothed trajectory which remains inside a smoothing area defined by a parameter. The use of projection for constraints, also helps us to synchronize axis trajectories. As these algorithms don't use iteration, optimization or random data, computations are direct and are able to be used online.

An algorithm using shortcuting heuristic is also presented to produce more natural-looking trajectories which is bounded in jerk, acceleration and velocity. We also extend this approach to be implemented on-line to smooth the trajectory during execution.

We also presented tools to simplify the approximation of trajectories. We compared 3^{rd}, 4^{th}, 5^{th} polynomial functions to approximate any given geometrical trajectory. The 3^{rd} degree polynomial trajectories shows high performance and simplicity for service robot applications, which encourages us to develop tools to manipulate these trajectories for building robot controllers and planners.

The proposed trajectory generation method could be compared to the learning approaches. We take into account the Online Trajectory Generation concept when planning the trajectories, thus the proposed method is able to used in real time, which made the robot more reactive. Otherwise, the learning approaches will require large set of data to learn, and the learned control policies may have problem to cope with a dynamic and unpredictable environment where a robot works (e.g. a service robot work at home, or an industrial robot shares the workspace with the human operator).

6.1.2 Trajectory Based Control

For the trajectory based control, we give some examples of usage in the industrial project *ICARO*. The proposed trajectory controller is capable to achieve complex tasks. The author of this document argues that the trajectory generation based control is easier to implement with different sensor systems, such as different vision systems, or when the perception is obtained through the fusion of different sensors. Compared to visual servoing, another advantage is that trajectory based system can be easier to integrate with a path planner. In addition, the trajectory controller is also easy to communicate with other components, such as the collision checker, robot controller and task/motion supervisor. Due to these properties, with different Human-Robot Interaction specifications, stopping, slowing down, and accelerating on a trajectory can be also achieved while the robot stays on the path, guaranteeing collision free motion.

6.2 Perspectives

6.2.1 Trajectory Generation

6.2.1.1 NonConstant Kinematic Motion Constraints

Trajectory generation with nonconstant motion constraints is still an open problem. The major limitation of the algorithms described in this thesis is that only constant kinematic motion constraints can be applied to them, that is,

$$\begin{aligned} |{}_jJ(t)| &\leq {}_jJ_{max} = \text{constant} \\ |{}_jA(t)| &\leq {}_jA_{max} = \text{constant} \\ |{}_jV(t)| &\leq {}_jV_{max} = \text{constant}. \end{aligned} \tag{6.1}$$

This algorithm may be confined in the event-based robot control system. The robot must react instantaneously in the moment the event is detected, which may ask the robot follow a trajectory in which one or more elements of the current state of motion exceed the value of the motion constraints.

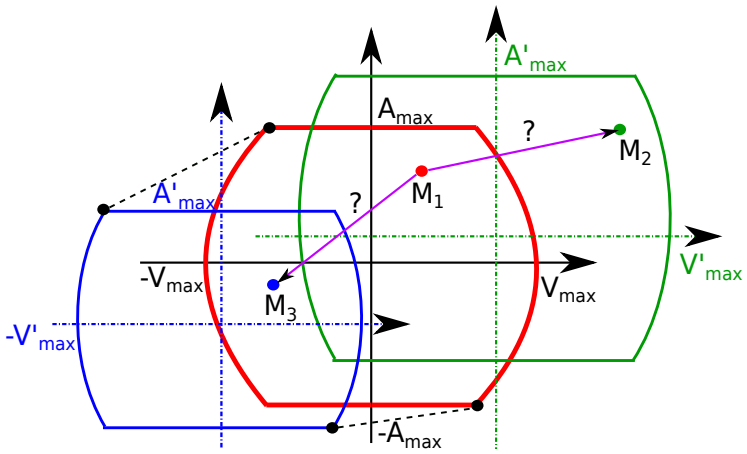


Figure 6.1: Nonconstant motion constraints. The robot at the state M_1 list in the red constraints frame is going to transfer a new state. The states M_2 and M_3 locate in new constraints frames which are denoted as green and blue. The kinematics motion bounds $({}_jJ_{max},\ {}_jA_{max},\ {}_jV_{max})$ are all changed.

Figure 6.1 illustrates two examples. The first one is that the robot has to move from state M_1 to state M_2 due to some reasons, such as to avoid an obstacle. However, the kinematic elements of M_2 has exceeded the current motion constraints. The problem of how to compute a trajectory to connect M_1 and M_2 within one control circle is expected to be solved. The second example happens when the kinematic motion constraints decline, the values of the current motion conditions are greater than the new boundaries. In this case, the trajectory generator proposed in this thesis is not able to produce motions any more. As

in our algorithms, this situation is treated as illegal.

Therefore, an extension to the proposed trajectory generator is necessary to deal with nonconstant motion constraints in real time. The advantage of the nonconstant constraints trajectory generation is that the values of the kinematic motion constraints can be abruptly increased or decreased, such that motion trajectory parameters can be adapted online [Kroger 12a].

6.2.1.2 Considering Robot Dynamics

In this thesis, we only considered the kinematic model of the robot during the trajectory planning procedure. The algorithms presented do not take into account dynamically changing acceleration capabilities based on maximum actuator forces/torques. Although it is already sufficient for many field of applications, it is of a disadvantage in application that requires high-performance robot motions. A large number of off-line trajectory generation approaches consider the system dynamics. [Katzschmann 13] combines the online trajectory generation concept with the robot dynamics to produce trajectories that will bring at least one of the actuators into force/torque saturation are computed instantaneously in the moment unforeseen sensor signals or events happen. As a result, robots can employ their dynamic capabilities immediately and react instantaneously.

Thus in the future work, new OTG algorithms shall be developed to take the robot dynamics and torque limits into consideration. If we consider the robot dynamics to extend our OTG algorithms, the input kinematic constraints ${}_jA_{max}$ can no longer be considered as constant [Kröger 10a]. The values are relative to the forces and torques of the actuators. So this algorithms will have something in common with the one previously presented in section C.5.2.1.

6.2.2 Flexible Controller

One of the advantages of trajectory generation based control is the ability to react to different sensory events and informations. In this thesis, we have shown an example integrated with a force/torque sensor. The challenge is to build a robot around trajectories, so trajectories are expected to be extended to various control systems, such as force tracking control, impedance control, sensor-guided control and sensor guarded control. New methods shall be investigated.

The whole body control of a mobile manipulator is another issue on which we are studying. The motion planner plans path for a robot to follow, including the base and the arms. While synchronized, the robot finishes complex tasks, such as navigation and manipulation in the same time, while avoiding obstacles. The problem requires more study because the navigation of the robot and the motion of the arms should slow down or stop to avoid moving obstacles, and the two should be synchronized. But the dynamic and precision of trajectory following of the robot base and arms are different, hence new strategies shall be proposed to achieve manipulation during navigation.

A

Quaternions and Rotations

Quaternions gives a compact and effective representation for three dimentional rotations. This annexe gives the basics of quaternion, its relations with several other common representations and stops at the perturbations and time-derivatives of quaternions, which are used in this thesis for 6D tracking of object.

A.1 Axis-Angle Representation

The axis-angle representation parametrizes the rotation of a rigid body in a three dimentional space by two values: a unit vector $\boldsymbol{u}$ which defines the direction of rotation, and a rotation angle ϕ the magnitude. Axis-angle is useful to interpolate rotations of rigid body and easy to convert from and to quaternions.

A.2 Definition of Quaternion

Quaternions can be seen as the extension of the complex numbers. A quaternion q is written as:

$$q = q_0 + q_1 i + q_2 j + q_3 k \tag{A.1}$$

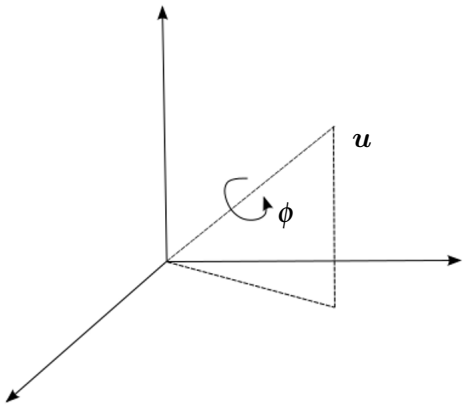


Figure A.1: Axis-Angle Representation

where $q_0, q_1, q_2, q_3 \in \mathbb{R}$, and i, j, k are defined so that:

$$\begin{aligned} i^2 &= j^2 = k^2 = -1 \\ ij &= -ji = k \\ jk &= -kj = i \\ ki &= -ik = j \end{aligned} \tag{A.2}$$

Quaternions can be written in vector representation, simply as:

$$q = q_0 + \overrightarrow{q} = (q_0, \overrightarrow{q}) \tag{A.3}$$

In which $\overrightarrow{q}$ is the imaginary or vector part of quaternion. While complex numbers with unit lengh, written as $z = e^{i\theta}$ can encode rotations in the 2D plane, quaternions of unit length encode rotations in 3D space, although the computation is not as straightforward as for complex numbers. Given the rotation in vector-angle form, $\boldsymbol{v} = \phi \boldsymbol{u}$, a rotation of ϕ rad along the axis given by the unit vector $\boldsymbol{u} = (u_x, u_y, u_z)$, we have the unit quaternion to represent the rotation:

$$q = (cos(\phi/2), \boldsymbol{u} sin(\phi/2)) \tag{A.4}$$

And

$$\phi = arctan(\| q \|, a) \tag{A.5}$$

$$\boldsymbol{u} = q/\| q \| \tag{A.6}$$

The multiplication of two quaternions is defined as:

$$q = \tilde{q} \bigotimes \bar{q} = \tilde{Q}\bar{q} \tag{A.7}$$

In which $\tilde{q} = [\tilde{q_1}, \tilde{q_2}, \tilde{q_3}, \tilde{q_4}]$, $\bar{q} = [\bar{q_1}, \bar{q_2}, \bar{q_3}, \bar{q_4}]$, and $\tilde{Q}$ is the matrix:

$$\tilde{Q} = \begin{bmatrix} \tilde{q_1} & -\tilde{q_2} & -\tilde{q_3} & -\tilde{q_4} \\ \tilde{q_2} & \tilde{q_1} & -\tilde{q_4} & \tilde{q_3} \\ \tilde{q_3} & \tilde{q_4} & \tilde{q_1} & -\tilde{q_2} \\ \tilde{q_4} & -\tilde{q_3} & \tilde{q_2} & \tilde{q_1} \end{bmatrix} \tag{A.8}$$

A.3 Rotation matrix

Rotation matrix is used commonly in numerical computation libraries. Given a rotation vector $\boldsymbol{v}$,

$$\boldsymbol{R} = e^{[\boldsymbol{u}]_\times} \tag{A.9}$$

where the operator $[\bullet]_\times$ is operator defined by:

$$[\boldsymbol{u}]_x \triangleq \begin{bmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{bmatrix} \tag{A.10}$$

Rotation matrix $\boldsymbol{R}$ is then given as:

$$\boldsymbol{R} = \begin{bmatrix} cos(\theta) + u_x^2(1-cos\theta) & u_x u_y(1-cos\theta) - u_z sin\theta & u_x u_z(1-cos\theta) + u_y sin\theta \\ u_y u_x(1-cos\theta) - u_z sin\theta & cos(\theta) + u_y^2(1-cos\theta) & u_y u_z(1-cos\theta) + u_x sin\theta \\ u_z u_x(1-cos\theta) - u_y sin\theta & u_z u_y(1-cos\theta) + u_x sin\theta & cos(\theta) + u_z^2(1-cos\theta) \end{bmatrix} \tag{A.11}$$

Which can be written as:

$$\boldsymbol{R} = \boldsymbol{I} cos\theta + sin\theta[\boldsymbol{u}]_\times + (1-cos\phi)\boldsymbol{u} \bigodot \boldsymbol{u} \tag{A.12}$$

where $\bigodot$ is the tensor product (which is often written as $\bigotimes$, which being used to represent quaternion product in this document), and defined as:

$$\boldsymbol{u} \bigodot \boldsymbol{u} = \begin{bmatrix} u_x^2 & u_x u_y & u_x u_z \\ u_x u_y & u_y^2 & u_y u_z \\ u_x u_z & u_y u_z & u_z^2 \end{bmatrix} \tag{A.13}$$

A.4 Rotations and Compositions

Write a rotation in quaternion as q, and in rotation matrix as R. The rotation applied to a vector $\boldsymbol{x}$ into a new vector $\boldsymbol{x}'$ is given by:

$$\bar{\boldsymbol{x}}' = q \bigotimes \bar{\boldsymbol{x}} \bigotimes q* \tag{A.14}$$

$$\boldsymbol{x}' = R\boldsymbol{x} \tag{A.15}$$

with:

$$\bar{\boldsymbol{x}} = [0 \;\; \boldsymbol{x}^T]^T \tag{A.16}$$

And $q* = (q_0, -q_1, -q_2, -q_3)$ is the conjugate quaternion. The composition of two rotations are straigtforward with the definiton of multiplication of quaternions introduced above:

$$q = \tilde{q} \bigotimes \bar{q} \tag{A.17}$$

$$R = \tilde{R}\bar{R} \tag{A.18}$$

Another useful representation is homogeneous transformation matrix, which is defined as:

$$T = \begin{bmatrix} & & & x \\ & R_{3\times 3} & & y \\ & & & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.19}$$

For two frames, F_2 is obtained by translation $trans = [x, y, z]$ and rotation R from F_1, then the transformation matrix is written as above. If we write the pose of a point in a frame $F1$ as $P_1 = [P_x, P_y, P_z, 1]$, then the pose of this point in frame $F2$ is given as:

$$P_2 = TP_1 \tag{A.20}$$

And transformation matrix has the same composition rule:

$$T = \tilde{T}\bar{T} \tag{A.21}$$

A.5 Perturbations and Derivatives

When using quaternions to represent rotations and build a dynamic model for motion, one important aspect is the computation of perturbations and time-derivatives. Given a quaternion q, and the perturbation written as Δq, expressed in the local body frame. Then the new quaternion can be written as:

$$\tilde{q} = q \bigotimes \Delta q \tag{A.22}$$

The same for rotation matrix:

$$\tilde{R} = R \bigotimes \Delta R \tag{A.23}$$

In the case the perturbation angle $\Delta\theta$ is small ($\Delta\theta$ represents the rotation around an axis $\boldsymbol{u}$), then the perturbation quaternion and rotation matrix can be approximated by the first terms

of the Taylor expansion. Which means:

$$\Delta q = \begin{bmatrix} 1 \\ \frac{1}{2}\Delta\theta \end{bmatrix} + \mathscr{O}(|\Delta\theta|^2) \tag{A.24}$$

$$\Delta R = \boldsymbol{I} + [\Delta\theta]_\times + \mathscr{O}(|\Delta\theta|^2) \tag{A.25}$$

where $\mathscr{O}(|\Delta\theta|^2)$ is the remainder of Taylor expansion. If at time $t = kT$, the rigid body rotation is written as $q = q(t)$, and $\tilde{q} = q(t+\Delta t)$, the derivative of $q(t)$ given as:

$$\frac{dq(t)}{dt} \triangleq \lim_{\Delta t \to 0} \frac{q(t+\Delta t) - q(t)}{\Delta t} \tag{A.26}$$

And writing the angular velocity as $\omega(t)$, expressed in local body frame. The development of the derivative is given as:

$$\begin{aligned}
\dot{q} &\triangleq \lim_{\Delta t \to 0} \frac{q(t+\Delta t) - q(t)}{\Delta t} \\
&= \lim_{\Delta t \to 0} \frac{q \bigotimes \Delta q - q}{\Delta t} \\
&= \lim_{\Delta t \to 0} \frac{Q(\Delta q)q - q}{\Delta t} \\
&= \lim_{\Delta t \to 0} \frac{(\boldsymbol{I} + \frac{1}{2}\begin{bmatrix} 0 & -\Delta\theta^T \\ \Delta\theta & -[\Delta\theta]_\times \end{bmatrix} q + \mathscr{O}(|\Delta\theta|^2))q - q}{\Delta t} \\
&= \lim_{\Delta t \to 0} \frac{\frac{1}{2}\begin{bmatrix} 0 & -\Delta\theta^T \\ \Delta\theta & -[\Delta\theta]_\times \end{bmatrix} q + \mathscr{O}(|\Delta\theta|^2)}{\Delta t} \\
&= \frac{1}{2}\begin{bmatrix} 0 & -\omega^T \\ \omega & -[\omega]_\times \end{bmatrix} q
\end{aligned} \tag{A.27}$$

$$\tag{A.28}$$

The details of this matrix is given as:

$$\Omega(\omega) \triangleq \begin{bmatrix} 0 & -\omega^T \\ \omega & -[\omega]_\times \end{bmatrix} = \begin{bmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & \omega_x \\ \omega_z & \omega_y & -\omega_x & 0 \end{bmatrix} \tag{A.29}$$

B

Computation of Approximations

B.1 Constraints type II

In this constraint type, we define a function $f(X_0, X_F, V_0, V_F, A_0, A_F, T_{imp})$ to compute the parameters. As two segments are needed for the trajectory construction, they can be expressed as:

$$
\begin{aligned}
J_1 &= S_1 * T_1 + J_0 \\
A_1 &= S_1 * T_1^2/2 + J_0 * T_1 + A_0 \\
V_1 &= S_1 * T_1^3/6 + J_0 * T_1^2/2 + A_0 * T_1 + V_0 \\
X_1 &= S_1 * T_1^4/24 + J_0 * T_1^3/6 + A_0 * T_1^2/2 + V_0 * T_1 + X_0 \\
J_2 &= S_2 * T_2 + J_1 \\
A_2 &= S_2 * T_2^2/2 + J_1 * T_2 + A_1 \\
V_2 &= S_2 * T_2^3/6 + J_1 * T_2^2/2 + A_1 * T_2 + V_1 \\
X_2 &= S_2 * T_2^4/24 + J_1 * T_2^3/6 + A_1 * T_2^2/2 + V_1 * T_2 + X_1 \\
[S_1, S_2, T_1, T_2, J_0] &= f(X_F = X_2, V_F = V_2, A_F = A_2, T_1 = Timp/2, T_1 = T_2)
\end{aligned}
$$

By solving the above equations, we can compute the following parameters to construct the trajectories:

$$S_1 = \frac{192X_0 - 192X_F}{Timp^4} + \frac{108V_0 + 84V_F}{Timp^3} + \frac{22A_0 - 10A_F}{Timp^2} \tag{B.1}$$

$$S_2 = \frac{192X_F - 192X_0}{Timp^4} - \frac{84V_0 + 108V_F}{Timp^3} + \frac{22A_F - 10A_0}{Timp^2} \tag{B.2}$$

$$J_2 = -\frac{48X_0 + 18X_F}{Timp^3} - \frac{30V_0 + 18V_F}{Timp^2} + \frac{2A_F - 8A_0}{Timp} \tag{B.3}$$

B.2 Constraints type III

In the constraint type III, we use the same function $f(X_0, X_F, V_0, V_F, A_0, A_F, T_{imp})$ to compute the parameters. The difference to the constraints type II is that one intermediate variable is required during the expression of the trajectory segments.

$$\begin{aligned}
J_1 &= S_1 * T_1 + J_0 \\
A_1 &= S_1 * T_1^2/2 + J_0 * T_1 + A_0 \\
V_1 &= S_1 * T_1^3/6 + J_0 * T_1^2/2 + A_0 * T_1 + V_0 \\
X_1 &= S_1 * T_1^4/24 + J_0 * T_1^3/6 + A_0 * T_1^2/2 + V_0 * T_1 + X_0 \\
J_2 &= S_2 * T_2 + J_{noCont} \\
A_2 &= S_2 * T_2^2/2 + J_{noCont} * T_2 + A_1 \\
V_2 &= S_2 * T_2^3/6 + J_{noCont} * T_2^2/2 + A_1 * T_2 + V_1 \\
X_2 &= S_2 * T_2^4/24 + J_{noCont} * T_2^3/6 + A_1 * T_2^2/2 + V_1 * T_2 + X_1 \\
[S_1, S_2, T_1, T_2, J_0, J_{noCont}] &= f(X_F = X_2, V_F = V_2, A_F = A_2, T_1 = Timp/2, T_1 = T_2)
\end{aligned}$$

J_{noCont} is the intermediate jerk which is a temporary variable during the computation. Similarly as for constraints type II, we solve the associated system as:

$$S_1 = \frac{192X_0 - 192X_F}{Timp^4} + \frac{72V_0 + 120V_F}{Timp^3} + \frac{4A_0 - 28A_F}{Timp^2} \tag{B.4}$$

$$S_2 = \frac{192X_F - 192X_0}{Timp^4} - \frac{120V_0 + 72V_F}{Timp^3} + \frac{4A_F - 28A_0}{Timp^2} \tag{B.5}$$

$$J_2 = -\frac{48X_0 + 48X_F}{Timp^3} - \frac{24V_0 + 24V_F}{Timp^2} + \frac{5A_F - 5A_0}{Timp} \tag{B.6}$$

C

Résumé en Français

C.1 Introduction

C.1.1 Introduction

Les robots interactifs commencent à arriver sur les chaînes d'assemblage et de production. Ils constituent un progrès significatif par rapport à la première génération de robots qui effectuaient des opérations répétitives de manière rapide et précise. L'industrie manufacturière a acquis une grande expérience dans l'exploitation intensive des robots dans des environnements relativement statiques. En raison de leurs avantages par rapport aux opérateurs humains, les bras manipulateurs sont utilisés pour de nombreuses applications industrielles telles que manutention, soudage, usinage, peinture, etc. Mais le développement de cette robotique industrielle classique est confronté à la difficulté de la programmation et au coût des protections destinées à isoler les humains des robots. L'introduction de robots interactifs simplifie la conception des lignes de production et la programmation des robots. La figure C.1 montre un scénario où un opérateur humain coopère avec un bras manipulateur KUKA-LWR pour réaliser une tâche d'assemblage.

C.1.2 Motivation

Pour expliquer la motivation de ce travail, nous proposons une analogie avec le cas de deux opérateurs humains qui doivent réaliser l'assemblage d'un élément sur une ligne de production. Le premier opérateur s'y focaliserait sur l'assemblage proprement dit, tandis que le deuxième opérateur serait chargé de préparer le poste de travail, d'amener les outils

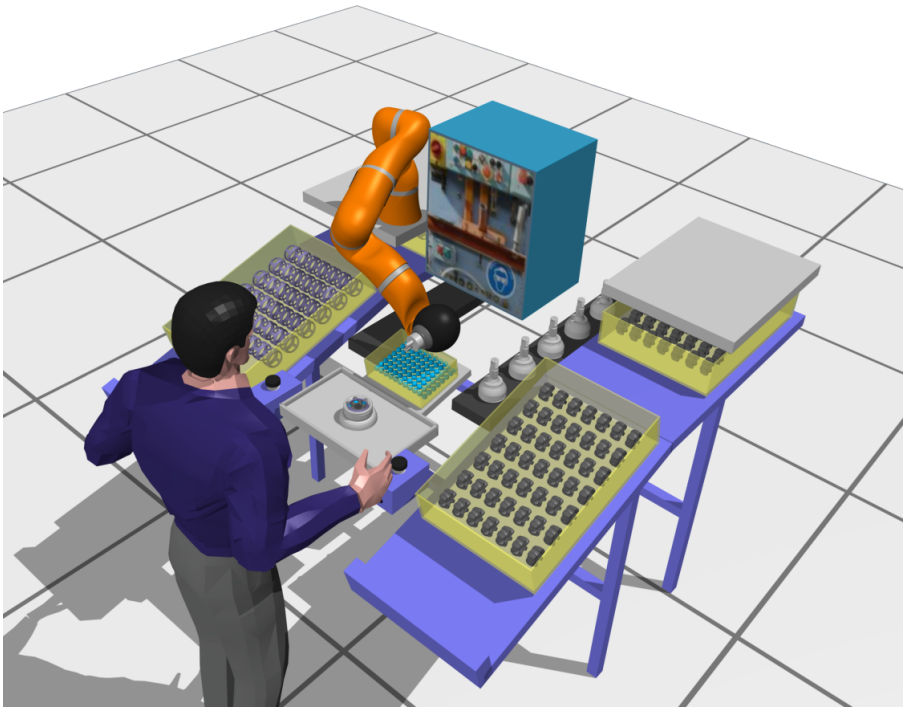


Figure C.1: Le robot manipulateur partage l'espace de travail avec un opérateur humain pour réaliser l'assemblage collaboratif d'un joint Rzeppa. L'illustration provient du projet *ANR-ICARO*.

au bon moment et de maintenir les pièces à assembler pendant l'assemblage.

Pour rester compétitives, les entreprises modernes sont contraintes de limiter le coût de la main d'oeuvre. Dans ce contexte, l'extension des capacités des robots pour leur permettre de travailler avec des humains et de les aider s'avère intéressante. Cette thèse propose de développer un tel robot capable de travailler à coté des humains pour réaliser des tâches en collaboration. Nous proposons notamment de construire un contrôleur plus flexible qui permet de commuter entre plusieurs capteurs et plusieurs lois de contrôle afin d'améliorer la réactivité du système. Pour cela nous proposons d'utiliser intensivement des trajectoires pour échanger des informations entre les composants du robot :planificateur de mouvement, détecteur de collision, système sensoriel (force et vision) et le contrôleur bas niveau du robot.

C.1.3 Objectifs poursuivis

L'objectif de ce travail est d'améliorer l'interactivité des robots collaborants avec des humains par l'introduction d'un contrôleur de trajectoire à un niveau intermédiaire de la structure logicielle du contrôleur. Ainsi, le contrôleur doit fournir une méthode pour générer en temps réel des trajectoires lisses et optimales en temps. Il doit aussi être capable de réagir aux changements de l'environnement. Toutefois, l'algorithme de génération de trajectoire

doit rester simple et les trajectoires doivent prendre en compte les limitations physiques du robot en bornant non seulement la vitesse, mais aussi l'accélération et le jerk.

Ce contrôleur de trajectoire devra accepter en entrée les trajectoires produites par les planificateur de mouvement et pouvoir approximer tous types de trajectoires. Ceci permet d'élargir le type de mouvements que le robot peut réaliser et ainsi réaliser des tâches plus complexes. Le contrôleur de trajectoire proposé est basé sur le concept de Génération de trajectoire en ligne (OTG : Online Trajectory Generation) qui permet de calculer des trajectoires en temps réel. Cette approche basée sur les trajectoires facilite la communication entre les différents composants du robot comme le planificateur de chemin, le générateur de trajectoire, le détecteur de collision et le contrôleur. Ce type contrôleur permet aussi de déformer localement une trajectoire ou de passer d'une trajectoire initiale à une nouvelle trajectoire.

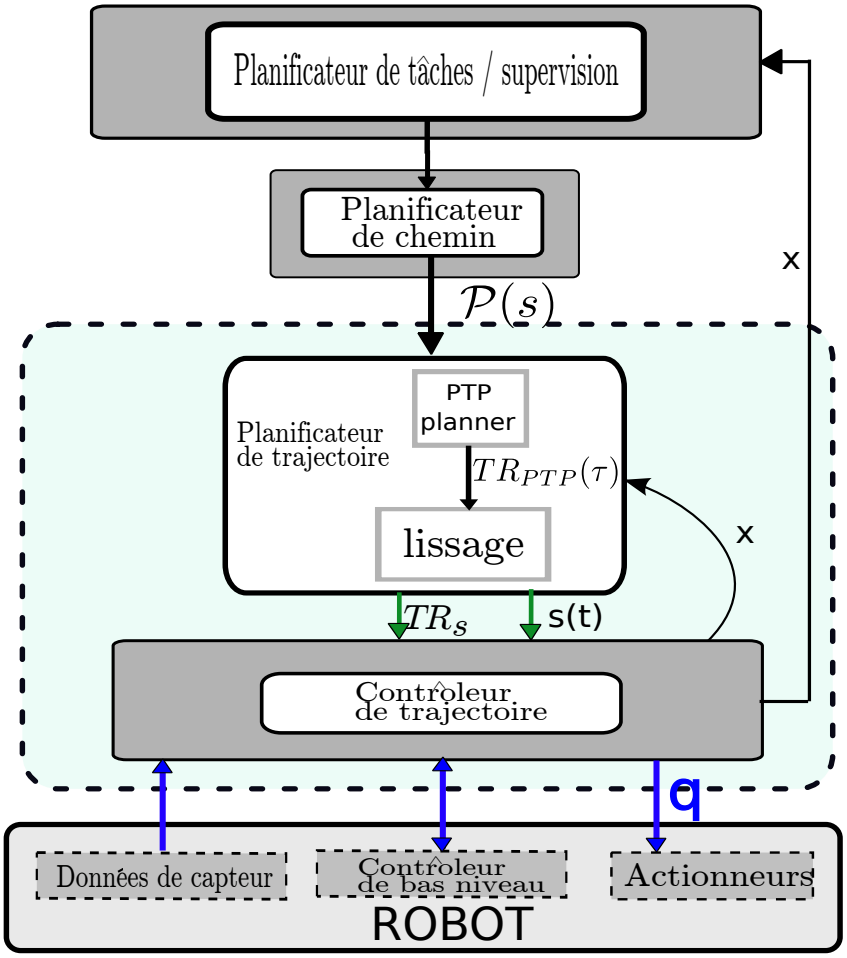


Figure C.2: Le contrôleur de trajectoire se trouve à un niveau intermédiaire de l'architecture du contrôleur, entre les contrôleurs rapides des axes du robots et le niveau planification plus lent.

La figure C.2 illustre le niveau intermédiaire du contrôleur de trajectoire dans l'architecture

du contrôleur. Ce contrôleur intègre des informations provenant d'autres éléments du système comme le raisonnement géométrique ou la planification de mouvements qui prennent en compte la présence des humains. À partir des informations issues des capteurs, le planificateur de trajectoire peut générer des fonctions non-linéaires pour moduler le temps ou replanifier la trajectoire pour réagir à une modification de l'environnement. Un avantage de cette approche est de proposer des contrôleurs plus réactifs à l'évolution dynamique de l'environnement. Un second avantage est La simplicité d'utilisation du contrôleur constitue un deuxième avantage.

C.2 Génération de trajectoire

C.2.1 Introduction

L'introduction d'un générateur de trajectoire à l'intérieur du contrôleur permet au robot de connaître à l'avance une description du proche avenir de ses mouvements. Ces trajectoires permettent au robot d'anticiper différents événements comme les collisions. Cette anticipation donne un avantage important aux couches supérieures d'un contrôleur hiérarchique en leur permettant notamment de mieux assurer la sécurité des êtres humains en interaction. Dans ce chapitre, nous présentons des outils pour définir et manipuler des trajectoires avec l'objectif de construire des contrôleurs de robot multi-niveaux. Nous aborderons plus particulièrement deux types d'outils : les générateurs de trajectoires et l'approximation de trajectoire.

C.2.2 Génération d'une trajectoire à partir d'un chemin

Nous nous intéressons ici à la conversion d'un chemin défini par une liste de points de passage en une trajectoire pouvant être réalisée directement par un robot. Le chemin d'entré $\mathscr{P}$ est fourni par un planificateur de chemin qui peut, par exemple, être de type PRM (Probabilistic Road Map) ou RRT (RApidly Exploring Random Tree).

La première étape de la génération consiste à calculer une trajectoire passant par tous les points de passage du chemin $\mathscr{P}$. Cette trajectoire, que nous appelons $\mathscr{T}_{PTP}$ se compose de mouvements linéaires point à point (Voir section 3.2.2.2) qui s'arrêtent à chaque point de passage.

La méthode de limitation du jerk en ligne par synchronisation de phase que nous proposons permet de résoudre de nombreux problèmes comme assurer la synchronisation de phase, fournir un calcul rapidement compatible avec les applications temps réel, ne nécessite pas de calcul itératif ou d'optimisation et prend en compte les contraintes A vérifier, j'ai modifié. Dans les paragraphes suivants, nous prenons comme exemple le cas de la génération de trajectoires articulaires pour détailler notre approche.

C.2.3 Trajectoires avec phases synchronisées

C.2.3.1 Génération de trajectoire et synchronisation

Notons $\mathscr{C} = \mathbb{R}^n$ l'espace des configurations de dimension n, $q_0 \in \mathscr{C}$ la position initiale et $q_f \in \mathscr{C}$ la position finale des articulations. Une fois la position finale q_f choisie, la trajectoire est générée en bornant les trois premières dérivées des positions articulaires :

$$\begin{aligned} |\dot{q}| &\leq V_{max} \\ |\ddot{q}| &\leq A_{max} \\ |\dddot{q}| &\leq J_{max} \end{aligned} \tag{C.1}$$

où J_{max}, A_{max} et V_{max} sont des paramètres choisis par l'utilisateur pour limiter le jerk, l'accélération et la vitesse. Ces limites sont choisies à partir d'études sur les mouvements acceptables par les humains et les performances des axes contrôlés.

La génération des trajectoires monoaxiales est bien maîtrisée, mais le cas des mouvements multi-axiaux est plus complexe. Dans ce cas, la génération de trajectoires monoaxiales de même durée n'est pas suffisante et il est nécessaire de synchroniser la phase tout au long de la trajectoire afin de définir la forme du chemin. Par exemple, pour générer une ligne droite, le rapport entre la vitesse de chaque axe doit rester constant.

Notre générateur de trajectoire prend en entré les valeurs $[q_0,\ q_f,\ V_{max},\ A_{max},\ J_{max}]$ et génère un profil de jerk bang-bang pour chaque axe $\dddot{q}(t)$. Ce profil de jerk est ensuite intégré trois fois pour obtenir la trajectoire à suivre $q(t)$.

Definition 2. *La synchronisation de phase est la synchronisation en position, vitesse, accélération et jerk. Elle correspond à une progression simultanée de toutes les variables qui, à un instant donné, doivent avoir parcouru le même pourcentage de la trajectoire.*

pour un mouvement linéaire dans un espace de dimension n, le mouvement vérifie :

$$\frac{{}_iq(t) - {}_iq(t_I)}{{}_jq(t) - {}_jq(t_I)} = \frac{{}_iq(t) - {}_iq_0}{{}_jq(t) - {}_jq_0} = \lambda \quad \forall i,j \in [1,n], t \in [t_0,t_f] \tag{C.2}$$

où λ est une constante.

Le facteur λ est calculé simplement à partir des états initial et final :

$$\lambda = \frac{{}_iq(t_f) - {}_iq_0}{{}_jq(t_f) - {}_jq_0} = \frac{{}_iq_f - {}_iq_0}{{}_jq_f - {}_jq_0} \tag{C.3}$$

C.2.4 Génération de trajectoires lisses

Les trajectoires polygonales $\mathscr{T}_{ptp}$ obtenues précédemment sont faisables, mais elles ne sont pas satisfaisantes car au voisinage des points de passage la vitesse varie beaucoup pour

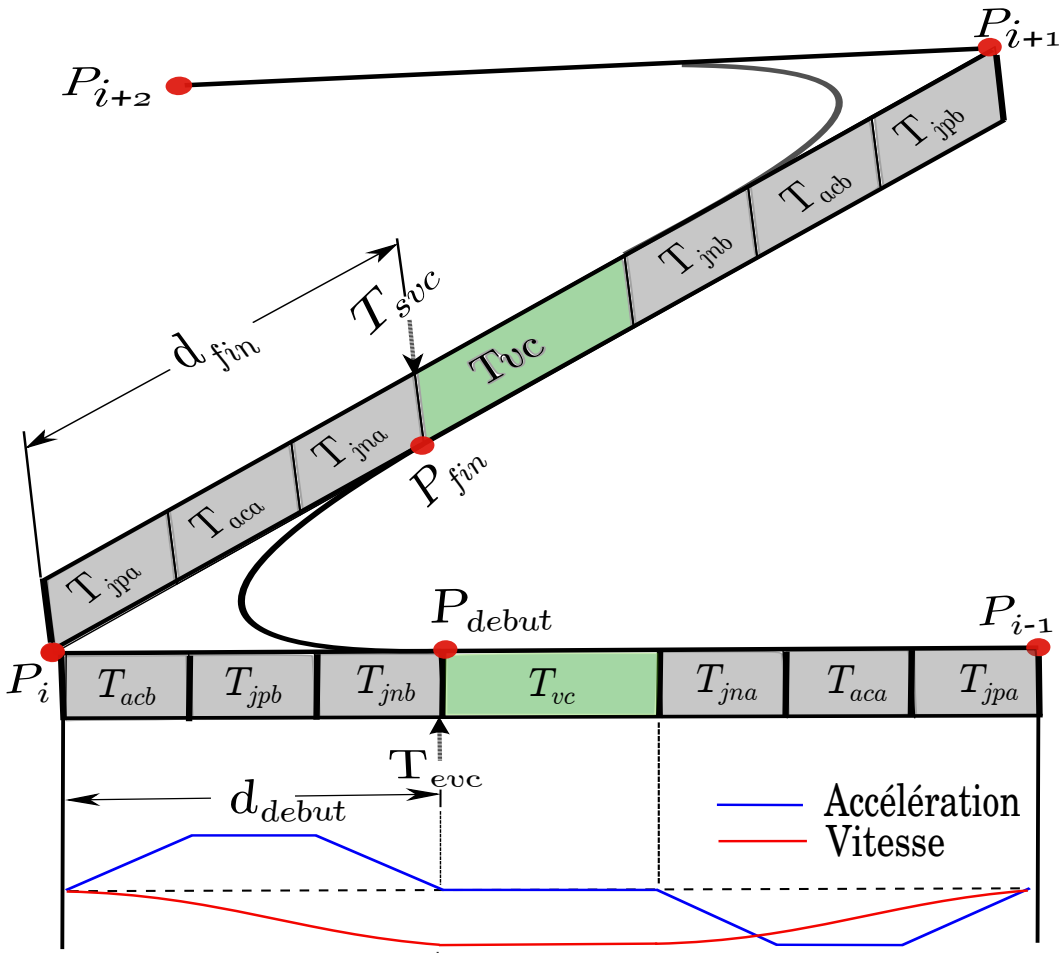


Figure C.3: Influence du choix des points initiaux et finaux délimitant la zone lissée

s'annuler. Ces arrêts peuvent être évité en autorisant la trajectoire à dévier un peu des points de passage, ce qui permet d'arrondir les angles pour adoucir la trajectoire au niveau des points de changement de direction.

Sans perte de généralité, nous considérons trois points adjacents (P_{i-1}, P_i, P_{i+1}) pour étudier le lissage de la trajectoire au niveau du point intermédiaire P_i. Nous calculons d'abord la trajectoire polygonale constitué des deux segments $\mathscr{T}_{P_{i-1}P_i}$ et $\mathscr{T}_{P_iP_{i+1}}$.

Nous choisissons ensuite deux points (P_{debut}, P_{fin}) à partir des distances d_{debut} et d_{fin} au point P_i. Ces deux points définissent la zone lissée qui est donc définie par les deux distances. Le choix de ces deux points est très large et conduit à des trajectoires très différentes, voir figure C.3.

Comme nous souhaitons un mouvement en temps minimal, une première idée simple consiste à conserver au maximum les segments parcourus à vitesse maximum et de lisser la zone qui n'est pas parcourue à vitesse constante. Aussi nous choisissons les deux instants T_{evc} et T_{svc} qui limitent respectivement la fin du segment à vitesse constante $\mathscr{T}_{P_{i-1}P_i}$ et le début du segment à vitesse constante $\mathscr{T}_{P_iP_{i+1}}$ (voir figure C.3).

Dans ce cas, les points P_{debut} et P_{fin} sont respectivement associés avec les états du mouvement $M_{start} = (X_{evc},\ V_{evc},\ A_{evc})$ et $M_{end} = (X_{svc},\ V_{svc},\ A_{svc})$. Comme ces deux états appartiennent chacun à un segment parcouru à vitesse constante, les accélérations A_{evc} et A_{svc} sont nulles.

Nous calculons ensuite le temps de parcours minimal pour chaque axe indépendamment entre P_{debut} et P_{fin} en utilisant la méthode des sept segments proposée par Broquère dans [Broquere 08]. Comme chaque variable est indépendante, le temps d'exécution minimum T_{imp} entre M_{start} et M_{end} est déterminé par le temps mis par le mouvement de l'axe qui prend le plus de temps. Ceci détermine un nouveau problème qui consiste à calculer un mouvement qui dure ce même temps pour chacun des autres $n-1$ axes.

Nous définissons une fonction f qui calcule le temps optimal pour un axe en utilisant la méthode des sept segments. À partir des états initial et final (M_{start} et M_{end}) du mouvement et des bornes cinématiques (J_{max}, A_{max} et V_{max}) pour un axe, la fonction f retourne le temps de transit $f(M_{start},\ M_{end},\ J_{max},\ A_{max},\ V_{max})$ Aussi,

$$T_{imp} = \max_{j\in[1,n]}(f({}_jM_{start}, {}_jM_{end}, {}_jJ_{max}, {}_jA_{max}, {}_jV_{max})) \tag{C.4}$$

T_{imp} définit donc le temps nécessaire pour parcourir le mouvement associé à l'un des axes. Pour chacun des autres axes j, le problème revient à construire un mouvement qui dure le temps prédéfini T_{imp} pour aller de l'état ${}_jM_{start}$ à l'état ${}_jM_{end}$. Il existe de nombreuses solutions, chacune définit un lissage différent. Ici nous en retenons trois qui sont associés à des paramètres de définition de la trajectoire différents.

C.2.4.1 Interpolation par trois segments

A partir des états du mouvement $M_{start} = (X_I,\ V_I,\ A_I)$ et $M_{end} = (X_F,\ V_F,\ A_F)$ définis à l'instant initial t_i et à l'instant final t_f. Une solution simple pour relier ces deux états consiste à définir une séquence de trois segments de trajectoires à jerk constants qui durent T_{imp} en tout. Nous choisissons tris segments car nous souhaitons avoir un minimum de segments et il n'existe généralement pas de solution avec un ou deux segments.

Le système à résoudre est défini par treize contraintes : les états initial et final (six contraintes), la continuité en position, vitesse et accélération pour chacun des deux changements de trajectoire et le temps total. Chaque segment de trajectoire est défini par quatre paramètres et le temps. Si nous fixons la durée des trois segments $T_1 = T_2 = T_3 = \frac{T_{imp}}{3}$, nous obtenons un système défini par treize paramètres où uniquement les trois jerks sont inconnus [Broquère 10]. Comme le contrôleur est périodique de période T, le temps $T_{imp}/3$ doit être un multiple de la période T. Dans cette étude nous choisissons T_{imp} comme un multiple de $3T$.

C.2.4.2 Interpolation par trois segments avec jerk borné

L'interpolation par trois segments permet de générer une trajectoire en temps borné entre deux états. Comme le temps T_{imp} est plus long que le temps minimum nécessaire pour relier les deux états, une solution au problème défini par trois segments existe généralement. Toutefois les jerks obtenus ne sont pas toujours bornés. Nous introduisons ici une variante à la méthode des trois segments en choisissant deux jerks à priori.

Comme nous l'avons vu, le problème est défini par treize contraintes et quinze variables. Nous choisissons ici deux contraintes additionnelles de manière différente en fixant le jerk des premier et dernier segments ($|J_1| = |J_3|$) à l'intérieur des limites cinématiques. Dans ce cas, les inconnus sont le jerk J_2 du segment intermédiaire et les durées des segments. Nous obtenons ainsi un système de quatre équations à quatre inconnus (J_2, T_1, T_2 et T_3) :

$$A_F = J_3 T_3 + A_2 \tag{C.5}$$

$$V_F = J_3 \frac{T_3^2}{2} + A_2 T_3 + V_2 \tag{C.6}$$

$$X_F = J_3 \frac{T_3^3}{6} + A_2 \frac{T_3^2}{2} + V_2 T_3 + X_2 \tag{C.7}$$

$$T_{imp} = T_1 + T_2 + T_3 \tag{C.8}$$

où

$$\begin{aligned} A_2 =& J_2 T_2 + J_1 T_1 + A_I \\ V_2 =& J_2 \frac{T_2^2}{2} + (J_1 T_1 + A_I) T_2 + J_1 \frac{T_1^2}{2} + A_I T_1 + V_I \\ X_2 =& J_2 \frac{T_2^3}{6} + (J_1 T_1 + A_I) \frac{T_2^2}{2} + (J_1 \frac{T_1^2}{2} + A_I T_1 + V_I) T_2 \\ &+ J_1 \frac{T_1^3}{6} + A_I \frac{T_1^2}{2} + V_I T_1 + X_I \end{aligned}$$

Pour choisir les valeurs imposées au jerk, nous regardons les vitesses V_I et V_F. Si $V_I - V_F > 0$ nous choisissons $J_1 = -J_3 = J_{max}$ et si $V_I - V_F < 0$ nous choisissons $J_1 = -J_3 = -J_{max}$. Si $V_I - V_F = 0$ nous regardons de la même manière les valeurs des accélérations A_I et A_F.

C.2.4.3 Interpolation avec jerk, accélération et vitesse bornés

Nous proposons maintenant une méthode pour borner à la fois le jerk, la vitesse et l'accélération. La méthode décrite en C.2.4.2 permet de borner le jerk, mais nécessite de réajuster le jerk pour limiter l'accélération et la vitesse. Lorsque nous calculons les temps minimum pour chaque axe afin de choisir le temps T_{imp}, le jerk, l'accélération ou la vitesse est saturé suivant les segments. Aussi, on peut essayer d'augmenter la duré pour tous les axes à l'exception de

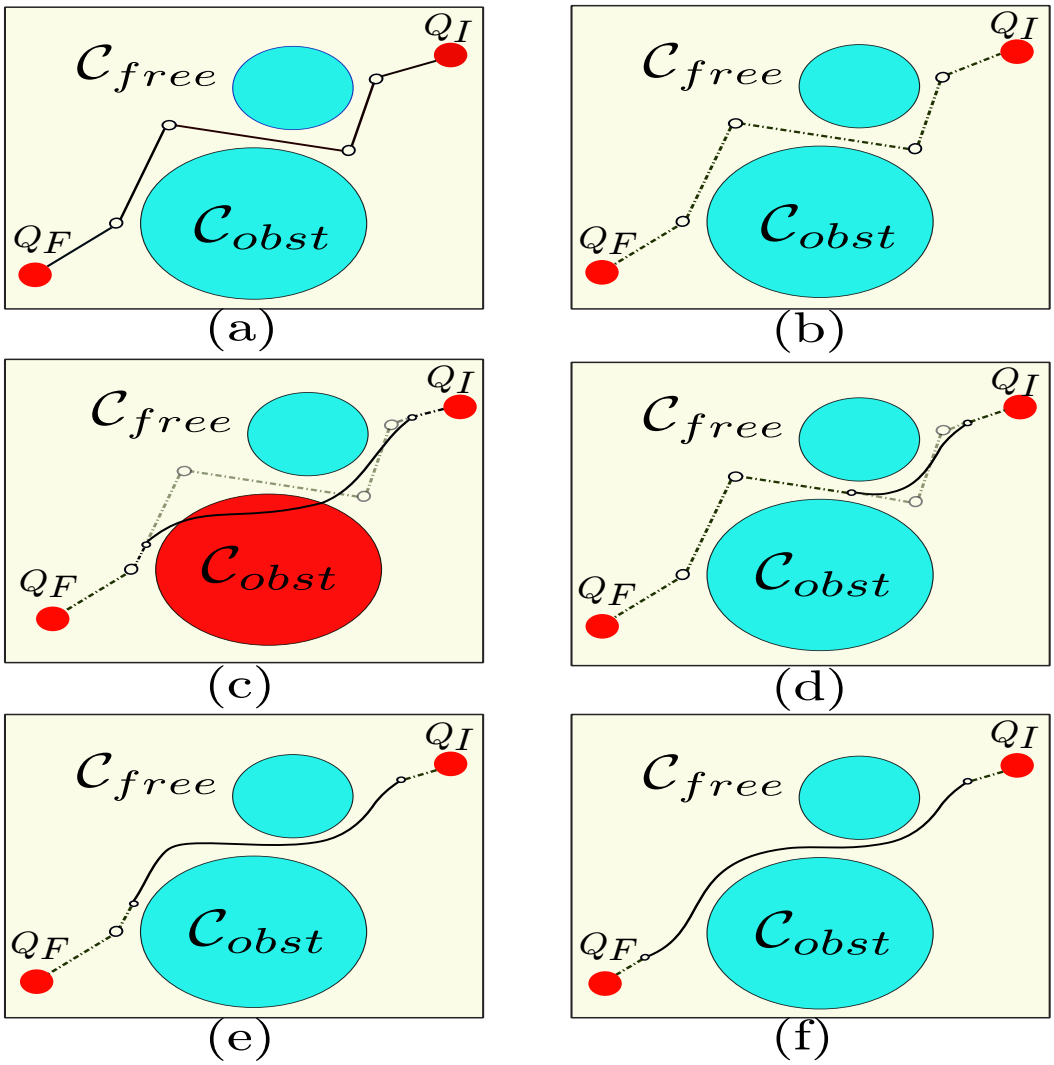


Figure C.4: Algorithme de lissage par raccourci. (a) La ligne polygonale initiale. (b) Conversion en une trajectoire qui s'arrête à chaque point de passage. (c) Une trajectoire plus courte en collision. (d-e) Deux trajectoires plus courtes réussies. (f) la trajectoire finale.

celui associé au temps le plus long T_{imp} en désaturant les éléments saturés tout en conservant le nombre de segment $\mathcal{N}_j$ pour chaque axe. Nous nommons cette méthode *ralentissement du déplacement*.

C.2.5 Lissage par raccourci

Les trajectoires lissées obtenues dans les paragraphes précédents permet d'éviter les arrêts au niveau des points e passage et de diminuer le temps de parcours. Ces méthodes sont utilisées lorsque la trajectoire doit rester au voisinage du chemin initial défini par les points de passage. Toutefois, il nous faut développer de nouvelles solutions pour construire des trajectoires qui paraissent plus naturelles.

Pour produire des mouvements de robot imitant mieux les humains, nous introduisons une variante de la méthode par raccourci couramment utilisé en robotique et en animation graphique. Le procédé utilise d'abord une heuristique pour sélectionner deux points le long de la trajectoire initiale. Ici, nous appliquons simplement une fonction aléatoire uniforme pour sélectionner les deux points. Dans un deuxième temps, nous générons un segment de trajectoire entre les états du mouvement au niveau des deux points. Ensuite, si le segment généré est sans collision, le nouveau segment remplace la partie de la trajectoire initiale. La figure C.4 illustre l'algorithme de lissage et présente quatre itérations de l'algorithme sur un chemin polygonal. La première trajectoire proposée (figure C.4 c) ne satisfait pas au test de collision. Après deux autres tentatives, l'algorithme fourni une solution possible.

C.2.6 Conclusions

Dans ce chapitre, nous avons présenté un générateur de trajectoire synchronisée à partir d'un chemin. Nous avons aussi introduit deux algorithmes de lissage de trajectoires rapides basés sur des fonctions polynomiales de degrés 3 pour des manipulateurs avec de nombreux degrés de liberté. Les principales contributions sont :

1. La proposition d'un algorithme simple et rapide qui opère dans l'espace d'état des configurations, vitesse et accélération.

2. Nous avons proposé une solution analytique optimale en temps qui borne la vitesse, l'accélération et le jerk pour interpoler une trajectoire entre deux états définis par la position, la vitesse et l'accélération. Pour un axe unique, l'interpolation en temps optimal est obtenue sous forme fermée. Dans le cas multi-axes, le système détecte l'axe associé au temps de parcours le plus long, puis interpole les axes restants pour construire des trajectoires bornées en jerk, accélération et vitesse et de durée définie.

C.3 Approximation de trajectoire

C.3.1 Introduction

Un de nos objectifs est de construire des contrôleurs de robots simples et capables de réaliser une grande variété de tâches et, en particulier, des tâches où hommes et robots collaborent.

Comme nous l'avons vu, beaucoup de tâches peuvent être décrite par une trajectoire et une primitive de contrôle. De même, les possibilités mathématiques pour décrire des trajectoires sont illimitées et les modèles obtenus ne sont pas toujours compatibles avec le calcul en temps réel. Aussi des outils d'approximation sont nécessaire pour transformer ces trajectoires en un type de trajectoire que le contrôleur peut accepter en entrée. L'objectif de ce chapitre est de choisir un petit ensemble de modèle de trajectoire et de construire des outils pour approximer tous les types de trajectoire à l'aide de ces modèles.

C.3.2 Différentes solutions d'approximation

Comme nous souhaitons des fonctions de classe C^2, nous devons utiliser des polynômes de degrés supérieur à 2. Un mouvement en temps imposé entre deux états définit sept contraintes : trois conditions initiales (X_0, A_0, V_0) , trois conditions finales (X_F, A_F, V_F) et le temps imposé T_{imp}. Nous allons comparer des solutions d'approximations d'une trajectoire basées sur des fonctions de degrés trois, quatre et cinq.

Comme chaque contrainte de continuité de classe C^2 entre deux segments introduit trois équations de contrainte, l'approximation par des segments de fonctions polynomiales cubiques nécessite au moins trois segments [Broquère 10]. Chaque segment de trajectoire cubique est défini par 5 paramètres.

Fonctions cubiques:

$$X(t) = \frac{J}{6}(t-T_I)^3 + \frac{A}{2}(t-T_I)^2 + V(t-T_I) + X_0 \tag{C.9}$$

Un segment de trajectoire polynomiale quartique est défini par six paramètres, aussi deux segments de trajectoire quartiques sont suffisants pour représenter le mouvement.

Fonctions quartiques:

$$X(t) = \frac{S}{24}(t-T_I)^4 + \frac{J}{6}(t-T_I)^3 + \frac{A}{2}(t-T_I)^2 + V(t-T_I) + X_0 \tag{C.10}$$

Une seule fonction polynomiale de degré 5 est caractérisé par 7 paramètres qui sont donc suffisant pour utiliser un seul segment.

Fonctions quintiques:

$$X(t) = \frac{C}{120}(t-T_I)^5 + \frac{S}{24}(t-T_I)^4 + \frac{J}{6}(t-T_I)^3 + \frac{A}{2}(t-T_I)^2 + V(t-T_I) + X_0 \tag{C.11}$$

C.3.3 Conclusions

Dans ce chapitre, nous avons abordé le problème de l'approximation de trajectoire et la possibilité de construire un générateur de trajectoire simple basé sur l'approximation de trajectoire. Nous avons d'abord proposé trois solutions pour l'approximation de trajectoire par des fonctions polynomiales de degré 3, 4 et 5. Dans chaque cas, nous avons calculé les différents paramètres des trajectoires. L'approche globale peut être résumée comme suit :

- définir la trajectoire comme la somme de K segments de trajectoire.
- Définir chaque segment par une fonction polynomiale de degré k dont les coefficients sont à calculer.
- Calculer ces coefficients à partir des contraintes du mouvement (vitesse, accélération, limites cinématiques, etc.) et en minimisant un critère d'optimalité (erreur d'approximation, nombre de segment, etc.).

- Une fois les coefficients de la trajectoire calculés, ils sont transporté dans l'espace articulaire en utilisant la jacobienne et le modèle géométrique inverse.

C.4 Le contrôle de trajectoire pour le projet ICARO

C.4.1 Introduction

Das ce chapitre, nous avons présenté l'application d'un contrôleur de trajectoire dans un contexte industriel proposé par un partenaire du projet ICARO. Ce scénario est idéal pour le contrôle de trajectoire car un travailleur est au centre de la cellule assemblage où il est nécessaire de contrôler les efforts et d'éviter les obstacle. Le contrôleur de trajectoire permet au robot de réaliser la tâche d'assemblage complète tout en assurant la sécurité de l'opérateur.

C.4.2 Éléments du projet ICARO

C.4.2.1 Détection de gestes

L'objectif de ce module est de remplacer l'utilisation de boutons pour dialoguer avec le robot par des gestes de l'opérateur. L'utilisation de gestes devrait aider l'opérateur a interagir plus facilement avec le robot. Ce module utilise la bibliothèque OpenNI pour créer un graphe de micro-noeuds pour transformer les données brutes en du capteur en un nuage de points, des images de disparité et d'autres éléments utiles pour la visualisation et les calculs.

C.4.2.2 Planification réactive

La planification réactive intervient essentiellement au niveau du noeud de contrôle qui interagit essentiellement avec le planificateur de chemin et les noeuds qui contrôlent la tâche globale. Ces noeuds ROS et celui contrôlant la trajectoires sont regroupés dans la pile *ICARO*. Le développement d'un module pour la planification réactive de chemin dans l'environnement ROS constitue une contribution du groupe SIEMENS au projet ICARO. À partir de la tâche à réaliser et de l'état des collisions, le moniteur de trajectoire demande au module de planification de chemin de produire un chemin qui est transformé en trajectoire par le générateur de trajectoire. Dès qu'une nouvelle trajectoire est produite, le contrôleur de trajectoire commute vers la nouvelle trajectoire.

C.4.2.3 Positionnement de la fusée

La fusée doit d'abord être alignée pour que la direction de ses gorges soit alignées avec la direction du mouvement d'insertion des billes. Nous utilisons pour cela une empreinte imprimé en 3D qui a une forme complémentaire et sur laquelle le robot dépose la fusée avant de la ressaisir dans la bonne orientation. La procédure d'orientation correspond à :

- Descente du robot à partir d'une position située au dessus de l'empreinte jusqu'au contact avec celle-ci. Une variation de la force verticale permet de détecter le contact.
- Le robot fait ensuite tourner la fusée jusqu'à ce que la fusée soit orienté, c'est à dire jusqu'à une diminution de la force verticale associée à une reprise du mouvement de descente jusqu'à une position prédéfinie.
- Le robot ouvre alors la pince et la fusée termine son orientation par gravité.
- Le robot oriente la pince dans la bonne direction.
- le robot ressaisit la fusée.
- Le robot dégage la fusée vers le haut.

C.4.2.4 Tâche d'insertion des billes

L'insertion des six billes d'un joint RZEPPA est une tâche répétitive et complexe pour laquelle l'emploie d'un robot est logique. Comme il est nécessaire de contrôler les forces et les moments durant l'insertion, cette tâche est délicate pour le robot. Le robot ne peut pas être rigides pour pouvoir réagir aux modifications des forces. Aussi nous utilisons un contrôleur de trajectoire basé sur le contrôleur par impédance. Pour ce travail, nous utilisons une méthode qui utilise un générateur de trajectoire en ligne et un contrôleur d'évènement associés à une structure de contrôle qui permet un bon suivi des trajectoires générés et le choix des propriétés d'impédance. Les forces appliqués à l'organe terminal constituent le seul élément pris en compte avec la position pour assurer le contrôle. Le *LIRMM*, partenaire du projet ICARO, a développé le noeud ROS *Coinserter* qui surveille les forces et les moments au niveau de l'organe terminal et calcule les consignes de position et vitesse. Ces forces donnent des informations sur la direction de déplacement souhaité pour l'organe terminal. Comme le noeud *Coinserter* travaille au niveau cartésien à partir du torseur d'effort fourni par le contrôleur en impédance du bras Kuka, le modèle cinématique inverse est nécessaire pour calculer les consignes au niveaux des axes du bras.

C.4.2.5 Surveillance des mains

Pendant la tâche d'insertion des billes présentée dans le paragraphe précédent, les mains de l'opérateur sont proche de la fusée où des risques sérieux de blessure existent. En particulier les doigts de l'opérateur peuvent se faire coincer dans les alvéoles destinées aux billes. Aussi, pour des raisons de sécurité, le mouvement des mains est surveillé par vision durant le mouvement d'insertion des billes. Le mouvement du robot n'est possible que si les doigts de l'opérateur sont à une distance minimale de la fusée.

C.4.3 Conclusion

Nous avons présenté un contrôleur de trajectoire réactif ainsi que quelques résultats relatifs à une application industrielle. Les premiers résultats présentés illustrent la polyvalence du contrôleur basé sur la génération en temps réel de trajectoire. Dans l'exemple présenté, le contrôleur de l'application intègre de nombreux composants comme un planificateur de chemin, un détecteur de collision et un contrôleur de robot. Il est capable de passer d'une trajectoire à une nouvelle et de suspendre puis reprendre le mouvement pour s'adapter à l'activité de l'opérateur.

C.5 Conclusions et perspectives

C.5.1 Conclusions

Ce travail porte sur le développement de la génération de trajectoire en ligne (OTG) pour le contrôle de trajectoire. L'objectif est de construire un robot plus facile à contrôler, en particulier quand le robot interagit avec des humains dans un environnement dynamique. La prise en compte du temps à travers l'utilisation de trajectoires semble complexifier le contrôleur, mais en réalité de nombreuses opérations deviennent plus simples :

- Lissage des chemins : l'introduction des contraintes liées au robot et à la tâche pour générer les trajectoires fournit directement des trajectoires lisse.

- Commutation entre deux trajectoire : l'utilisation du concept de contrôle de primitive par génération de trajectoire en ligne permet de basculer de la trajectoire courante à une nouvelle trajectoire en calculant simplement une trajectoire de connexion en temps réel.

- Réactivité : le contrôleur de trajectoire peut facilement basculer entre les modes de contrôle et changer de trajectoire d'entrée, ce qui rend le système robotique réactif en lui permettant de faire face à des événements imprévus détecté par les capteurs.

C.5.2 Perspectives

C.5.2.1 Contraintes cinématiques variables

La génération de trajectoire avec des contraintes variables constitue encore un problème ouvert. La principale limitation des algorithmes décrits dans cette thèse est qu'ils ne peuvent être appliqués qu'à des systèmes associé à des contraintes cinématiques constantes telles

que :

$$\begin{aligned} |_jJ(t)| &\leq {}_jJ_{max} = \text{constant} \\ |_jA(t)| &\leq {}_jA_{max} = \text{constant} \\ |_jV(t)| &\leq {}_jV_{max} = \text{constant}. \end{aligned} \tag{C.12}$$

Ces algorithmes sont utilisé dans la boucle de contrôle pour réagir aux évènements. Le robot doit réagir instantanément à l'instant où l'évènement est détecté, ce qui peut conduire le robot qui suit une trajectoire à avoir un état courant cinématique qui soit en dehors des limites cinématiques.

C.5.2.2 Prise en compte de la dynamique du robot

Dans cette thèse, nous n'avons considéré que le modèle cinématique du robot pour planifier les trajectoires. Les algorithmes présentés ne tiennent pas compte de l'évolution des capacités dynamiques et en particulier des forces et moments délivrés par les actionneurs au niveau des axes. Même si de nombreuses applications peuvent être réalisées sans prise en compte de la dynamique, elle est nécessaire pour obtenir des mouvements de robots très performants. Dans le futur, le développement d'algorithme d'OTG prenant en compte la dynamique du robot sera nécessaire.

C.5.2.3 Contrôleur flexible

L'ensemble du contrôle du corps d'un manipulateur mobile est une autre question sur laquelle nous étudions. Le planificateur de mouvement prévoit chemin pour un robot à suivre, y compris la base et les bras. Bien synchronisée, le robot se termine tâches complexes, telles que la navigation et la manipulation en même temps, tout en évitant les obstacles. Le problème nécessite une étude plus approfondie, car la navigation du robot et le mouvement des bras devrait ralentir ou d'arrêter pour éviter les obstacles en mouvement, et les deux doivent être synchronisés. Mais la précision dynamique et de suivre la trajectoire de la base du robot et les bras sont différents, donc de nouvelles stratégies doivent être proposées pour atteindre la manipulation pendant la navigation.

C.5.2.4 Contrôleur flexible

Le contrôle du corps complet d'un robot manipulateur mobile constitue une autre question intéressante. Les planificateurs de chemin savent produire des mouvements incluant la base et les bras. Le robot devrait être capable de manipuler et naviguer de manière coordonnée tout en évitant les obstacles. D'autres travaux sont nécessaires pour que la navigation du robot et le mouvement des bras puissent être ralenti ou accéléré pour éviter un obstacle tout en maintenant la coordination. Comme la dynamique et la précision de la trajectoire à suivre

sont différentes pour la base et les bras, de nouvelles stratégies doivent être développées pour réussir à manipuler durant la navigation.

Bibliography

[Ahn 04] Kitak Ahn, Wan Kyun Chung & Youngil Yourn. *Arbitrary states polynomial-like trajectory (ASPOT) generation*. In Industrial Electronics Society, 2004. IECON 2004. 30th Annual Conference of IEEE, volume 1, pages 123–128 Vol. 1, Nov 2004. 16

[Alami 06] Rachid Alami, Raja Chatila, Aurelie Clodic, Sara Fleury, Matthieu Herrb, Vincent Montreuil & Emrah Akin Sisbot. *Towards Human-Aware Cognitive Robots*. In National Conference on Artificial Intelligence, 2006. 38

[Arai 10] T. Arai, R. Kato & M. Fujita. *Assessment of operator stress induced by robot collaboration in assembly*. {CIRP} Annals - Manufacturing Technology, vol. 59, no. 1, pages 5 – 8, 2010. 38

[Argall 09] Brenna D. Argall, Sonia Chernova, Manuela Veloso & Brett Browning. *A Survey of Robot Learning from Demonstration*. Robot. Auton. Syst., vol. 57, no. 5, pages 469–483, May 2009. 21, 25

[Bellman 03] Richard Ernest Bellman. Dynamic programming. Dover Publications, Incorporated, 2003. 22

[Berchtold 94] S. Berchtold & B. Glavina. *A scalable optimizer for automatically generated manipulator motions*. In IEEE/RSJ Int. Conf. on Intel. Rob. And Sys., 1994. 21

[Biagiotti 08] Luigi Biagiotti & Claudio Melchiorri. Trajectory planning for automatic machines and robots. Springer Science & Business Media, 2008. XV, 13, 14, 34, 40

[Bobrow 85] J.E. Bobrow, S. Dubowsky & J.S. Gibson. *Time-Optimal Control of Robotic Manipulators Along Specified Paths*. The International Journal of Robotics Research, vol. 4, no. 3, pages 3–17, 1985. 16

[Bounab 08] B. Bounab, D. Sidobre & A. Zaatri. *Central axis approach for computing n-finger force-closure grasps*. Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on, pages 1169–1174, May 2008. 28

[Brady 82] M. Brady, J. Hollerbach, T. Johnson & T. Lozano-Perez. Robot motion, planning and control. The MIT Press, Cambridge, Massachusetts, 1982. 14, 16

[Broquere 08] Xavier Broquere, Daniel Sidobre & Ignacio Herrera-Aguilar. *Soft motion trajectory planner for service manipulator robot*. Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on, pages 2808–2813, Sept. 2008. 17, 34, 39, 40, 47, 72, 76, 78, 80, 117

[Broquère 10] X. Broquère & D. Sidobre. *From motion planning to trajectory control with bounded jerk for service manipulator robots*. In IEEE Int. Conf. Robot. And Autom., 2010. 34, 40, 48, 67, 68, 117, 121

[Broquère 11] X. Broquère. *Planification de trajectoire pour la manipulation d'objets et l'interaction Homme-robot*. PhD thesis, LAAS-CNRS and Université de Toulouse, Paul Sabatier, 2011. 34, 39, 74

[Buttazzo 94] G Buttazzo, B. Allotta & F. Fanizza. *Mousebuster: A robot for real-time catching*. IEEE Control Systems Magazine, vol. 14(1), 1994. 24

[Calinon 04] S. Calinon & A. Billard. *Stochastic gesture production and recognition model for a humanoid robot*. In Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on, volume 3, pages 2769 – 2774 vol.3, sept.-2 oct. 2004. 25

[Calinon 07] S. Calinon & A. Billard. *Incremental learning of gestures by imitation in a humanoid robot*. In Human-Robot Interaction (HRI), 2007 2nd ACM/IEEE International Conference on, pages 255–262, March 2007. 21

[Cao 94] Bailin Cao & G.I. Dodds. *Time-optimal and smooth joint path generation for robot manipulators*. In Control, 1994. Control '94. International Conference on, volume 2, pages 1122–1127 vol.2, March 1994. 16

[Cao 97] B. Cao, G.I. Dodds & G.W. Irwin. *Constrained time-efficient and smooth cubic spline trajectory generation for industrial robots*. Control Theory and Applications, IEE Proceedings -, vol. 144, no. 5, pages 467–475, Sep 1997. 51

[Cao 98] Bailin Cao, Gordon I. Dodds & George W. Irwin. *A Practical Approach to Near Time-Optimal Inspection-Task-Sequence Planning for Two Cooperative Industrial Robot Arms*. The International Journal of Robotics Research, vol. 17, no. 8, pages 858–867, 1998. 16

[Castain 84] Ralph H. Castain & Richard P. Paul. *An On-Line Dynamic Trajectory Generator*. The International Journal of Robotics Research, vol. 3, no. 1, pages 68–72, 1984. 16

[Chaumentte 06] F. Chaumentte & S.A. Hutchinson. *Visual servo control. Part I: Basic approaches*. IEEE Robotics and Automation Magazine, vol. 4(13), December 2006. 24

[Chaumentte 07] F. Chaumentte & S.A. Hutchinson. *Visual servo control. Part II: Advanced approaches*. IEEE Robotics and Automation Magazine, vol. 1(14), March 2007. 24

[Chwa 05] Dongkyoung Chwa, Junho Kang & Jin-Young Choi. *Online trajectory planning of robot arms for interception of fast maneuvering object under torque and velocity constraints*. Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on, vol. 35, no. 6, pages 831–843, Nov 2005. 16

[Constantinescu 00] D. Constantinescu & E. A. Croft. *Smooth and time-optimal trajectory planning for industrial manipulators along specified paths*. Journal of Robotic Systems, vol. 17, no. 5, pages 233–249, 2000. 16

[Dahl 90] O. Dahl & L. Nielsen. *Torque-limited path following by online trajectory time scaling*. Robotics and Automation, IEEE Transactions on, vol. 6, no. 5, pages 554–561, Oct 1990. 16

[De Luca 08] A. De Luca & L. Ferrajoli. *Exploiting robot redundancy in collision detection and reaction*. In Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on, pages 3299 –3305, sept. 2008. 29

[Duchaine 09] Vincent Duchaine & C. Gosselin. *Safe, Stable and Intuitive Control for Physical Human-Robot Interaction*. In Robotics and Automation, 2009. ICRA '09. IEEE International Conference on, pages 3383–3388, May 2009. 91

[Farrokh 11] Janabi-Sharifi Farrokh, Deng Lingfeng & J.Wilson William. *Comparison of Basic Visual Servoing Methods*. IEEE/ASME Transactions on Mechatronics, vol. 16, no. 5, October 2011. 24

[Gerelli 10] O. Gerelli & C.G.L. Bianco. *A discrete-time filter for the on-line generation of trajectories with bounded velocity, acceleration, and jerk*. In Robotics and Automation (ICRA), 2010 IEEE International Conference on, pages 3989–3994, May 2010. 40

[GmbH 08] KUKA Roboter GmbH. vol. Lightweight Robot 4 Operating Instructions, 2008. 58, 60

[Gosselin 93] G Gosselin, J. Cote & D Laurendeau. *Inverse kinematic functions for approach and catching operations*. IEEE Trans. Systems, Man, and Cybernetics, vol. 23(3), 1993. 24

[Haddadin 08] S. Haddadin, A. Albu-Schaffer, A. De Luca & G. Hirzinger. *Collision detection and reaction: A contribution to safe physical Human-Robot Interaction*. In Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on, pages 3356–3363, Sept 2008. 38

[Hall 63] Edward T Hall. *A system for the notation of proxemic behavior1*. American anthropologist, vol. 65, no. 5, pages 1003–1026, 1963. 18

[Haschke 08a] R. Haschke, E. Weitnauer & H. Ritter. *On-line planning of time-optimal, jerk-limited trajectories*. In Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on, pages 3248–3253, Sept 2008. 16

[Haschke 08b] R. Haschke, E. Weitnauer & H. Ritter. *On-Line Planning of Time-Optimal, Jerk-Limited Trajectories*. In IEEE/RSJ International Conference on Intelligent Robots and Systems, 2008. IROS 2008, pages 3248–3253, 2008. 17, 40

[He 13a] Wuwei He. *Reactive control and sensor fusion for mobile manipulators in human robot interaction*. Theses, Université Paul Sabatier - Toulouse III, October 2013. 25

[He 13b] Wuwei He, Daniel Sidobre & Ran Zhao. *A Reactive Trajectory Controller for Object Manipulation in Human Robot Interaction*. In Proceedings of the 10th International Conference on Informatics in Control, Automation and Robotics, pages 19–28, 2013. 29

[He 15] Wuwei He, Daniel Sidobre & Ran Zhao. *A Reactive Controller Based on Online Trajectory Generation for Object Manipulation*. In Jean-Louis Ferrier, Oleg Gusikhin, Kurosh Madani & Jurek Sasiadek, editeurs, Informatics in Control, Automation and Robotics, volume 325 of *Lecture Notes in Electrical Engineering*, pages 159–176. Springer International Publishing, 2015. XVI, 26, 27

[Herrera 05] I. Herrera & D. Sidobre. *On-line trajectory planning of robot manipulators end effector in cartesian space using quaternions*. In 5th Int. Symposium on Measurement and Control in Robotics, 2005. 87

[Hornung 13] Armin Hornung, KaiM. Wurm, Maren Bennewitz, Cyrill Stachniss & Wolfram Burgard. *OctoMap: an efficient probabilistic 3D mapping framework based on octrees*. Autonomous Robots, vol. 34, no. 3, pages 189–206, 2013. 84

[Howard 09] Matthew Howard, Stefan Klanke, Michael Gienger, Christian Goerick & Sethu Vijayakumar. *A novel method for learning policies from variable constraint data*. Autonomous Robots, vol. 27, no. 2, pages 105–121, 2009. 22

[Ikeura 95] R. Ikeura & H. Inooka. *Variable impedance control of a robot for cooperation with a human*. In Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on, volume 3, pages 3097–3102 vol.3, May 1995. 91

[Ikeura 02] R. Ikeura, T. Moriguchi & K. Mizutani. *Optimal variable impedance control for a robot and its application to lifting an object with a human*. In Robot and Human Interactive Communication, 2002. Proceedings. 11th IEEE International Workshop on, pages 500–505, 2002. 91

[Jaillet 10] L. Jaillet, J. Cortés & T. Siméon. *Sampling-based path planning on configuration-space costmaps*. IEEE Transactions on Robotics, 2010. 20

[Jetchev 11] Nikolay Jetchev & Marc Toussaint. *Task Space Retrieval Using Inverse Feedback Control*. In (ICML 2011), 2011. 22

[Katzschmann 13] R. Katzschmann, T. Kroger, T. Asfour & O. Khatib. *Towards online trajectory generation considering robot dynamics and torque limits*. In Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on, pages 5644–5651, Nov 2013. 102

[Kavraki 96] L.E. Kavraki, P. Svestka, J.-C. Latombe & M.H. Overmars. *Probabilistic roadmaps for path planning in high-dimensional configuration spaces*. Robotics and Automation, IEEE Transactions on, vol. 12, no. 4, pages 566–580, Aug 1996. 33

[Kavraki 08] L.E. Kavraki & S.M. LaValle. Motion Planning In Springer Handbook of Robotics, by B. Siciliano and O. Khatib. 2008. 20

[Khalil 99] W. Khalil & E Dombre. Modélisation identification et commande des robots, volume 56. Hermes, 1999. 14, 16

[Kim 07] Joon-Young Kim, Dong-Hyeok Kim & Sung-Rak Kim. *On-line minimum-time trajectory planning for industrial manipulators*. In Control, Automation and Systems, 2007. ICCAS '07. International Conference on, pages 36–40, Oct 2007. 16

[Kober 13] J. Kober, J. Andrew (Drew) Bagnell & J. Peters. *Reinforcement Learning in Robotics: A Survey*. International Journal of Robotics Research, July 2013. 3

[Kröger 06] T. Kröger, A. Tomiczek & F.M. Wahl. *Towards on-line trajectory computation*. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Beijing, China. Citeseer, 2006. 17

[Kröger 10a] T. Kröger. On-line trajectory generation in robotic systems, volume 58 of *Springer Tracts in Advanced Robotics*. Springer, Berlin, Heidelberg, Germany, first edition, jan 2010. 34, 102

[Kroger 10b] Torsten Kroger & Friedrich M Wahl. *Online trajectory generation: basic concepts for instantaneous reactions to unforeseen events*. Robotics, IEEE Transactions on, vol. 26, no. 1, pages 94–111, 2010. 14

[Kröger 11] Torsten Kröger, Bernd Finkemeyer & FriedrichM. Wahl. Manipulation primitives: A universal interface between sensor-based motion control and robot programming, volume 67 of *Springer Tracts in Advanced Robotics*. Springer Berlin Heidelberg, 2011. 27

[Kroger 12a] T. Kroger. *On-line trajectory generation: Nonconstant motion constraints*. In Robotics and Automation (ICRA), 2012 IEEE International Conference on, pages 2048–2054, May 2012. 102

[Kröger 12b] T. Kröger & Jose Padial. *Simple and Robust Visual Servo Control of Robot Arms Using an On-Line Trajectory Generator*. 2012 IEEE International Conference on Robotics and Automation, 2012. 24

[Kyriakopoulos 88] K.J. Kyriakopoulos & G.N. Saridis. *Minimum jerk path generation*. In Robotics and Automation, 1988. Proceedings., 1988 IEEE International Conference on, pages 364–369 vol.1, Apr 1988. 16

[Lambrechts 04] Paul Lambrechts, M. Boerlage & M. Steinbuch. *Trajectory planning and feedforward design for high performance motion systems*. In American Control Conference, 2004. Proceedings of the 2004, volume 5, pages 4637–4642 vol.5, June 2004. 16

[Larsen 99] E. Larsen, S. Gottschalk, M.C. Lin & D. Manocha. *Fast proximity queries with swept sphere volumes*. 1999. 29

[Lasota 14] P.A. Lasota, G.F. Rossano & J.A. Shah. *Toward safe close-proximity human-robot interaction with standard industrial robots*. In Automation Science and Engineering (CASE), 2014 IEEE International Conference on, pages 339–344, Aug 2014. 2

[LaValle 01a] S. M. LaValle & J. Kuffner. *Rapidly-exploring random trees: Progress and prospects*. In Workshop on the Algorithmic Foundations of Robotics, 2001. 20

[LaValle 01b] Steven M. LaValle & James J. Kuffner. *Randomized Kinodynamic Planning*. The International Journal of Robotics Research, vol. 20, no. 5, pages 378–400, 2001. 11, 33

[LaValle 06] S.M. LaValle. Planning algorithms. Cambridge Univ Pr, 2006. 20

[Lecun 06] Yann Lecun, Sumit Chopra, Raia Hadsell, Fu J. Huang, G. Bakir, T. Hofman, B. Schölkopf, A. Smola & B. Taskar Eds. *A tutorial on energy-based learning*. In Predicting Structured Data, 2006. 24

[Liu 02a] Steven Liu. *An on-line reference-trajectory generator for smooth motion of impulse-controlled industrial manipulators*. In Advanced Motion Control, 2002. 7th International Workshop on, pages 365–370, 2002. 16

[Liu 02b] Steven Liu. *An on-line reference-trajectory generator for smooth motion of Impulse-Controlled Industrial Manipulators*. In 7th International Workshop on Advanced Motion Control, pages 365–370, 2002. 17

[Macfarlane 03] S. Macfarlane & E.A. Croft. *Jerk-bounded manipulator trajectory planning: design for real-time applications*. Robotics and Automation, IEEE Transactions on, vol. 19, no. 1, pages 42–52, Feb 2003. 16

[Mainprice 10] Jim Mainprice, E Akin Sisbot, Thierry Siméon & Rachid Alami. *Planning Safe and Legible Hand-over Motions for Human-Robot Interaction*. In IARP workshop on technical challenges for dependable robots in human environments, volume 2, page 7, 2010. 11

[Mainprice 11] J. Mainprice, E.A. Sisbot, L. Jaillet, J Cortés, T. Siméon & R. Alami. *Planning Human-aware motions using a sampling-based costmap planner*. In IEEE Int. Conf. Robot. And Autom., 2011. 21

[Mainprice 13] J. Mainprice & D. Berenson. *Human-robot collaborative manipulation planning using early prediction of human motion*. In Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on, pages 299–306, Nov 2013. 38

[Mallet 10a] A. Mallet, C. Pasteur, M. Herrb, S. Lemaignan & F. Ingrand. *GenoM3: Building middleware-independent robotic components*. In Robotics and Automation (ICRA), 2010 IEEE International Conference on, pages 4627–4632, May 2010. 10, 58

[Mallet 10b] Anthony Mallet, Cédric Pasteur, Matthieu Herrb, Séverin Lemaignan & Félix Ingrand. *GenoM3: Building middleware-independent robotic components*. In Robotics and Automation (ICRA), 2010 IEEE International Conference on, pages 4627–4632. IEEE, 2010. 60, 78

[Mallet 11] Anthony Mallet & Matthieu Herrb. *Recent developments of the GenoM robotic component generator*. In 6th National Conference on Control Architectures of Robots, page 4 p., Grenoble, France, May 2011. INRIA Grenoble Rhône-Alpes. 10

[M.E 71] Kahn M.E & Roth B. *The Near-Minimum-Time Control Of Open-Loop Articulated Kinematic Chains*. Journal of Dynamic Systems, Measurement, and Control, vol. 93, pages 164–172, 1971. 16

[Meisner 08] Eric Meisner, Volkan Isler & Jeff Trinkle. *Controller design for human-robot interaction*. Autonomous Robots, vol. 24, no. 2, pages 123–134, 2008. 38

[Nikolaidis 13] S. Nikolaidis, P. Lasota, G. Rossano, C. Martinez, T. Fuhlbrigge & J. Shah. *Human-robot collaboration in manufacturing: Quantitative evaluation of predictable, convergent joint action*. In Robotics (ISR), 2013 44th International Symposium on, pages 1–6, Oct 2013. 38

[Owen 04] W.S. Owen, E.A. Croft & B. Benhabib. *Real-time trajectory resolution for dual robot machining*. In Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on, volume 5, pages 4332–4337 Vol.5, April 2004. 16

[Pan 12] Jia Pan, Liangjun Zhang & Dinesh Manocha. *Collision-free and smooth trajectory computation in cluttered environments*. The International Journal of Robotics Research, page 0278364912453186, 2012. 51

[PCL 10] *PCL: Point Cloud Library*. vol. www.pointclouds.org, 2010. 84

[Pomerleau 91] Dean A. Pomerleau. *Efficient training of artificial neural networks for autonomous navigation*. Neural Computation, vol. 3, page 97, 1991. 21

[Quinlan 95] Sean Quinlan. *Real-Time Modification of Collision-Free Paths*. Rapport technique, Stanford, CA, USA, 1995. 55

[Ratliff 06] Nathan D. Ratliff, J. Andrew Bagnell & Martin A. Zinkevich. *Maximum Margin Planning*. In Proceedings of the 23rd International Conference on Machine Learning, ICML '06, pages 729–736, New York, NY, USA, 2006. ACM. 23

[Russell 09] Stuart Russell & Peter Norvig. Artificial intelligence: A modern approach. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009. 22

[Rusu 11] R.B. Rusu & S. Cousins. *3D is here: Point Cloud Library (PCL)*. In Robotics and Automation (ICRA), 2011 IEEE International Conference on, pages 1–4, May 2011. 84

[Rybski 12] Paul Rybski, Peter Anderson-Sprecher, Daniel Huber, Christopher Niessl & Reid Simmons . *Sensor Fusion for Human Safety in Industrial Workcells*. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), October 2012. 38

[Saut 12] Jean-Philippe Saut & Daniel Sidobre. *Efficient Models for Grasp Planning with a Multi-Fingered Hand*. Robotics and Autonomous Systems, vol. 60, March 2012. 28

[Schreiber 10] G. Schreiber, A. Stemmer & R. Bischoff. *The Fast Research Interface for the KUKA Lightweight Robot*. In Proc. of the IEEE ICRA 2010 Workshop on ICRA 2010 Workshop on Innovative Robot Control Architectures for Demanding (Research) Applications - How to Modify and Enhance Commercial Controllers, pages 15–21, 2010. 58

[Schwarzer 04] Fabian Schwarzer, Mitul Saha & Jean-Claude Latombe. *Exact Collision Checking of Robot Paths*. In Jean-Daniel Boissonnat, Joel Burdick, Ken Goldberg & Seth Hutchinson, editeurs, Algorithmic Foundations of Robotics V, volume 7 of *Springer Tracts in Advanced Robotics*, pages 25–41. Springer Berlin Heidelberg, 2004. 55

[Shiller 94] Z. Shiller. *Time-energy optimal control of articulated systems with geometric path constraints*. In Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on, pages 2680–2685 vol.4, May 1994. 16

[Siciliano 08a] B. Siciliano & O. Khatib. Springer Handbook of Robotics. 2008. 15

[Siciliano 08b] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani & Giuseppe Oriolo. Robotics: Modelling, planning and control. Springer Publishing Company, Incorporated, 1st edition, 2008. XV, 12

[Sidobre 12] D. Sidobre, X. Broquère, J. Mainprice, E. Burattini, A. Finzi, S. Rossi & M. Staffa. *Human–Robot Interaction*. Advanced Bimanual Manipulation, pages 123–172, 2012. 27, 34

[Sisbot 07a] E. A. Sisbot, L. F. Marin-Urias, R. Alami & T. Siméon. *Spatial Reasoning for Human-Robot Interaction*. In IEEE/RSJ Int. Conf. on Intel. Rob. And Sys., San Diego, CA, USA, November 2007. 18

[Sisbot 07b] E.A. Sisbot, L.F. Marin-Urias, R. Alami & T. Simeon. *A Human Aware Mobile Robot Motion Planner*. Robotics, IEEE Transactions on, vol. 23, no. 5, pages 874–883, Oct 2007. 10, 38

[Sisbot 10] EmrahAkin Sisbot, LuisF. Marin-Urias, Xavier Broquère, Daniel Sidobre & Rachid Alami. *Synthesizing Robot Motions Adapted to Human Presence*. International Journal of Social Robotics, vol. 2, no. 3, pages 329–343, 2010. 38

[Sisbot 11] E.Akin Sisbot, Raqual Ros & Rachid Alami. *Situation Assessment for Human-Robot Interactive Object Manipulation*. 20th IEEE International Symposium on Robot and Human Interactive Communication, July-August 2011. 26

[Şucan 12] Ioan A. Şucan, Mark Moll & Lydia E. Kavraki. *The Open Motion Planning Library*. IEEE Robotics & Automation Magazine, vol. 19, no. 4, pages 72–82, December 2012. http://ompl.kavrakilab.org. 84

[Tanaka 12] S. Tanaka, Young Min Baek, N. Sugita, T. Ueta, Y. Tamaki & M. Mitsuishi. *Minimum-jerk trajectory generation for master-slave robotic system*. In Biomedical Robotics and Biomechatronics (BioRob), 2012 4th IEEE RAS EMBS International Conference on, pages 811–816, June 2012. 52

[Tonietti 05] G. Tonietti, R. Schiavi & A. Bicchi. *Design and Control of a Variable Stiffness Actuator for Safe and Fast Physical Human/Robot Interaction*. In Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on, pages 526–531, April 2005. 38

[Tsochantaridis 05] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann & Yasemin Altun. *Large Margin Methods for Structured and Interdependent Output Variables*. J. Mach. Learn. Res., vol. 6, pages 1453–1484, December 2005. 24

[Tsumugiwa 02] T. Tsumugiwa, R. Yokogawa & K. Hara. *Variable impedance control based on estimation of human arm stiffness for human-robot cooperative calligraphic task*. In Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on, volume 1, pages 644–650 vol.1, 2002. 91

[Vakanski 12] A. Vakanski, I. Mantegh, A. Irish & F. Janabi-Sharifi. *Trajectory Learning for Robot Programming by Demonstration Using Hidden Markov Model and Dynamic Time Warping*. Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, vol. 42, no. 4, pages 1039 –1052, aug. 2012. 25

[Whitney 69] D. Whitney. *Resolved motion rate control of manipulators and human prostheses*. IEEE Trans. Man-Machine Syst, vol. 10, pages 47–53, 1969. 79

[Wu 09] Wenxiang Wu, Shiqiang Zhu & Songguo Liu. *Smooth joint trajectory planning for humanoid robots based on B-splines*. In Robotics and Biomimetics

(ROBIO), 2009 IEEE International Conference on, pages 475–479, Dec 2009. 52

[xiu Kong 13] Min xiu Kong, Chen Ji, Zheng sheng Chen & Rui feng Li. *Smooth and near time-optimal trajectory planning of robotic manipulator with smooth constraint based on cubic B-spline*. In Robotics and Biomimetics (ROBIO), 2013 IEEE International Conference on, pages 2328–2333, Dec 2013. 52

[Yang 04] Jingzhou Yang, R Timothy Marler, HyungJoo Kim, Jasbir Arora & Karim Abdel-Malek. *Multi-objective optimization for upper body posture prediction*. In 10th AIAA/ISSMO multidisciplinary analysis and optimization conference, volume 30, 2004. 18