

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Institut National Polytechnique de Toulouse (INP Toulouse)

Discipline ou spécialité :

Robotique

Présentée et soutenue par :

M. JOSEPH MIRABEL

le mardi 21 février 2017

Titre :

Manipulation planning for documented objects

École doctorale :

Systèmes (Systèmes)

Unité de recherche :

Laboratoire d'Analyse et d'Architecture des Systèmes (L.A.A.S.)

Directeur de Thèse :

M. FLORENT LAMIRAUX

Rapporteurs :

M. EMMANUEL MAZER, INRIA GRENOBLE - RHONE ALPES

M. FRANK VAN DER STAPPEN, UNIVERSITEIT UTRECHT PAYS-BAS

Membres du jury :

M. JEAN-PAUL LAUMOND, LAAS TOULOUSE, Président

M. FLORENT LAMIRAUX, LAAS TOULOUSE, Membre

M. MICAEL MICHELIN, NEIKER TECNALIA, Membre

Honoré de Balzac

Il y a deux Histoires: l'Histoire officielle, menteuse qu'on enseigne, l'Histoire "ad usum delphini" ; puis l'Histoire secrète, où sont les véritables causes des évènements, une Histoire honteuse.

Remerciements

Les travaux de recherche dont ce manuscrit fait l'objet n'auraient pu voir le jour sans le concours de maintes personnes. Parce qu'il m'est plus facile de leur exprimer ma gratitude dans ma langue maternelle, je préfère employer dans cette page la langue de Molière à celle de Shakespeare.

La première personne que je tiens à remercier est Florent Lamiraux, mon directeur de thèse. Sa disponibilité associée à la confiance qu'il m'a accordé ont sans nul doute contribué à rendre notre travail commun intéressant et fructueux.

Je remercie Emmanuel Mazer et Frank Van Der Stappen de m'avoir fait l'honneur d'être les rapporteurs et membres de Jury de soutenance de cette thèse. Leurs nombreuses remarques et commentaires m'ont permis d'améliorer ce manuscrit d'une part et de prendre du recul quant à ses potentielles applications d'autre part.

Merci à Jean-Paul Laumond et Micael Michelin d'avoir accepté de faire partie du Jury de soutenance.

Je souhaite remercier chaleureusement les anciens et actuels membres de l'équipe GEPETTO. De par les discussions que j'ai pu avoir avec chacun-e, l'équipe a été à tour de rôle un soutien, une source d'inspiration et de motivation ainsi qu'un environnement de travail amical et enrichissant.

Pour finir, un grand merci à toutes les personnes avec qui j'ai vécu durant ces années pour leur soutien, sous quelque forme qu'il fut exprimé. Je pense notamment à ma famille, mes colocataires et Isabelle.

Contents

Contents v

List of Figures List ix

of Tables List of xi

Algorithms xiii

Introduction 1

Problem statement . . . 1
Contributions . . . 2
Chapter organization . . . 3
Publications . . . 4

1 State of the art 5

1.1 Motion planning . . . 5
1.1.1 Other problem formulations . . . 8
1.1.2 Constrained motion planning . . . 9
1.1.3 Motion planning for humanoid robot . . . 12
1.2 Task planning . . . 13
1.3 Manipulation planning . . . 14
1.3.1 Multi-layer manipulation planners . . . 15
1.3.2 Single layer planners . . . 16

2 Constrained motion planning 19

2.1 Notations and definitions . . . 20
2.1.1 Configuration space . . . 20
2.1.2 Constraints . . . 22
2.1.3 Path . . . 24
2.2 Continuous path on manifolds . . . 25
2.2.1 Newton-Raphson algorithm . . . 25
2.2.2 Continuity of the Newton-Raphson iteration function . . . 26
2.2.3 Two path projection algorithms . . . 27
2.2.4 Continuous planning algorithm . . . 30
2.3 Static stability . . . 35
2.3.1 Static stability constraint . . . 35
2.3.2 Integration to a motion planner . . . 37

3 Manipulation planner **41**
3.1 Constraint Graph . . . 42
3.1.1 States and transitions . . . 42
3.1.2 Problem statement . . . 46
3.1.3 Manipulation RRT . . . 47
3.2 Crossed foliation issue . . . 50
3.2.1 Example . . . 50
3.2.2 Conditions . . . 51
3.2.3 Crossed foliation transition . . . 56
3.3 Generalized reduction property . . . 57
3.3.1 Generalized reduction property . . . 57
3.3.2 Grasps and placements . . . 62
3.3.3 Limitations . . . 63
3.4 Narrow passages . . . 63
3.4.1 Low sampling probability . . . 63
3.4.2 Way-point transition . . . 65
3.4.3 Experimental results . . . 66

4 Affordance **69**
4.1 Documented objects . . . 69
4.1.1 Grippers and handles . . . 70
4.1.2 Contact surfaces . . . 71
4.2 Constraint graph generation . . . 73
4.2.1 Building the states . . . 74
4.2.2 Transition detection . . . 75

5 Results **79**
5.1 Humanoid Path Planner . . . 79
5.1.1 Library architecture . . . 80
5.1.2 Results . . . 81
5.2 Manipulator arm . . . 84
5.2.1 Rearrangement planning . . . 84
5.2.2 Tool use inference . . . 85
5.3 Humanoid robots . . . 86
5.3.1 Quasi-static walking motion . . . 86
5.3.2 Romeo holding a placard . . . 90
5.3.3 Grasping behind a door . . . 91

Conclusion **93**
Perspectives . . . 93

A Bound of the Hessian matrix of a kinematic chain **95**
A.1 Notations . . . 95
A.2 Jacobian . . . 95

A.3 Hessian 96
A.3.1 Element of the Hessian matrix 96
A.3.2 Bounds 97

B Static stability constraints **101**

Bibliography **103**

Glossary **111**

Notations **113**

List of Figures

2.1 Path projection example with UR5 robot 20
2.2 Quaternion interpolation 21
2.3 2D illustration of discontinuous path projection 25
2.4 Progressive path projection algorithm 28
2.5 Global path projection algorithm 30
2.6 Path projection results with UR5 robot 33
2.7 HRP2 opening a door 34
2.8 Non-coplanar friction-less multi-contact criterion 35
2.9 Robot HRP-2 quasi-statical stairs climbing 38

3.1 Graph of Constraint for a pick-and-place problem 41
3.2 Illustration of grasp and placement constraints 44
3.3 Example of manipulation path 45
3.4 Steps of the Manipulation-RRT algorithm 48
3.5 Grasp and placement with unique grasp 50
3.6 The crossed foliation issue 51
3.7 Biased projection 53
3.8 Reachability of configurations generated by Manipulation-RRT . . . 54
3.9 Graph of Constraint with crossed foliation transitions 56
3.10 Counter examples to the reduction property 58
3.11 Small Space Manipulability 59
3.12 Narrow passages with a simple example 64
3.13 Way-point transition with a pre-grasp 64
3.14 Way-point transition in the configuration space 65
3.15 Way-point transition with pre-grasp and pre-placement 66
3.16 Benchmark of way-point transition using UR5 66

4.1 Documentation of gripper, handle and contact surfaces 70
4.2 Distance between polygons 72

5.1 Robot programming with HPP 83
5.2 Rearrangement planning with Baxter robot 84
5.3 Constraint graph for Rearrangement Planning benchmark 85
5.4 PR2 robot picks up a box in a drawer 86
5.5 Graph of Constraint to pick up a box in a drawer 87
5.6 Robot HRP-2 quasi-static walk on flat floor 88
5.7 Robot HRP-2 quasi-static stepping motion 88
5.8 Constraint graph for quasi-static walking 89
5.9 Robot Romeo with two continuous grasps. 90
5.10 Robot Romeo puts a box in a fridge. 91

B.1 Schematic model of a legged robot. 101

List of Tables

2.1 Benchmark of path projection algorithms 32
2.2 Benchmark of path projection algorithms with UR5 34
2.3 Benchmark of friction-less multi-contact equilibrium criterion 38

3.1 Benchmark of way-point transitions with UR5 67

4.1 Constraints of way-point states 75
4.2 Way-point transition constraints 76

5.1 Types of joint in the HPP library 80
5.2 Comparison of HPP and OMPL 82
5.3 Benchmark of rearrangement planning with Baxter robot 85
5.4 Benchmark of planning for Romeo holding a placard 90

A.1 Upper bound for $\sigma(v, \chi, \kappa)$. 97
A.2 Upper bound for $\sigma(\omega, \chi, \kappa)$. Omitted combination are null. 98

List of Algorithms

1.1 Constrained-RRT 11
1.2 Standard RRT Extension 12
2.1 Progressive continuous projection 29
2.2 Global continuous projection 31
2.3 Continuous Constrained RRT extension 32
3.1 Manipulation-RRT 47
3.2 Continuous constrained RRT extension using the Constraint Graph . 49
3.3 Connect two configuration with the Graph of Constraint 49
4.1 Generate the constraint graph 74
4.2 Make a state of the constraint graph 75
4.3 Make a transition - Short version 76
4.4 Make a transition - Full version 77

Introduction

Robotic systems first reached the industry in 1962 in a General Motors automobile factory. Since then, their number has not ceased to increase. However, most of these robots are specifically designed for a small set of similar tasks and cannot achieve other tasks. Installing, programming and maintaining them is very costly and they are usually not suited to work with humans for safety reasons. This three facts make them inaccessible to small-sized factories. These factories need affordable robots that can be adapted or programmed for a different task when the production line changes.

Motion and manipulation planning are key areas to reduce these costs. As fields of research, they have raised interest since the last fifty years. Generally speaking, progress in both fields have mostly not yet reached robots in factories. A few recent start-up companies, such as Rethink Robotics, Mujin or Delft Robotics, work on robots with sufficient capabilities to achieve a wide range of tasks and to be reprogrammed to accomplish various tasks sequentially. Also, many robots have been made safer and share the environment with humans. However, their autonomy and robustness to errors is rather limited. Programming them to react to unlikely or unexpected events, to complex tasks still requires a lot of engineering skills when possible.

Problem statement

This thesis addresses the manipulation planning problem with an emphasis on three aspects.

The first aspect concerns the formulation of the manipulation problem. While motion and manipulation planning are rather easy tasks for human beings, the general manipulation planning problem is still out of reach in reasonable time for computers. This thesis explores the idea that, with guidance information provided by a human, the problem becomes accessible for computers. To be useful, it must be an easy task for a human to produce it and to translate it into data a planning algorithm can understand. For instance, human beings easily analyse geometrical information and understand how an object can be grasped or where to release it. To integrate this affordance information in the specification of the manipulation planning problem, the formulation of the problem must model it. In this context, it is a key aspect to understand the structure of the manipulation problem and formulate it as a model comprehensible both by computers and human beings.

The second aspect concerns the resolution of the problem formulated above. Manipulation planning is known to be hard because it combines the search of the sequence of tasks and the motions to accomplish each of these tasks. For instance, to move an object with a robot in a simple scenario, one has to discover a motion to go and pick up the object, a second motion to move it to its goal position while

holding it and a third move the robot. However, the way the robot grasps the object is an internal variable of the problem and is to be decided by the planning algorithm itself. Even in this simple scenario, each task has an infinite number of ways of being achieved and potentially influences all other tasks.

The third and last aspect concerns the variety of robots considered. To be applicable in a wide variety of industrial scenarios, several types of robots ought to be considered. Manipulator arms are the most common. However, cable robots are promising for their large working space. Parallel robots can be both accurate and fast. Mobile manipulators extends the manipulator workspace. Finally, legged robots, such as humanoid robots, have high capabilities of motion. Each of these robots have their own constraint. It is necessary to abstract these constraints to handle these robots in a unified manner.

To summarize, this thesis goals are the following.

- Model the manipulation planning problem which takes into account guidance information. This information is provided as input by a human.
- Propose an algorithm to solve the above problem.
- Abstract the problem so that it handles in a unified manner robots with various capabilities.

Contributions

This thesis tackles the problem stated above both from a theoretical and a practical point of view. It contains four main theoretical contributions and four algorithmic contributions. Their relation with the problem is explained below. Another important outcome is the development of the Humanoid Path Planner (HPP) open-source library.

First, the constrained motion planning problem is addressed. This is a necessary step to express the manipulation planning with constraints. Most state-of-the-art approaches to this problem fail to guarantee the continuity of the solution. Hauser [2013] proposed an algorithm to check for discontinuities. I show that this solution is however not suited to constrained motion planning and I propose the *Progressive path projection* and *Global path projection* algorithms as efficient methods to generate continuous constrained paths. They emerge from a theoretical analyse of the Newton-Raphson algorithm. They rely on a *per iteration continuity condition*, easy to verify in practice. I propose an updated version of the Constrained-RRT [Dalibard et al., 2013] which integrates these algorithms.

Next, the manipulation planning problem is addressed. I consider a problem with robots and objects. The objects can be free-floating, articulated or static. An analyse of the manipulation rules naturally leads to the definition of validation constraint and parametrization constraint. From these definitions, I model the interaction between robots and objects - static or movable - with the *Graph of Constraint*, whose vertices are called states and edges are called transitions. This

model extends and formalizes notions presented by [Dalibard et al., 2010, Berenson et al., 2011, Jentzsch et al., 2015, Hauser and Ng-Thow-Hing, 2011]. I propose the *Manipulation-RRT (M-RRT)* planning algorithm. The Graph of Constraint models the structure of the Cartesian product of the configuration spaces of robots and objects, which allows the M-RRT to plan manipulation paths.

Then, I extend the analyse of the foliated structure of configuration space by Siméon et al. [2004]. In their analyse, they introduced the reduction property which reduces the complexity of the problem of a robot manipulating one free-floating object. I generalize it as the *generalized reduction property* to take into account articulated objects and objects manipulated by two grippers. The generalization relies on the notion of manipulability. An analyse of the configuration space also leads me to identify the *crossed foliation issue*. Randomized manipulation planners that do not handle this issue are unable to solve manipulation problems when two foliations cross each other. This issue had never been identified before because the reduction property allows to bypass it in pick and place scenarios with one object. From a practical perspective, I introduce two new types of transition in the Graph of Constraint. The first is the crossed foliation transition. It models the crossed foliation transition and enable the M-RRT to address the above issue. The second is the way-point transition. It addresses narrow passage issues intrinsic to manipulation problems.

Finally, a simple documentation for robots, and objects is introduced. This documentation provides a geometrical description of possible interactions. Thanks to the documentation, I propose an *algorithm to automatically build a Graph of Constraint*. The graph automatically built is filled with crossed foliation transition and way-point transition where these are required. This means that the simple documentation allows to easily define a manipulation problem in a way that is understood by the M-RRT.

Chapter organization

The chapters of this thesis are organized as follows. Chapter 1 presents the state of the art in motion and manipulation planning. It also briefly presents task planning.

Chapter 2 recalls some mathematical concepts and introduces two algorithms to validate the continuity of constrained path. It proposes a variant of the Constrained-RRT and a method to achieve constrained motion planning in the set of quasi-static configurations.

Chapter 3 describes how the manipulation problem is modelled and defined. It introduces the Graph of Constraint as a way to represent the structure of the configuration space. Then it describes the M-RRT planning algorithm and shows how issues arising from the structure of the configuration space are solved.

Chapter 4 describes a documentation of robots, objects and environments. It introduces an algorithm to automatically generate a Graph of Constraint from this documentation.

Chapter 5 briefly presents the HPP library, as an outcome of this thesis. It also shows some simulations and benchmarks of the presented approach. It has been applied to a wide range of robots in various environments.

Finally, the last chapter points out the conclusions of this thesis and some possible future works.

Publications

Joseph Mirabel and Florent Lamiraux. Manipulation planning: addressing the crossed foliation issue. In *IEEE International Conference on Robotics and Automation (ICRA)*, June 2017.

Joseph Mirabel, Steve Tonneau, Pierre Fernbach, Anna-Kaarina Seppälä, Mylène Campana, Nicolas Mansard, and Florent Lamiraux. HPP: a new software for constrained motion planning. In *IEEE/RSJ Intelligent Robots and Systems (IROS)*, October 2016. 15, 79

Chapter 1

State of the art

Contents

1.1 Motion planning 5
1.1.1 Other problem formulations 8
1.1.2 Constrained motion planning 9
1.1.3 Motion planning for humanoid robot 12
1.2 Task planning 13
1.3 Manipulation planning 14
1.3.1 Multi-layer manipulation planners 15
1.3.2 Single layer planners 16

This chapter provides an overview of existing solutions to the problems related to manipulation planning. Section 1.1 focuses on motion planning and emphasizes the elements which are essential to manipulation planning. Section 1.2 briefly describes the task planning problem. Section 1.3 describes existing approaches to integrate task and motion planning.

1.1 Motion planning

Motion planning consists of finding a collision-free path from an initial robot configuration to a desired goal. It is a continuous geometrical problem. Among the first formulations, the piano's mover problem [Schwartz and Sharir, 1986], recalled in Definition 1.1, shows the case where the robot is a single unarticulated body. The problem has found industrial applications for disassembly problems [Laumond, 2006]. However, motion planning becomes both more interesting and more complex when considering articulated robots.

Definition 1.1 (Piano's mover problem)**.** *Given a robot, a set of static obstacles and an initial and a goal configuration of the robot, find a collision-free path for the robot from the initial to the goal configuration.*

The first main contribution to the problem is the definition of the *Configuration Space*, denoted by $\mathcal{CS}$, introduced by Lozano-Perez [1983]. It is the Cartesian product of the interval of definition of each joint parameter. For a robot with n degrees of freedom (DoFs), $\mathcal{CS}$ is a n-dimensional manifold that contains all the configurations of the robot. It represents the pose of each body of the robot as a

single point so a trajectory of the robot is a continuous path in $\mathcal{CS}$. The problem is thus transformed into finding a trajectory for a point in $\mathcal{CS}$ instead of a trajectory of several bodies in the Euclidean space.

The two following subspaces of $\mathcal{CS}$ are commonly used:

- $\mathcal{CS}_{obs}$: the subset of $\mathcal{CS}$ for which at least one body of the robot collides with an obstacle or with another robot body.
- $\mathcal{CS}_{free}$: the subset of collision-free configurations, *i.e.* $\mathcal{CS}_{free} = \mathcal{CS} \setminus \mathcal{CS}_{obs}$. It is an open set[1].

The motion planning problem can be stated as finding a continuous path $\mathbf{p}(t)$ from a start configuration $\mathbf{p}(0) = \mathbf{q}_{init}$ to a goal configuration $\mathbf{p}(1) = \mathbf{q}_{goal}$ such that $\forall t \in [0,1], \mathbf{p}(t) \in \mathcal{CS}_{free}$.

Three classes of methods tackling this problem can be found in the literature: deterministic approaches, randomized approaches and optimization-based approaches. The reader may refer to the books of Latombe [1991], Choset [2005], LaValle [2006] for a broad overview of existing methods.

Deterministic methods Deterministic methods always compute the same valid solution. Methods such as cellular decomposition, Voronoi diagrams, visibility graphs and Canny's algorithm rely on the construction of an explicit and exact representation of $\mathcal{CS}_{obs}$ to build a graph, or roadmap, that represents the connectivity of $\mathcal{CS}_{free}$ [Canny, 1988, O'Rourke and Goodman, 2004, Indyk and Matousek, 2004]. Other approaches approximate $\mathcal{CS}$ by discretizing it. Motion planning is then reduced to a search in a graph. Although complete, these approaches quickly become intractable as the dimension of $\mathcal{CS}$ increases.

Algorithms based on potential field scale well to high-dimensional configuration space [Khatib, 1986]. However, they are in general not complete because the planner can be trapped in a local minima. Harmonic functions allow to formulate a potential with only one local minimum, which is the global minimum. However, in the general case, such functions are expressed only through a differential equation and the explicit solution is not known.

Sampling based methods Sampling based methods randomly sample $\mathcal{CS}$ and build a graph of configurations connected by collision-free paths. This graph is usually called *roadmap* and approximates the connectivity of $\mathcal{CS}_{free}$. Sampling based algorithms are commonly classified in two groups: single query and multiple query algorithms.

Multiple query algorithms work in two steps. They first build a roadmap that represents as well as possible the connectivity of $\mathcal{CS}_{free}$. Then, motion planning

[1] This supposes that the configurations in contact are in $\mathcal{CS}_{obs}$. Indeed, let $d(\mathbf{q})$ be the smallest distance between bodies in configuration $\mathbf{q}$. $d(\mathbf{q}) > 0 \iff \mathbf{q} \in \mathcal{CS}_{free}$. As d is continuous, $\mathcal{CS}_{free}$ is open.

queries are solved by connecting the initial and final configurations to the previously computed roadmap. If the connectivity of $\mathcal{CS}_{free}$ is well captured during the first step, the queries in the second step are not time consuming. Most popular algorithms are Probabilistic RoadMaps (PRM) [Kavraki et al., 1996] and Visibility PRM (V-PRM) [Siméon et al., 2000]. Instead of PRM, V-PRM tries to identify configurations which add connectivity information to the roadmap. The resulting roadmap is smaller, which makes proximity queries faster.

Single query algorithms do not seek to fully represent the connectivity of $\mathcal{CS}_{free}$. Instead they represent part of $\mathcal{CS}_{free}$ around the initial and, in some case, goal configuration(s). A tree of configurations is grown by iteratively extending the nearest neighbour of a randomly sampled configuration, towards this random configuration. Although it is hard to give a brief summary of the existing algorithms, the most famous family of algorithms is the variants of Rapidly exploring Random Tree (RRT) [Lavalle, 1998].

Randomized algorithms are very efficient in solving high-dimensional problems. However, they suffer the few following drawbacks.

- These algorithms are only *probabilistically complete*. It means that, if a solution exists, the probability of finding one solution converges to 1 as the number of iterations increases. However, if no solution exists, they will run forever without detecting it.

- The solution path contains a lot of erratic motion and is most likely far from optimal. Optimization is usually considered as a post-processing step. Nevertheless, variations that asymptotically converge towards the global minimum solution path have been proposed, like PRM* and RRT* by Karaman and Frazzoli [2011]. These solutions add a step to recompute the shortest path between pairs of configurations in the roadmap. This "rewiring" step is of constant time but nevertheless time-consuming.

- The time required to find a solution quickly increases with the presence of *narrow passages*. This is due to the fact that it is unlikely to randomly sample configurations in those passages. Some techniques prove to be efficient in some cases, such as dimensionality reduction using a Principal Component Analysis [Dalibard and Laumond, 2011] or the bridge test [Hsu et al., 2003].

- Zero volume sub-manifolds cannot be sampled randomly. Random sampling is thus unable to solve problems where the set of feasible configurations is a zero volume sub-manifold of $\mathcal{CS}$. This is for instance the case of closed-loop systems or humanoid robots. Constrained variant exists for most algorithms, such as Constrained-RRT detailed in Section 1.1.2.

Optimization-based methods Optimization based methods formulate motion planning as a trajectory optimization problem. Given a naive initial trajectory, possibly in collision, they iteratively pull the trajectory out of collision while optimizing

a cost. The optimization uses both a user-defined cost - path length, energy... - and collision detection. Collision avoidance is transformed into a constraint of positiveness of the signed distance function. This function is defined as follows. A positive distance corresponds to the shortest distance between non-colliding objects while a negative one corresponds to the penetration between colliding objects.

The two main methods in the literature are Covariant Hamiltonian Optimization for Motion Planning (CHOMP) [Ratliff et al., 2009] and Stochastic Trajectory Optimization for Motion Planning (STOMP) [Kalakrishnan et al., 2011]. CHOMP reduces the overall cost based on covariant gradient information. STOMP generates noisy trajectories to explore the space around the current trajectory. They are then combined to produce an updated trajectory of lower cost. However, STOMP cannot be expected to solve typical motion planning problems like the alpha puzzle in a reasonable amount of time [Kalakrishnan et al., 2011].

1.1.1 Other problem formulations

The above defined motion planning problem does not cover all the situations where motion planning for a robotic system is needed. So some variants of the original motion planning problem have been studied in the literature. Some are briefly presented here.

Optimal motion planning Generally speaking, motion planning does not seek optimal solution but rather feasible solution. Optimization-based methods can only prove local optimality, at the cost of possibly being trapped in infeasible local optimum. However, some algorithms, like PRM* or RRT*, are guaranteed to converge to the globally optimal solution.

Non-holonomic robots Some robotic systems cannot move in all directions. For instance, the velocity of a car is always forward or backward. Its sidewise component is always zero. To handle such cases, one has to design specific local planner, called *steering method*. A steering method links two configurations without considering obstacles. For a car, the optimal steering method was found by Reeds and Shepp [1990].

Achieving a task It is often convenient to specify the goal as a set of robot configurations instead of a single configuration. This set is often called a *task*. For instance, the following is a useful goal set: *robot hand must go to a specific position with respect to the door handle*. This specification is convenient because many configurations can achieve the desired goal. Beyond convenience, reducing the whole set of configurations achieving a goal to a subset of it - one or several configurations - can make a feasible problem infeasible.

Constrained systems For many robotic systems, the set of feasible configurations is a sub-manifold of $\mathcal{CS}$. This is the case for closed-loop system, under-actuated

robots like humanoid robots, cable robots... It is also the case for robots subject to constraints, like equilibrium constraint. Motion planning for these robots falls into constrained motion planning. It is the focus of next section.

1.1.2 Constrained motion planning

For closed-loop systems, humanoid robots or under actuated robots, the feasible configuration space is of the form $\{\mathbf{q} \in \mathcal{CS} \mid \mathbf{f}(\mathbf{q}) = \mathbf{0}\}$, where $\mathbf{f}$ represents the constraint, such as a loop closure constraint, an equilibrium constraint, *etc.* The set of admissible configurations is defined only implicitly and it has measure zero in $\mathcal{CS}$ when $\mathbf{f}$ is non trivial.

Formally, for a constraint $\mathbf{f}$ from $\mathcal{CS}$ to $\mathbb{R}^n$ such that a configuration $\mathbf{q}$ is feasible if and only if $\mathbf{f}(q) = \mathbf{0}$, the motion planning problem becomes finding a continuous path $\mathbf{p}$ such that $\forall t \in [0,1], \mathbf{p}(t) \in \mathcal{CS}_{free}$ and $\mathbf{f}(\mathbf{p}(t)) = 0$.

Most approaches found in the literature tackle only a simpler version of this problem. Berenson et al. [2009], Dalibard et al. [2013] have introduced constrained versions of randomized planners but their algorithms do not guarantee continuity of the solution path.

The following section explains how most methods, not considering continuity, generate admissible configurations and paths. Then, I describe a method proposed by Hauser [2013] to generate continuous constrained paths. The last section describes the Constrained-RRT.

1.1.2.1 Point-wise constraint satisfaction

All constrained motion planning algorithms need a method to generate configurations that satisfy the constraint. In some rare case, the constraint satisfaction problem is explicit. However in most cases, this problem only has an implicit formulation, which is given below.

Definition 1.2 (Constraint satisfaction problem)**.** *Given a constraint error function* $\mathbf{f} : \mathcal{CS} \mapsto \mathbb{R}^n$ *and* $\mathbf{q}_0 \in \mathcal{CS}$*, find* $\mathbf{q} \in \mathcal{CS}$ *such that* $\mathbf{f}(\mathbf{q}) = \mathbf{0}$*.*

For practical reasons, the above problem is often relaxed by using a constraint violation tolerance $\varepsilon > 0$. A configuration $\mathbf{q}$ satisfies the constraints if and only if $||\mathbf{f}(\mathbf{q})|| \leq \varepsilon$.

Stilman [2010] proposes a comparison of several algorithms addressing this problem: Randomized Gradient Descent (RGD), Tangent Space Sampling (TS) and Newton-Raphson (NR) (called First Order Retraction in the paper).

RGD and NR iteratively update the input configuration in order to decrease the constraint violation $||\mathbf{f}(\mathbf{q})||_2$. At each step, RGD randomly samples configurations in the neighbourhood of the current configuration until it has found a configuration with a lower constraint violation. NR uses the Jacobian of the constraint to improve the configuration. TS projects the input configuration onto the tangent space of the constraint at another configuration and then applies RGD.

NR and TS require differentiable constraints. Fortunately, for robotic application, geometric constraints are differentiable and the Jacobian is explicit, as detailed in Appendix A. NR tends to be preferred because it has better a success ratio without being significantly slower [Stilman, 2010]. Moreover, it is a deterministic algorithm so the result is repeatable. It can be written as a *projector*, as defined below.

Definition 1.3 (Projector)**.** *A* projector *on constraint* $\mathbf{f}(\mathbf{q}) = 0$ *is a mapping* P *from a subset* D_P *of* $\mathcal{CS}$ *to* $\mathcal{CS}$ *such that*

$$\forall \mathbf{q} \in D_P, \mathbf{f}(P(\mathbf{q})) = 0$$

This repeatability property of NR is essential. Indeed, non-deterministic algorithms cannot be written as a *projector* so they cannot be continuous with respect to the input configuration.

The continuity of a projector is a sufficient condition to ensure the continuity of a constrained path. However, a projector is often only defined on a subset of $\mathcal{CS}$ and is continuous on a subset of its interval of definition.

1.1.2.2 Constrained paths

Definition 1.4 (Continuous constrained path)**.** *A* continuous constrained path $\mathbf{p}$*, under constraint* $\mathbf{f}$*, is a continuous mapping from* $[0, T]$ *to* $\mathcal{CS}$ *such that*

$$\forall s \in [0, T], \mathbf{f}(\mathbf{p}(s)) = 0$$

Let $(\mathbf{q}_0, \mathbf{q}_1) \in \mathcal{CS}^2$ and $\mathbf{straight}(\mathbf{q}_0, \mathbf{q}_1) : [0, 1] \to \mathcal{CS}$ be the linear interpolation from $\mathbf{q}_0$ to $\mathbf{q}_1$. Most motion planning approaches build paths with basic interpolation method between configurations, *e.g.* $\mathbf{straight}(\mathbf{q}_0, \mathbf{q}_1)$. However, in the constrained case, this is not sufficient. The basic interpolation will most likely not satisfy the constraint, even if the endpoints satisfy it.

A first naive solution to continuous constrained path is to discretize the path, project each sample and use a linear interpolation between the projected samples. The resulting path will be continuous but the point wise projection has two drawbacks. First, the resulting path does not satisfy the constraint between samples. Second, in some cases, it *jumps* between two solution sets that cannot be connected by continuous path satisfying the constraint. In these cases, it gives a wrong solution to a problem which may be infeasible. This last point is explained in full details in Section 2.2.

The Recursive Hermite Projection (RHP) proposed by Hauser [2013] addresses the problem of generating C^1 paths that satisfy a set of non-linear constraints. The basic interpolation between samples is cubic Hermite curve so the velocity can be made continuous. RHP addresses constraints $\mathbf{f}(\mathbf{q}) = 0$ for which there exists a Lipschitz constant M such that $\forall (p, q) \in \mathcal{CS}^2, ||\mathbf{f}(\mathbf{q}_0) - \mathbf{f}(\mathbf{q}_1)|| \leq M||\mathbf{q}_0 - \mathbf{q}_1||$. Note that the norm and the minus operator on $\mathcal{CS}$ will be defined rigorously later. For

a constraint violation threshold $\varepsilon > 0$, the algorithm recursively splits consecutive interpolation points (IPs) in two until the distance between them is less than $\frac{2\varepsilon}{M}$. There are three drawbacks with this approach.

First, for an initial path of length L, it generates at least $\frac{M}{2\varepsilon}L$ points. For the simulations which will be presented in this paper, ε is about 10^{-3} and typically[2] $1 \leq M \leq 10$. Thus the number of points would be greater than $10^3 L$ for a single path. Motion planning extensively uses the steering method. An increase in the time to build a path can be dramatic to the overall performance.

Second, the algorithm assumes the IPs exactly satisfy the constraint, which is of course impossible in practice. One could work around this issue by using two thresholds: a small one to compute accurate IPs and a large one to check for path continuity. This would make the algorithm harder to implement. Because the continuity interval would decrease as the small threshold increases, one would have to make a trade-off between the time to compute accurate IPs and the distance between them.

Third, I consider random exploration of the configuration space. As such, I only consider continuity and not differentiability. I prefer to explore the configuration space of the system and to address differentiability in a post-processing step. When applicable, this approach is known to be more efficient than kinodynamic motion planning that explores the state space of the system and returns differentiable solutions. As such, a slight modification of RHP is required to make it suited for motion planning in $\mathcal{CS}$.

In Section 2.2, I propose a different approach and give a more detailed comparison between my method and the RHP.

1.1.2.3 Constrained-RRT

The Constrained-RRT is very similar to the original RRT so I present directly the constrained version. The unconstrained version can be obtained by considering the trivial constraint $f : q \mapsto 0$.

Algorithm 1.1 Constrained-RRT

```
1: function EXPLORETREE(q_0)
                                  ▷ Constrained-RRT from q_0, s.t. constraint f.
2:     T.INIT(q_0)
3:     for i = 1 → K do
4:         q_rand ← RAND(CS)
5:         q_near ← NEAREST(q_rand, T)
6:         q_proj ← PROJECT(q_rand, f)
7:         EXTEND(q_near, q_proj, T)
```

The idea is to extend a tree of configurations toward randomly sampled points.

[2] M can be computed using the Appendix A.

Pseudo-code is proposed in Algorithm 1.1. The algorithm initializes a tree of configurations $\mathcal{T}$ with the input configuration. It randomly samples $\mathcal{CS}$ and finds the nearest neighbour in $\mathcal{T}$. Such a neighbour always exists since $\mathcal{T}$ is not empty. At Line 6-7, the random configuration is projected as explained in Section 1.1.2.1 and one of the standard RRT extension methods is called. Algorithm 1.2 gives one extension method. It adds to the tree the part of the straight interpolation which is collision-free.

Algorithm 1.2 Standard RRT extension

1: **function** EXTEND($\mathbf{q}_{near}$,$\mathbf{q}_{rand}$,T)
2: $p_1 \leftarrow$ INTERPOLATE($\mathbf{q}_{near}, \mathbf{q}_{rand}$)
3: $p_2 \leftarrow$ TESTCOLLISION(p_1)
4: $\mathbf{q}_{new} \leftarrow$ FINALCONFIGURATION(p_2)
5: $\mathcal{T}$.INSERTCONFANDPATH($\mathbf{q}_{new}$, p_2)

1.1.3 Motion planning for humanoid robot

Full body motion planning for humanoid robot is known to be hard and is almost never addressed directly. The approaches presented below reduce the complexity of the problem, each of them with a different approach.

Decoupled planning Dalibard et al. [2013] addresses the class of problem where the floor is always flat. They show that a collision-free path for a humanoid robot sliding on the ground can always be converted in a dynamically stable walking trajectory. The path of the sliding robot must satisfy the following constraints:

- position and orientation of the feet so that they are always in contact with the floor,
- horizontal position of the Center of Mass (CoM) with respect to the feet,
- constant vertical position of the CoM.

Thanks to this property, planning is divided in two steps. First plan a collision-free path for the humanoid robot subject to the above mentioned constraints. Then generate an admissible walking trajectory following this path.

Multi-contact planning Tonneau et al. [2015] addresses the problem of generating contact plans for multiped robot, such as standing up, climbing stairs using a handrail... They define a reachability condition which verifies that the root configuration of a robot is close enough to allow contact creation, but not too close to avoid collision. The reachability condition turns the high-dimensional computation of finding configuration in contact into a collision checking problem. With this approximation of the space of admissible root configurations, the contact planning

problem is decomposed into two simpler sub-problems. First plan a guide path for the root without considering the whole-body configuration. Then generate a discrete sequence of whole-body configurations in static equilibrium along this path. Although not complete, the approach is extremely efficient to generate contact plan for a wide range of robots. The efficiency comes at the cost of exhaustiveness.

1.2 Task planning

The task planning problem is to find a sequence of elementary actions that accomplish a given task. This section recalls the problem statement and some general notions of task planning domain. Although task planning is not the focus of this thesis, these notions will be useful later. It is also worth while to make some analogies between the two fields, namely motion planning and task planning. Further details can be found in [Fikes and Nilsson, 1971, Hoffmann and Nebel, 2001].

Definition 1.5 (State)**.** *A state S is a finite set of logical atoms.*

Definition 1.6 (STRIPS actions)**.** *A* STRIPS action *o is a triple $o = (pre(o), add(o), del(o))$ where $pre(o)$ are the preconditions of o, $add(o)$ is the add list of o and $del(o)$ is the delete list of the action, each being a set of atoms. For an atom $f \in add(o)$, we say that o achieves f. The result of applying a single STRIPS action to a state is defined as follows:*

$$Result(S, \langle o \rangle) = \begin{cases} (S \cup add(o)) \setminus del(o) & \text{if } pre(o) \in S \\ undefined & \text{otherwise} \end{cases}$$

In the first case, where $pre(o) \in S$, the action is said to be applicable in S. The result of applying a sequence of more than one action to a state is recursively defined as

$$Result(S, \langle o_1, \ldots, o_n \rangle) = Result(Result(S, \langle o_1, \ldots, o_{n-1} \rangle), \langle o_n \rangle).$$

From a set of actions, the goal is to find a sequence of actions that transform, by their addition and deletion lists, an initial set of atoms in a final set of atoms. A rigorous definition is recalled below.

Definition 1.7 (Planning Task)**.** *A planning task $\mathcal{P} = (\mathcal{O}, \mathcal{I}, \mathcal{G})$ is a triple where $\mathcal{O}$ is the set of actions, and $\mathcal{I}$ (the initial state) and $\mathcal{G}$ (the goals) are sets of atoms.*

Definition 1.8 (Plan)**.** *Given a planning task $\mathcal{P} = (\mathcal{O}, \mathcal{I}, \mathcal{G})$. A plan is a sequence $P = \langle o_1, \ldots, o_n \rangle$ of actions in $\mathcal{O}$ that solves the task, i.e., for which $\mathcal{G} \subseteq Result(\mathcal{I}, P)$ holds.*

Many approaches use relaxed task planning to guide the search. The relaxation of a task planning problem is the same problem with the exception that the delete list of actions is empty.

Hoffmann and Nebel [2001] introduced the Fast Forward (FF) heuristic. It uses the relaxed problem to guide their search algorithm called *enforced hill-climbing*. It is a forward search engine, which means it never backtracks and always tries to append actions to the current task plan. Decisions are never revised. The only way to recover from the selection of a *poor action* is to invert the effects of this action. Thus, although it has been successful in many scenarios, the output can contain an arbitrarily large number of *action* and its *inverse action*, being very suboptimal. The approach is complete if the input problem contains no reachable dead-end states [Hoffmann and Nebel, 2001]. A state is a dead-end if and only if no sequence of actions achieves the goal from it. Some realistic criteria, like invertible planning task [Koehler and Hoffmann, 2000, Hoffmann and Nebel, 2001], make many problems dead-end free.

Limitations As a purely symbolic problem, task planning is of rather limited use in robotics. Actions are assumed to be always feasible while the underlying geometrical problem may be conditionally feasible or even infeasible. Indeed, an action such as *Take object on the table* is feasible only if the object is at a reachable position on the table. In the case of continuous grasps and/or placements, it is not possible to represent symbolically each grasp and each placement separately so the case is even harder.

To overcome this limitation, task planning has to be combined with a geometrical motion planner, as described in next section.

1.3 Manipulation planning

Manipulation planning is a combination of task and motion planning.

Definition 1.9 (Manipulation planning problem)**.** *Given a set of robots, objects, static obstacles, an initial configuration $\mathcal{I}$ for all robots and objects and a set of goal configurations $\mathcal{G}$, find a path, for all robots and all objects, from $\mathcal{I}$ to a final configuration in $\mathcal{G}$. The path must be collision-free and must satisfy the* manipulation rules*.*

As a computational geometry problem, it has raised a lot of interest for the past forty years. Pioneering works by Wilfong [1988] and Alami et al. [1989] first considered low dimensional problems where robots and objects move in translation. Dacre-Wright et al. [1992], Alami et al. [1994] are the first works that apply random motion planning methods developed a few years earlier [Kavraki et al., 1996] to the manipulation planning problem. Recently, the domain has regained interest with various variants where papers propose approaches that tackle the inherent complexity of manipulation planning. The domain is traditionally divided into several categories. Navigation Among Movable Obstacles (NAMO) consists of planning a path for a robot that needs to move obstacles in order to reach the goal configuration [Stilman and Kuffner, 2008, Nieuwenhuisen et al., 2008, Dalibard et al., 2010].

Rearrangement planning consists of automatically finding a sequence of manipulation paths that move several objects from initial configurations to specified goal configurations [Krontiris and Bekris, 2015, Ota, 2004, Lertkultanon and Pham, 2015]. Multi-arm motion planning has also given rise to a lot of papers [Gharbi et al., 2009, Harada et al., 2014, Dobson and Bekris, 2015]. From a geometric point of view, manipulation planning is a hybrid problem where discrete states (gripper A holds object B) are defined by continuous constraints on the positions of objects and robots. States are connected by manipulation trajectories that give rise to the underlying structure of a graph whose nodes are the discrete states [Mirabel et al., 2016]. This structure, although not expressed as such, is present in various papers [Dalibard et al., 2010, Berenson et al., 2011, Jentzsch et al., 2015, Hauser and Ng-Thow-Hing, 2011]. The partially discrete nature of the problem has also given rise to integration of task and motion planning techniques [Cambon et al., 2009, Barry et al., 2013, Garrett et al., 2015, Srivastava et al., 2014, Havur et al., 2014].

Considering a set of robots and objects, the problem is to find both a sequence of elementary actions, as well as robots and objects paths for each action, in order to accomplish a set of goals. Compared to task planning, this intends to solve the limitations presented above, *i.e.* the output is guaranteed feasible. The added complexity resides in the fact that solving motion planning problems are time-consuming and most algorithms are only probabilistically complete. They cannot state a problem infeasibility and it is very difficult, if possible at all, to have an estimation of search progress in the general case.

The following sections propose a classifications of manipulation planner in two categories: multi-layer planners and single layer planners. The latter use a unique data structure to organize the data gathered during the search, while the former use a hierarchy of data structure to represent them.

1.3.1 Multi-layer manipulation planners

These planners tackle the problem with two or three planning layers. A high level symbolic planner generates task plans. A low level geometric planner generates paths for elementary actions.

Unfortunately, symbolic and geometric planners do not share the same language. The former reasons on discrete variables while the latter reasons on continuous variables. For instance, a symbolic action, such as *Take object on the table* or *Put object on the table*, does not specify how the object must be grasped or where it must be released. However, the geometric planner needs to know it. Reducing these possibilities to one is not, in the general case, sufficient because the choice is case-dependant. Thus, the challenging issue is the communication between the two levels.

Srivastava et al. [2014] proposes an intermediate layer which refines task plans into motion plan. For each action of the task plan, this layer calls the motion planner for each possible values of the effects of the action, until it has found one which

succeeds. When no motion planning request succeeds, the algorithm backtracks to the previous action. If the refinement fails, a partial plan, up to failure, is found and the algorithm seeks for a possible reason and update the task planner before starting again. For instance, action *Put object on the table* would lead to generate a subset of object pose on the table and try to plan only considering this subset of object pose. If the problem cannot be solved, then a new subset is generated.

Cambon et al. [2009] proposes an multi-layer approach using similar tools as single layer planners. They make use of the reduction property, recalled below, which makes their approach suitable for continuous grasps and placements.

Stilman and Kuffner [2005] addresses Navigation Among Movable Obstacles. The robot is permitted to reconfigure the environment by moving obstacles and clearing free space for a path. A heuristic finds a path without considering collisions with movable obstacles. This paths is used to determine what objects should be moved. Object are moved when they allow two distinct connected components of $\mathcal{CS}$ to merge. Their approach however would not move obstacles which do not block the robot, even if they prevent other movable objects from being moved.

A major issue of multi-layered approaches is that motion planning algorithm cannot prove infeasibility. Most assumes a problem infeasible when some threshold is reached, in terms of number of iterations or elapsed time. Although this is sufficient for many problem, it is a very weak strategy in general and it requires parameter tuning.

Another issue comes from the task planning techniques most approaches use: the FF heuristic or one variant of it. As stated above, the task plan returned by FF can contain an arbitrarily large number of actions and its inverse. In a pick and place scenario,it means the robot can pick up and put down the object several times in an awkward way[3]. From a theoretical point of view, this aspect is not important. However, in practice, this generates awkward and useless motions which makes it not directly usable in practical situations.

1.3.2 Single layer planners

Instead of using layered planners, the problem is represented using the Cartesian product of the configuration space of robots and objects. As this is the framework which will be used in the following, I know recall important notions from the work of Siméon et al. [2004].

The reduction property Consider a problem with a robot and an object. The configuration space of the system is $\mathcal{CS} = \mathcal{CS}_{robot} \times \mathcal{CS}_{object}$. The domain in $\mathcal{CS}$ corresponding to valid placements of the object, *i.e.* stable placements where the object can rest when released by the robot, is denoted by $\mathcal{CP}$. The domain in $\mathcal{CS}$ corresponding to valid grasps of the object by the robot, is denoted by $\mathcal{CG}$. Both $\mathcal{CG}$ and $\mathcal{CP}$ are sub-manifolds of $\mathcal{CS}$.

[3]As, for instance, in the video experiment of Srivastava et al. [2014]: `https://youtu.be/7DUw5L5bx7s`.

A solution to a manipulation planning problem corresponds to a constrained path in $\mathcal{CS}_{free}$. Such a solution path is an alternate sequence of two types of sub-paths verifying the specific constraints of the manipulation problem, and separated by grasp/ungrasp operations.

- *Transit paths* where the robot moves alone while the object stays stationary in a valid placement. They lie in $\mathcal{CP}$. However a path in $\mathcal{CP}$ is not generally a transit path since such path has to belong to the sub-manifold corresponding to a fixed placement of the object. They induce a foliation of $\mathcal{CP}$.

- *Transfer paths* where the robot moves while holding the object with the same grasp. The position of the object with respect to the robot end-effector is constant. They lie in $\mathcal{CG}$ and induce a foliation of $\mathcal{CG}$.

For completeness, I recall the notion of foliation [Haefliger, 1970].

Definition 1.10 (Foliation)**.** *A foliation of a n-dimensional manifold M is an indexed family L_α of arc-wise connected m-dimensional sub-manifolds $m < n$, called leaves of M, such that:*

- $L_\alpha \cup L_{\alpha'} = \emptyset$ *if* $\alpha \neq \alpha'$
- $\cup_\alpha L_\alpha = M$
- *every point in M has a local coordinate system such that $n - m$ coordinates are constant.*

An example of discrete manipulation problem is navigation inside a building, made of floors and staircases. To move from one floor to another, you must navigate in a staircase. To move from one staircase to another, you must navigate in a floor. Allowed motions induce a foliation of the set of floors, in which each floor is a leaf, and a foliation of the set of staircases. Navigating in this space requires to find an alternate sequence of floors and staircases.

Two foliation structures are defined in $\mathcal{CG} \cap \mathcal{CP}$, which implies the following property, shown by Dacre-Wright et al. [1992].

Theorem 1.1 (Reduction property)**.** *Any path lying in $\mathcal{CG} \cap \mathcal{CP}$ where the robot is not in collision with static obstacles can be transformed into a finite sequence of* transit *and* transfer *paths.*

This property reduces the manipulation problem to a problem of discovering the connectivity of the various components of $\mathcal{CG} \cap \mathcal{CP}$ by *transit* and *transfer* paths. They provide two multiple-query algorithms based on PRM.

This result is the main theoretical result in manipulation planning. To my best knowledge, there has been no significant theoretical breakthrough about the geometrical problem since.

Other approaches Another successful algorithm is FFrob, introduced by Garrett et al. [2015]. It is an extension of the Fast Forward heuristic that accounts for geometrical information. A set of useful object poses and robot configurations is sampled offline and stored in a conditional reachability graph. When the planner failed to solve a problem, new object poses and robot configurations are sampled and the graph is updated. Thanks to this offline computations, FFrob is able to solve challenging problems in a reasonable amount of time. However, many parameters are to be tuned. Most of them are very case specific such as the number of sampled object poses, the number of grasp configurations, the number of iterations of the RRT to solve a motion planning problem...

The Diverse Action RRT (DA-RRT) algorithm, proposed by Barry et al. [2013], tackles the Diverse Action Manipulation problem. The inputs to the problem are a mobile robot, a set of movable objects, and a set of diverse, possibly non-prehensile manipulation actions. This algorithm finds a high-level sequence of transfer manipulations by planning a path only for objects in the domain. It then attempts to achieve each transfer manipulation individually. A similar approach to this one is Sampling-based Motion and Symbolic Action Planner, introduced by Plaku and Hager [2010]. They grow a tree of configurations by sequentially applying actions. The action applied is selected using a measure of the utility of actions.

Chapter 2

Constrained motion planning

Contents

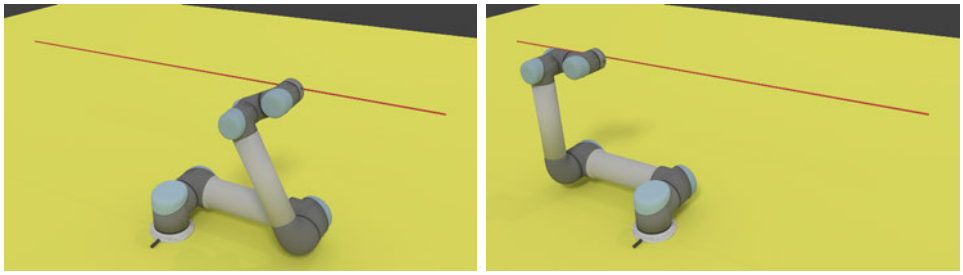
2.1 Notations and definitions 20
2.1.1 Configuration space 20
2.1.2 Constraints 22
2.1.3 Path 24
2.2 Continuous path on manifolds 25
2.2.1 Newton-Raphson algorithm 25
2.2.2 Continuity of the Newton-Raphson iteration function 26
2.2.3 Two path projection algorithms 27
2.2.4 Continuous planning algorithm 30
2.3 Static stability 35
2.3.1 Static stability constraint 35
2.3.2 Integration to a motion planner 37

This chapter addresses important issues raised by constrained planning and manipulation planning. Manipulation planning relies on constrained motion planning, with the additional difficulty that the constraints are not the same in the whole configuration space. Manipulation paths alternates between different constraints.

The first section presents the formalism necessary to address problems addressed in this thesis. It includes basic operations on the configuration space and the velocity space, two types of constraints, naturally emerging from the analysis of manipulation rules, and continuous paths.

The second section presents two algorithms with continuity certificate and a proof of this certificate. As explained in the previous chapter, most approaches addressing constrained motion planning do not guarantee the continuity of the solution. I also propose continuous constrained motion planning algorithm. This two algorithms constitute a major contribution of this thesis.

In the last section, I formulate the problem of generating quasi-static motion for a humanoid robot as a manipulation problem. I express a friction-less multi-contact equilibrium criterion as a constraint, which enables to plan full-body motion in the sub-manifold of quasi-static configurations.



(a) Initial configuration (b) Goal configuration

Figure 2.1: Path projection example with UR5 robot. UR5 end-effector must follow the red line with constant orientation. Planning in configuration space with pointwise path projection can generate discontinuous path.

2.1 Notations and definitions

I consider a set of robots R, a set of movable objects M and a set of static obstacles. The configuration space $\mathcal{CS}$ always denotes the Cartesian product of the configurations space of each robot and object. This means I always consider that robots and objects are one single kinematic tree.

2.1.1 Configuration space

In general, $\mathcal{CS}$ is a manifold and not a vector space. The velocity is defined as the derivative of the joint parameters with respect to time or any other abscissa.

For one dimensional bounded rotations and for translations, the configuration space is a vector space and the velocity space is identical to the configuration space. In this context, unbounded rotations are represented by the circle group $SO(2)$, *i.e.* $\mathbf{q} = (\cos\theta, \sin\theta)$, to overcome issues due to the multiplicity of angles representing the same rotation. The velocity space is $\mathbb{R}^1$ and velocities are defined by

$$\frac{\partial \mathbf{q}}{\partial s} = \frac{\partial \theta}{\partial s}$$

Three dimensional rotations are represented by with unit quaternions, described by the unit sphere in $\mathbb{R}^4$. This representation is non-singular, which is not the case of Euler angles, and more compact than matrices. In the following, I give some brief elements about the special orthogonal group $SO(3)$ derived from formal Lie group theory which is not described here. The user may refer to Kirillov [2008]. The velocity space is $\mathbb{R}^3$ and the velocity ω is defined by

$$[\omega]_\times = \frac{d\mathbf{R}}{dt}\mathbf{R}^T$$

where $\mathbf{R}$ is the rotation matrix.

Both unbounded rotations and three dimensional rotations have a velocity space whose dimension is lower than their configuration space.

2.1.1.1 Operations

In order to treat each joint configuration space uniformly, two operations are defined in $SO(2)$ and $SO(3)$: an *addition* between a joint configuration and a joint speed and a *difference* between two joint configurations. These operators extend the addition and difference on joint configuration spaces which are vector spaces.

Operations in $SO(2)$ Let $(\mathbf{p}_1, \mathbf{p}_2) \in SO(2)^2$ and θ_i such that $\mathbf{p}_i = (\cos\theta_i, \sin\theta_i)$.

The difference $(\theta) = \mathbf{p}_2 - \mathbf{p}_1 \in]-\pi, \pi]$ is such that $\theta \equiv \theta_2 - \theta_1 + 2k\pi, k \in \mathbb{Z}$. The addition of velocity (θ) to joint configuration $\mathbf{p}_1$ is the rotation obtained by integrating velocity (θ) during unit time, *i.e.* $\mathbf{p}_1 + (\theta) \equiv (\cos(\theta_1 + \theta), \sin(\theta_1 + \theta))$.

Operations in $SO(3)$ Let (q_1, q_2) be two quaternions representing two elements of $SO(3)$. Because of the double covering of quaternions, q_i and $-q_i$ represents the same element of $SO(3)$. As the joint configuration space is $SO(3)$, there is no reason to abandon this duplicity. The two operations defined below consider this duplicity when necessary.

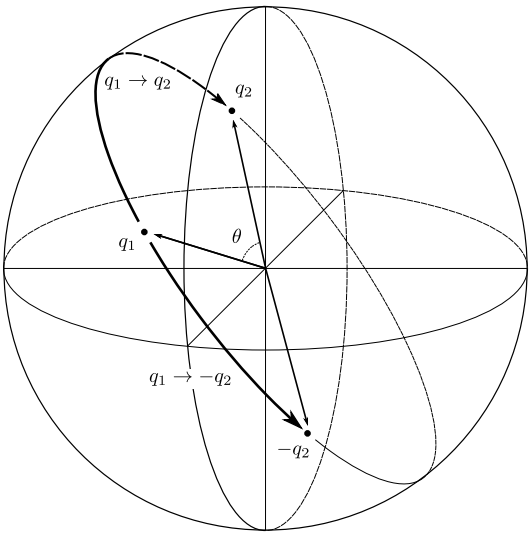


Figure 2.2: Schematic representation of the quaternion interpolation. It represents the unit sphere of $\mathbb{R}^3$. q_1 and q_2 are two elements of the sphere. $-q_2$ represents the same rotation as q_2. When $cos(\theta) \geq 0$, path $q_1 \to q_2$ is shorter than path $q_1 \to -q_2$. When $cos(\theta) < 0$, path $q_1 \to -q_2$ is shorter.

The difference $\omega = q_2 - q_1 \in \mathbb{R}^3$ is defined as the speed to go in unit time from q_1 to q_2. It corresponds to a rotation of $||\omega||$ around $\omega/||\omega||$. A path from q_1 to q_2 and from q_1 to $-q_2$ is of different length in general. This is depicted on Figure 2.2. For instance, if $q_1 = q_2$, the former is of angle 0 while the latter is of angle 2π. The difference operation chooses the one with smallest angle. Let $\otimes$ be the quaternion product and $\overline{q_1}$ be the conjugate of quaternion q_1. Finally, let $q = q_2 \otimes \bar{q_1} = (w, \mathbf{v})$, where $w \in \mathbb{R}$ is the scalar part of q and $\mathbf{v} \in \mathbb{R}^3$ is the vector part of q. It can be defined as:

$$\omega \equiv 2\frac{\mathbf{v}}{||\mathbf{v}||} \times \begin{cases} \arccos(w) & \text{, if } w \geq 0 \\ -\arccos(-w) & \text{, otherwise} \end{cases}$$

The addition of velocity ω to configuration q is the rotation obtained by inte-

grating velocity ω during unit time, *i.e.*

$$q + \omega = \left(\cos\frac{||\omega||}{2}, \sin\frac{||\omega||}{2}\frac{\omega}{||\omega||}\right) \otimes q$$

These operations are continuous on $SO(3)$, although they are not on the unit sphere of $\mathbb{R}^4$. Moreover, the difference operation gives geodesics in $SO(3)$. However, one has to be careful with the arithmetic. Indeed, $q_2 - q_1 = q_2 - (-q_1)$ so $q_1 - (-q_1) = 0_{\mathbb{R}^3}$. The quaternion $q_1 + (q_2 - q_1)$ is either q_2 or $-q_2$ depending whether their scalar product in $\mathbb{R}^4$, $\langle q_1, q_2 \rangle$, is positive or negative.

Operations in $\mathcal{CS}$ The above operations are extended to the robot configuration space. The difference in each joint configuration space defines a difference in the robot configuration space. Similarly, the addition operation between a robot configuration and a robot velocity is defined.

I define the two following operations.

- distance $\mathbf{dist}_D(\mathbf{q}_0, \mathbf{q}_1) = ||\mathbf{q}_1 - \mathbf{q}_0||_D = (\mathbf{q}_1 - \mathbf{q}_0)^T D(\mathbf{q}_1 - \mathbf{q}_0)$ is the weighted norm of the difference. The weight D is a square matrix of dimension $n \times n$, where n the size of the velocity space.

- straight interpolation from $\mathbf{q}_0$ to $\mathbf{q}_1$,

$$\mathbf{straight}(\mathbf{q}_0, \mathbf{q}_1) : \begin{array}{ll} [0,1] & \to \mathcal{CS} \\ s & \mapsto \mathbf{q}_0 + s\,(\mathbf{q}_1 - \mathbf{q}_0) \end{array} \tag{2.1}$$

  The difference operation in $SO(3)$ means the straight interpolation is the path of shortest length.

2.1.1.2 Jacobian

The Jacobian of the robot with n degrees of freedom (DoFs) is a $6 \times n$ matrix. The matrix maps the velocity in configuration space to velocities in Euclidean space. The three top rows correspond to the linear part and the three bottom ones correspond to the angular part. The columns correspond to DoFs. For instance, unbounded rotations are represented by one column while three dimensional rotations span three columns. Appendix A gives the analytical expressions.

2.1.2 Constraints

This section defines two types of constraints useful in manipulation planning. To picture these types, consider an end-effector manipulating an object. There are two possible states for a configuration. Either the end-effector grasps the object in a stable way or the object is in a stable placement. They constitute one type of constraint, called *validation constraint*. They are immutable, *i.e.* they depends only on inputs of the planning problem.

There are two possible states for a path. Either the robot moves the object or the object does not move. In the first case, each configuration of the path is a valid grasp configuration and the object pose relatively to the end-effector is constant. The grasp is said constant. In the second case, the object remains still, in a valid placement. The placement is said constant. The two constraints *grasp is constant* and *placement is constant* constitute the other type of constraint, called *parametrization constraint*. The are mutable, *i.e.* they depend on problem inputs and on internal parameters. For instance, the internal parameter for *placement is constant* is the pose of the object.

The notion of constraint generalizes the notion of grasp and placement. They are related to *atoms* in Task Planning, at the exception that I define two different types of constraints.

Consider a continuous grasp, *i.e.* when the set of valid grasps is not countable. Let us assume the end-effector and the object involved in this grasp are such that two functions from $\mathcal{CS}$ can be defined. The first function $\mathbf{f}_{valid}$ defines what is a grasp. The output set is $\mathbb{R}^n$. It evaluates to zero if and only if the configuration is a valid grasp, *i.e.*

$$\mathbf{f}_{valid}(\mathbf{q}) = 0_{\mathbb{R}^n} \Leftrightarrow \mathbf{q} \in \mathcal{CG}.$$

The second function $\mathbf{f}_{param}$ uniquely defines each grasp. For any two grasping configurations $\mathbf{q}_0$ and $\mathbf{q}_1$, the relation $\mathbf{f}_{param}(\mathbf{q}_0) = \mathbf{f}_{param}(\mathbf{q}_1)$ means $\mathbf{q}_0$ and $\mathbf{q}_1$ defines the same grasp. Thus, a path $\mathbf{p} \in C\left([0,1], \mathcal{CS}\right)$ is a valid *transfer path* if and only if for all t in $[0,1]$, $\mathbf{p}(t) \in \mathcal{CG}$ and $\mathbf{f}_{param}(\mathbf{p}(t)) = \mathbf{f}_{param}(\mathbf{p}(0))$.

Functions with the similar meaning can be defined for continuous placements. Thus these two functions are an abstract representation of grasp and placement. This defines the framework of this thesis regarding what type of grasps, placements (and actions in general) are considered. There must exist two functions to define it: a validation function and a parametrization function.

From the above remark, it becomes natural to define the following constraints, where $C^1\left(\mathcal{CS}, \mathbb{R}^n\right)$ denotes the space of continuously differentiable function from $\mathcal{CS}$ to $\mathbb{R}^n$.

Definition 2.1 (Parametrization constraint)**.** *A* parametrization constraint *is a constraint of the form* $\mathbf{f}(\mathbf{q}) = \mathbf{b}$ *where* $\mathbf{f} \in C^1\left(\mathcal{CS}, \mathbb{R}^n\right)$, $\mathbf{q} \in \mathcal{CS}$ *and* $\mathbf{b} \in \mathbb{R}^n$. $\mathbf{b}$ *is the* parameter *and* n *is the dimension.*

Configuration $\mathbf{q} \in \mathcal{CS}$ *satisfies the* parametrization constraints with parameter $\mathbf{b}$ *if and only if* $\mathbf{f}(\mathbf{q}) = \mathbf{b}$.

As a stack of parametrization constraints is also a parametrization constraint, I always consider only one parametrization constraint. For instance, a stack of two parametrization constraints with parameters $\mathbf{b}_0$ and $\mathbf{b}_1$, $\mathbf{f}_0(\mathbf{q}) = \mathbf{b}_0$ and $\mathbf{f}_1(\mathbf{q}) = \mathbf{b}_1$, is the following parametrization constraint.

$$\begin{pmatrix} \mathbf{f}_0 \\ \mathbf{f}_1 \end{pmatrix} (\mathbf{q}) = \begin{pmatrix} \mathbf{b}_0 \\ \mathbf{b}_1 \end{pmatrix}$$

Definition 2.2 (Validation constraint)**.** *A* validation constraint *is a parametrization constraint whose parameter* $\mathbf{b}$ *is zero, i.e. of the form* $\mathbf{f}(\mathbf{q}) = 0$ *where* $\mathbf{f} \in C^1(\mathcal{CS}, \mathbb{R}^n)$ *and* $\mathbf{q} \in \mathcal{CS}$*.*

Similarly to parametrization constraint, a stack of validation constraints is a validation constraint. I always consider only one validation constraint. $\mathcal{C}_V$ denotes the space of all validation constraints, $\mathcal{C}_P$ the space of all parametrization constraints and $\mathcal{C}$ the union of both.

As mentioned in Section 1.1.2.1, the preferred method for constraint resolution is a Newton-Raphson (NR) algorithm. This choice implies that the constraint must be differentiable. The Jacobian of a constraint is the Jacobian of $\mathbf{f}$. It is denoted by $\mathbf{J}_\mathbf{f}$.

2.1.3 Path

A path $\mathbf{p} \in C([0,1], \mathcal{CS})$ is a continuous mapping from $[0,1]$ to $\mathcal{CS}$.

When solving a path planning problem where the robot is subject to a constraint, an operator called *steering method* is used. This operator takes as input two configurations satisfying the constraint. It returns (in case of success) a path satisfying the constraint and linking the end configurations.

$$\mathcal{SM} : \begin{array}{ccc} \mathcal{CS} \times \mathcal{CS} \times \mathcal{C} & \rightarrow & C([0,1], \mathcal{CS}) \\ (\mathbf{q}_0, \mathbf{q}_e, c) & \mapsto & \mathbf{p} \end{array}$$

such that $\forall t \in [0,1]$, $\mathcal{SM}(\mathbf{q}_0, \mathbf{q}_e, c)(t)$ satisfies constraint c.

From an implementation point of view, it is possible to discretize the linear interpolation between $\mathbf{q}_0$ and $\mathbf{q}_e$ into N steps, project each sample configuration on the constraint and make the steering method return linear interpolations between projected sample configurations. However, the point wise projection has several drawbacks.

First, a discretization step needs to be chosen for each application. This adds a parameter to be set.

Second, the resulting path may not satisfy the constraint between samples. Some algorithms that assume that constraints are satisfied everywhere may fail because the assumption is not satisfied. And third, the projector is, in general, not continuous and the path obtained after projection may not be continuous.

The second issue is addressed by applying the constraints at evaluation.

$$\mathcal{SM}(\mathbf{q}_0, \mathbf{q}_e, c)(t) = \mathit{proj}(\mathbf{interpolate}(\mathbf{q}_0, \mathbf{q}_e)(t), c)$$

where $\mathit{proj}(., c)$ is a projector on c and **interpolate** is the interpolation used internally by the *steering method*. The naive interpolation function is the straight line interpolation **straight** defined in (2.1).

The third issue is addressed by constructing a suitable function **interpolate** for each $(\mathbf{q}_0, \mathbf{q}_e)$. This is detailed in next section.

2.2 Continuous path on manifolds

In this section, I address the problem of guaranteeing the continuity of constrained path. As explained in Section 1.1.2.1, the NR algorithm is best suited to constraint resolution. However, the algorithm is not continuous with respect to the input configuration. Figure 2.3 gives an elementary example where a continuous input path yields a discontinuous projected path. Figure 2.1 gives a realistic example where point-wise projection fails. This example is detailed in the end of this section.

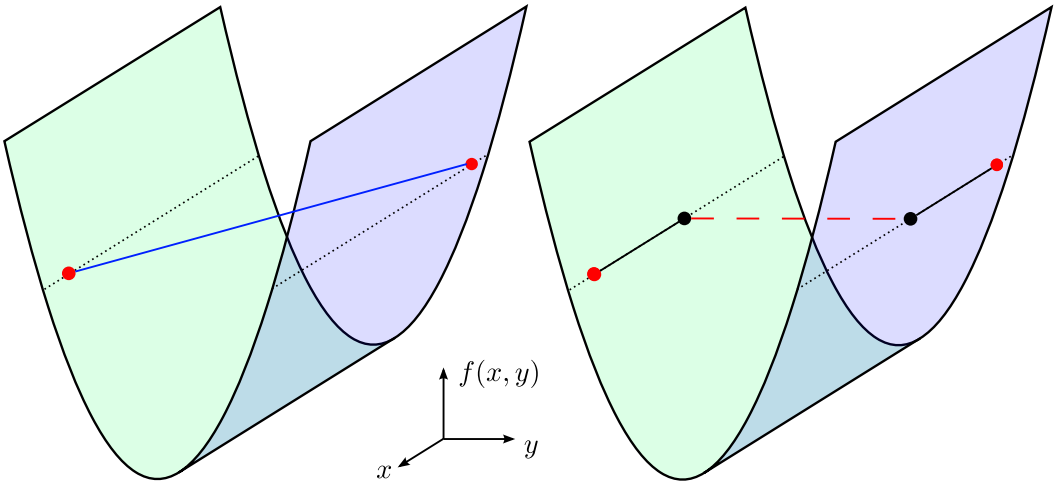


Figure 2.3: This 2D example, where (x, y) are the configuration parameters, shows the graph of $\mathbf{f}((x,y)) = y^2 - 1$. The 2 dotted horizontal line are the solutions of $\mathbf{f}((x,y)) = 0$. The 2 red circles are two configurations satisfying $\mathbf{f}(\mathbf{q}) = 0$. On the left, the blue line is **straight** $[\mathbf{q}_0, \mathbf{q}_e]$ and on the right, the black solid line is its pointwise projection. The discontinuity is highlighted by the black circles and the red dashed line.

In this section, I derive a sufficient condition of continuity of one iteration of the NR algorithm. This condition is easy to check in practice. Then, I introduce two algorithms to check for path continuity.

2.2.1 Newton-Raphson algorithm

The NR algorithm iteratively updates the robot configuration so as to decrease the norm of the constraint value $\mathbf{f}(\mathbf{q})$. Let $\alpha > 0$ and $P_\alpha \in \mathcal{F}(\mathcal{CS}, \mathcal{CS})$ be the NR iteration function:

$$P_\alpha(\mathbf{q}) = \mathbf{q} - \alpha \mathbf{J}(\mathbf{q})^\dagger \mathbf{f}(\mathbf{q}) \tag{2.2}$$

where $A^\dagger$ is the Moore-Penrose pseudo-inverse of A and $\mathbf{J}(\mathbf{q})$ is the Jacobian matrix of $\mathbf{f}$ in $\mathbf{q}$. $P_\alpha(\mathbf{q})$ is the configuration obtained after one iteration of the NR algorithm, starting at $\mathbf{q}$.

For a given sequence $(\alpha_n) \in]0,1]^{\mathbb{N}}$ and a given numerical tolerance $\varepsilon > 0$, let

$\mathcal{P}_N(\mathbf{q}) = P_{\alpha_N}(\cdots(P_{\alpha_0}(\mathbf{q})))$. The projection of a configuration $\mathbf{q}$ is $\mathcal{P}_N(\mathbf{q})$ where N is such that:

- $\forall i \in [\![0, N-1]\!], P_{\alpha_i}(\cdots(P_{\alpha_0}(\mathbf{q}))) \geq \varepsilon$,
- $\mathcal{P}_N(\mathbf{q}) < \varepsilon$.

Note that the projection is not always defined as N might not exist.

2.2.2 Continuity of the Newton-Raphson iteration function

Let $\mathcal{B}(\mathbf{q}, r) = \{\tilde{\mathbf{q}} \in \mathcal{CS}, ||\tilde{\mathbf{q}} - \mathbf{q}||_2 < r\}$ be the open ball of center $\mathbf{q}$ and of radius r. The continuity of P_α is expressed as follows.

Lemma 2.1 (Continuity of the NR iteration function)**.** *Let $\mathbf{f} \in \mathcal{C}^1(\mathcal{CS}, \mathbb{R}^m)$. Let $\mathbf{J}(\mathbf{q})$ be its Jacobian and $\sigma(\mathbf{q})$ be the smallest non-zero singular value of $\mathbf{J}(\mathbf{q})$. Finally, let $r = \max_{\mathbf{q} \in \mathcal{CS}}(rank(\mathbf{J}(\mathbf{q})))$.*

If $\mathbf{J}$ is a Lipschitz function, of constant K, then, $\forall \mathbf{q} \in \mathcal{CS}$

$$rank(\mathbf{J}(\mathbf{q})) = r \Rightarrow P_\alpha \text{ is continuous on } \mathcal{B}\left(\mathbf{q}, \tfrac{\sigma(\mathbf{q})}{K}\right)$$

Proof. Let $\mathbf{f}$ be continuously differentiable function, K be a Lipschitz constant of its Jacobian, and $r = \max_{\mathbf{q}}(rank(\mathbf{J}(\mathbf{q})))$ be known.

As $\mathbf{f}$ is continuously differentiable, P_α is continuous where the pseudo-inverse application is continuous. The first part of the proof reminds some continuity condition of the pseudo-inverse. The second part proves that the latter condition is satisfied on the interval of Lemma 2.1.

Condition of continuity of the pseudo-inverse Let $\mathbf{q}$ be a *regular point*, *i.e.* $rank(\mathbf{J}(\mathbf{q})) = r$. As the set of *regular points* is open [Lewis, 2009] and $\mathbf{J}$ is continuous, there exists a neighbourhood $\mathcal{U}$ of $\mathbf{q}$ where the rank of $\mathbf{J}$ is constant. The continuity of the Moore-Penrose pseudo inverse can be expressed as follows [Rakočević, 1997].

Theorem 2.1 (Continuity of the pseudo inverse)**.** *If $(A_n) \in (\mathbb{R}^{m \times d})^{\mathbb{N}}$, $A \in \mathbb{R}^{m \times d}$ and $A_n \mapsto A$, then*

$$A_n^\dagger \mapsto A^\dagger \Leftrightarrow \exists n_0, \forall n \geq n_0, rank(A_n) = rank(A)$$

Theorem 2.1 proves that $\mathbf{J}^\dagger$ is a continuous function of $\mathbf{q}$ on $\mathcal{U}$. In the following section, I prove that $\mathcal{U} = \mathcal{B}(\mathbf{q}, \frac{\sigma}{K})$ is a suitable neighborhood.

Interval of continuity of the pseudo-inverse The norm on $\mathbb{R}^{m \times n}$ I consider is the Frobenius norm (L2-norm), denoted $|||.|||_F$. Mirsky's theorem [Mirsky, 1960, Theorem 5], restricted to the Frobenius norm, is:

Theorem 2.2 (Mirsky)**.** *If $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$ and $\tilde{\sigma}_1 \geq \tilde{\sigma}_2 \geq \cdots \geq \tilde{\sigma}_n$ are the singular values of two matrices of the same size, $\mathbf{B}$ and $\tilde{\mathbf{B}}$, then*

$$|||diag(\tilde{\sigma}_i - \sigma_i)|||_F \leq |||\tilde{\mathbf{B}} - \mathbf{B}|||_F$$

Lemma 2.2. *Let $(\mathbf{J}, \mathbf{dJ}) \in (\mathbb{R}^{m\times d})^2$ and σ be the smallest non-zero singular value of $\mathbf{J}$. Then,*

$$|||\mathbf{dJ}|||_F < \sigma \Rightarrow rank(\mathbf{J}) \leq rank(\mathbf{J} + \mathbf{dJ})$$

Proof. Let p, resp. q, be $rank(\mathbf{J})$, resp. $rank(\mathbf{J} + \mathbf{dJ})$. Let $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p > 0$, resp. $\tilde{\sigma}_1 \geq \tilde{\sigma}_2 \geq \cdots \geq \tilde{\sigma}_q > 0$, be the non-zero singular values of $\mathbf{J}$, resp. $\mathbf{J} + \mathbf{dJ}$. I apply Theorem 2.2 with $\mathbf{B} = \mathbf{J}$ and $\tilde{\mathbf{B}} = \mathbf{J} + \mathbf{dJ}$.

$$\begin{aligned} |||\mathbf{dJ}|||_F < \sigma_p &\Rightarrow |||diag(\tilde{\sigma}_i - \sigma_i)|||_F < \sigma_p \\ &\Rightarrow \forall i \leq p, \tilde{\sigma}_i > \sigma_i - \sigma_p \\ &\Rightarrow \forall i \leq p, \tilde{\sigma}_i > 0 \\ &\Rightarrow p \leq q \end{aligned}$$ □

Note that the ball has to be open. At this point, I have an interval for the Jacobian in which the rank does not decrease. I use the Lipschitz constant K to have an interval in the configuration space.

$$\forall(\mathbf{q}, \tilde{\mathbf{q}}) \in \mathcal{CS}^2, |||\mathbf{J}(\tilde{\mathbf{q}}) - \mathbf{J}(\mathbf{q})|||_F \leq K||\tilde{\mathbf{q}} - \mathbf{q}||_2$$

Let $\mathbf{q} \in \mathcal{CS}$ and σ be the smallest non-zero singular value of $\mathbf{J}(\mathbf{q})$. Then,

$$\begin{aligned} \tilde{\mathbf{q}} \in \mathcal{B}\left(\mathbf{q}, \tfrac{\sigma}{K}\right) &\Rightarrow |||\mathbf{J}(\tilde{\mathbf{q}}) - \mathbf{J}(\mathbf{q})|||_F \leq K||\tilde{\mathbf{q}} - \mathbf{q}||_2 < \sigma_p \\ &\Rightarrow rank(\mathbf{J}(\tilde{\mathbf{q}})) \geq rank(\mathbf{J}(\mathbf{q})) \end{aligned}$$

If $\mathbf{q}$ is a *regular point*, $rank(\mathbf{J}(\mathbf{q}))$ has rank $r = \max_{\mathbf{q}}\left(rank\left(\mathbf{J}(\mathbf{q})\right)\right)$. Thus $\mathbf{J}(\tilde{\mathbf{q}})$ has a constant rank r on $\mathcal{B}\left(\mathbf{q}, \frac{\sigma}{K}\right)$. By Theorem 2.1, $\mathbf{J}(\mathbf{q})^\dagger$ is continuous. P_α is the composition of continuous functions so it is continuous on $\mathcal{B}\left(\mathbf{q}, \frac{\sigma}{K}\right)$.

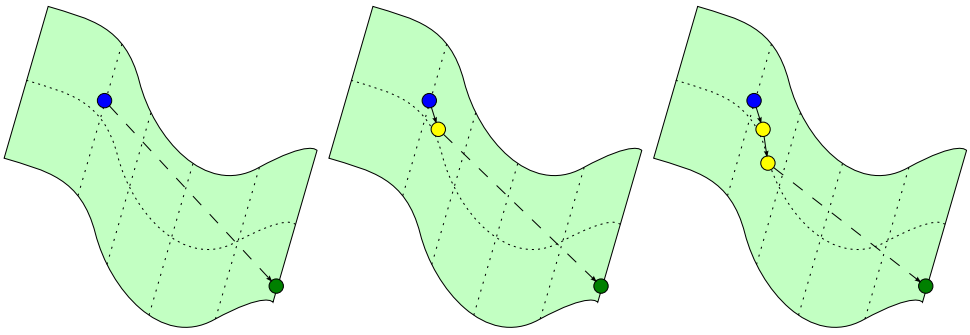
This proves Lemma 2.1. □

2.2.3 Two path projection algorithms

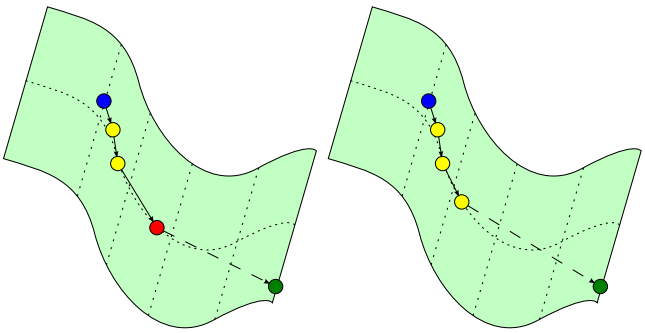
This section presents two path projection algorithms with continuity certificate. From an initial constrained path $\mathcal{SM}(\mathbf{q}_0, \mathbf{q}_e, \mathbf{f})$, the algorithms generate a set of interpolation points (IPs) $(\mathbf{q}_0, \cdots, \mathbf{q}_n)$ where $\mathbf{f}(\mathbf{q}_i) = 0$ and n is decided by the algorithm. The sequence is such that the continuity interval of consecutive IPs overlay so the NR iteration function is continuous on the straight line joining consecutive IPs. The resulting path is the concatenation of $\mathcal{SM}(\mathbf{q}_i, \mathbf{q}_{i+1}, \mathbf{f}), \forall i \in [0, n[$. When the algorithms succeed, $\mathbf{q}_n = \mathbf{q}_e$. When they fail to project a path, they return the longest part along the path, starting at $\mathbf{q}_0$, that has been validated.

To benefit from the continuity interval of P_α, a Lipschitz constant must be computed for the Jacobian of the constraint. Appendix A proposes a method to bound from above the norm of the Hessian for constraints involving joint placements. This upper bound is a Lipschitz constant of the Jacobian. This method also extends to constraint involving the Center of Mass (CoM) of the robot as the CoM is a weighed sum of joint positions.

The maximum number of IPs on unit length paths N_{max} is set to 20 and the minimum interpolation distance λ_m is set to 0.001. These parameters ensure the algorithms terminate. When a limit is reached, the algorithms return the left part of a path that has been successfully projected, as stated above.



(a) Input path (b) Step 1: Success (c) Step 2: Success



(d) Step 3: $\mathbf{q}$ too far (e) Step 3: Success

Figure 2.4: Progressive projection method. The green surface is $\mathbf{f}(\mathbf{q}) = 0$. (a) shows the input path. (b) and (c) shows two successful iterations. The two first IPs are added without refinement because they are close enough from the previous IP. (d) shows a rejected IP. As it is too far from the previous IP, it is refined by dividing λ by two. It results in (e) and the new IP is finally added.

Progressive projection is presented in Alg 2.1 and depicted in Figure 2.4. The sequence of IPs is generated recursively, starting from q_0. Each IP is computed from the previous IP and the final configuration q_e.

From $\mathbf{q}_0$, it builds a configuration satisfying the constraint, within the continuity interval of $\mathbf{q}_0$. The configuration is chosen towards $\mathbf{q}_e$ (Line 7). When $\mathbf{q}_e$ is within

Algorithm 2.1 Progressive continuous projection

1: **function** PROJECT($\mathbf{q}_0, \mathbf{q}_e, depth$)
▷ Continuously project the direct path $(\mathbf{q}_0, \mathbf{q}_e)$ onto the submanifold $\mathbf{f}(\mathbf{q}) = 0$
2: **if** $K \times ||\mathbf{q}_0 - \mathbf{q}_e||_2 < \sigma_r(\mathbf{q}_0)$ **then return** $(\mathbf{q}_0, \mathbf{q}_e)$
3: **if** $depth > N_{max} \times ||\mathbf{q}_0 - \mathbf{q}_e||_2$ **then return** $(\mathbf{q}_0)$
4: $\lambda \leftarrow \sigma_r(\mathbf{q}_0)/K$
5: **repeat**
6: **if** $\lambda < \lambda_m$ **then return** $(\mathbf{q}_0)$
7: $\mathbf{q} \leftarrow \mathcal{SM}(\mathbf{q}_0, \mathbf{q}_e, f)(\frac{\lambda}{||\mathbf{q}_0 - \mathbf{q}_e||_2})$
8: $\lambda \leftarrow \frac{\lambda}{2}$
9: **until** $K||\mathbf{q} - \mathbf{q}_0||_2 < \sigma_r(\mathbf{q}_0)$
10: **return** $\{\mathbf{q}_0\} \cup$ PROJECT$(\mathbf{q}, \mathbf{q}_e, depth + 1)$

the continuity interval of $\mathbf{q}_0$ (Line 2), the algorithm succeeds.

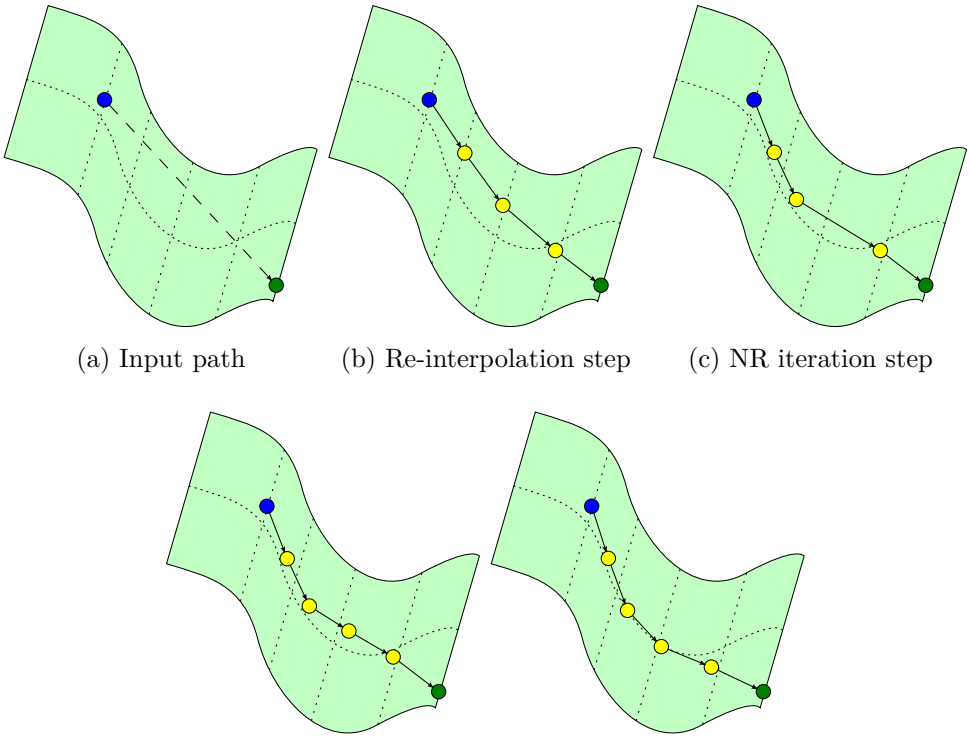
From 2.4(a) to 2.4(b), the path is cut in two at parameter λ from the start configuration. λ is gradually reduced so that the projected configuration is within the continuity ball of $\mathbf{q}_0$ (Lines 4-9). When $\lambda < \lambda_m$ (Line 6), the projection locally increases the distances more than $\sigma_r(\mathbf{q}_k)/(\lambda_m K)$. The path is considered discontinuous and the algorithm fails. If $\mathbf{q}$ is found in $\mathcal{B}\left(\mathbf{q}_0, \frac{\sigma(\mathbf{q}_0)}{K}\right)$, the left part satisfies the condition of Lemma 2.1. The right part is then projected using the same procedure.

Global projection method is presented in Alg 2.2 and depicted in Figure 2.5. The algorithm starts by computing IPs along the straight path such that the continuity interval of consecutive IPs overlay. However, they do not satisfy the constraints. The constraint violation of each IPs is then iteratively decreased while keeping overlaying continuity intervals.

The algorithm works in two steps. First, the IPs are improved in order to decrease the constraint violation, by applying the NR iteration function (Line 8). Second, it checks whether the distance between each pair of consecutive IPs $(\mathbf{q}_k, \mathbf{q}_{k+1})$ is within the union of the two continuity balls (Line 14). If this check fails, a new IP $\mathbf{q}$ is added at the border of the continuity ball of $\mathbf{q}_k$. Next iteration will consider the two consecutive points $(\mathbf{q}, \mathbf{q}_{k+1})$.

For clarity of the pseudo-code, a limit on the number of iterations of constraint violation reduction loops has been omitted (Line 6). Such a limit avoids infinite loops due to local minima. This limit is set to 40 in my implementation and the counter is reset whenever an IP is added.

Figure 2.5 shows the path after some iterations. From 2.5(b) and 2.5(c), the projection loop (Line 6) reduces the constraint violation point-wise. Between 2.5(c) and 2.5(d), an IP is added (Line 18).



(a) Input path (b) Re-interpolation step (c) NR iteration step

(d) Re-interpolation step (e) NR iteration step

Figure 2.5: Global projection method. The green surface is $\mathbf{f}(\mathbf{q}) = 0$. (a) shows the input path. (b) and (d) shows two re-interpolation steps. New IPs are added when the distance between consecutive IPs is too big. (c) and (e) shows the constraint violation reduction step. Each IP is updated in order to decrease the constraint violation.

2.2.4 Continuous planning algorithm

The above algorithms can be integrated into motion planning to achieve continuous constrained motion planning. I propose here an algorithm based on Constrained-Rapidly exploring Random Tree (RRT) [Dalibard et al., 2013] but similar changes could be made to other algorithms. The projection algorithm are then tested on various problems.

2.2.4.1 Continuous Constrained-RRT

Algorithm 2.3 provides function CONSTRAINEDEXTEND. This function builds a straight interpolation $\mathbf{p}_1$ between $\mathbf{q}_{near}$ and a configuration $\mathbf{q}_{proj}$, which satisfies *constraint*. Continuous path projection is applied to $\mathbf{p}_1$ and generates a path $\mathbf{p}_2$ satisfying $\mathbf{p}_2(0) = \mathbf{q}_{near}$. In case of failure, the left part that has successfully been projected is returned. In case of success, we have $\mathbf{p}_2(1) = \mathbf{q}_{proj}$. Then $\mathbf{p}_2$ is tested for collision. Again, in case of failure, a left collision-free part of the path is returned. $\mathbf{p}_3$ is then a continuous collision-free path satisfying the constraint. Many variants

Algorithm 2.2 Global continuous projection

1: **function** PROJECT($\mathbf{q}_0, \mathbf{q}_e, \mathbf{f}$)
▷ Continuously project the direct path $(\mathbf{q}_0, \mathbf{q}_e)$ onto the submanifold $\mathbf{f}(\mathbf{q}) = 0$
2: $\mathbf{Q} \leftarrow (\mathbf{q}_0, \mathbf{q}_e)$
3: $repeat \leftarrow True$
4: **while** $repeat$ **do**
5: $repeat \leftarrow False$
6: **for all** $\mathbf{q}_k \in \mathbf{Q}$ **do**
7: **if** $||f(\mathbf{q}_k)||_2 > \varepsilon$ **then**
8: $\mathbf{q}_k \leftarrow P_\alpha(\mathbf{q}_k)$
9: $repeat \leftarrow True$
10: **for all** Consecutive $\mathbf{q}_k, \mathbf{q}_{k+1} \in \mathbf{Q}$ **do**
11: **if** $\sigma_r(\mathbf{q}_k) < K\lambda_m$ **then**
12: $\mathbf{Q} \leftarrow (\mathbf{q}_0, \cdots, \mathbf{q}_k)$ **and break**
13: $d \leftarrow \sigma_r(\mathbf{q}_k) + \sigma_r(\mathbf{q}_{k+1})$
14: **if** $d < K \times ||\mathbf{q}_k - \mathbf{q}_{k+1}||_2$ **then**
15: $\mathbf{q} \leftarrow$ INTERPOLATE$(\mathbf{q}_k, \mathbf{q}_{k+1}, \frac{\sigma_r(\mathbf{q}_k)}{K})$
16: $\mathbf{Q} \leftarrow (\mathbf{q}_0, \cdots, \mathbf{q}_k, \mathbf{q}, \mathbf{q}_{k+1}, \cdots)$
17: $repeat \leftarrow True$
18: **if** LENGTH$(\mathbf{Q}) > N_{max} \times ||\mathbf{q}_0 - \mathbf{q}_e||_2$ **then**
19: $\mathbf{Q}$.REMOVELASTELEMENT
20: **return Q**

of the RRT could be used here, such as using fixed progression step or using only a ratio of the valid path.

This function replaces function EXTEND, Line 7 of Algorithm 1.1.

2.2.4.2 Simulations

I compared both continuous path projection algorithms and the Recursive Hermite Projection (RHP) Hauser [2013] to each other in two settings, each described in the two following paragraphs. The benchmarks are run using the Humanoid Path Planner (HPP) software framework, in which the 3 algorithms have been implemented.

Quadratic problems I first compare the Progressive and Global path projection algorithms and the RHP for various parameters in the following problems.

- Circle: the configuration space is $[-1, 1]^2$, subject to constraint $f(x, y) = x^2 + y^2 - 1 = 0$. A Lipschitz constant of f is $M = 2\sqrt{2}$ and a Lipschitz constant of its Jacobian is $K = 2\sqrt{2}$. I project line segments between $(1, 0)$ and $(\cos\theta, \sin\theta)$ for $\theta \in [\pi/2, \pi]$. None of the algorithms were able to find a continuous path for the singular case $\theta = \pi$. The Global projection method did not need any IPs to return an answer.

Algorithm 2.3 Continuous Constrained RRT extension

```
1: function CONSTRAINEDEXTEND(q_near, q_proj, T, constraint)
2:     p_1 ← INTERPOLATE(q_near, q_proj)
3:     pathProj ← PROJECTOR(constraint)
4:     p_2 ← pathProj.APPLY(p_1)
5:     p_3 ← TESTCOLLISION(p_2)
6:     q_new ← FINALCONFIGURATION(p_3)
7:     T.INSERTCONFANDPATH(q_new, p_3)
```

- Parabola: the configuration space is $[-1, 1] \times [0, 2]$, subject to constraint $f(x, y) = y^2 - 1 = 0$. This constraint has two disjoint sets of solution: $y = -1$ and $y = 1$. Figure 2.3 illustrates this case. A Lipschitz constant of f is $M = 2$ and a Lipschitz constant of its Jacobian is $K = 2$. I project line segments between $(0, 1)$ and $(\tau, -1)$ for $\tau \in [0, 2]$. No continuous path can both connect any pair of these points and satisfy the constraints at all time. All the algorithms were able to detect the discontinuity.

Results are presented in Table 2.1. Be aware that the units for the RHP are not the same as for the two other algorithms. The global projection method outperforms the progressive method on these quadratic problems.

Global proj.	Circle	Parabola
t_{avg}/t_{max} (µs)	16/90	201/231
$d_{min}/d_{avg}/d_{max}$ (mm)	-	0.2/51/316
N_{ip}	0	10

Progressive proj.	Circle	Parabola
t_{avg}/t_{max} (µs)	78/134	173/207
$d_{min}/d_{avg}/d_{max}$ (mm)	284/462/512	1/71/316
N_{ip}	4.75	14

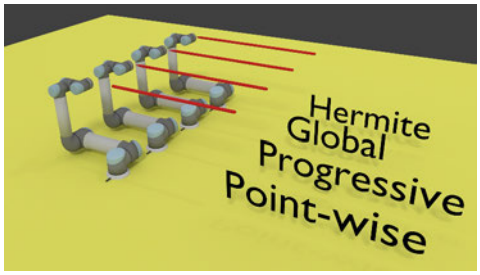
Recursive hermite proj.	Circle	Parabola
t_{avg}/t_{max} (ms)	503/900	0.017/0.03
$d_{min}/d_{avg}/d_{max}$ (µm)	50/75/100	-
N_{ip}	28946	0

Table 2.1: Quadratic problems benchmarks. Each case is run 10 times. The rows correspond to the average and maximum computation time t_{avg}, t_{max}, the average, minimum and maximum distance between consecutive IPs d_{avg}, d_{min}, d_{max}, and the average number of interpolation points N_{ip}.

Constrained planning



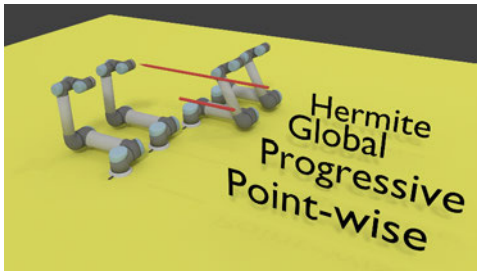
(a) Initial configuration of path 1



(b) Final configuration of path 1



(c) Initial configuration of path 2



(d) Final configuration of path 2

Figure 2.6: Path projection results with UR5 robot for two input paths. UR5 end-effector must follow an horizontal line, with fixed orientation. The red lines give the left part of the input paths that was successfully projected. In both cases, the point-wise projection failed. Hermite refers to the RHP algorithm. The three algorithms manage to project the input path 1, from (a) to (b). Only the Progressive projection method managed to project the input path 2, from (c) to (d). In this case, Global projection and RHP did not succeed at all.

UR5 Consider a welding problem with UR5 robot. The end-effector of the robot must move along a line with a fixed orientation, as shown in Figure 2.1. I compare the behaviour of each algorithm on motions that switch between two inverse kinematic solutions while satisfying the problem constraint.

Using Appendix A, a Lipschitz constant of the constraint is $M = 6$ and a Lipschitz constant of its Jacobian is $K = 7.14$. Table 2.2 and Figure 2.6 summarize the results obtained for various line segment. Note that, without motion planning, the three algorithms are able to project paths between two different inverse kinematic solutions. For instance, from Figure 2.6(c) to Figure 2.6(d), the robot must find a configuration in which its arm and forearm are aligned, while still being on the line. An inverse kinematic solver would have to detect this and handle it as a specific case by finding the intermediate configuration with aligned arm and forearm.

When the projection method returns a false negative, the longest validated part of the input path is returned. In the context of randomized motion planning, the high rate of false negatives of global projection method does not block the search. The expected effect is an increase of the size of final roadmap.

Projection method	Global	Progressive	RHP
t_{avg}/t_{max} (ms)	85/746	6.5/9.5	160/175
N_{ip}	123	72	3403
False negative	54%	0%	8%

Table 2.2: Results of UR5 case. The rows have the same meaning as in Table 2.1. The number of false negative corresponds to the ratio of rejected path over all tests, while a continuous path exists.

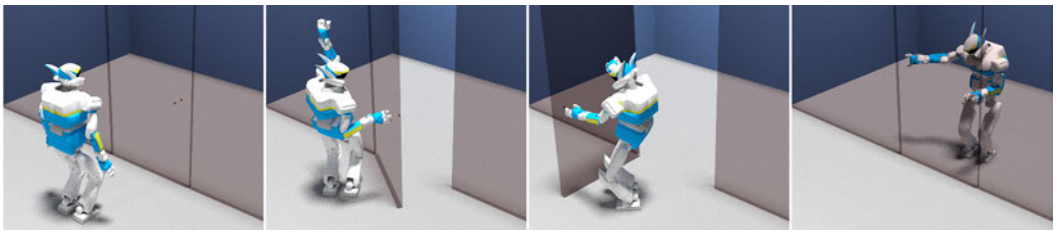


Figure 2.7: HRP2 opening a door. This illustrates a problem where continuous path projection is needed. Without continuous projection, the algorithm returns discontinuous solutions.

Integration in a manipulation planner The Continuous Constrained RRT has been tested to plan a manipulation path where the HRP2 robot opens a door. As proposed by Dalibard et al. [2013], the planning is split in two phases. A quasi-static full-body motion for the sliding robot is first computed. Additionally to the manipulation rules, quasi-static constraints are taken into account. Then, the motion is post-processed to obtain a dynamically-feasible walking trajectory.

Figure 2.7 shows the result of the first phase. No optimization were run. The motion without continuous path projection contains several discontinuities. This demonstrates both the necessity to check for continuity and that these algorithms perform as expected.

2.2.4.3 Discussion

The two continuous path projection algorithms have the following guarantees. They provide a path with IPs satisfying the constraints. Moreover, they ensure that the NR iteration function is continuous along the lines connecting consecutive IPs. The piecewise straight interpolation is closer to constraint satisfaction than the input path and one iteration of NR is continuous. This leads to good chances to have the resulting path continuous. In practice, no discontinuity have been encountered.

Compared to the presented method, the RHP gives continuity, at the cost of being first, computationally less efficient, second, unable to return the continuously projected part of the path and third requires to introduce velocities. The efficiency of the presented method, compared to RHP, comes from the expected distances between IPs. Indeed, RHP generates a lot more IPs than the two proposed algo-

rithms. The reason is the following. The distance between IPs is less than $\varepsilon/K_{\mathbf{f}}$ where ε is the constraint satisfaction tolerance and $K_{\mathbf{f}}$ is a Lipschitz constant of the constraint. In the presented case, this distance is around $\sigma/K_{\mathbf{J}}$, where σ is the smallest singular value of the Jacobian and $K_{\mathbf{J}}$ is a Lipschitz constant of the Jacobian of the constraint. In part of the configuration space far from singularities, σ is orders of magnitude bigger than ε, set to 10^{-4} in the experiments. The comparison with RHP in previous section emphasizes this theoretical approach.

2.3 Static stability

Motion planning for humanoid robots is challenging for two reasons. First $\mathcal{CS}$ is a high dimensional space. Second equilibrium must be taken into account. As I am only considering geometrical problems, without taking time into account. It is thus a static stability criterion.

Ignoring time is of course a questionable choice. Many problems require dynamical motions and cannot be solved while keeping static stability at all time. For instance, static stability cannot plan jumps. However, taking dynamics into account in a full-body motion planner is still an open problem.

In this section, I propose a formulation of a subclass of static stability problem as a differentiable constraint.

2.3.1 Static stability constraint

Consider a robot in contact with the environment. I make the simplifying assumption of punctual contacts. For a humanoid robot, the foot can be modelled as four contact points. I do not consider friction, although the method I present could easily be extended to consider them.

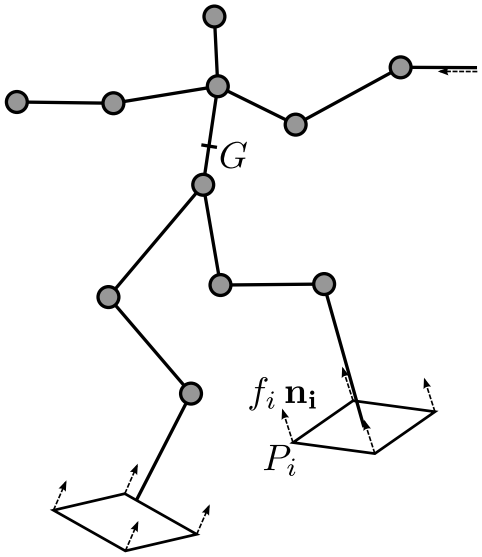


Figure 2.8: Non-coplanar friction-less multi-contact criterion. The circle represents the robot joints. The dashed arrows represent forces applied to the robot. P_i is a contact point and $f_i\,\mathbf{n}_i$ is the force applied in P_i.

Let $(C_i)_{i\in[\![1,n]\!]}$ be n contacts. G denotes the center of gravity of the robot

and m its mass. Each contact C_i is defined by a point P_i and a normal $\mathbf{n}_i$. It defines an effort $f_i\,\mathbf{n}_i$ in P_i. P_i and $\mathbf{n}_i$ are defined respectively to a robot joint frame. $\mathbf{f} = (f_1, \ldots, f_n)$ denotes the vector of forces. Figure 2.8 summarizes the notations. The laws of classical mechanics give the following frictionless static stability criterion:

$$\exists \mathbf{f} \in [0, +\infty[^n, \phi\mathbf{f} + m\mathbf{G} = 0_{\mathbb{R}^6} \tag{2.3}$$

where $\mathbf{G} = (0, 0, -9.81, 0, 0, 0)^T \in \mathbb{R}^6$ is the gravity vector extended with 3 zeros and

$$\phi = \begin{pmatrix} \ldots & \mathbf{n}_i & \ldots \\ \ldots & \mathbf{P_i G} \times \mathbf{n}_i & \ldots \end{pmatrix} \in \mathbb{R}^{6\times n} \tag{2.4}$$

It is easy to see that ϕ is a continuously differentiable function of the robot configuration.

Problem formulation I now formulate the problem of finding $\mathbf{f}$ in (2.3) as a quadratic problem. Let $\mathbf{H} \in \mathbb{R}^{n\times n}$ and $\mathbf{g} \in \mathbb{R}^n$ be $\mathbf{H} = \phi^T\phi$ and $\mathbf{g} = m\phi^T\mathbf{G}$. Thus,

$$\frac{1}{2}||\phi\mathbf{f} + m\mathbf{G}||_2^2 = \frac{1}{2}\mathbf{f}^T\mathbf{H}\mathbf{f} + \mathbf{f}^T\mathbf{g} + \frac{1}{2}m^2\mathbf{G}^T\mathbf{G} \tag{2.5}$$

Then, the frictionless static stability criterion (2.3) can be written as the following optimisation problem.

$$\begin{array}{ll} \min & \frac{1}{2}\mathbf{f}^T\mathbf{H}\mathbf{f} + \mathbf{f}^T\mathbf{g} \\ s.t. & \mathbf{f} \geq 0 \end{array} \tag{2.6}$$

Let $C(\mathbf{f}) = \frac{1}{2}\mathbf{f}^T\mathbf{H}\mathbf{f} + \mathbf{f}^T\mathbf{g}$ be the cost, $\mathbf{f}^*$ be the optimal solution to the above problem, $\mathbf{y} \in [0, +\infty[^n$ be the dual variable and $\mathbf{y}^*$ be the dual variable at optimum. $C(\mathbf{f}^*)$, which depends on $\mathbf{q}$, is the output of the static stability criterion.

At the optimum, the Karush–Kuhn–Tucker conditions gives

$$\mathbf{H}\mathbf{f}^* + \mathbf{g} - \mathbf{y}^* = 0 \tag{2.7}$$

$$\mathbf{y}^{*T}\mathbf{f}^* = 0 \tag{2.8}$$

Derivative of the optimal cost I denote partial derivatives of a function a with respect to the robot configuration by $\frac{\partial a}{\partial \mathbf{q}}$. When a is a matrix valued function, $\frac{\partial a}{\partial \mathbf{q}}$ is a tensor.

The derivative of the cost function is

$$\frac{\partial C(\mathbf{f})}{\partial \mathbf{q}} = \frac{1}{2}\mathbf{f}^T\frac{\partial \mathbf{H}}{\partial \mathbf{q}}\mathbf{f} + \mathbf{f}^T\frac{\partial \mathbf{g}}{\partial \mathbf{q}} + \frac{\partial \mathbf{f}}{\partial \mathbf{q}}^T(\mathbf{H}\mathbf{f} + \mathbf{g})$$

At the optimum, using (2.7), it becomes

$$\frac{\partial C(\mathbf{f}^*)}{\partial \mathbf{q}} = \frac{1}{2}\mathbf{f}^{*T}\frac{\partial \mathbf{H}}{\partial \mathbf{q}}\mathbf{f}^* + \mathbf{f}^{*T}\frac{\partial \mathbf{g}}{\partial \mathbf{q}} + \frac{\partial \mathbf{f}^*}{\partial \mathbf{q}}^T \mathbf{y}^*$$

In case the strict complementary slackness (2.8) holds, either of the following is true:

- $y_i^* = 0$ and $f_i^* > 0$, so $\frac{\partial f_i^*}{\partial \mathbf{q}} y_i^* = 0$,
- $y_i^* > 0$ and $f_i^* = 0$, so[1] $\frac{\partial f_i^*}{\partial \mathbf{q}} = 0$.

Finally the derivative of the optimal cost when strict complementary slackness holds is:

$$\frac{\partial C(\mathbf{f}^*)}{\partial \mathbf{q}} = \frac{1}{2}\mathbf{f}^{*T}\frac{\partial \mathbf{H}}{\partial \mathbf{q}}\mathbf{f}^* + \mathbf{f}^{*T}\frac{\partial \mathbf{g}}{\partial \mathbf{q}} \tag{2.9}$$

The tensor $\frac{\partial \mathbf{H}}{\partial \mathbf{q}}$ and the matrix $\frac{\partial \mathbf{g}}{\partial \mathbf{q}}$ only depends on forward kinematics so they are easy to compute. It is thus possible to define a piecewise differentiable function as follows. Given a set of contacts, the static stability function returns a one dimensional vector whose value is the optimal cost $C(\mathbf{f}^*)$. The derivative in Equation (2.9) holds when strict complementarity holds. I make the simplifying assumption that strict complementarity always holds.

2.3.2 Integration to a motion planner

Constraint solver The above function has two drawbacks. First, compared to geometrical functions, it is time-consuming because one has to solve a quadratic program and compute the derivative of matrix ϕ. Second, the function is built with a set of contacts. Evaluation of the function for robot configurations not satisfying these contacts does not make sense. The function is only defined on the sub-manifold of $\mathcal{CS}$ where the contacts are satisfied.

Fortunately, there is no compromise between the two. It is possible to evaluate the constraint only where defined and to reduce the number of optimization problem solved at the same time. The solution is to use a hierarchical method similar to the Stack Of Tasks [Mansard and Chaumette, 2007]. This method uses several layers of constraints (called tasks). The constraints are solved using a layered NR method. Each layer refines the descent step in the null space of the previous layers. It stops whenever the null space has dimension zero.

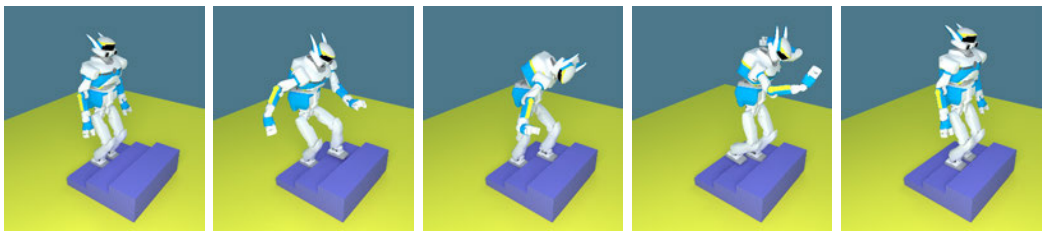
The method I propose uses the same algorithm at one exception. Instead of only stopping when the null space has dimension zero, the algorithm also stops if the constraint of the current layer is not satisfied. So a layer is reached only if the constraints of all the previous layers are satisfied.

The hierarchy of constraint I propose contains two layers although more could be added. The highest priority layer contains kinematic constraints of the contacts.

[1] I assume solutions to the optimization problem are continuous with respect to the inputs so that $y_i > 0$ holds on a ball. $f_i = 0$ is then true on this ball so the derivative at the center is zero.

The lowest priority layer contains the static stability constraint. So the static stability constraint is only evaluated for configurations in contact.

Figure 2.9 shows a motion that was generated using this criterion and the manipulation planner detailed in next chapter. The example set-up is detailed in Section 5.3.1. The quadratic program in Equation (2.6) is solved using the qpOASES library [Ferreau et al., 2014]. Table 2.3 shows benchmarks between two other criteria and this one. It seems prioritizing the constraint slightly decreases the performances. The method is about three times slower and has a lower success ratio. The deterioration of the success ratio might be due to the assumption that strict complementary slackness always holds. However, the method is able to move the position of the feet on the stairs to achieve equilibrium.



(a) Initial config. (b) (c) (d) (e) Goal config.

Figure 2.9: Robot HRP-2 climbs stairs quasi-statically. This path was generated using the friction-less multi-contact equilibrium criterion described in Section 2.3 and the manipulation planning algorithm in Chapter 3. It is detailed in Section 5.3.1.

	CoM above center	CoM above line	QP	QP with priorities
T_{eval} (µs)	345	362	545	545
T_{proj} (ms)	3.90	2.93	9.46	10.3
Success ratio	99%	99%	78%	75%

Table 2.3: Benchmark of friction-less multi-contact equilibrium criterion. Several quasi-static equilibrium criteria are used to project configurations in the scenario of Figure 2.9. The values are averaged over the same set of 10000 random configurations. T_{eval} is the time to evaluate the function and its Jacobian. T_{proj} is the time to project a random configuration. The last row is the ratio of random configuration successfully projected. "CoM above center" refers to the criterion in Dalibard et al. [2013], "CoM above line" to the second criterion in Appendix B, "QP" to this criterion without the priorities detailed in Section 2.3.2 and "QP with priorities" with the priorities.

Comparison with existing methods This method has mainly two advantages. First, it does not reduce the space of statically stable configurations to a sub-space of it, as [Dalibard et al., 2013]. This method can be used for as many contact as desired and for non coplanar contacts. Second, while most methods require fully specified contacts, they can be specified only partially with this method. As with

grasps and placements, a contact has a validation constraint and a parametrization function. For instance, one can specify that one foot should be on one step, the other on another and the hand on the handrail. The position on each step and on the handrail can be let free. The constraint solver moves the feet and arm while keeping them in contact with the environment. It uses those DoFs to generate statically stable configurations.

However, this approach has drawbacks. First, the derivative is only fully known where the complementary slackness is strict. Second, ignoring both friction and dynamics is very limiting. A wide range of motions uses at least one of them. The approach could integrate friction, at the cost of a larger[2] optimization problem to be solved. Third, evaluating the constraint and its derivative is extremely costly compared to other geometrical constraints. As a constrained motion planner evaluates both very often, motion planning becomes slow. This justifies why most methods use simpler stability criteria. The efficiency of this method could be improved using a different formulation of the optimization problem. Prete et al. [2016] proposes a lot more efficient stability criterion which uses optimization. However, they do not provide a derivative of the criterion. Brossette et al. [2015] formulates the problems as a Sequential Quadratic Program and proposes a solution which takes friction into account.

[2]The number of variables is multiplied by 3. The number of constraints stays the same.

Chapter 3

Manipulation planner

Contents

3.1 Constraint Graph 42
3.1.1 States and transitions 42
3.1.2 Problem statement 46
3.1.3 Manipulation RRT 47
3.2 Crossed foliation issue 50
3.2.1 Example 50
3.2.2 Conditions 51
3.2.3 Crossed foliation transition 56
3.3 Generalized reduction property 57
3.3.1 Generalized reduction property 57
3.3.2 Grasps and placements 62
3.3.3 Limitations 63
3.4 Narrow passages 63
3.4.1 Low sampling probability 63
3.4.2 Way-point transition 65
3.4.3 Experimental results 66

This chapter describes a theoretical framework to model admissible motions of robots and objects. From the manipulation rules, this model provides the structure of $\mathcal{CS}$. This structure is represented as a graph. It expresses admissible motions and proposes a generic way of dealing with the foliations they induce.

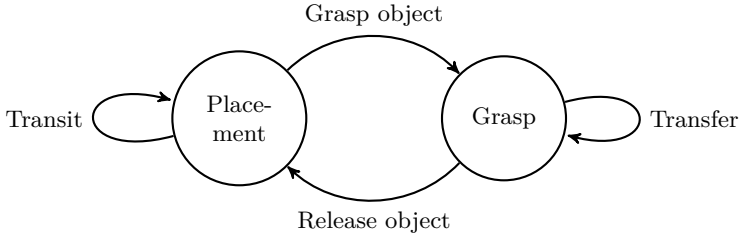


Figure 3.1: Graph of Constraint for a problem with a robot with one gripper manipulating one object.

The first section formally introduces the graph and the Manipulation-RRT (M-RRT) algorithm. The vertices of the graph represent a combination of facts, like *Robot gripper holds object* or *Object is on the table*. The edges represent transitions, like *Grasp object* or *Put down object*. Thanks to this graph, the M-RRT solves manipulation problems. It uses the graph to generate admissible motions. A tree of configurations is grown by choosing randomly edges in the graph at each iteration.

The second section describes the crossed foliation transitions. The necessity of this type of transition arises from the crossed foliation issue. It is encountered in cases of continuous grasps and continuous placements. In those cases, randomly choosing new grasps and new placements has a probability zero to solve a manipulation problem.

The third section addresses the topology induced by the manipulation rules. I emphasize a limitation of the reduction property. I extend the property to articulated object, under-actuated end-effector and to the case of simultaneous grasps.

The last section describes the way-point transitions. This type of transition tackles the issue raised by narrow passages of $\mathcal{CS}$. These passages are known to drastically increase resolution time. Manipulation planning is intrinsically subject to narrow passages as objects must be close to either supporting objects or grippers.

3.1 Constraint Graph

3.1.1 States and transitions

I represent manipulation rules in the form of a graph called *Graph of Constraint*. The vertices of the graph are called *states* and the edges *transitions*. A state defines a subset of $\mathcal{CS}$, defined by a constraint[1].

Definition 3.1 (State)**.** *A* state S *contains a validation constraint* S.constraint *and a set of outgoing transitions* S.transitions.

A robot configuration $\mathbf{q}$ *is in* S*, denoted* $\mathbf{q} \in S$*, if and only if* $\mathbf{q}$ *satisfies* S.constraint.

A transition defines the set of admissible motions from a state.

Definition 3.2 (Transition)**.** *A* transition T *contains a parametrization function* $T.\mathbf{f}$*, an origin state* T.origState*, a destination state* T.dstState *and a state* T.state.

In the definition above, a transition T has three states. $T.origState$ and $T.dstState$ correspond to the states which are connected by the transition. The last state, $T.state$, corresponds to the state in which admissible paths lie. This state can be any state although, most of the time, it is $T.origState$ or $T.dstState$.

The parametrization function $T.\mathbf{f}$ is used to build parametrization constraints. Consider the foliation of the set $T.state$. This foliation is parametrized by $T.\mathbf{f}$. Given a configuration $\mathbf{q}_0$, the parametrization constraint $T.\mathbf{f}(\mathbf{q}) = \mathbf{b}$ with $\mathbf{b} = T.\mathbf{f}(\mathbf{q}_0)$ is satisfied for configurations on the same leaf as $\mathbf{q}_0$.

[1] I will often abusively use a state in place of the subset of $\mathcal{CS}$ it defines.

Definition 3.3 (Admissible motion)**.** *A path* $\mathbf{p} \in C([0,1], \mathcal{CS})$ *is* admissible for transition T*, or* T-admissible*, if and only if the following conditions are all satisfied:*

- $\mathbf{p}(0) \in T.\text{origState}$,
- $\forall t \in [0,1], \mathbf{p}(t) \in T.\text{state}$,
- $\forall t \in [0,1], T.\mathbf{f}(\mathbf{p}(t)) = T.\mathbf{f}(\mathbf{p}(0))$.

A path is admissible *if there exists a transition such that it is admissible for this transition.*

It is important to understand that the set of admissible motions from $T.origState$ is larger than the set of admissible motions from $T.origState$ to $T.dstState$. Another important point is that when $T.origState \cap T.state$ is empty, T admits no admissible paths and when $T.dstState \cap T.state$ is empty, T admits no admissible paths reaching the destination. Admissible motions do not take collisions into account. By design, I want the restriction of a path $\mathbf{p}$ to an interval of the form $[0,t] \subset [0,1]$ to still be admissible. This restriction happens in most motion planning algorithms, when a collision is detected at a time $T \in]t,1[$. Formally, the following property is satisfied[2]:

Definition 3.4 (Admissibility inheritance property)**.**

$$\forall \mathbf{p} \in C([0,1], \mathcal{CS}), \mathbf{p} \text{ is admissible } \Rightarrow \forall t \in [0,1], \mathbf{p}|_{[0,t]} \text{ is admissible}$$

From the notion of *admissible motions*, I define the *reachability set* as the set of reachable configurations from configuration $\mathbf{q}_0$ with transition T. To be reachable, a configuration must be on the same leaf as $\mathbf{q}_0$ of the foliation of $T.state$ induced by $T.\mathbf{f}$. In other words, it must satisfy the parametrization constraint $T.\mathbf{f}(\mathbf{q}) = T.\mathbf{f}(\mathbf{q}_0)$. Again, collisions are not taken into account. The *reachability set* is:

$$\mathcal{R}(\mathbf{q}, T) = \{\mathbf{q}\prime \in T.state \mid T.\mathbf{f}(\mathbf{q}\prime) = T.\mathbf{f}(\mathbf{q})\} \tag{3.1}$$

Example of Graph of Constraint Let us consider the problem of manipulation defined by a cylindrical vertical object like a bottle, a table and a simple 6 degrees of freedom (DoFs) manipulator arm with a gripper. The configuration space of the system is $\mathcal{CS} \equiv [-\pi, \pi]^6 \times SE(3)$. Figure 3.2 shows this set-up.

Figure 3.1 shows the Graph of Constraint of this simple example. This graph has two states

- *placement*: the subspace of configurations of the system where the object is standing on the table.
- *grasp*: the subspace of configurations where the object is grasped by the robot.

[2] $\mathbf{p}|_{[0,t]}$ is the restriction of $\mathbf{p}$ to $[0,t]$, *i.e.* $\mathbf{p}|_{[0,t]} = \begin{cases} [0,t] \to \mathcal{CS} \\ s \mapsto \mathbf{p}(s) \end{cases}$

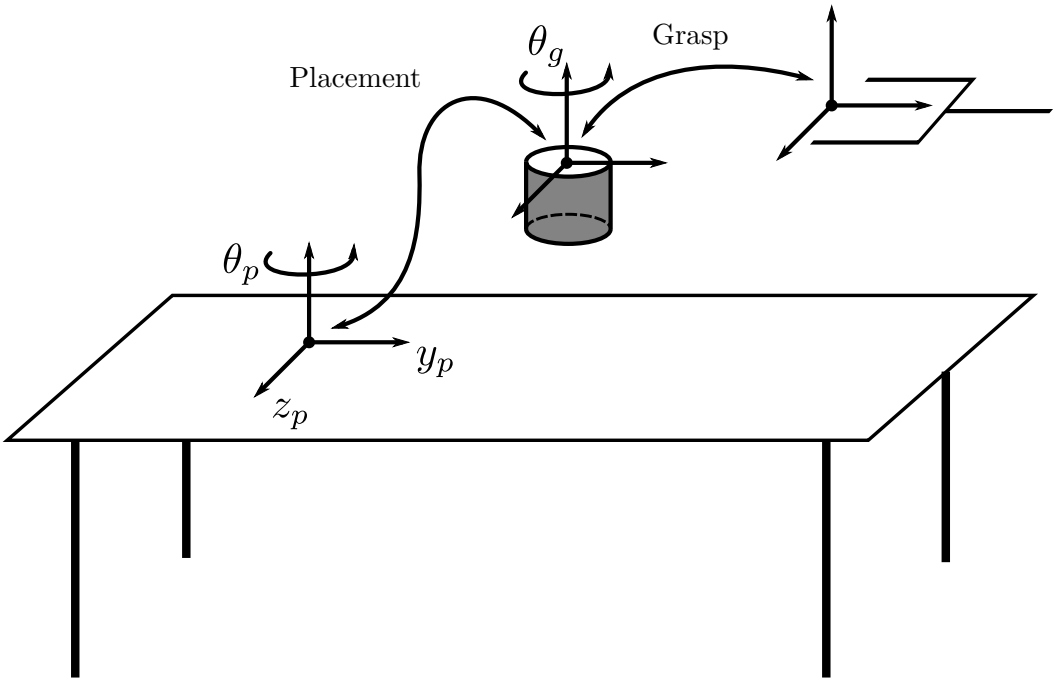


Figure 3.2: Illustration of grasp and placement constraints. A gripper can grasp a cylindrical object with a DoF in rotation. The cylindrical object can be placed on the table. The parameters y_p, z_p and θ_p are constrained by the placement parameterization constraint. They parameterize the foliation of the placement space. The parameter θ_g is constrained by the grasp parameterization constraint. It parameterizes the foliation of the grasp space.

Placement constraints Let $(x_p, y_p, z_p, \theta_p, \phi_p, \psi_p) \in \mathbb{R}^6$ be the position and orientation of the object. A configuration $\mathbf{q}$ can be written as $\mathbf{q} = (q_0, \ldots, q_5, x_p, y_p, z_p, \theta_p, \phi_p, \psi_p)$. The object is standing on the table when its height, roll and pitch are equal[3] to 0. The *validation constraint* of *placement* is defined by $\mathbf{P}(\mathbf{q}) = \mathbf{P}(q_0, \ldots, q_5, x_p, y_p, z_p, \theta_p, \phi_p, \psi_p) = [x_p, \phi_p, \psi_p] = \mathbf{0}$. Along *transit* paths, the position of the object on the table remains constant. This constraint defines a *foliation* of *placement*. Leaves of the foliation are parameterized by the 3 remaining DoFs of the object on *placement*: horizontal translation and yaw angle of the object. The *parametrization function* of placements is defined by $\bar{\mathbf{P}}(\mathbf{q}) = \bar{\mathbf{P}}(q_0, \ldots, q_5, x_p, y_p, z_p, \theta_p, \phi_p, \psi_p) = [y_p, z_p, \theta_p]$.

Grasp constraints In the discrete case, where *grasp* is defined by a fixed relative transformation of the gripper with respect to the object, *grasp* is not foliated (or composed of only one leaf). The validation constraint is a fixed relative transformation of the gripper with respect to the object. There is no parametrization function.

[3]There always exists a coordinate system in which this is true.

The case of continuous grasps is a more interesting. The validation constraint sets the value of the 3 translations and 2 of the 3 rotations of the relative transformation of the gripper with respect to the cylindrical object. The parametrization function returns the relative angle around the cylinder axis, θ_g on Figure 3.2, left free by the validation constraint.

Graph of Constraint State *grasp* contains the grasp validation constraint and *placement* contains the placement validation constraint. Transitions *transit* and *grasp object* contain the placement parametrization function. Their *state* is *placement* because admissible motions for these transition must lie in *placement*. Transitions *transfer* and *release object* contain no function in the fixed grasp case and the grasp parametrization function in the continuous grasp case. Their *state* is *grasp* because admissible motions for these transitions must lie in *grasp*.

Figure 3.3 shows the configuration space for a pick-and-place problem. The two foliations of $\mathcal{CG}$ and $\mathcal{CP}$ are crossing. Moving in $\mathcal{CS}$ requires to find motions in A_{f_i} and B_{g_i} alternatively.

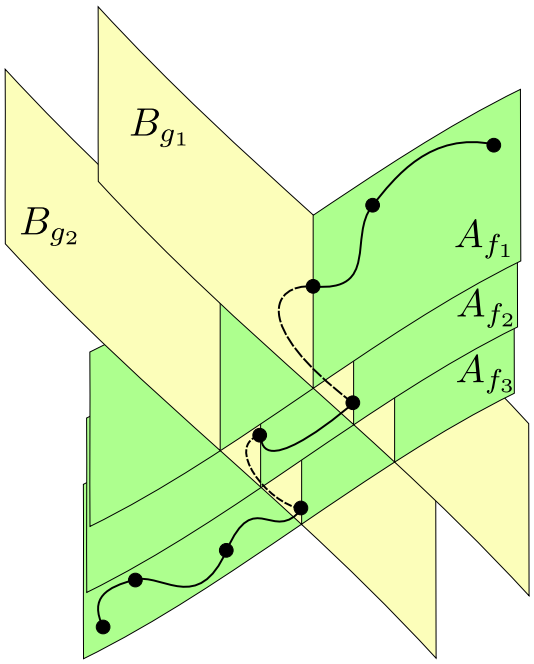


Figure 3.3: Example of manipulation path and reachability sets with two foliations. Each A_{f_i} is a leaf of $\mathcal{CP}$. Each B_{g_i} is a leaf of $\mathcal{CG}$. f_i and g_i are continuous and only some values are shown. Solid lines are *transit paths*, along the A_{f_i} and dashed lines are *transfer paths*, along the B_{g_i}.

State of a configuration It is very often useful to obtain a state from a configuration $\mathbf{q} \in \mathcal{CS}$. This is however ill-defined for the two following reasons.

$\mathbf{q}$ may satisfy the constraint of several states. For instance, a configuration in

the intersection of *grasp* and *placement* satisfies both constraint of *placement* and *grasp*. I chose to overcome the issue by prioritizing the states. If $\mathbf{q}$ satisfies the constraint of two states, it belongs to the state of highest priority. If a state is included in another state, the former must have a higher priority than the latter. When there is no inclusion relation, the order is ill-defined and the algorithm relies on user input. For *placement* and *grasp*, it is usually better to consider *placement* with lower priority than *grasp*. Articulated object, like a door, is always in a valid placement so $\mathcal{CP} = \mathcal{CS}$ and $\mathcal{CG} \subset \mathcal{CP}$ and $\mathcal{CG}$ must have higher priority.

$\mathbf{q}$ may also violate the constraint of all states. Then, $\mathbf{q}$ does not belongs to any states. Except before projecting random configurations, this does not happen. This is one of the motivation for the *state* attribute of transition. Configurations along a motion generated using the graph lie in *state*.

Relation with task planning notions In task planning, the manipulation rules are encoded in the action preconditions and effects. The notion of action does not exist in this formalism. States encode preconditions and effects. Transitions encode admissible motions.

3.1.2 Problem statement

A manipulation planning problem can be defined as follows.

Definition 3.5 (Manipulation planning problem)**.** *Given a set of robots, objects, static obstacles and a Graph of Constraint, an initial configuration for all robots and objects and a set of goal configurations, then a* manipulation planning problem *is to find a path, for all robots and all objects, from the initial to a final position in the set of goal configurations. The path must be collision-free and must satisfy the* manipulation rules*.*

In the above definition, the goal specification is the most abstract. A set of configuration can be specified by a validation constraint. Though this work could be extended in order to handle a set of goal configurations, I only focus on a discrete countable set of goal configurations. This restriction affects only the proposed algorithm, which would have to be slightly modified. In the literature, this problem is known as Rearrangement planning. All that follows focuses on the following problem:

Definition 3.6 (Rearrangement planning problem)**.** *Given a set of robots, objects, static obstacles and a Graph of Constraint, an initial and one or several goal configuration(s) for all robots and objects then a* manipulation planning problem *is to find a path, for all robots and all objects, from the initial to a final position in the set of goal configurations. The path must be collision-free and must satisfy the* manipulation rules*.*

3.1.3 Manipulation RRT

In this section, I explain how to use the Graph of Constraint to plan manipulation paths. The RRT-like planning algorithm developed to solve manipulation problems is called the Manipulation-RRT algorithm.

For each state of the Graph of Constraint, I define a probability distribution over all transitions starting from this state. By default the uniform distribution is a reasonable option.

Algorithm 3.1 Manipulation-RRT

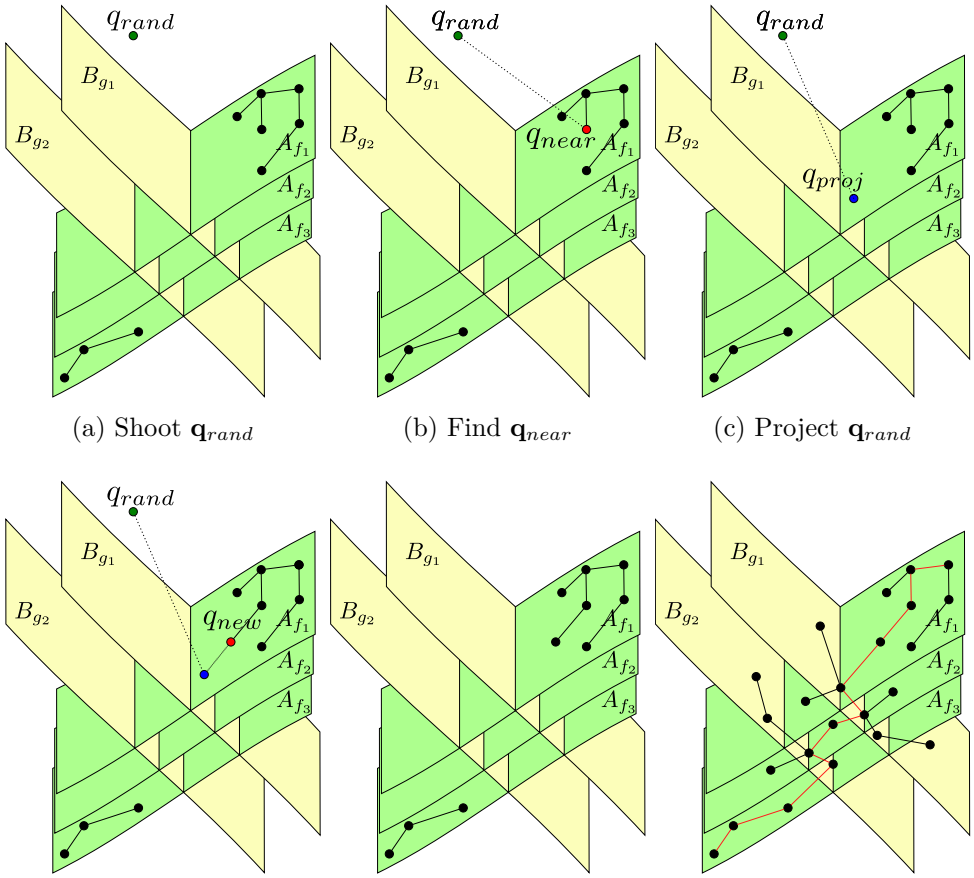
▷ Rapidly exploring Random Tree (RRT) from $\mathbf{q}^I$ to one goal $\mathbf{q}_i^G$ using the constraint graph

1: **function** FINDPATH($\mathbf{q}^I, \left\{\mathbf{q}_1^G, \ldots, \mathbf{q}_n^G\right\}$)
2: $\mathcal{T}$.INSERT($\mathbf{q}^I$, $\mathbf{q}_1^G$, …, $\mathbf{q}_n^G$)
3: **for** $i = 1 \to K$ **do**
4: $\mathbf{Q} \leftarrow$ EMPTYSET
5: $\mathbf{q}_{rand} \leftarrow$ RANDOM($\mathcal{CS}$)
6: **for all** connected component cc of $\mathcal{T}$ **do**
7: $\mathbf{q}_{near} \leftarrow$ NEAREST($\mathbf{q}_{rand}$,cc)
8: $T \leftarrow$ CHOOSETRANSITION($\mathbf{q}_{near}$)
9: $\mathbf{q}_{new}, path \leftarrow$ CONSTRAINEDEXTEND($\mathbf{q}_{near}$,$\mathbf{q}_{rand}$,T)
10: **if** last step succeeded **then**
11: $\mathbf{Q}$.APPEND($\mathbf{q}_{new}$)
12: $\mathcal{T}$.INSERT($\mathbf{q}_{new}$, $path$)
13: **for all** $(\mathbf{q}_1, \mathbf{q}_2) \in \mathbf{Q}^2 \mid \mathbf{q}_1 \neq \mathbf{q}_2$ **do**
14: CONNECT($\mathbf{q}_1, \mathbf{q}_2$)
15: **for all** $\mathbf{q}^G \in \left\{\mathbf{q}_1^G, \ldots, \mathbf{q}_n^G\right\}$ **do**
16: **if** SAMECONNECTEDCOMPONENT($\mathbf{q}^I$, $\mathbf{q}^G$) **then return** path found

Algorithm 3.1 shows a pseudo code for the M-RRT algorithm. Figure 3.4 pictures some of the steps, with the example of Figure 3.3. The algorithm consists in exploring the transitions of the Graph of Constraint as follows.

- Figure 3.4(a), Line 5: shoot a random configuration $\mathbf{q}_{rand}$.
- Figure 3.4(b), Line 7: in each connected component of the current roadmap, find the closest node $\mathbf{q}_{near}$.
- Line 8: choose an outgoing transition of the state of node $\mathbf{q}_{near}$ in the Graph of Constraint. In all the simulations of this work, this choice is random and follows the uniform probability distribution over all outgoing transitions of each state.
- Figure 3.4(d), Line 9: extend $\mathbf{q}_{near}$ along the transition up to $\mathbf{q}_{new}$, further detailed in Algorithm 3.2.

- Line 14: try to connect the new nodes together, using Algorithm 3.3.
- Figure 3.4(f), Line 16: the algorithm succeeds when $\mathbf{q}^I$ and one $\mathbf{q}_i^G$ are in the same connected component.



(a) Shoot $\mathbf{q}_{rand}$ (b) Find $\mathbf{q}_{near}$ (c) Project $\mathbf{q}_{rand}$

(d) Extend toward $\mathbf{q}_{proj}$ (e) Update the roadmap (f) Solution found

Figure 3.4: Steps of the M-RRT algorithm. (a)-(e) shows the steps of one iteration of the algorithm. $\mathbf{q}_{rand}$ is a random configuration. $\mathbf{q}_{near}$ is its nearest neighbour. $\mathbf{q}_{proj}$ is the projection of $\mathbf{q}_{rand}$ onto the reachability set. $\mathbf{q}_{new}$ is the configuration obtained after extension. (f) shows the roadmap when a solution was found, after more iterations.

Constrained extension Algorithm 3.2 describes the constrained extension step above (Line 9 of Algorithm 3.1). It extends the functionalities of Algorithm 2.3. C_t denotes the *parametrization constraint* that defines $\mathcal{R}(\mathbf{q}_{near}, T)$, the *reachability set* from $\mathbf{q}_{near}$ with transition T (Line 3). C_s denotes the concatenation of the *validation constraint* of target state $T.dstState$ and transition state $T.state$ (Line 2). $\mathbf{q}_{proj}$ is obtained by projecting $\mathbf{q}_{rand}$ onto $C_t \cap C_s$. This corresponds to Line 5 and Figure 3.4(c). One of the path projection method of Section 2.2 is used to

Algorithm 3.2 Continuous constrained RRT extension using the Constraint Graph

```
1: function CONSTRAINEDEXTEND(q_near, q_rand, T)
2:     C_s ← T.dstState.constraint ∩ T.state.constraint
3:     C_t ← PARAMETRIZATIONCONSTRAINT(f = T.f, b = T.f(q_near))
4:     proj ← PROJECTOR(C_s, C_t)
5:     q_proj ← proj.APPLY(q_rand)
6:     p_1 ← INTERPOLATE(q_near, q_proj)
7:     pathProj ← PROJECTOR(T.state.constraint, C_t)
8:     p_2 ← pathProj.APPLY(p_1)
9:     p_3 ← TESTCOLLISION(p_2)
10:    q_new ← FINALCONFIGURATION(p_3)
11:    return q_new, p_3
```

project the path onto the transition constraint (Line 8). Then, a collision checking procedure validates the path and the algorithm returns.

In case of collision or projection failure, p_3 is the left part of the path that is successfully projected and collision-free. In this case, configuration $\mathbf{q}_{new}$ is likely not to be in $T.dstState$. Thanks to the admissibility inheritance property 3.4, p_3 is still an admissible path.

Algorithm 3.3 Connect two configurations with the Graph of Constraint

```
1: function CONNECT(q_1, q_2)
2:     S_1 ←STATE(q_1)
3:     S_2 ←STATE(q_2)
4:     T ←TRANSITION(S_1, S_2)
5:     if T is None then return failure
6:     if T.f(q_2) ≠ T.f(q_1) then return failure
7:     C_t ← PARAMETRIZATIONCONSTRAINT(f = T.f, b = T.f(q_1))
8:     proj ← PROJECTOR(C_t)
9:     p_1 ← INTERPOLATE(q_near, q_proj)
10:    p_2 ← PROJECTPATH(p_1, proj)
11:    p_3 ← TESTCOLLISION(p_2)
12:    if no collision and projection success then return p_3
13:    else return failure
```

Termination condition Algorithm 3.3 tries to connects the new nodes of the roadmap created by Algorithm 3.2 to other connected components of the roadmap, as in any classical implementation of RRTConnect algorithm. First the function looks for a transition between the configurations. If a transition exists, the function checks that $\mathbf{q}_1$ and $\mathbf{q}_2$ are on the same leaf of the transition foliation. If not, the function returns failure.

In Figure 3.4(f), the leaf B_{g_1} and B_{g_2} were selected randomly at Line 5 of Algorithm 3.2. This leads to the crossed foliation issue, explained in the following section.

3.2 Crossed foliation issue

This section introduces the *crossed foliation issue* first by an example, then by a formal approach. Finally, I explain the solution I used to handle it.

3.2.1 Example

Consider the example given in Section 3.1.3: a cylindrical vertical object, a table and a simple 6 DoFs manipulator arm. Consider continuous placements of the object on the table. In the fixed grasp case, *grasp* is not foliated (or composed of only one leaf). Extending a RRT-tree from the initial configuration and from the goal configuration will solve the problem of moving the object at another configuration. The trees will connect in *grasp* space. Figure 3.5 illustrates this case.

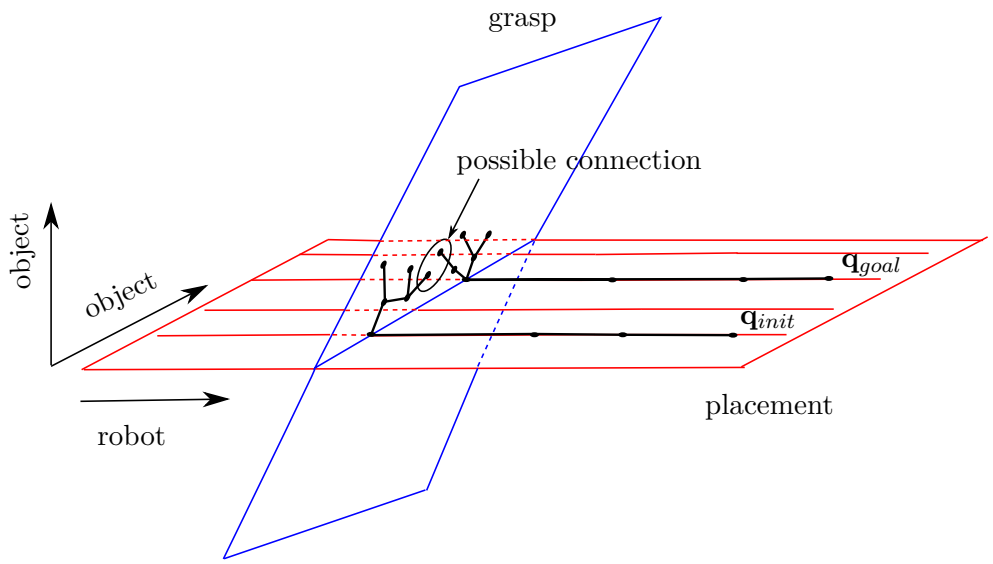


Figure 3.5: Grasp and placement with unique grasp: If the grasp is unique, any pair of grasp configurations are connectible by a *transfer path*.

In the general case, objects can be grasped in a continuous way. In the above example, the relative orientation of the object with respect to the gripper is free around the vertical axis of the object. In this case, *grasp* is also foliated. The foliation is parametrized by the relative angle of the object with respect to the gripper along the object axis. In this case, RRT-trees rooted in placement will never meet, since the probability that both trees start exploring *grasp* with the same grasp is equal to zero. I call this issue the *crossed foliation issue*. Figure 3.6 illustrates this case.

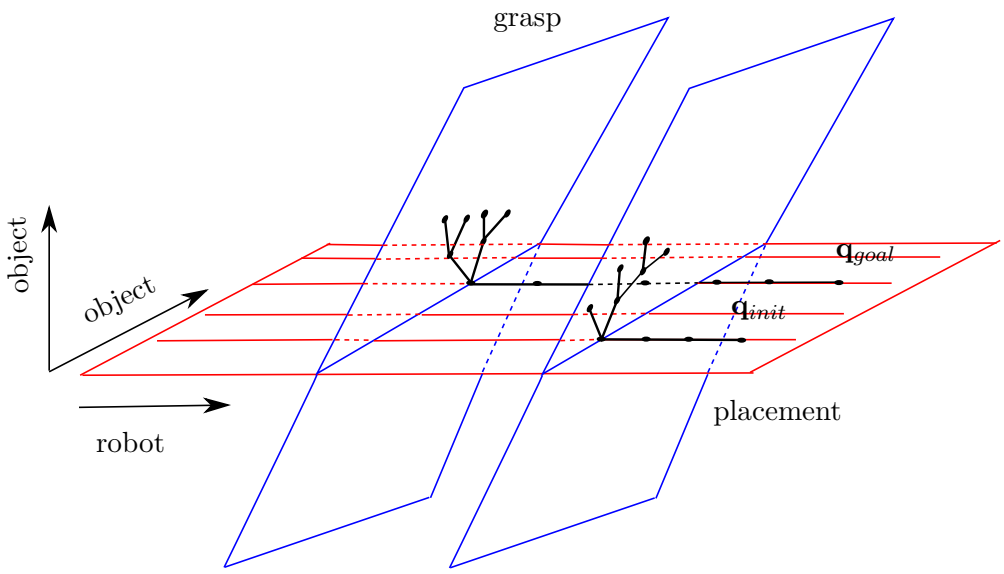


Figure 3.6: The crossed foliation issue: if *grasp* and *placement* are foliated, trees rooted at initial and goal configurations will never meet.

To my best knowledge, this issue has never been stated and solved in a general way. Note that Siméon et al. [2004], Harada et al. [2014], Lertkultanon and Pham [2015] overcome the issue by using the reduction property. However, the issue also arises manipulating two objects with discrete grasps and continuous placement. In the case, the reduction property does not apply.

3.2.2 Conditions

The *crossed foliation issue*, pictured in the above section, arises when two transitions connecting two nodes back and forth are both foliated. It happens at Line 6 of Algorithm 3.3, if states S_1 and S_2 are foliated. As explained in Figure 3.6, it will always return failure and will never succeed in connecting configurations from different trees. The main result of this section, Corollary 3.1, gives the conditions raising the issue.

To formalize the issue, I first define the leaf parameter set and one useful property for projectors.

Definition 3.7 (Leaf parameter set)**.** *For a transition T and a graph $\mathcal{T}$, the set of leaf parameters, of foliation induced by T, reached by $\mathcal{T}$ is defined by*

$$L(T, \mathcal{T}) = \{T.\mathbf{f}(\mathbf{q}) | \mathbf{q} \in \mathcal{T} \cap T.\text{state}\}$$

For instance, Figure 3.4 shows two foliations parametrized by function f for the A_{f_i} and g for the B_{g_i}. On Figure 3.4(a), the two leaf parameter sets are $L(T_f, \mathcal{T}) = \{f_1, f_3\}$ and $L(T_g, \mathcal{T}) = \emptyset$, while on Figure 3.4(f), they are $L(T_f, \mathcal{T}) = \{f_1, f_2, f_3\}$ and $L(T_g, \mathcal{T}) = \{g_1, g_2\}$.

Property 3.1 (Unbiased projector)**.** *Let M be a sub-manifold of $\mathcal{CS}$ of positive dimension and C be a constraint satisfied only on M, i.e. $M = \{\mathbf{q} \in \mathcal{CS} \mid \mathbf{q}$ satisfies $C\}$. Finally, let* proj *be a projector onto constraint C.*

Then, proj *is an* unbiased *projector if the pre-image of any subset of M of measure zero by* proj *is a subset of $\mathcal{CS}$ of a measure zero.*

Property 3.1 can be interpreted as follows. An unbiased projector does not prefer any subset of the output manifold M of measure zero. Let $\mathbf{q}^R$ be a uniformly distributed random configuration of $\mathcal{CS}$ and $\mathbf{q}^R_{proj}$ be its projection. Indeed, $\mathbf{q}^R_{proj}$ has a probability zero of reaching any subset of M of measure zero:

$$\forall S \subset M \mid |S| = 0, p(\mathbf{q}^R_{proj} \in S) = 0$$

Note that an unbiased projector may - and will in the general case - prefer some subset of M of strictly positive measure. Two subsets of M of same non-zero measure have not, in general, the same probability of being reached by $\mathbf{q}^R_{proj}$.

In the context of constrained motion planning, this is a desirable property. It is a necessary - but not sufficient - condition to have a uniform distribution for variable $\mathbf{q}_{proj}$ in M from a uniform distribution for variable $\mathbf{q}_{rand}$ in $\mathcal{CS}$. In this framework, most constraints are infinitely differentiable. As projection uses a Newton-Raphson (NR) method, it is reasonable that the property is true[4]. The pre-image of a configuration is the set of configurations obtained by following forward and backward the gradient of the constraint. The property can be violated by the placement constraint that I introduce later. This constraint is only piecewise infinitely differentiable. Figure 3.7 gives an example of biased projector because of a placement constraint.

The following theorem formalizes this issue.

Theorem 3.1 (Probability to generate reachable configuration)**.** *Let S_1 and S_2 be two states. For $1 \leq i, j \leq 2, i \neq j$, $T_{i,j}$ be transitions from S_i to S_j such that $T_{i,j}$ induces a foliation of $T_{i,j}$.state $= S_i$ parametrized by $T_{i,j}.\mathbf{f}$.*

Let $(\mathbf{q}^1, \mathbf{q}^2) \in {S_1}^2$. For $k \in \mathcal{N}$, let $\mathcal{T}^1_k$ and $\mathcal{T}^2_k$ be two sequences of trees such that $\mathcal{T}^i_0 = \{\mathbf{q}^i\}$ and $\mathcal{T}^i_{k+1}$ is the tree obtained from $\mathcal{T}^i_k$ after one iteration of Algorithm 3.1, using only transition $T_{1,2}$. Let q^i_{k+1} be the configuration added to $\mathcal{T}^i_k$ to obtain $\mathcal{T}^i_{k+1}$.

Finally, let E_k and F_k be the events "$L(T_{2,2}, \mathcal{T}^1_k) \cap L(T_{2,2}, \mathcal{T}^2_k) = \emptyset$" and "$\mathbf{q}^1_k \notin L(T_{2,2}, \mathcal{T}^2_k)$ and $\mathbf{q}^2_k \notin L(T_{2,2}, \mathcal{T}^1_k)$" respectively. Then, the probability of $E_k, k \geq 1$ is:

$$p(E_k) = p(E_0) \prod_{i=0}^{k-1} p(F_{i+1} \mid E_i)$$

Before providing, I explain what are each term in the above theorem. The two trees $\mathcal{T}^i_k$ are the tree normally grown by the M-RRT. Event E_k means that, at the begining of iteration $k+1$, no configuration from $\mathcal{T}^1_k$ are reachable from

[4]Smooth enough functions have the 0-property, which corresponds to Property 3.1. See Ponomarev [1987] for more details.

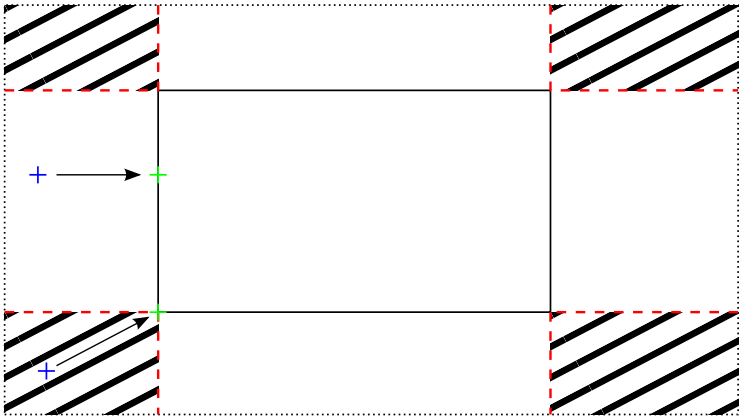


Figure 3.7: This simple example shows a biased projector. The space represents the position of an object in a two dimensional plane. The square at the center represents the set of stable placement (like a table). Assume the projection returns the stable placement closest to the input, as shown by the two arrows. An object in the hatched regions is unstable and its projection would be at the closest corner of the table. Thus, one corner has a non-zero probability of being sampled after projection. If the table had round corners, the projector would be unbiased.

a configuration in $\mathcal{T}_k^2$ and vice-versa. Event F_{k+1} means that configuration $\mathbf{q}_{k+1}^1$ generated by the $k+1$-th iteration is not reachable from configurations in $\mathcal{T}_{k+1}^2$ and vice versa.

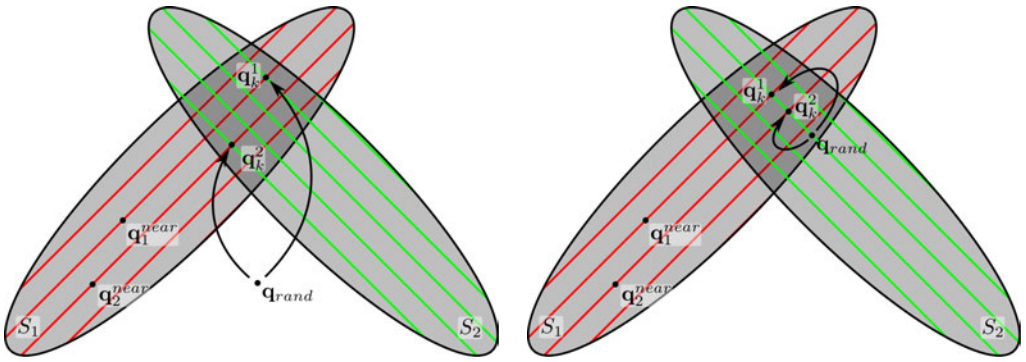
Assume Figure 3.4 shows the $k+1$-th iteration. Then, Figure 3.4(a) shows the roadmap obtained at the begining of interation $k+1$, $\mathcal{T}_k^i$. Event E_k happens since the two trees $\mathcal{T}_k^i$ are on different B_{g_i}. $\mathbf{q}_{k+1}^1$, the configuration $\mathbf{q}_{new}$ on Figure 3.4(d), is not on a B_{g_i} reached by $\mathcal{T}_{k+1}^2$. Assuming the same assertion is true for $\mathbf{q}_k^2$, not on this Figure, then event F_k happens. In other words, the two trees could not be connected before iteration $k+1$ (event E_k) and iteration $k+1$ did not change it (event F_{k+1}).

Thus the probability of E_k is the probability that $T_{1,2}$ generates configuration that are reachable from each other using transition $T_{2,2}$. When $p(F_{k+1} \mid E_k) < 1$, then $\lim_{k\to\infty} p(E_k) = 0$, which is desirable otherwise the problem cannot be solved. I now prove the above theorem.

Proof. After iteration k, event E_{k+1} happens if $L(T_{2,2}, \mathcal{T}_{k+1}^1) \cap L(T_{2,2}, \mathcal{T}_{k+1}^2) = \emptyset$. As $\mathcal{T}_k^i \subset \mathcal{T}_{k+1}^i$, event E_{k+1} implies $L(T_{2,2}, \mathcal{T}_k^1) \cap L(T_{2,2}, \mathcal{T}_k^2) = \emptyset$, $\mathbf{q}_{k+1}^1 \notin L(T_{2,2}, \mathcal{T}_{k+1}^2)$ and $\mathbf{q}_{k+1}^2 \notin L(T_{2,2}, \mathcal{T}_{k+1}^1)$, *i.e.* events E_k and F_{k+1}. So event E_{k+1} happens only if E_k happened and F_{k+1} happens knowing E_k happened. Thus, we get the following recursive rule.

$$p(E_{k+1}) = p(E_k) \times p(F_{k+1} \mid E_k)$$

So $P(E_k) = P(E_0) \prod_{i=0}^{k-1} p(F_{i+1} \mid E_i)$ □



(a) $\mathbf{q}_k^1$ and $\mathbf{q}_k^2$ are not reachable.

(b) $\mathbf{q}_k^1$ and $\mathbf{q}_k^2$ are reachable.

Figure 3.8: Reachability of configurations generated by Manipulation-RRT at iteration k. The two gray sets are states S_1 and S_2. The coloured lines are leaves of the foliation of each state. $\mathbf{q}_k^i$ is the projection of $\mathbf{q}_{rand}$ onto the intersection of $S_1 \cap S_2$ with the red line passing by $\mathbf{q}_i^{near}$. (a) shows the general case where $\mathbf{q}_k^1$ and $\mathbf{q}_k^2$ are not in the same foliation of S_2, *i.e.* on the same green line. (b) shows a particular case where the projection is made orthogonally to the green lines.

I now analyse the value of $p(F_k \mid E_{k-1})$. Figure 3.8 shows how $\mathbf{q}_k^1$ and $\mathbf{q}_k^2$ are generated by M-RRT. Let $\mathbf{q}_{rand} \in \mathcal{CS}$ be a random configuration and $\mathbf{q}_i^{near}$ its nearest neighbour in $\mathcal{T}_{k-1}^i$. F_k can be rewritten as "$\mathbf{q}_k^1 \notin L(T_{2,2}, \mathcal{T}_{k-1}^2)$ and $\mathbf{q}_k^2 \notin L(T_{2,2}, \mathcal{T}_{k-1}^1)$ and $T_{2,2}.\mathbf{f}(\mathbf{q}_k^1) \neq T_{2,2}.\mathbf{f}(\mathbf{q}_k^2)$". Then,

$$\begin{aligned} p(F_k \mid E_{k-1}) = &(1 - p(T_{2,2}.\mathbf{f}(\mathbf{q}_k^1) \in L(T_{2,2}, \mathcal{T}_{k-1}^2) \mid E_{k-1})) \\ &\times (1 - p(T_{2,2}.\mathbf{f}(\mathbf{q}_k^2) \in L(T_{2,2}, \mathcal{T}_{k-1}^1) \mid E_{k-1})) \\ &\times (1 - p(T_{2,2}.\mathbf{f}(\mathbf{q}_k^1) = T_{2,2}.\mathbf{f}(\mathbf{q}_k^2) \mid E_{k-1})) \end{aligned}$$

Let $C(\mathbf{q})$ be the intersection of S_2 and the leaf of S_1 of parameter $T_{1,2}.\mathbf{f}(\mathbf{q})$. When extending $\mathcal{T}_{k-1}^i$ with $T_{1,2}$, the constraint applied to generate $\mathbf{q}_k^i$ are $C(\mathbf{q}_i^{near})$. Let $F^{-1}(\mathbf{q}^*, \mathbf{l}) = \{\mathbf{q} \in C(\mathbf{q}^*) \mid T_{2,2}.\mathbf{f}(\mathbf{q}) = \mathbf{l}\}$ be the pre-image of $\mathbf{l}$ by $T_{2,2}.\mathbf{f}$ in $C(\mathbf{q}^*)$. Let $P^{-1}(\mathbf{q}^*, \mathbf{l}) = \{\mathbf{q} \in \mathcal{CS} \mid \mathit{projector}(\mathbf{q}, C(\mathbf{q}^*)) \in F^{-1}(\mathbf{q}^*, \mathbf{l})\}$ be the pre-image of $F^{-1}(\mathbf{q}^*, \mathbf{l})$ by *projector* onto $C(\mathbf{q}^*)$). It is the subset of $\mathcal{CS}$ whose projection onto $C(\mathbf{q}^*)$ is in the leaf of parameter $\mathbf{l}$, *i.e.* $T_{2,2}.\mathbf{f}(\mathbf{q}_{proj}) = \mathbf{l}$. Let's consider each term of the above equation independently.

- The first term is $p(T_{2,2}.\mathbf{f}(\mathbf{q}_k^1) \in L(T_{2,2}, \mathcal{T}_{k-1}^2) \mid E_{k-1})$. We have $\{\mathbf{q}_{rand} \in \mathcal{CS} \mid T_{2,2}.\mathbf{f}(\mathbf{q}_k^1) \in L(T_{2,2}, \mathcal{T}_{k-1}^2)\} \subset \bigcup_{(\mathbf{q}_1', \mathbf{q}_2') \in \mathcal{T}_{k-1}^1 \times \mathcal{T}_{k-1}^2} P^{-1}(\mathbf{q}_1', T_{2,2}.\mathbf{f}(\mathbf{q}_2'))$. It is only a subset because of the nearest neighbour search. In the general case, $F^{-1}(\mathbf{q}^*, \mathbf{l})$ is a measure zero subset of $C(\mathbf{q}*)$. This happens for instance when $T_{2,2}.\mathbf{f}$ is continuous and non-redundant with constraint $C(\mathbf{q}*)$. In this case, $P^{-1}(\mathbf{q}^*, \mathbf{l})$ is the pre-image of a set of measure zero by the projector. If the projector is unbiased, then $P^{-1}(\mathbf{q}^*, \mathbf{l})$ has measure zero and $p(T_{2,2}.\mathbf{f}(\mathbf{q}_i^{proj}) \in L(T_{2,2}, \mathcal{T}_{k-1}^{3-i}) \mid E_{k-1})) = 0$.

- The second term $p(T_{2,2}.\mathbf{f}(\mathbf{q}_k^2) \in L(T_{2,2}, \mathcal{T}_{k-1}^1) \mid E_{k-1})$ is symmetrical.
- The third term is $p(T_{2,2}.\mathbf{f}(\mathbf{q}_k^1) = T_{2,2}.\mathbf{f}(\mathbf{q}_k^2) \mid E_{k-1})$. This probability is the probability of sampling a random configuration whose projections $\mathbf{q}_k^1$ and $\mathbf{q}_k^2$ are such that $T_{2,2}.\mathbf{f}(\mathbf{q}_k^)) = T_{2,2}.\mathbf{f}(\mathbf{q}_k^2)$. This is $p(\mathbf{q}_{rand} \in \bigcup_{\mathbf{l}\in\mathbb{R}^n} \left(P^{-1}(\mathbf{q}_1^{near}, \mathbf{l}) \cap P^{-1}(\mathbf{q}_2^{near}, \mathbf{l})\right)$. I consider two cases: $T_{1,2}.\mathbf{f}(\mathbf{q}_1^{near}) = T_{1,2}.\mathbf{f}(\mathbf{q}_2^{near})$ and $T_{1,2}.\mathbf{f}(\mathbf{q}_1^{near}) \neq T_{1,2}.\mathbf{f}(\mathbf{q}_2^{near})$.

  In the first case, the constraints applied to generate $\mathbf{q}_k^1$ and $\mathbf{q}_k^2$ are the same. The two projected configurations are thus equal. This case is not very interesting because it means the two trees are already reachable by $T_{1,1}$.

  In the second case, the constraints applied to generate $\mathbf{q}_k^1$ and $\mathbf{q}_k^2$ are different. It is hard in the general case to estimate the probability. Figures 3.8(a) and 3.8(b) show what can happen. On Figure 3.8(b), the projection is orthogonal to the green lines. This means that the projection did not influence $T_{2,2}.\mathbf{f}$. This can happen is the configuration space is a vector space, $T_{1,2}.\mathbf{f}$ and $T_{2,2}.\mathbf{f}$ are linear and the green lines are orthogonal to the red ones. Although I cannot provide a proof, I believe that with a non-linear configuration space and non-linear constraints, the measure of $\bigcup_{\mathbf{l}\in\mathbb{R}^n} \left(P^{-1}(\mathbf{q}_1^{near}, \mathbf{l}) \cap P^{-1}(\mathbf{q}_2^{near}, \mathbf{l})\right)$ is very low, if not zero. And so is the probability. This seemed verified in practice as random exploration without the crossed foliation transition, detailed in next section, was never able to solve problems with crossing foliations in the amount of time I let it run.

Definition 3.8 (Correlated constraint)**.** *A projector onto $C(\mathbf{q})$ is correlated with parametrization function $\mathbf{f}$ if and only if $\forall(\mathbf{q}_1, \mathbf{q}_2) \in \mathcal{CS}^2$,*

$$\left| \bigcup_{\mathbf{l}\in\mathbb{R}^n} \left(P^{-1}(\mathbf{q}_1, \mathbf{l}) \cap P^{-1}(\mathbf{q}_2, \mathbf{l})\right) \right| = 0$$

where $P^{-1}(\mathbf{q}, \mathbf{l})$ is the subset of $\mathcal{CS}$ whose projection onto $C(\mathbf{q})$ is in the pre-image of $\mathbf{l}$ by $\mathbf{f}$.

Corollary 3.1 (Crossed foliation issue)**.** *Consider the case of Theorem 3.1. Let $C(\mathbf{q})$ be the intersection of S_2 and the leaf of S_1 of parameter $T_{1,2}.\mathbf{f}(\mathbf{q})$ and let* proj *be the projector onto $C(\mathbf{q})$. When* proj *is unbiased,* proj *and $T_{2,2}.\mathbf{f}$ are correlated and when $T_{1,1}.\mathbf{f}(\mathbf{q}^1) \neq T_{1,1}.\mathbf{f}(\mathbf{q}^2)$, the probability of randomly generating reachable configurations for $T_{2,2}$ is zero.*

Proof. From the comments above, $p(F_{k+1} \mid E_k) = 1$ so, using Theorem 3.1, $p(E_k) = 1$. □

When there are no correlation as defined in Definition 3.8, a value very close to one of $p(F_{k+1} \mid E_k)$ means the likelihood to solve the problem is very low. When this is true for both transition $T_{2,1}$ and $T_{1,2}$, then, the M-RRT algorithm will run forever. To address this issue, next section introduces the *cross foliation*

transition. This transition ensures that $p(T_{2,2}.\mathbf{f}(\mathbf{q}_k^1) \in L(T_{2,2}, \mathcal{T}_{k-1}^2) \mid E_{k-1}) = 1$ and $p(T_{2,2}.\mathbf{f}(\mathbf{q}_k^2) \in L(T_{2,2}, \mathcal{T}_{k-1}^1) \mid E_{k-1}) = 1$.

3.2.3 Crossed foliation transition

To overcome the *crossed foliation issue*, I define a new type of transition called *crossed foliation transition*. This transition generates new configurations in leaves that have already been reached by other connected components. To achieve this, the roadmap builds and stores the leaf parameter set reached by each connected component of the roadmap, of each transition of this type. In the Graph of Constraint, a transition of this type must be inserted parallel to transitions inducing a foliation. It must also know the parametrization function of the crossing foliation. Figure 3.9 shows how the graph of Figure 3.1 must be modified to overcome the issue. Two *crossed foliation transitions* have been added, from *placement* to *grasp* and vice versa. Transition *Grasp object with specific grasp* knows the function of transition *Transfer*.

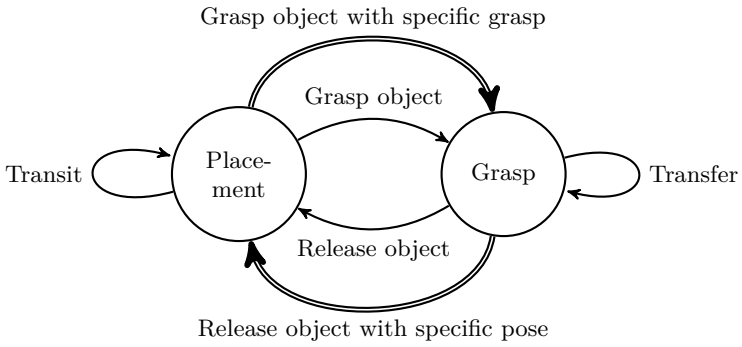


Figure 3.9: Graph of Constraint for a problem with a robot with one gripper manipulating one object. The double line represents *crossed foliation transition*.

The graph on Figure 3.9 contains two states S_1 and S_2, four transitions $T_{i\to j}, 1 \leq i, j \leq 2$ from S_i to S_j and two crossed foliation transition $T_{1\to 2}^{cf}$ and $T_{2\to 1}^{cf}$. When extending a node belonging to state S_1, Algorithm 3.1 randomly chooses between the three outgoing transitions. When it selects $T_{1\to 2}^{cf}$, the extension algorithm picks randomly an element $\mathbf{l}$ of the leaf parameter set of $T_{2\to 2}$ reached by any other connected component of the current roadmap. Constraint $T_{2\to 2}.\mathbf{f}(\mathbf{q}) = \mathbf{l}$ is then added to the projector (Line 4 of Algorithm 3.2).

By definition of the leaf parameter set $L(T_{2\to 2}, \mathcal{T})$, $\mathbf{l}$ is associated to one or several configuration(s) of the roadmap lying in state S_2 and satisfying $T_{2\to 2}(\mathbf{q}) = \mathbf{l}$. As a consequence, $\mathbf{q}_{proj}$ is on the same leaf as those configurations for the transition $T_{2\to 2}$. Eventually, Algorithm 3.3 may connect these configurations.

3.3 Generalized reduction property

Siméon et al. [2004], Harada et al. [2014], Lertkultanon and Pham [2015] do not encounter the crossed foliation issue. They overcome the issue by using the reduction property. This property reduces the complexity of the problem with a robot manipulating one object to a constrained motion planning problem. The intersection of grasp and placement spaces is not foliated. However, the reduction property, recalled in Section 1.3.2, only covers cases where a full actuated gripper grasps a free floating object. It does not cover the cases of two grippers manipulating the same object, of "non-holonomic" grippers and of articulated objects. In this section, I propose a generalized reduction property. It extends the reduction property to the cases mentioned above. This generalization relies on the notion of manipulability which I propose a definition of.

The reduction property relies on the notion of *grasp* and of *placement*. I define them through the notion of gripper and handle as follows.

A *handle* is a frame, attached to a static or movable body. It is defined by a forward kinematic function $e_h : \mathcal{CS} \to SE(3)$. A *gripper* is a full-actuated handle, *i.e.* all the gripper velocities, $Im(\mathbf{J}_{e_g})$, are actionable. This is always the case if all the joints from the root joint of the kinematic tree and the joint are actuated. It is sometimes not verified. For instance, a non-holonomic mobile platform, with a manipulator arm with strictly less that six DoFs mounted on it has non-actionable velocities in some configurations.

Definition 3.9 (Grasp, Placement)**.** *A* placement *is a fixed relative transformation constraint between two handles h and h', i.e. $e_h(\mathbf{q})^{-1} e_{h'}(\mathbf{q}) = M \in SE(3)$.*

A grasp *is a fixed relative transformation constraint between a gripper g and a handle h, i.e. $e_g(\mathbf{q})^{-1} e_h(\mathbf{q}) = M' \in SE(3)$.*

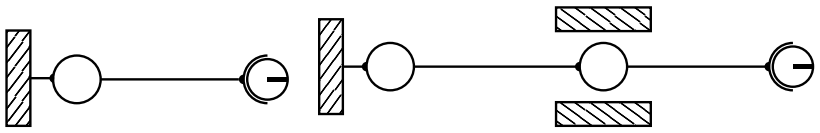
The reduction property is given as input a path in $\mathcal{CG} \cap \mathcal{CP}$. Grasps and placements along this path are valid and only these sets of grasp and placement are used in the following paragraph.

3.3.1 Generalized reduction property

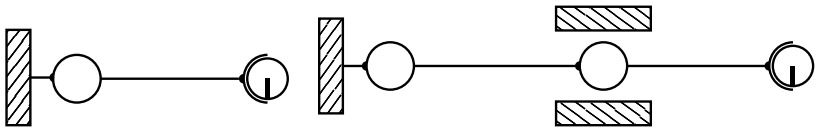
The reduction property, as stated in Dacre-Wright et al. [1992], is not complete. Figure 3.10 gives two counter examples. It shows two different robots manipulating an cylindrical object.

On Figures 3.10(a) and 3.10(c), the robot has one rotational DoF and the object can move in the plane (3 DoFs). In both positions, the object is grasped by the robot. There is a motion in $\mathcal{CG} \cap \mathcal{CP}$ that solves the problem: the robot stay still and the object rotates around its axis. However, clearly, the problem has no solution because the robot cannot execute this motion. The reduction property does not apply in this case.

On Figures 3.10(b) and 3.10(d), the case is slightly different. The robot has two rotational DoFs and the object can move in the plane (3 DoFs). As in the previous



(a) 1 DoF robot (b) 2 DoFs robot



(c) 1 DoF robot (d) 2 DoFs robot

Figure 3.10: Two counter examples to the reduction property. Circles are rotation joints: 1 rotation on (a) and (c), 2 rotations on (b) and (d). The circle with a radius is an object that moves on the plane (3 DoFs). Hatched areas are obstacles. Start configurations are the top figures and goal configurations are the bottom ones. From (a) to (c), the problem has no solution independently of the obstacles. From (b) to (d), the problem has no solution because the obstacles.

case, in both position, the object is grasped by the robot. And again, there is a motion in $\mathcal{CG} \cap \mathcal{CP}$ that solves the problem: the robot stay still and the object rotates around its axis. In this set-up, given an initial object position (x_0, y_0, θ_0), reaching (x_0, y_0, θ_1) requires to switch between inverse kinematic solutions. When using only one inverse kinematic solution, the only reachable object angle in (x_0, y_0) is θ_0. If not considering collisions, the path in $\mathcal{CG} \cap \mathcal{CP}$ can be *transformed* into a valid sequence of transit and transfer paths. However, it cannot be *approximated*. So, with the obstacles, the problem has no solution because a path going from one inverse kinematic solution to the other will always collide either with the object or with one obstacle. The reduction property does not apply in this case too.

In the following, $\mathcal{M}^i(\mathbf{q}, \varepsilon)$ denotes the set of admissible paths from $\mathbf{q}$ and included in $\mathcal{B}(\mathbf{q}, \varepsilon)$, *i.e.* $\{\mathbf{p} \in \mathcal{C}^i([0,1], \mathcal{B}(\mathbf{q}, \varepsilon)) \mid \mathbf{p}(0) = \mathbf{q}\}$. Paths in $\mathcal{M}^i(\mathbf{q}, \varepsilon)$ are i times continuously differentiable. $\mathcal{B}(x, \varepsilon)$ is the open ball of center x and radius ε.

Definition 3.10 (Small Space Controllability)**.** *A robot R is* small space controllable *in $\mathbf{q} \in \mathcal{CS}$ if and only if $\forall \varepsilon > 0, \exists \eta > 0, \forall \mathbf{q}' \in \mathcal{B}(\mathbf{q}, \eta), \exists \mathbf{p} \in \mathcal{M}^1(\mathbf{q}, \varepsilon)$ such that $\mathbf{p}(1) = \mathbf{q}'$.*

These two robots are small space controllable. However, in both cases, the robot end-effector is not small space controllable in the configuration space of the object. For instance, consider the 2 DoFs robot on Figure 3.10(b). Let (ϕ_r, ψ_r) be the two angles of the robot and $(x_e, y_e, \theta_e) = f(\phi_r, \psi_r)$ be the position and orientation of the end-effector. The set of angles that can be reached in (x_e, y_e), *i.e.* $\{\theta \in [0, 2\pi[\mid \exists (\phi_r, \psi_r), f(\phi_r, \psi_r) = (x_e, y_e, \theta)\}$, has only two elements so the end-

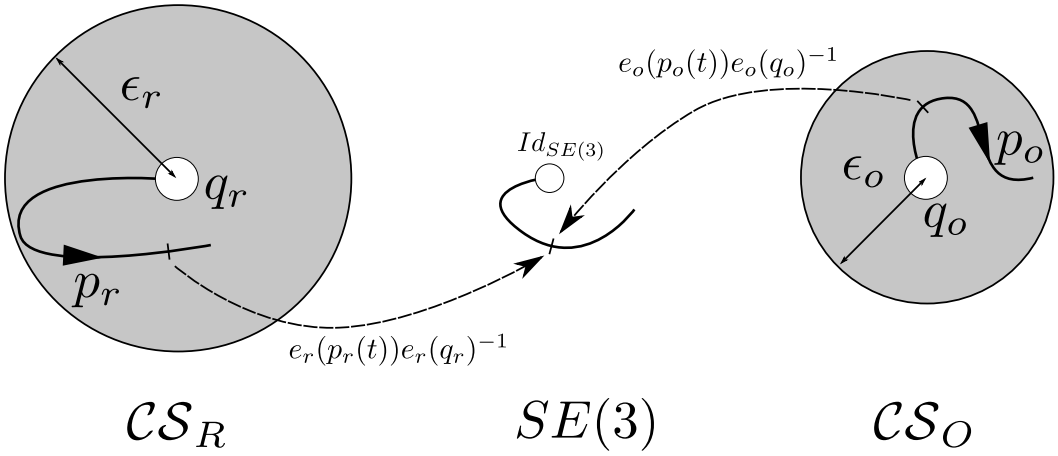


Figure 3.11: Small Space Manipulability. For all path $\mathbf{p}_o$ included in $\mathcal{B}(\mathbf{q}_o, \varepsilon_o))$ starting in $\mathbf{q}_o$, there exists a path $\mathbf{p}_r$ included in $\mathcal{B}(\mathbf{q}_r, \varepsilon_r))$ starting in $\mathbf{q}_r$ such that the relative position of O with respect to R is constant.

effector is not small-space controllable in the configuration space of the object. This can be formalized by the Small Space Manipulability (SSM) property, which also applies to articulated objects.

Definition 3.11 (Small Space Manipulability)**.** *Let R be a robot and O be an object. Let $e_r : \mathcal{CS}_R \to SE(3)$, resp. $e_o : \mathcal{CS}_O \to SE(3)$ be the continuous forward kinematic function that maps the robot, resp. object, configuration to the gripper, resp. handle, pose.*

O is small space manipulable by R from $(\mathbf{q}_r, \mathbf{q}_o) \in \mathcal{CS}_R \times \mathcal{CS}_O$ if and only if $\forall \varepsilon_r > 0$, $\exists \varepsilon_o > 0, \forall \mathbf{p}_o \in \mathcal{M}^1(\mathbf{q}_o, \varepsilon_o), \exists \mathbf{p}_r \in \mathcal{M}^1(\mathbf{q}_r, \varepsilon_r)$ such that

$$e_r(\mathbf{p}_r(t))^{-1} e_o(\mathbf{p}_o(t)) = e_r(\mathbf{q}_r)^{-1} e_o(\mathbf{q}_o)$$

Figure 3.11 pictures this condition. The "Jacobian" of e_r can be defined[5] by $\frac{de_r \circ \mathbf{p}_r}{dt}(t) = \mathbf{J}_r(\mathbf{p}_r(t)) \frac{d\mathbf{p}_r}{dt}(t)$. The Jacobian $\mathbf{J}_o$ of e_o is defined similarly.

Lemma 3.1 (Necessary condition and sufficient condition of SSM)**.** *Using the notations of Definition 3.11 the following assertions verifies $(i) \implies (ii) \implies (iii)$.*

(i) $\mathbf{q}_r$ *is a regular point of* $\mathbf{J}_r$ *and* $\exists \varepsilon_r > 0, \exists \varepsilon_o > 0$ *such that,* $\forall (\mathbf{q}'_r, \mathbf{q}'_o) \in \mathcal{B}(\mathbf{q}_r, \varepsilon_r) \times \mathcal{B}(\mathbf{q}_o, \varepsilon_o)$,

$$Im(\mathbf{J}_o(\mathbf{q}'_o)) \subset \{v_r \times e_r(\mathbf{q}_r)^{-1} e_o(\mathbf{q}_o) \mid v_r \in Im(\mathbf{J}_r(\mathbf{q}'_r))\}$$

(ii) $\forall \varepsilon_r > 0$, $\exists \varepsilon_o > 0$, $\forall \mathbf{p}_o \in \mathcal{M}^1(\mathbf{q}_o, \varepsilon_o)$, $\exists \mathbf{p}_r \in \mathcal{M}^1(\mathbf{q}_r, \varepsilon_r)$ *such that*

$$e_r(\mathbf{p}_r(t))^{-1} e_o(\mathbf{p}_o(t)) = e_r(\mathbf{q}_r)^{-1} e_o(\mathbf{q}_o)$$

[5] In the following, I use a formalism not described here: $\frac{de_r \circ \mathbf{p}_r}{dt}(t) \in \mathfrak{se}(3)$ and $\frac{d\mathbf{p}_r}{dt}(t)$ is in the tangent to the joint space. This formalism is well described in the book of Featherstone [2008].

(iii) $Im(\mathbf{J}_o(\mathbf{q}_o)) \subset \{v_r \times e_r(\mathbf{q}_r)^{-1} e_o(\mathbf{q}_o) \mid v_r \in Im(\mathbf{J}_r(\mathbf{q}_r))\}$

Before providing a proof, I explain what (i) and (iii) mean. (iii) means that all admissible velocities of the object are admissible for the robot end-effector. (i) means that on a neighbourhood of a regular robot configuration, all the admissible velocities of the object on a neighbourhood of its original position are admissible for the robot end-effector, keeping the grasp constant.

In the previous examples, we immediately see that (iii) is false, thus the object is nowhere small-space manipulable by the robot.

Consider the cases where the Jacobian of the robot end-effector is of rank 6. By using results of Section 2.2.2, there exists a neighbourhood where the rank is constant. Thus, on this neighbourhood, $Im(\mathbf{J}_r) = \mathfrak{se}(3)$ and (i) is true. So all objects are small-space manipulable by the robot in considered configuration. This is the case which was considered by Dacre-Wright et al. [1992].

Proof. I first prove that $(ii) \implies (iii)$. Assume assertion (ii) is true.

Let $v_o \in Im(\mathbf{J}_o)$. There exists $\dot{\mathbf{q}}_o$ such that $v_o = \mathbf{J}_o \dot{\mathbf{q}}_o$. Let $\mathbf{p}_o \in \mathcal{M}^1(\mathbf{q}_o, \varepsilon_o)$ such that $\frac{d\mathbf{p}_o}{dt}(0) = \dot{\mathbf{q}}_o$. There exists a path $\mathbf{p}_r \in \mathcal{M}^1(\mathbf{q}_r, \varepsilon_r)$ such that

$$e_r(\mathbf{p}_r(t))^{-1} e_o(\mathbf{p}_o(t)) = e_r(\mathbf{q}_r)^{-1} e_o(\mathbf{q}_o) \tag{3.2}$$

By rearranging Eq. (3.2) and taking the derivative, we have:

$$\left(\mathbf{J}_r(\mathbf{p}_r(t)) \frac{d\mathbf{p}_r}{dt}(t)\right) \times e_r(\mathbf{q}_r)^{-1} = \left(\mathbf{J}_o(\mathbf{p}_o(t)) \frac{d\mathbf{p}_o}{dt}(t)\right) \times e_o(\mathbf{q}_o)^{-1} \tag{3.3}$$

which implies that $v_o = \left(\mathbf{J}_r(\mathbf{q}_r) \frac{d\mathbf{p}_r}{dt}(0)\right) \times e_r(\mathbf{q}_r)^{-1} e_o(\mathbf{q}_o)$. So (iii) is true.

I now prove that $(i) \implies (ii)$. Assume assertion (i) is true. Note that if (ii) is true for some $\bar{\varepsilon}_r$, it is obviously true for all $\varepsilon_r > \bar{\varepsilon}_r$.

Let $\mathbf{p}_o$ be an object path. Consider the differential equation $\frac{d\mathbf{p}}{dt}(t) = f(t, \mathbf{p}(t))$, of unknown $\mathbf{p}$, and the initial condition $e_r(\mathbf{p}_r(0)) = e_r(\mathbf{q}_r)$. We seek a path in $\mathcal{M}^1(\mathbf{q}_r, \varepsilon_r)$, *i.e.* a continuously differentiable solution. The function f is

$$f : t, \mathbf{p} \mapsto \mathbf{J}_r(\mathbf{p}(t))^\dagger \left(\mathbf{J}_o(\mathbf{p}_o(t)) \frac{d\mathbf{p}_o}{dt}(t)\right) \times e_o(\mathbf{q}_o)^{-1} e_r(\mathbf{q}_r)$$

As $\mathbf{q}_r$ is a regular point and $\mathbf{q} \mapsto \mathbf{J}_r(\mathbf{q})$ is continuous, Theorem 2.1 applies. There exists ε_r^0 such that $\mathbf{q} \mapsto \mathbf{J}_r(\mathbf{q})^\dagger$ is continuous on $\mathcal{B}(\mathbf{q}_r, \varepsilon_r^0)$.

Assertion (i) implies that there exists $\varepsilon_r^1 > 0$ and $\varepsilon_o > 0$ such that $\forall (\mathbf{q}_r', \mathbf{q}_o') \in \mathcal{B}(\mathbf{q}_r, \varepsilon_r^1) \times \mathcal{B}(\mathbf{q}_o, \varepsilon_o)$,

$$Im(\mathbf{J}_o(\mathbf{q}_o')) \subset \{v_r \times e_r(\mathbf{q}_r)^{-1} e_o(\mathbf{q}_o) \mid v_r \in Im(\mathbf{J}_r(\mathbf{q}_r'))\} \tag{3.4}$$

Thus, let $\varepsilon_r > 0$ be fixed such that $\varepsilon_r < \min(\varepsilon_r^0, \varepsilon_r^1)$. Assertion (i) is still true for $\varepsilon_r < \varepsilon_r^0$. Let $\mathbf{p}_0$ be fixed in $\mathcal{M}^1(\mathbf{q}_o, \varepsilon_o)$. Cauchy–Lipschitz theorem states that there

is a unique maximal solution $\mathbf{p}_r \in \mathcal{M}^1(\mathbf{q}_r, \varepsilon_r)$. Moreover, Equation (3.4) implies

$$\left(\mathbf{J}_o(\mathbf{p}_o(t))\frac{d\mathbf{p}_o}{dt}(t)\right) \times e_o(\mathbf{q}_o)^{-1}e_r(\mathbf{q}_r) \in Im(\mathbf{J}_r(\mathbf{p}_r(t)))$$

Thus,

$$\mathbf{J}_r(\mathbf{p}(t))f(t,\mathbf{p}_r(t)) = \mathbf{J}_r(\mathbf{p}(t))\mathbf{J}_r(\mathbf{p}(t))^\dagger \left(\mathbf{J}_o(\mathbf{p}_o(t))\frac{d\mathbf{p}_o}{dt}(t)\right) \times e_o(\mathbf{q}_o)^{-1}e_r(\mathbf{q}_r)$$
$$\mathbf{J}_r(\mathbf{p}(t))\frac{d\mathbf{p}_r}{dt}(t) = \left(\mathbf{J}_o(\mathbf{p}_o(t))\frac{d\mathbf{p}_o}{dt}(t)\right) \times e_o(\mathbf{q}_o)^{-1}e_r(\mathbf{q}_r)$$

The above equation integrates to Eq. 3.2 so assertion (ii) is true.

□

The following theorem formulates the generalized reduction property.

Theorem 3.2 (Generalized reduction property)**.** *Let R be a robot and O be an object. Any path $\mathbf{p}$ lying in $\mathcal{CG} \cap \mathcal{CP}$ satisfying the following conditions at all time can be transformed into a finite sequence of* transit *and* transfer *paths:*

- *the robot is not in collision with static obstacles at $\mathbf{p}(t)$,*
- *R is small space controllable.*
- *O is small space manipulable by R from $\mathbf{p}(t)$.*

Proof. Let $\mathbf{p} : [0,1] \to \mathcal{CG} \cap \mathcal{CP}$. Let $\mathbf{p}_r$, resp. $\mathbf{p}_o$, be the projection of $\mathbf{p}$ onto $\mathcal{CS}_R$, resp. $\mathcal{CS}_O$. Let $t_0 \in [0,1]$ be a fixed time.

As, R is not in collision with static obstacle in $\mathbf{p}_r(t_0)$, we can find $\varepsilon > 0$ such that $\mathcal{B}(\mathbf{p}_r(t_0), \varepsilon) \subset \mathcal{CS}_{R,free}$.

As O is small-space manipulable by R from $\mathbf{p}(t_0)$, there exists $\varepsilon_r > 0$ and $\varepsilon_o > 0$ such that all configurations of $\mathcal{B}(\mathbf{p}_r(t_0), \varepsilon_r)$ are reachable by paths from $\mathbf{q}_r$ included in $\mathcal{B}(\mathbf{p}_r(t_0), \varepsilon)$, thus collision free, and such that $\forall \mathbf{p}_o \in \mathcal{C}^0([0,1], \mathcal{B}(\mathbf{q}_o, \varepsilon_o)) \mid \mathbf{p}_o(0) = \mathbf{q}_o$, $\exists \mathbf{p}_r \in \mathcal{C}^0([0,1], \mathcal{B}(\mathbf{q}_r, \varepsilon_r)) \mid \mathbf{p}_r(0) = \mathbf{q}_r$ such that

$$e_r(\mathbf{p}_r(t))^{-1}e_o(\mathbf{p}_o(t)) = e_r(\mathbf{q}_r)^{-1}e_o(\mathbf{q}_o)$$

As $\mathbf{p}$ is continuous, there exists $\delta > 0$ such that

$$\forall \tau \in]t_0 - \delta, t_0 + \delta[, \mathbf{p}_r(\tau) \in \mathcal{B}(\mathbf{p}_r(t_0), \varepsilon_r) \text{ and } \mathbf{p}_o(\tau) \in \mathcal{B}(\mathbf{p}_o(t_0), \varepsilon_o)$$

$$\forall \tau \in]t_0 - \delta_o, t_0 + \delta_o[, \mathbf{p}_o(\tau) \in \mathcal{B}(\mathbf{p}_o(t_0), \varepsilon_o)$$

As $\mathbf{p}$ is continuous, there exists $\delta_r, \delta_o > 0$ such that

$$\forall \tau \in]t_0 - \delta_r, t_0 + \delta_r[, \mathbf{p}_r(\tau) \in \mathcal{B}(\mathbf{p}_r(t_0), \varepsilon_r)$$

$$\forall \tau \in]t_0 - \delta_o, t_0 + \delta_o[, \mathbf{p}_o(\tau) \in \mathcal{B}(\mathbf{p}_o(t_0), \varepsilon_o)$$

Let $\delta = \min(\delta_r, \delta_o)$, $\tau \in]t_0, t_0 + \delta[$ and

$$\boldsymbol{\pi}_o : \begin{array}{ccc} [t_0, \tau] & \rightarrow & \mathcal{B}\left(\mathbf{p}_o(t_0), \varepsilon_o\right) \\ t & \mapsto & \mathbf{p}_o(t) \end{array}$$

There exists $\boldsymbol{\pi}_r \in \mathcal{C}^0([t_0, \tau], \mathcal{B}\left(\mathbf{q}_r, \varepsilon_r\right))$ such that $\boldsymbol{\pi}_r(t_0) = \mathbf{q}_r$ and $e_r(\boldsymbol{\pi}_r(t))^{-1} e_o(\boldsymbol{\pi}_o(t)) = e_r(\mathbf{q}_r)^{-1} e_o(\mathbf{q}_o)$. Note that the time interval $[0, 1]$ as been rescaled to $[t_0, \tau]$. The path $(\boldsymbol{\pi}_r, \boldsymbol{\pi}_o)$ constitutes a valid transfer path. $\mathcal{R}$ is small-space controllable, so as $\boldsymbol{\pi}_r(\tau)$ and $\mathbf{p}_r(\tau)$ are in $\mathcal{B}\left(\mathbf{q}_r, \varepsilon_r\right)$, there exists an admissible collision-free path $\boldsymbol{\pi}'_r$ from $\boldsymbol{\pi}_r(\tau)$ to $\mathbf{p}_r(\tau)$. $\mathbf{p} : t \rightarrow (\boldsymbol{\pi}'_r(t), \mathbf{p}_o(\tau))$ is a valid transit path.

The path from $(\mathbf{p}_r(t_o), \mathbf{p}_o(t_o))$ to $(\mathbf{p}_r(\tau), \mathbf{p}_o(\tau))$ has been transformed into a valid transfer path followed by a valid transit path.

As $\mathbf{p}_r$ is a compact set included in an open set of $\mathcal{CS}_{R,free}$, This local transformation can be applied on a finite covering of $[0, 1]$. Thus, we get a finite number of elementary manipulation paths.

□

3.3.2 Grasps and placements

The definition of *grasp* and *placement* given above implies that a *grasp* is a *placement*. The reverse is not true. The difference is that a *grasp* can induce a motion.

Theorem 3.2 generalizes the reduction property to articulated objects. It makes use of two forward kinematic functions e_r and e_o. The DoFs involved in e_o are not necessarily actuated while the DoFs involved in e_r are.

Now consider an object O and two grippers G_1 and G_2. Indexes 1 and 2 in this paragraph refers to one of these grippers. Both gripper can hold the object at the same time. Then, three cases can happen.

First case, one gripper is not enough to carry the object. Both are needed at the same time, for instance for a heavy object. Thus, there is only one *grasp*, which happens when both grippers hold the object. Though the generalized reduction property does not cover this case, I believe it could be extended by considering the following function as gripper function:

$$e_g : \left\{ \begin{array}{ccc} \mathcal{CS} & \rightarrow & SE(3) \times SE(3) \\ \mathbf{q} & \mapsto & (e_{g_1}(\mathbf{q}), e_{g_2}(\mathbf{q})) \end{array} \right.$$

Second case, each gripper is sufficient to hold the object. The generlized reduction property applies for paths in $\mathcal{CG}_1 \cap \mathcal{CP}$, in $\mathcal{CG}_2 \cap \mathcal{CP}$ or in $\mathcal{CG}_1 \cap \mathcal{CG}_2$ Consider a path $\mathbf{p}$ lying in $\mathcal{CG}_1 \cap \mathcal{CG}_2$. It is not in general a valid manipulation path, for the same reason as a path in $\mathcal{CG} \cap \mathcal{CP}$ is not. Let us temporarily view $\mathcal{CG}_2$ as a set of valid placement. Then, when the hypothesis of Theorem 3.2 holds for $e_r = e_{G_1}$ and $e_o = e_{G_2}$, the decomposition given in the proof gives a path which is a alternative sequence of

- valid transfer by gripper G_1, lying in $\mathcal{CG}_1 \cap \mathcal{CG}_2$,

– transit for gripper G_1, and where G_2 is static and holds O, thus also valid.

This path is thus a valid manipulation path. This is expressed by the following corollary of Theorem 3.2.

Third case, gripper G_1 can carry the object alone but G_2 cannot. If the problem has no solution considering grasps with G_1 only, then it has no solution at all.

Corollary 3.2 (Corollary to the generalized reduction property)**.** *Let R be a robot with two grippers and O be an object. Any path $\mathbf{p}$ lying in $\mathcal{CG}_1 \cap \mathcal{CG}_2$ satisfying the following conditions at all time can be transformed into a finite sequence of* transit *and* transfer *paths:*

– *the robot is not in collision with static obstacles at $\mathbf{p}(t)$,*

– *R is small space controllable.*

– *O is small space manipulable by R from $\mathbf{p}(t)$.*

3.3.3 Limitations

In general, the reduction property does not cover coupled DoFs. Indeed, the above demonstrations all assume the e_r and e_o are decoupled. For instance, the DoFs of a humanoid robot are all coupled by an equilibrium constraint. However, in this case, the constraint is infinitely differentiable. It seems reasonable, although merely an assumption, that the generalized reduction applies.

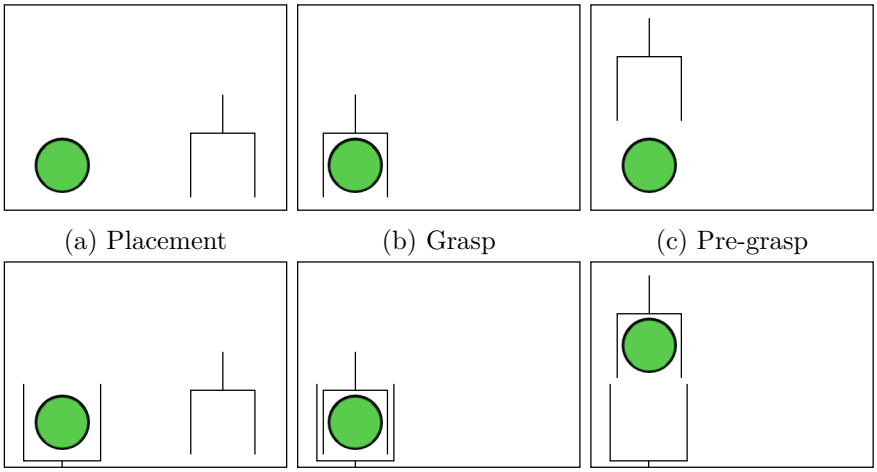
3.4 Narrow passages

Narrow passages are known to be challenging for motion planning algorithms. Unfortunately, manipulation planning encounters them very often. To overcome this issue, I introduce the notion of way-points.

3.4.1 Low sampling probability

A search for a path to grasp an object necessarily goes close to collisions because the object has to be in the gripper. Figure 3.12 shows a simple case where a gripper with two fingers manipulates a ball. From Figure 3.12(a) to Figure 3.12(b), the gripper goes to a grasp pose, close to the object. Without a hint on how to approach the ball, randomized motion planning will spend a lot of time searching for a solution through this narrow passage. In this case, a basic hint would be the direction of approach of the gripper. A straight line approach along this direction is likely to be collision-free.

Figure 3.13 shows a Graph of Constraint which integrates an intermediate state corresponding to poses like the one showed on Figure 3.12(c). Transitions *Grasp ball* and *Move gripper up* compute motion from Figure 3.12(c) to 3.12(b) and vice versa. While poses where the ball is grasped are very cluttered, pre-grasp poses, like on Figure 3.12(c), are much less cluttered.



(a) Placement (b) Grasp (c) Pre-grasp

(d) Placement in box (e) Grasp in box (f) Pre-placement

Figure 3.12: This simple example illustrates *narrow passages*. (a) and (b) show a gripper and a ball. (c) gives a possible approaching pose. (d) and (e) show the same environment but the ball is in a box. The passage is now even narrower and lifting the ball up is a second *narrow passage*. (f) gives a possible lifting pose.

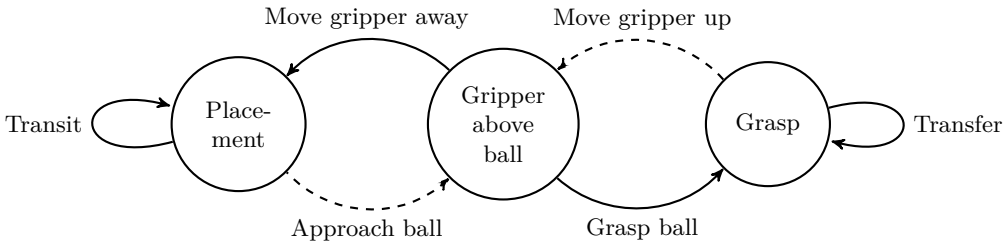


Figure 3.13: A Graph of Constraint for the problem on Figures 3.12(a) and 3.12(b) with a intermediate state similar to Figure 3.12(c). Dashed line represents *way-point transition*.

The most natural method to compute this path is to constrain the gripper to move on the line passing through the two ends. A straight line in Euclidean space means the path in $\mathcal{CS}$ is not straight. However, if the distance between the two configurations is small, the straight line in $\mathcal{CS}$ is very close to a straight line for the gripper, in Euclidean space. So a straight line in $\mathcal{CS}$ does not decrease much the likelihood of a collision-free path. Moreover, the projection algorithm requires to solve a system of linear equations and makes use of singular values for the continuity criterion. Those operations are of cubic complexity so the marginal increase of time for projection for each additional constraint increases. Straight lines in $\mathcal{CS}$ are - potentially a lot - less time-consuming. I experimented both strategies and preferred the second one as it speeds up projections.

Figure 3.15 shows a Graph of Constraint which integrates three intermediate

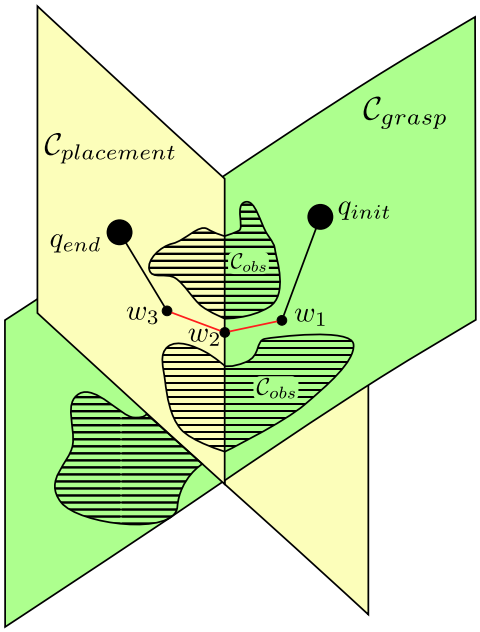


Figure 3.14: Effect of a way-point transition in the configuration space. $\mathcal{C}_{grasp}$ and $\mathcal{C}_{placement}$ are represented. A path from $\mathbf{q}_{init} \in \mathcal{C}_{grasp}$ to $\mathbf{q}_{end} \in \mathcal{C}_{placement}$ built with graph on Figure 3.15 will contains three way-points: w_1 in "Ball above ground", w_2 in "Grasp - Placement" and w_3 in "Gripper above ball". The two red lines shows the sub-paths which are expected to be short.

states. *Gripper above ball* corresponds to poses like on Figure 3.12(c), already explained. *Ball above ground* corresponds to poses like on Figure 3.12(f). The robot holds the ball and the ball is at a pre-placement position. The intermediate state encodes the hypothesis that, going from pre-placement to pre-grasp, from Figure 3.12(f) to 3.12(c), requires to find a pose like on Figure 3.12(e). This restricts the set of solutions. This is illustrated in Figure 3.14.

3.4.2 Way-point transition

In the examples above, I use states to specify approaching poses. This fastens the search by giving hints on how to execute an action. The hints are given as a pose constraint for the end-effector or the object.

However, hints should not be states as it would slow down the search for the two following reasons. First, two or more iterations would be required to compute a path between *grasp* and *placement*. The fact that transition *Grasp ball* should directly follow *Approach ball* is ignored. Second, the approaching configuration is close to the grasping configuration so generating the grasping configuration from the approaching configuration is likely to succeed. The path between the two configurations is very short so, if the approaching configuration is collision-free, the path is also likely to be collision-free. Two steps mean two different random configurations. More likely, it will result in a projection failure or in a longer path, more

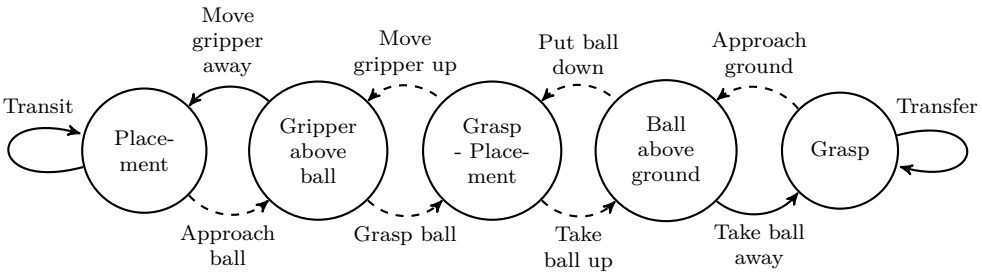
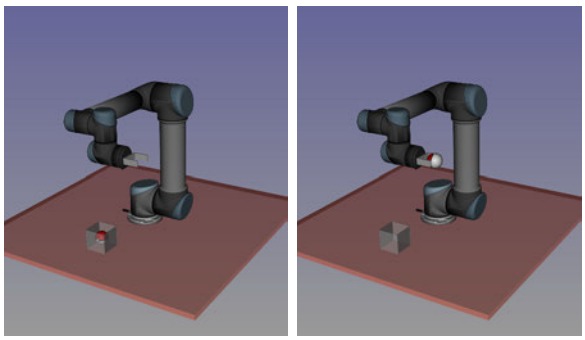


Figure 3.15: A Graph of Constraint for the problem on Figures 3.12(d) and 3.12(e) with intermediate states similar to Figures 3.12(c) and 3.12(f). Dashed line represents *way-point transition*.

likely not collision-free.

To overcome these issues, I introduce the *way-point transition*. This type of transition tells the planner to chain the transition. For instance, on Figure 3.13, whenever *Approach ball* is selected and generates $\mathbf{q}_{proj}$ from a random $\mathbf{q}_{rand}$ and a neighbour $\mathbf{q}_{near}$, *Grasp ball* is called and generates a configuration from $\mathbf{q}_{rand}$ and $\mathbf{q}_{proj}$. These transitions are also useful to specify transitions representing motions lying in several states. For instance on Figure 3.13, the way-point transition from *placement* to *grasp* represents motion that lies first in *placement* (*Approach ball* and *Grasp ball*) and then in *grasp* (*Take ball up* and *Take ball away*).

3.4.3 Experimental results



(a) Initial configuration (b) Goal configuration

Figure 3.16: Benchmark of way-point transition using UR5.

Three different Graphs of Constraint have been tested in the following set-up. The UR5 manipulator arm must move a ball from inside a box to a pose where the ball is in its gripper. This two configurations are shown in Figure 3.16. The results are summarized in Table 3.1. The columns correspond to the following cases, respectively.

The first graph corresponds to the one of Figure 3.1. It contains only two states and no guidance information.

The second graph corresponds to the one of Figure 3.9, where transitions are not way-point transitions but standard transitions. This means the algorithm does not chain them. For instance, *Grasp ball* will not necessarily be applied to configuration obtained by *Approach ball*.

The third graph corresponds to the one of Figure 3.9, with way-point transitions. This means the algorithm will apply systematically *Approach ball*, *Grasp ball*, *Take ball up* and *Take ball away* in sequence.

The results show that properly using guidance information has a huge effect on the computation time. With way-point transitions, the problem was solved more than 300 times faster than without guidance information.

Guidance	No	With transitions	With way-point transitions
t_{avg}/t_{max} (s)	74.8 / 296	1.52 / 8.92	0.234 / 0.245
N_{avg}/N_{max}	3553 / 13840	53 / 291	5 / 5

Table 3.1: Results of the benchmark of way-point transitions with UR5. Each case is run 100 times. t is the computation time. and N is the number of nodes generated. No guidance means that no intermediate poses were given. Guidance with transitions and way-point transitions means that intermediate poses were provided. In the former case, this information was provided using standard states and transitions while the latter case uses way-point transitions.

Chapter 4

Affordance

Contents

4.1 Documented objects 69
4.1.1 Grippers and handles 70
4.1.2 Contact surfaces 71
4.2 Constraint graph generation 73
4.2.1 Building the states 74
4.2.2 Transition detection 75

This chapter presents a method to generate a Graph of Constraint. The method relies on affordance. Each robot and object comes with a *documentation* containing intrinsic information about its capabilities.

The first section presents the documentation developed in the framework of this thesis. The second introduces an algorithm that generates a Graph of Constraint from the documentation.

The importance of this algorithm resides in the fact that it makes manipulation planning accessible to non-expert users. Indeed, the problem can be defined simply through two set of rules. The first set of rules specify pairs of end-effector and objects that can generate a grasp. The second set of rules concerns object stability. It contains a set of support surfaces on environment and contact surfaces on objects and robots.

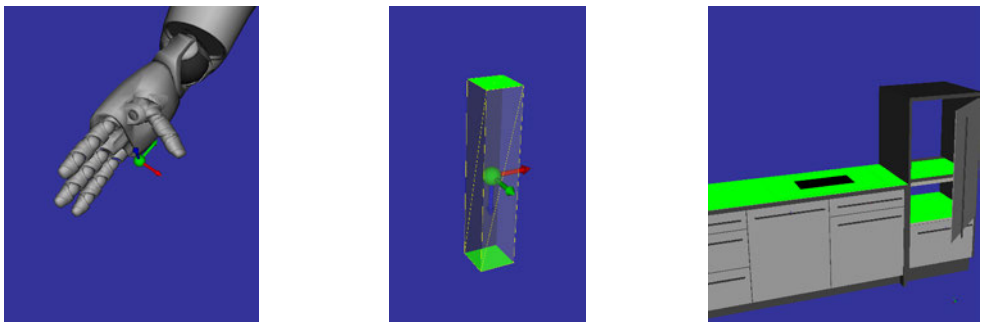
4.1 Documented objects

Models of robots, objects and environments are augmented with guidance information, referred to as *object documentation*. It contains geometrical information. Two types of interactions are proposed. The robot can interact with its surroundings by grasping with its end-effectors. It can also interact by creating contacts. Each type of interaction is presented in the two following sections.

I do not consider low-level grasping for complex grippers, such as human-like grippers. Such grippers could be documented through contact surfaces. However, full-body motion planning for a robot, considering 15 degrees of freedom (DoFs) per hand, becomes rather complex.

4.1.1 Grippers and handles

When a robot grasps an object o with an end-effector e, the relative transformation of the object with respect to the end-effector, ${}^eT_o$, is constant. However, there may be several or even infinite admissible values for ${}^eT_o$. Those admissible values depend on intrinsic parameters of e, intrinsic parameters of o and coupling parameters which depends on their association. In the following, I ignore coupling parameters.



(a) Robot gripper (b) Object handle and surfaces (c) Environment surfaces

Figure 4.1: Documentation of gripper, handle and contact surfaces. On (a) and (b), the frames represent the grasping positions. The X axis, in red, is the approaching axis. The Z axis, in blue, is the possible rotation axis. The green surfaces on (b) and (c) are contact surfaces. Object surfaces can go on environment surfaces.

The documentation of a gripper and a handle shares the following properties, some of which are shown on Figure 4.1(a) and 4.1(b).

- A *reference frame* $\mathcal{R}$ defines the center of the element, with respect to a specified body frame $\mathcal{B}(\mathbf{q})$. The X axis defines an approaching direction. The Z axis defines an axis for a potential rotation freedom.
- A positive real value, called *clearance* and denoted by δ, gives an idea of how big the element is. It should be more that the shortest distance along X axis, the approaching axis, at which a plane orthogonal to X axis would not be in collision with the element.

The documentation of a handle also contains information about the parametrization of grasps. Object can be *long*, *axial* or *long axial*. *long* means the translation along Z parametrizes grasps. *axial* means the rotation around Z parametrizes grasps. *long axial* means the two previous simultaneously. This could be extended to more types of parametrization.

In order to write the grasp constraints, let me denote the relative transformation of the handle with respect to the gripper by ${}^gT_h(\mathbf{q}) = \mathcal{R}_g^{-1} \times \mathcal{B}_g(\mathbf{q})^{-1} \times \mathcal{B}_h(\mathbf{q}) \times \mathcal{R}_h \in SE(3)$. As an element of $SE(3)$, it can be mapped to an element of $\mathfrak{se}(3)$ using the inverse of the exponential map $\exp : \mathfrak{se}(3) \to SE(3)$ that gives the smallest angle. I denote this element by

$$\log\left({}^gT_h(\mathbf{q})\right) = \left([\omega]_\times, \mathbf{v}\right) \tag{4.1}$$

where $\mathbf{v} = (v_x, v_y, v_z) \in \mathbb{R}^3$ and $\omega = (\omega_x, \omega_y, \omega_z) \in \mathbb{R}^3$ such that $||\omega||_2 \leq \pi$.

The validation and parametrization constraints can now be built. Consider the handle h is of type *axial*. Other types can be easily guessed from this example. Two validation constraints corresponding to pre-grasp poses and to grasp poses:

$$\mathbf{f}_{grasp}(\mathbf{q}) = [v_x, v_y, v_z, \omega_x, \omega_y] = 0 \tag{4.2}$$

$$\mathbf{f}_{pregrasp}(\mathbf{q}) = [v_x + \delta_h + \delta_g, v_y, v_z, \omega_x, \omega_y] = 0 \tag{4.3}$$

The parametrization constraints are the same for pre-grasp and grasp:

$$\bar{\mathbf{f}}_{grasp}(\mathbf{q}) = \bar{\mathbf{f}}_{pregrasp}(\mathbf{q}) = [\omega_z] = \mathbf{b} \tag{4.4}$$

This model of grasps is not meant to be exhaustive. It limits grippers and objects that can be considered. The reference frame is not really intrinsic to a gripper or an object. Indeed, one handle reference frame may not be sufficient for both a big and a small gripper. The handle reference frame may have to be shifted along the X axis. These values are not robust to big ranges of gripper and object size.

4.1.2 Contact surfaces

Possible contact surfaces are defined as convex planar polygons. This representation is generic enough to represent most objects (object meshes are typically composed of triangles). I only consider planar contact between polygons though sphere / plane and cylinder / plane contact models could also be considered. Users define a polygon by providing an ordered set of vertices and the body it is attached to. Figure 4.1(b) and 4.1(c) give an example.

Two parts A and B (B can be the environment) are in contact when two contact polygons, one for each part, are in contact, *i.e.* the distance between them is less than a threshold ε. The focus is only contact creation and not equilibrium criterion, which is assumed to be handled by some other means, like another constraint.

The main difficulty is to define the distance between two surfaces. This distance must take into account polygon positions and normals relative orientation. A distance is proposed in the following paragraph.

Distance between polygons For a polygon P, $\mathbf{n}_P$ denotes its normal, C_P the centroid of its vertices and

- $\mathcal{R}_P$ is a reference frame center at C_P, whose X axis is aligned with $\mathbf{n}_P$ and Y axis is aligned with one vertex.

- $Q_{P,S}$ the orthogonal projection of C_P onto the plane containing another planar polygon S.

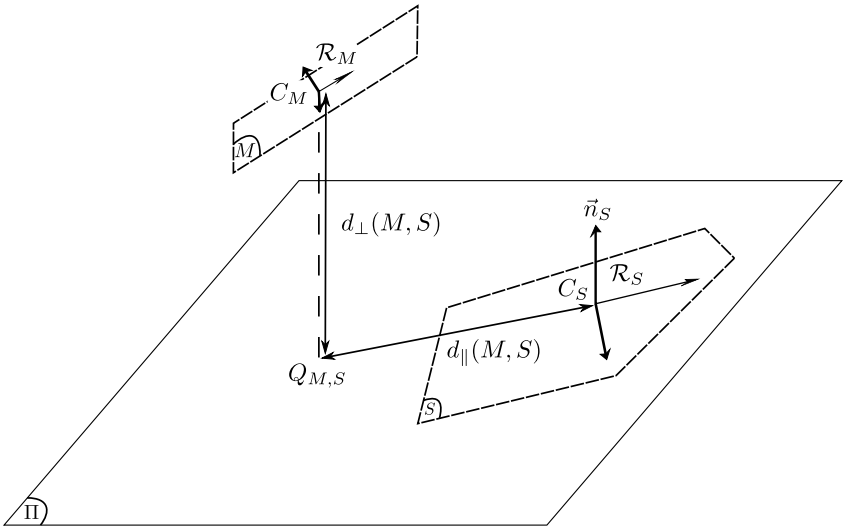


Figure 4.2: Distance between polygons. Support polygon S is included in plane Π. C_M is the centroid of the moving polygon M. $Q_{M,S}$ is its orthogonal projection onto Π. $d_\parallel(M,S)$ corresponds to Eq. (4.6) and $d_\perp(M,S)$ corresponds to Eq. (4.5)

Three distances[1] between a moving polygon M and a support polygon S are defined in Equation (4.7) and shown on Figure 4.2. Note that if $Q_{M,S} \notin S$ then, $d(M,S)$ is merely the euclidean distance between the centres. Otherwise, it is the distance along $\mathbf{n}_S$.

$$d_\perp(M,S) = \mathbf{C_S C_M}.\mathbf{n}_S \tag{4.5}$$

$$d_\parallel(M,S) = \begin{cases} ||\mathbf{C_S Q_{M,S}}||_2 & \text{if } Q_{M,S} \notin S \\ 0 & \text{otherwise} \end{cases} \tag{4.6}$$

$$d(M,S) = \sqrt{d_\perp(M,S)^2 + d_\parallel(M,S)^2} \tag{4.7}$$

Constraints The functions are build with two sets of polygons: a set of object polygons (M_i) and a set of support polygons (S_j).

Let $I, J = \arg\min_{i,j} d(M_i, S_j)$ be the indexes of the pair of the closest pair of polygons according to the distance function d. I denote by ${}^{S_j}T_{M_i}$ the relative transformation of $\mathcal{R}_{M_i}$ with respect to $\mathcal{R}_{S_j}$. The same decomposition as in (4.1) gives:

$$\log\left({}^{S_J}T_{M_I}(\mathbf{q})\right) = \log\left(\mathcal{R}_{M_I}(\mathbf{q})^{-1}\mathcal{R}_{S_J}(\mathbf{q})\right) = \left([\omega]_\times, \mathbf{v}\right)$$

where $\mathbf{v} = (v_x, v_y, v_z) \in \mathbb{R}^3$ and $\omega = (\omega_x, \omega_y, \omega_z) \in \mathbb{R}^3$ such that $||\omega||_2 \leq \pi$.

Let δ be a user-defined distance for pre-placements. I define the placement

[1] Formally, they are not distances as they are not symmetric.

validation constraints:

$$\mathbf{f}_{place}(\mathbf{q}) = \begin{cases} [v_x, 0, 0, \omega_y, \omega_z] & \text{if } Q_{M,S} \in S \\ [v_x, v_y, v_z, \omega_y, \omega_z] & \text{otherwise} \end{cases} = 0 \tag{4.8}$$

$$\mathbf{f}_{preplace}(\mathbf{q}) = \begin{cases} [v_x + \delta, 0, 0, \omega_y, \omega_z] & \text{if } Q_{M,S} \in S \\ [v_x + \delta, v_y, v_z, \omega_y, \omega_z] & \text{otherwise} \end{cases} = 0 \tag{4.9}$$

and the placement parametrization functions:

$$\bar{\mathbf{f}}_{place}(\mathbf{q}) = \begin{cases} [v_y, v_z, \omega_x] & \text{if } Q_{M,S} \in S \\ [0, 0, \omega_x] & \text{otherwise} \end{cases} \tag{4.10}$$

$$\bar{\mathbf{f}}_{preplace}(\mathbf{q}) = \begin{cases} [v_y, v_z, \omega_x] & \text{if } Q_{M,S} \in S \\ [0, 0, \omega_x] & \text{otherwise} \end{cases} \tag{4.11}$$

If $\mathbf{f}_{place}(\mathbf{q}) = \mathbf{0}$, the distance function is minimal, so polygons are in contact. The two constraints $\mathbf{f}_{place}(\mathbf{q}) = 0$ and $\bar{\mathbf{f}}_{place}(\mathbf{q}) = \mathbf{b}$ together fully constrain the relative motion between the closest polygons. Indeed, the two constraints together are equivalent to a constant ${}^{S_j}T_{M_i}(\mathbf{q})$.

The presented method is compatible with other primitives such as cylinder, resp. sphere, which would define linear, resp. punctual contacts. With additional calculations, one can similarly define distances for those primitives.

These functions are only piecewise continuously differentiable. As state in Section 3.2, the projector may not satisfy Property 3.1, *i.e.* it may be biased. The projection of a uniformly distributed configuration in $\mathcal{CS}$ has higher probability to be on the edges of the table than in the middle.

Though the discontinuity points may cause instability in the projection algorithm, none were encountered in all the cases and the efficiency of the projector did not seem altered. Escande et al. [2014] proposed a more complex approach with continuous gradient.

Eventually, the pair of surfaces used in the parametrization function is not identified in the output. In other words, specifying $\mathbf{b}$ in constraint $\bar{\mathbf{f}}_{place}(\mathbf{q}) = \mathbf{b}$ does not specify the pair. So the parametrization function does not correctly parametrize the placement space when there are more than one polygon in (M_i) and (S_j).

The example shown in Section 5.2.2 shows a case where the support surface is moving. When it is moving, the object put on it should follow its motion. This is automatically handled by the above constraints.

4.2 Constraint graph generation

In this section, I explain how a Graph of Constraint is automatically built from a set of objects and robot grippers. The objects, grippers, robots and the environment comes with the documentation defined above.

The number of nodes of the constraint graph may quickly grow when the number of objects and grippers increases. The user may provide as input only a subset of

relevant grasps to algorithm 4.1 so that the result remains tractable. This subset can be provided as a *grasp-placement table* as in Tournassoud et al. [1987], Lertkultanon and Pham [2015].

I denote by $GRASP(g, h)$ the validation constraint defined by the grasp of handle h by gripper g. I denote by $CONTACT(L_1, L_2)$ the validation constraint that enforces contact between either polygons of two sets L_1 and L_2. I denote by $FIXED(obj, g)$ the validation constraint that enforces a constant relative transformation between object obj and g, either a robot gripper or the environment.

Algorithm 4.1 Build the constraint graph

```
1: G ← set of grippers,
2: H ← set of handles,
3: function BUILDCONSTRAINTGRAPH
4:     states ← ∅
5:     for all subset G′ of G by increasing cardinal do
6:         for all injective mapping f1 from G′ to H do
7:             S1 ← MAKESTATE(f1, G′)
8:             states ← states ∪ {S1}
9:             for all g1 ∈ G′ do
10:                G″ ← G′ \ {g1}
11:                f0 ← f1 restricted to G″
12:                S0 ← states.GETSTATE(f0)
13:                MAKETRANSITIONS(S0, S1)
14:            MAKELOOPTRANSITION(S1, S1)
```

4.2.1 Building the states

Algorithm 4.1 describes the construction of the Graph of Constraint relative to a manipulation problem. One or several handles are attached to each object. The algorithm loops over all possible combinations of "some grippers hold some handles" f_1 (Lines 5, 6), by increasing number of gripper involved in a grasp.

States are created by increasing order because state priority is then the same as creation order. When there are no placement for an object, S_1 will be a subset of S_0 and it must have higher priority.

To reduce the combinatorial, I also introduced rules, which are not shown in Algorithm 4.1. A rule gives to the user the ability to forbid or authorize association of grippers with handles. This is done by checking where function f_1 in the loop Line 6 fulfils user-defined rules. If not, then, this f_1 is skip and directly go the next value for f_1. This is useful to reduce the exponential combinatorial. Also note that, when $G\prime$ has greater cardinal than H, there is no injective mapping from $G\prime$ to H.

The algorithm creates a state for each combination. Then, transitions are created to the new state S_1 from each state S_0 defined by the same set of grasps minus

the one involving g_1 (lines 9-13). Method GETSTATE returns the state built with the combination of grasps f_1 given as input (Line 12).

Algorithm 4.2 Build a state

```
    f_1 ← set of contact polygons of the environment
    G' ← set of grippers,
 1: O ← set of objects (with contact polygons),
 2: P ← set of contact polygons of the environment
 3: function MAKESTATE(f_1, G')
 4:     S ← EMPTYSTATE
 5:     for all obj ∈ O | f_1(G') ∩ obj.handles = ∅ do
 6:         C_contact ←CONTACT(obj.polygons, P)
 7:         S.constraints.ADD(C_contact)
 8:         S.transConstr.ADD(FIXED(obj, env))
 9:     for all g_1 ∈ G' do
10:         C_grasp ←GRASP(g_1, f_1(g_1))
11:         S.constraints.ADD(C_grasp)
12:         if C_grasp.dimension < 6 then
13:             S.transConstr.ADD(FIXED(f_1(g_1).obj, g_1))
14:     return S
```

Function MAKESTATE is shown in Algorithm 4.2. The set of handles of object *obj* is denoted by *obj.handles* (line 5). It loops over all ungrasped objects (Line 5) and creates constraints so that those objects stay in stable contact pose. Then it loops over each grasp g_1 defined by f_1 (Line 9) and creates the corresponding constraint. Each state stores a set of parametrization constraint that will be inserted in transitions by function MAKETRANSITIONS (Lines 8 and 13).

4.2.2 Transition detection

	S_0	W_1	W_2	W_3	S_1
$f_{grasp}(\mathbf{q}) = 0$			x	x	x
$f_{pregrasp}(\mathbf{q}) = 0$		x			
$f_{place}(\mathbf{q}) = 0$	x	x	x		
$f_{preplace}(\mathbf{q}) = 0$				x	
C_{sub}	x				x

Table 4.1: Constraints of way-point states. C_{sub} denotes the constraint of the sub-manifold in which the overall transition lies. For instance, it is composed of the grasp constraints of some pairs gripper / handle, and of the placement constraints of non-grasped objects. It does not contains the placement of the object of the current handle.

	T_{01}	T_{10}	T_{12}	T_{21}	T_{23}	T_{32}	T_{34}	T_{43}
state	S_0	S_0	S_0	S_0	S_1	S_1	S_1	S_1
$\overline{f_{grasp}}$			x	x	x	x	x	x
$\overline{f_{place}}$	x	x	x	x	x	x		
$\overline{f_{sub}}$	x	x	x	x	x	x	x	x

Table 4.2: Constraints of transition between way-point states. $\overline{f_{sub}}$ denotes the parametrization function of the foliation in which the overall transition lies. As in Table 4.1, it is composed of the grasp parametrization function of some pairs gripper / handle, and of the placement parametrization function of non-grasped objects.

Algorithm 4.3 shows function MakeTransition in cases with no pre-grasp and no pre-placement. Algorithm 4.4 shows the same function with pre-grasp and pre-placement. Similar algorithm have been designed for the two other cases.

Crossed foliation transitions are added even when there is only one foliated manifold. This increases the chance of solving a problem since the two random trees can meet both in the non-foliated and in the foliated manifold. Function MakeLoopTransition (Line 14 of Algorithm 4.1) behaves similarly to Algorithm 4.3 with only T_0 and T_1. T_0^* and T_1^* are not built.

Algorithm 4.3 Make a transition - No pre-grasp or pre-placement

```
1:  function MakeTransitions(S_0, S_1)
2:      T_0 ←EmptyTransitionBetween(S_0, S_1)
3:      T_1 ←EmptyTransitionBetween(S_1, S_0)
4:      T_0.constraints.add(S_0.transConstr)
5:      T_1.constraints.add(S_1.transConstr)
6:      if S_0.transConstr is not empty then
7:          T_1^* ←CrossedFoliationTransition(T_1)
8:          T_1^*.crossingParametrizationFunction(S_0.transConstr)
9:      if S_0.transConstr is not empty then
10:         T_0^* ←CrossedFoliationTransition(T_0)
11:         T_0^*.crossingParametrizationFunction(S_1.transConstr)
```

In Algorithm 4.3, lines 6 and 9, the *crossed foliation issue* corresponds to the parametrized constraints $S_1.transConstr$ and $S_2.transConstr$ to be both non-empty. Line 7 and 10, function CrossedFoliationTransition builds a crossed foliation transition that encodes the same motion as the transition passed as parameter. Line 8 and 11, function crossingParametrizationFunction sets the parametrization function of the crossing foliation.

Algorithm 4.4 builds a transition with three way-points, as in Figure 3.15. Table 4.1 shows the validation constraint of state S_0 and S_1 and of way-point states W_i. Table 4.2 shows the parametrization function of way-point transitions T_{ij}. *sub*

refers to the manifold of S_0 without placement constraint of the object involved in the grasp. Note that if the object is already involved in another gripper/handle pair, then only grasp and pre-grasp constraint should be used, and placement constraint should not be used. In this case, way-point states W_2 and W_3 of Table 4.1 and transitions T_{23}, T_{32}, T_{34} and T_{43} are not necessary. This is because one grasp is considered sufficient to have a stable position of an object.

Function WaypointTransition builds a way-point transition. The first argument is the ordered list of states. The second is the list of transition between the states. The W_i and the T_{ij} are internal to the way-points transitions. Line 8-15 does exactly as explained in the previous paragraph.

Algorithm 4.4 Make a transition - With pre-grasp and pre-placement

1: **function** MakeTransitions($S_0, S_1, f_0, f_1, G\prime, g_1$)
2: $h \leftarrow f_1(g_1)$
3: $o \leftarrow h.object$
Sub-manifold *sub* of the transitions: same as S_0 without placement for o.
4: $sub \leftarrow S_0$
5: *sub*.RemovePlacementOf(o)
Build way-point states as in Table 4.1 and way-point transitions as in Table 4.2.
6: $T_0 \leftarrow$WaypointTransition($(S_0, W_1, W_2, W_3, S_1)$, $(T_{01}, T_{12}, T_{23}, T_{34})$)
7: $T_1 \leftarrow$WaypointTransition($(S_1, W_3, W_2, W_1, S_0)$, $(T_{43}, T_{32}, T_{21}, T_{10})$)
8: **if** Grasp is foliated **then**
9: $T^*_{01} \leftarrow$CrossedFoliationTransition(T_{01})
10: T^*_{01}.crossingParametrizationFunction($\overline{f_{grasp}}$)
11: $T^*_0 \leftarrow$WaypointTransition($(S_0, W_1, W_2, W_3, S_1)$, $(T^*_{01}, T_{12}, T_{23}, T_{34})$)
12: **if** Placement is foliated **then**
13: $T^*_{43} \leftarrow$CrossedFoliationTransition(T_{43})
14: T^*_{43}.crossingParametrizationFunction($\overline{f_{place}}$)
15: $T^*_1 \leftarrow$WaypointTransition($(S_1, W_3, W_2, W_1, S_0)$, $(T^*_{43}, T_{32}, T_{21}, T_{10})$)

Chapter 5

Results

Contents

5.1 Humanoid Path Planner 79
5.1.1 Library architecture 80
5.1.2 Results 81
5.2 Manipulator arm 84
5.2.1 Rearrangement planning 84
5.2.2 Tool use inference 85
5.3 Humanoid robots 86
5.3.1 Quasi-static walking motion 86
5.3.2 Romeo holding a placard 90
5.3.3 Grasping behind a door 91

This chapter shows the results obtained with the algorithm described in Chapter 3. The continuous path projection algorithms and the Manipulation-RRT (M-RRT) algorithm have been integrated into the Humanoid Path Planner (HPP) software.

As an important outcome of this thesis, this library is briefly described in the first section. The two next sections show various results obtained with the implementation of the presented approach in HPP. First, I present results obtained with manipulator arms with and without mobile platform. Then, I present results with humanoid robots. Two types of problems, different in appearance, are treated: locomotion problems and manipulation problems.

5.1 Humanoid Path Planner

The Humanoid Path Planner[1] [Mirabel et al., 2016] is an open-source library addressing generic motion planning problem. It contains tools dedicated to humanoid robots, hence its name. The development of the software was initially impulsed by my PhD supervisor Florent Lamiraux. He and myself are the two main developers. Two well-known alternatives to HPP are Open Motion Planning Library (OMPL)[Şucan et al., 2012] and Open Robotics Automation Virtual Environment

[1] `https://humanoid-path-planner.github.io/hpp-doc/`

(OpenRAVE)[Diankov, 2010]. However, this two alternatives did not fit all our needs.

OMPL does not have an abstraction for continuous path. Instead, paths are always discretized. This is not desirable for the two following reasons. First, the discretization step has to be tuned for each case. Second, some algorithms need to evaluate a path at times which are not known beforehand. This is discussed in more details in Section 2.1.3.

OpenRAVE handles constraint through inverse kinematic solvers. Users can provide their own solvers via plugins. However, consider the case where a humanoid robot is to manipulate two objects, like in Section 5.3.3. The constraints to be considered are: equilibrium constraint of the robot and valid placement and valid grasp of each object. Each state has a different set of constraints. As these constraints are coupled, they cannot be solved separately. This means one would have to create a different inverse kinematic solver to handle each case separately. We preferred to put in the core an abstraction of the constraint.

5.1.1 Library architecture

Package architecture The software is organized in small packages. It relies on `hpp-fcl`, a modified version of `FCL` [Pan et al., 2012], for collision checking, and on `Pinocchio` [Mansard et al., 2014] for robot modelling and forward kinematics.

Package `hpp-pinocchio` wraps the Pinocchio library. It provides models of joints as described in Section 2.1 and summarized in Table 5.1.

Type	**Configuration space**	**Velocity space**
Prismatic (1D)	$\mathbb{R}$ (translation)	$\mathbb{R}$ (linear)
Unbounded revolute (1D)	$\mathbf{S}^1 \subset \mathbb{R}^2$ (unit complex)	$\mathbb{R}$ (angular)
Bounded revolute (1D)	$\mathbb{R}$ (angle)	$\mathbb{R}$ (angular)
Ball joints (3D)	$\mathbf{S}^3 \subset \mathbb{R}^4$ (unit quaternion)	$\mathbb{R}^3$ (angular)

Table 5.1: Main types of joints provided by default. For each joint, the representation of their configuration space ($\mathbf{q}$ vector) and their tangent space (velocity $\dot{\mathbf{q}}$ vector) is specified.

Package `hpp-core` is the main package. It contains an abstraction of tools for motion planning, such as random configuration generator, collision checker, projectors for configuration and paths and planning algorithms. It provides some implementations for each of these tools. The Progressive and Global path projection algorithms have been implemented in this package.

Package `hpp-manipulation` contains the Graph of Constraint and the M-RRT algorithm. Package `hpp-manipulation-urdf` contains the documentation parser.

Finally, package `hpp-tutorial` and `hpp-doc` contains tutorials and the API documentation.

Path abstraction The abstraction of a path contains a time interval $[t_1, t_2]$, a constraint C and a method **interpolate** : $\mathbb{R} \to \mathcal{CS}$. The interpolate method is handled differently depending on the type of path. For instance, straight path uses straight lines while car-like path uses Reeds-Shepp curves. Evaluation of the path at a time t is then done as follows.

$$p : \begin{cases} [t_1, t_2] & \to & \mathcal{CS} \\ t & \mapsto & \mathit{projector}(\textbf{interpolate}(t), C) \end{cases}$$

It corresponds to the abstraction described in Section 2.1.3.

Constraint Package `hpp-constraints` provides an abstraction of differentiable functions. This abstraction contains two methods. The first, $\mathit{value}(\mathbf{q})$, evaluates the function at a point passed as argument. The second, $\mathit{jacobian}(\mathbf{q})$, evaluates its derivative.

Then, package `hpp-core` contains an abstraction of validation and parametrization constraint and an implementation of the Newton-Raphson (NR) algorithm. A constraint C contains a differentiable function $\mathbf{f}$, an operator Δ among $\{=, >, <\}$ and a right hand side $\mathbf{b}$. The constraint is then $\mathbf{f}(\mathbf{q}) \,\Delta\, \mathbf{b}$. The NR algorithm then uses $C.\mathbf{f}.\mathit{value}(\mathbf{q})$ and $C.\mathbf{f}.\mathit{jacobian}(\mathbf{q})$ to generate configurations satisfying the constraints. It corresponds to the abstraction described in Section 2.1.2.

5.1.2 Results

Benchmarks We compared the performance of the implementation of the RRT-connect algorithm of OMPL and HPP. OMPL proposes other planners, not implemented in HPP. We did not find other benchmarks from other softwares to compare ourselves to. This benchmark shows that the efficiency of the default planner of HPP is comparable to OMPL.

To carry out the comparison, we used the benchmark database provided by OMPL, picking the three problems solved with RRT-Connect[2]. To provide a comparison as fair as possible, we had to take into account implementation details of OMPL and HPP. Firstly, OMPL uses a `range` parameter, which determines the maximum distance between two nodes in the roadmap, automatically computed for each scenario. Depending on the benchmark, this value can improve or slow down the computation time. The HPP implementation does not use such parameter.

Secondly, HPP includes a continuous collision checking method. It has a higher atomic cost than discretized collision checking, but has the advantage that only one test is required between two configurations, regardless of their distance.

To compare HPP and OMPL on an equivalent implementation of RRT-Connect, we consider on one hand a "no range" version of OMPL (OMPL-NR), where the range is set to a high value. On the other hand we consider a HPP implementation

[2]In the third scenario (Pipedream-Ring), no mesh of the ring-shaped robot was provided by OMPL, so we replaced it with a ring mesh of 982 triangles.

scenario	min time (s)				avg time (s)			
	HPP-D	HPP-C	OMPL	OMPL-NR	HPP-D	HPP-C	OMPL	OMPL-NR
Pipedream-Ring	0.065	0.043	0.458	0.618	1.24	2.05	3.00	4.23
Abstract	0.159	0.408	23.5	14.3	47.6	34.4	107	107
Cubicles	0.049	0.024	0.096	0.118	0.271	0.130	0.277	0.329

scenario	max time (s)				success rate (%)			
	HPP-D	HPP-C	OMPL	OMPL-NR	HPP-D	HPP-C	OMPL	OMPL-NR
Pipedream-Ring	6.52	7.35	10.4	14.1	100	100	100	100
Abstract	258	178	297	270	94	94	96	98
Cubicles	0.902	0.946	0.665	1.06	100	100	100	100

scenario	avg number of nodes				time-out (s)
	HPP-D	HPP-C	OMPL	OMPL-NR	
Pipedream-Ring	2283	2452	16100	22681	20
Abstract	11927	10807	177914	181427	300
Cubicles	495	302	261	307	20

Table 5.2: Results for 50 runs of each planner. Green values are used when the HPP implementation performs better than all OMPL implementations. Red values are used when HPP performs worse than at least one OMPL implementation. A planning is considered to have failed after running longer than the specified timeout value.

with discretized collision checking (HPP-D). The discretization step is the same in HPP and OMPL[3]. To be exhaustive we also included benchmarks performed with the specificities of the softwares: we thus also consider the standard "range" version of OMPL (OMPL), and the continuous collision checking version of HPP (HPP-C).

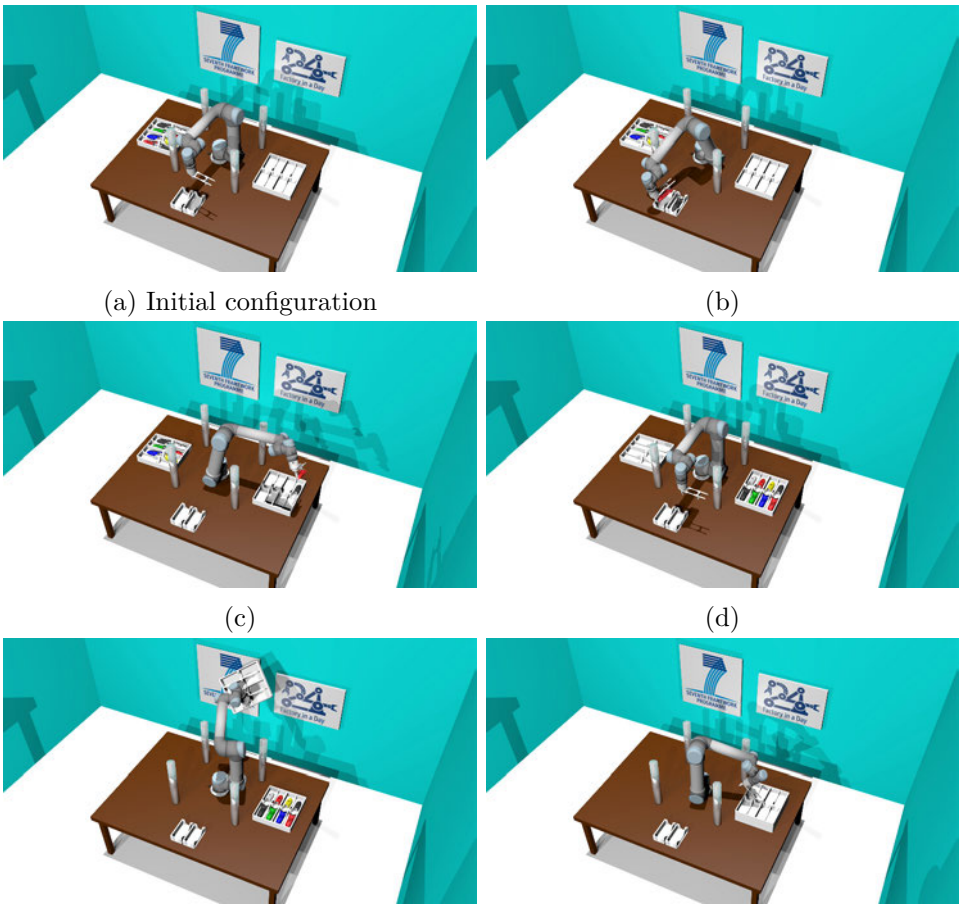
Table 5.2 presents the results for all three scenarios and implementations. The success rate represents the relative number of runs that succeeded before a given maximum time limit. When computing minimum, average and maximum time values, only successful runs were considered. The runs, single-threaded, were performed on a 64 bits computer with 8 processors of 1.2Ghz, 64Go or RAM.

In any considered case, HPP implementation presents equivalent or better average computation times compared to OMPL. The important point is that the performances remain in the same order of magnitude between HPP and OMPL.

Robot programming To my best knowledge, the earliest industrial application of motion planning is Kineo [Laumond, 2006]. Since this first successful attempt, many graphical interfaces have been developed to ease the use of motion planning algorithms [Coleman et al., 2014]. For instances, [A. Şucan and Chitta, 2013] developed MoveIt!, [Brunner et al., 2016] developed RAFCON, OpenRAVE comes with

[3]converted to the standard metric system from OMPL that uses inches.

its graphical interface. For the DARPA Robotic Challenge, [Marion et al., 2016] developed Director and [Rodehutskors et al., 2015] developed their own interface using virtual reality headset. The goal for these interfaces is to make robotic tools more accessible and user-friendly. A prototype graphical interface has also been developed with HPP. It provides to non-expert user an easy way of generating a Graph of Constraint and of solving manipulation problems. For instance, the sequence in Figure 5.1 was programmed within ten minutes only using the graphical interface and with the same tools as the one used to generate the examples in next sections.



(a) Initial configuration (b)

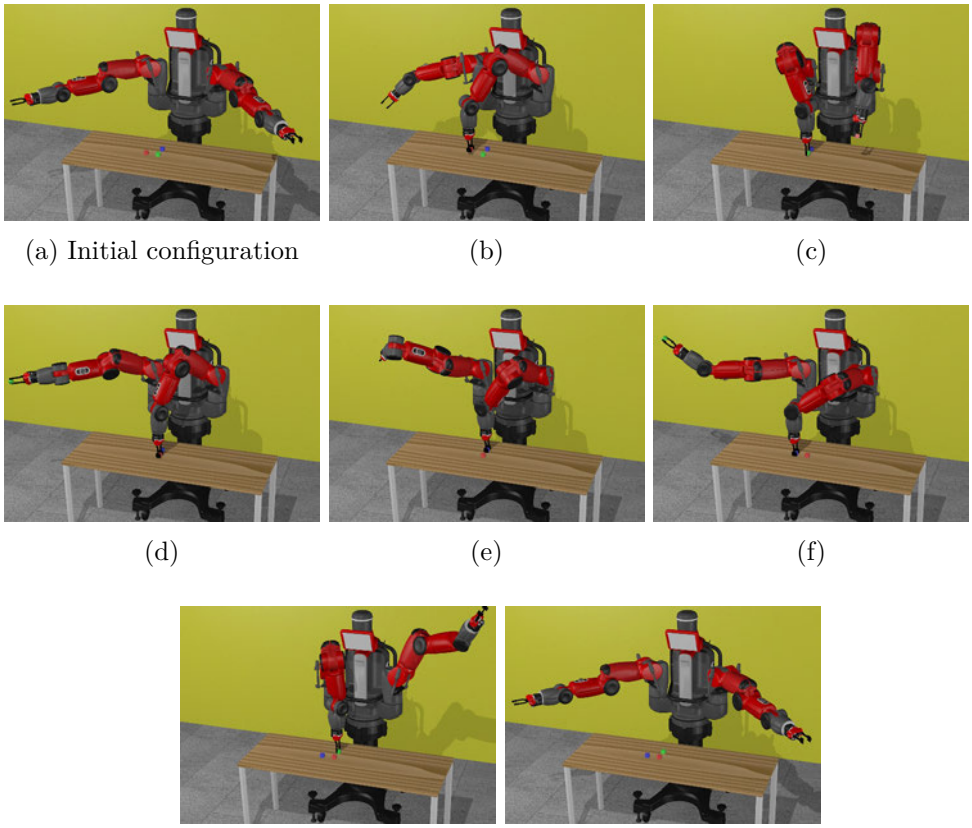
(c) (d)

(e) (f) Goal configuration

Figure 5.1: Robot programming with HPP. The robot UR5 is programmed with a graphical interface to move shaver parts. It must move them from the left carousel to the center one, where they are to be processed, and then move them to the right carousel. This is a use case of the European project Factory-In-A-Day.

5.2 Manipulator arm

5.2.1 Rearrangement planning



(a) Initial configuration (b) (c)

(d) (e) (f)

(g) (h) Goal configuration

Figure 5.2: Rearrangement planning: Baxter robot permutes the position of 3 boxes.

In these simulations, Baxter robot must move some boxes on a table in four different settings, numbered 1, 2, 3 and 4. In the first two cases, only the right arm of the robot is used. In the last two cases, both arms are used. The three first cases have only two boxes. Case 4 has three boxes. The constraint graph of case 1 is given in Figure 5.3. In the first case, the box positions are only shifted and the problem is monotone, *i.e.* there exists a solution which manipulates each box only once, one after the other. In the other 3 cases, the boxes are to be permuted. There are no monotone solution to these 3 problems. At least one intermediate box position must be found or simultaneous manipulation must occur.

Table 5.3 and Figure 5.2 summarizes the results. The solver is able to find solutions in all the four cases. For cases 1 and 2, the problem is not difficult and the solution comes quickly. Cases 3 and 4 corresponds to artificially-hard toy problems, yet the planner is able to discover a solution in a reasonable amount of

time.

From case 3 to case 4, the time to solve the problem explodes. This shows a limitation of the approach. It does not scale to a high number of objects because there is no task planner.

			Solving time (s)			Number of nodes		
Case	Nb	Arm	Min	Med	Max	Min	Med	Max
1	2	Right	0.15	1.4	3.5	11	55	141
2			1.1	4.6	10.6	42	199	482
3		Both	3.7	18	60	103	273	832
4	3		76	397	1028	664	3659	8830

Table 5.3: Benchmark of rearrangement planning with Baxter robot: Minimum, median and maximum solving time and number of nodes, over 20 runs, for the cases described in Section 5.2.1.

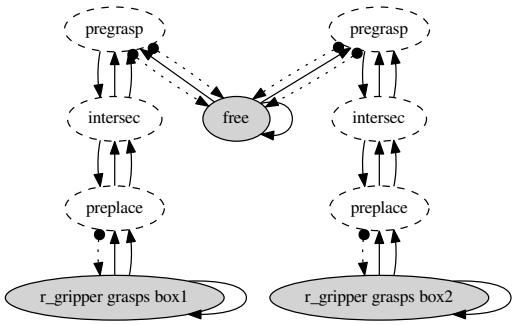
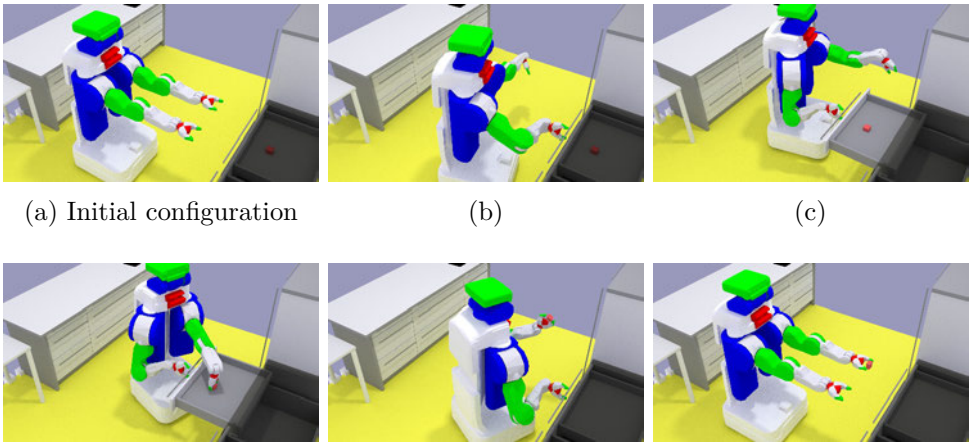


Figure 5.3: Constraint graph for Case 1 with Baxter robot: 2 boxes and considering only the right arm. The constraint graph is produced by an extension of Algorithm 4.1 that inserts way-point corresponding to approaching positions of the gripper in front of the object.

Case 1 shows that the presented approach is not as efficient as task planning based approaches on monotone cases. In contrast, it can solve non-monotone instances, as shown in cases 2, 3, 4. These cases shows the ability to discover new common valid placement. Case 3 and 4 also show the ability of the planner to consider simultaneous manipulation.

5.2.2 Tool use inference

In this example, PR2 robot picks up a box. Figure 5.4(a) and 5.4(f) show the initial and goal configurations. The planner must infer that it must open the drawer to pick up the box. Figure 5.4 shows the result path. Another interesting point is that interaction between the drawer and the object is inferred from the manipulation



(a) Initial configuration (b) (c)

(d) (e) (f) Goal configuration

Figure 5.4: PR2 robot picks up a box in a drawer

rules. The placement constraint ensures us the object moves as the drawer moves.

The constraint graph has been designed using the algorithm proposed in Section 4.2. The documentation consists of one handle and one contact surface for the object, one handle and one contact surface for the drawer and two grippers for PR2 robot.

The rules to limit the combinatorial were as follows. The left hand can grasp the box. The right hand can grasp the drawer. The drawer is always at a valid placement, An object pose is stable if the box surface and the drawer surface are in contact.

5.3 Humanoid robots

In this section, I apply the approach to humanoid robots. I demonstrate by some examples that the locomotion problem can be formulated as a manipulation problem. This is done by formulating quasi-static stability as a constraints. Then, I give some two examples of manipulation problems. They both illustrates simultaneous manipulation. The first also shows an articulated object. The second shows that it is able to cope with the crossed foliation issue.

5.3.1 Quasi-static walking motion

In this example, I model a locomotion problem as a manipulation planning problems. This approach is able to compute a quasi-static walking path for HRP-2. I consider three different scenarios. The environments are respectively a flat floor, a flat floor with an low obstacle and stairs. Contact constraints are build from contact surfaces in the documentation of the robot and the environment. The three following stability criteria are used.

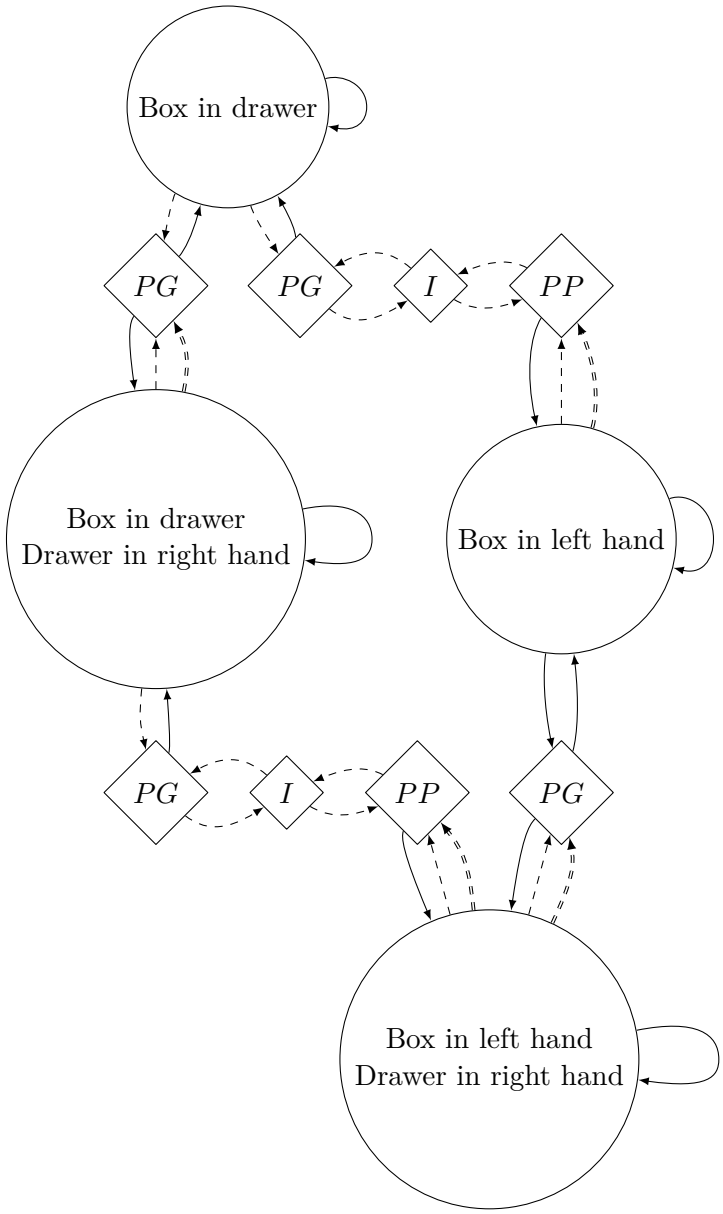
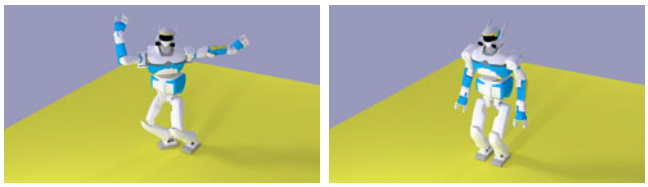


Figure 5.5: Graph of Constraint to pick up a box in a drawer. The diamond are way-point internal states: PG stands for pre-grasp, I for intersection of grasp and placement, and PP for pre-placement. The two foliated spaces are drawer placements and object placements.

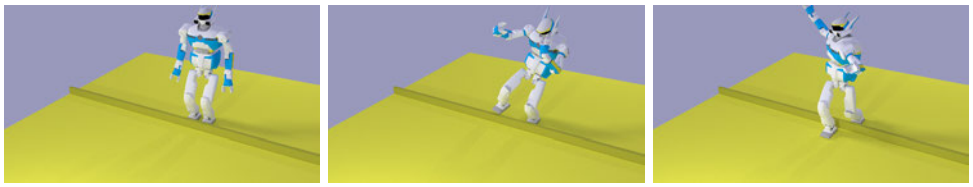


(a) Initial configuration (b) (c)

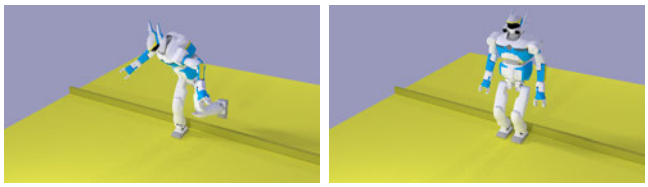


(d) (e) Goal configuration

Figure 5.6: Robot HRP-2 walking quasi-statically on flat floor.



(a) Initial configuration (b) (c)



(d) (e) Goal configuration

Figure 5.7: Robot HRP-2 quasi-statically steps over an obstacle on a flat floor.

The two first criteria are "Center of Mass (CoM) above one foot center", denoted by C^l_{CoM} and C^r_{CoM} for left and for right foot, and "CoM above the line segment linking the feet center", denoted by C^{line}_{CoM}, are described in Appendix B.

The last criterion is the one described in Section 2.3. It is a frictionless non-coplanar multiple contacts criterion. The three cases could be solved with the last stability criterion but, as already explained in Section 2.3, this criterion is time-consuming and has lower success ratio compared to the two others.

Figure 5.8 shows the graph for the two first cases. CoM_{lr} is constraint C^{line}_{CoM}, and CoM_l and CoM_r are constraint C^l_{CoM} and C^r_{CoM}. For the last scenario, only the frictionless non-coplanar multiple contacts constraint is used. There are six way-point transitions:

- two from double support to single support,

- two from single support to double support composed of only standard transition,
- two from single support to double support composed of one crossed foliation transition at the beginning and two standard transitions. This crossed foliation transition the algorithm to choose a foot pose on the floor for the free foot that has been visited by other connected components.

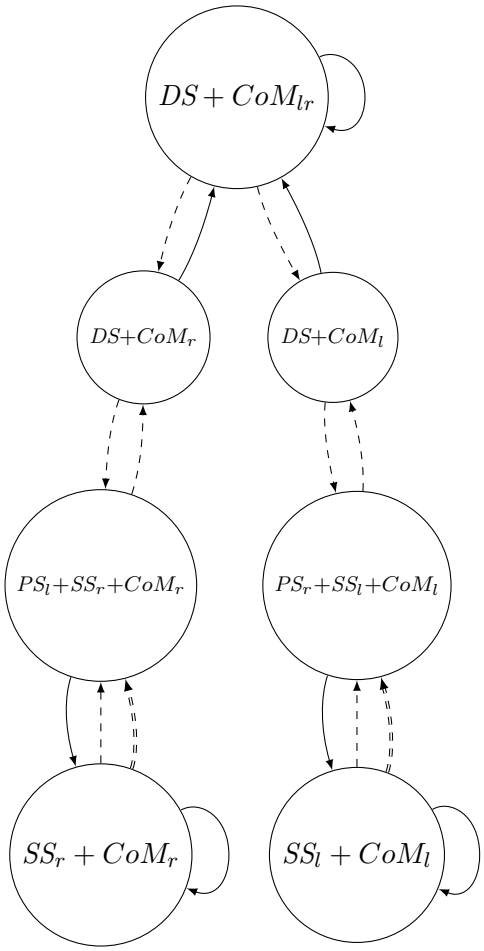
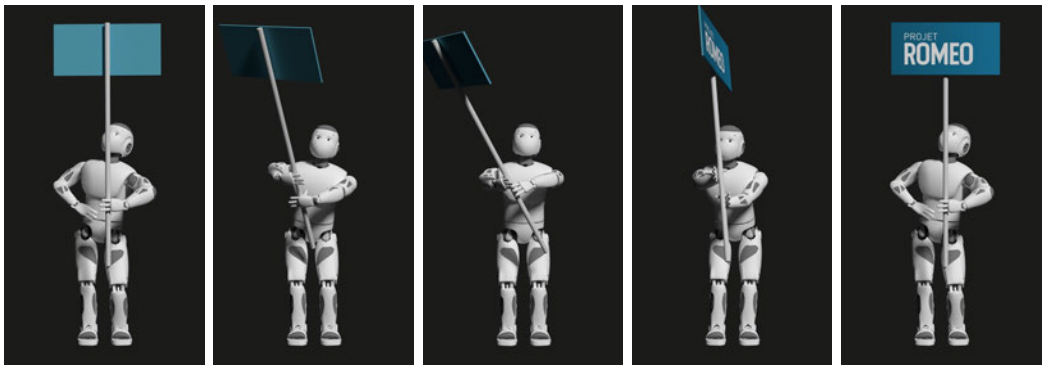


Figure 5.8: Constraint graph to generate quasi-static walking motions. l and r indexes stand for left and right foot, *DS* for Double Support, *SS* for Single Support, CoM_s for CoM on foot r and *PS* for pre Support, poses where the foot is close to its future pose on the floor.

Figure 5.6 and 5.7 shows the solution path found on flat floor and for stepping over. Figure 2.9 shows the path found for climbing stairs. The graph for this example is similar to the previous graph at the exception that it uses the third equilibrium criterion. These examples show the ability of finding common valid foot placement. Foot placement is considered in the same way as object placement. Indeed, in all these problems, there are two foliations crossing each other. They

also show that this approach is abstract enough to be applicable to a wide range of problems. This manipulation planner can thus be used to address locomotion problem.

5.3.2 Romeo holding a placard



(a) Initial config. (b) (c) (d) (e) Goal config.

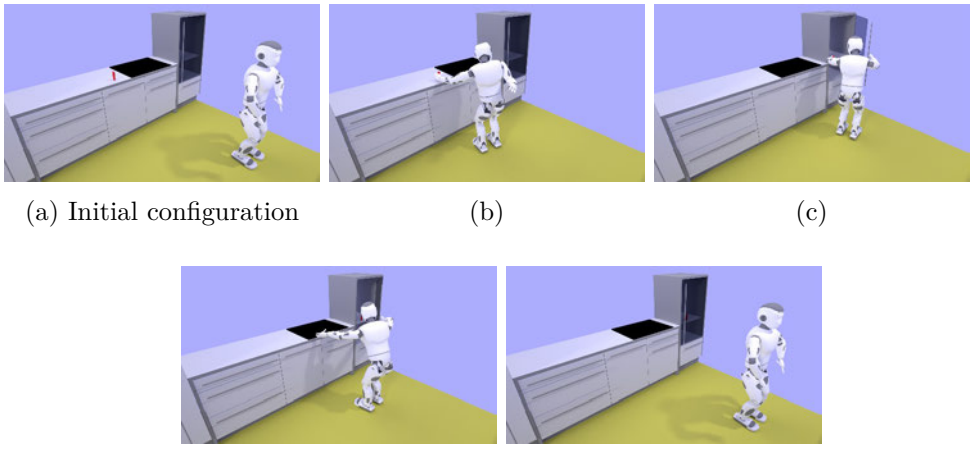
Figure 5.9: Robot Romeo holding a placard. In final configuration, the placard is rotated by 180 degrees. The manipulation planning algorithm needs to explore manifolds defined by right, left and both hand grasps.

In this example, Romeo holds a placard. The manipulation planning problem consists in rotating the placard around the vertical axis. To do so, the robot needs to go through a sequence of states where the robot holds the placard by the right hand, the left hand and both hands. Applying a task planning based approach for this problem seems difficult since the minimal sequence of tasks highly depends on the workspace of the robot arms and is very difficult to precompute.

	Min	Median	Max
Number of nodes	42	1370	7002
Solving time (s)	19	880	6500

Table 5.4: Benchmark of planning for Romeo holding a placard

The results obtained after 20 runs of the algorithm are displayed in Table 5.4. The variance of the computation time is surprisingly high. Path planning was not interrupted after a threshold time as usually done for difficult problems. Note that the model of the robot is very accurate (each hand has four fingers with 3 segments) and a lot of time is spent in collision checking. Moreover, the solution path goes through several “narrow passages” since grasping the pole requires to avoid a lot of collisions between the object and the fingers. Half of the time, the problem is solved in less than 15 minutes.



(a) Initial configuration (b) (c)

(d) (e) Goal configuration

Figure 5.10: Robot Romeo puts a box in a fridge.

5.3.3 Grasping behind a door

I now demonstrate the implementation of a more complex manipulation planning problem. A humanoid robot has to grasp an object and place it inside a fridge while opening the fridge door. The sequence of high level actions is not given. First, I generate a path for the sliding robot. Then the path can be post processed to generate a walking trajectory as proposed by Dalibard et al. [2013] or Carpentier et al. [2016]. I successfully planned a path where Romeo robot takes an object and puts it in a fridge. Figure 5.10 shows the solution found.

Conclusion

This thesis addressed the manipulation planning problem. I analysed the structure of the configuration space to model it. The contributions are organized around the Graph of Constraint. The overall approach to solve a manipulation planning problem is simple. First, the user documents the robots and the objects. Then, he specifies some elementary association rules which are used to automatically generate a Graph of Constraint. This graph provides useful information to the Manipulation-RRT to plan a manipulation path.

From a broader point of view, this approach could be used to program robots for industrial tasks. A prototype graphical interface based on this approach has had promising results in this direction. With a few interactions with the interface, it is possible to build a Graph of Constraint. This graph defines what are the interactions of the robot with the world. Then, manipulation planning problem can be defined using this graph and some other inputs, like position of a part to be processed. Then, this problem can be solved to generate a trajectory for each part. While the task is defined off-line in an abstract manner by the graph, the problem can be solved online with varying inputs. The big advantage of this approach is that it is robust to changes the inputs. For instance, the position of each input parts does not have to be the same. However, this assumes that some technical challenges are solved. For instance, the position of the parts has to be measured by some mean and three dimensional models of the environment and the objects must be available. Next section discusses possible future works.

Perspectives

Optimization of manipulation paths The manipulation paths produced by randomized algorithms contain erratic and undesirable movements. Although there exists optimization techniques in motion planning, they do not apply directly on manipulation paths because of the foliated structure of the configuration space. They can be split into a sequence of constrained path, to which standard optimization techniques applies. This method is implemented in Humanoid Path Planner (HPP) and gives good but insufficient results. Indeed, although it has the advantage of being easy to implement, the method cannot optimize the configuration linking the paths together. The method is doomed not to succeed in finding a path better than the linear interpolation between these configurations. For instance, this method is not able to change a "bad" grasp or a placement. Research in optimization for manipulation paths has already generated interest [Zhang and Shah, 2016, Hadfield-Menell et al., 2016].

Manipulator arms Many robots, like most manipulator arms, have an interval for revolute joints of width bigger than 2π. For instance, UR5 and IIWA robots have

intervals of width 4π. This means that, for n joints, there are 2^n configurations for the same robot geometry. Random configurations often are at more the 2π along one revolute joint coordinate from its nearest neighbour. This is not desirable as it leads to motions where the robot does a full loop around one articulation. Moreover, it greatly slows down the search because it generates longer paths more often colliding with the environment. On my experiments, changing the interval from $]-2\pi, 2\pi]$ to $]-\pi, \pi]$ has a non-negligible impact on solving time and improves a lot the quality of the solution path. However it is not a satisfactory solution since it reduces the capabilities of motion. Another possible solution I did not explore is to apply a modulus operation on rotation joints after having selected the nearest neighbour.

Variety of tasks This thesis did not consider tasks such as sweeping a surface. These actions are useful and appears often in factories. The problem of pushing object has already been addressed[Barry, 2013]. This framework considers actions which can be represented as constraint. It means they do not depend on time. Pushing could be written by a time-dependant constraint of the form $\mathbf{f} : \mathcal{CS} \times [0, 1] \to \mathbb{R}^n$. Sweeping a surface is different because it does not involve an object. The goal is not defined through the motion of an object.

Metric on the configuration space Randomized algorithm are sensitive to the chosen metric on the configuration space. Indeed, this metric is used to select the nearest neighbour and thus what part of the tree is extended. However, the foliated structure of the configuration space in manipulation planning should impact the nearest neighbour search. Indeed, configurations close to each other with a standard euclidean metric may be connected only by a very long path. For instance, two configurations corresponding to the same robot pose but a different object pose should account for the fact that the robot must also move.

Integration with task planning The novelty of the Manipulation-RRT (M-RRT) is not to call a motion planning for each action it takes. In the current version, it decides of what action to take randomly. This is interesting as it shows that, although it helps, a task planner is not required for simple tasks. However, the problem becomes intractable when the number of states in the Graph of Constraint increases. Integrating a task planner may help to keep this tractable. Moreover, the task planner could highly benefit from the Graph of Constraint. First, motion planners used by most integrated task and motion planning approaches return at most a graph of configuration along with a solution path when found. It does not provide information on the search. In its current implementation, the Graph of Constraint returns statistical information about elementary actions that were taken and the reason when they fail. A task planner may use these to help the M-RRT to choose actions to take.

Appendix A

Bound of the Hessian matrix of a kinematic chain

A.1 Notations

Consider a tree of joints with N joints and d degrees of freedom (DoFs). The types of joint considered are:

- *rotation*: denoted by R, 1 DoF;
- *translation*: denoted by T, 1 DoF;
- *spherical*: denoted by $SO(3)$, 3 DoFs.

Each joint can have one or several DoFs. $p(n)$ denotes the parent joint of joint n and $p^k(n) = p^{k-1}(p(n))$ is its k-th ancestor. O_n, resp. $\mathbf{n_n}$, is the center, resp. normal[1], of joint n. Let L be the longest possible distance between two centers of joint.

Let $I(n)$ be the set of joint indexes of the chain between the root joint and joint n. Let $I_R(n)$ (resp. $I_T(n)$, $I_{SO(3)}(n)$) be the subset of indexes of rotation (resp. translation, SO_3) joint in $I(n)$. They are such that $I(n) = I_R(n) \cup I_T(n) \cup I_{SO(3)}(n)$ and $I_R(n) \cap I_T(n) = I_T(n) \cap I_{SO(3)}(n) = I_{SO(3)}(n) \cap I_R(n) = \emptyset$.

$[\mathbf{u}]_\times$ is the cross matrix, *i.e.* $[\mathbf{u}]_\times \mathbf{v} = \mathbf{u} \times \mathbf{v}$. $[[\mathbf{M}_{3,n}]]_\times$ is the cross tensor, *i.e.* $[[\mathbf{M}_{3,n}]]_\times \mathbf{x}_n = [\mathbf{M}_{3,n}\mathbf{x}_n]_\times$.

A.2 Jacobian

The Jacobian of the placement of joint n is ${}^n\mathbf{J} \in \mathbb{R}^{6\times d}$. ${}^n\mathbf{J}_j^C$ is the block corresponding to joint j in ${}^n\mathbf{J}$. ${}^n\mathbf{J}_j^v$ denotes the velocity part of ${}^n\mathbf{J}_j^C$. ${}^n\mathbf{J}_j^\omega$ denotes the angular velocity part of ${}^n\mathbf{J}_j^C$. Both ${}^n\mathbf{J}_j^v$ and ${}^n\mathbf{J}_j^\omega$ have 3 rows. Their number of columns is the number of DoF of the joint type. This is summarized in (A.1).

$$ {}^n\mathbf{J} = \begin{pmatrix} {}^n\mathbf{J}_0^C & \dots & {}^n\mathbf{J}_d^C \end{pmatrix} = \begin{pmatrix} {}^n\mathbf{J}_0^v & \dots & {}^n\mathbf{J}_d^v \\ {}^n\mathbf{J}_0^\omega & \dots & {}^n\mathbf{J}_d^\omega \end{pmatrix} \tag{A.1} $$

The elements of ${}^n\mathbf{J}$ are as follows. The unlisted blocks are matrices of zeros.

[1] This is an extended notation. $SO(3)$ joints have no normal and $\mathbf{n_n}$ is never used.

- Joint $p^k(n)$ is a rotation:

$$ {}^n\mathbf{J}^v_{p^k(n)} = \mathbf{O_n O_{p^k(n)}} \times \mathbf{n_{p^k(n)}},\ {}^n\mathbf{J}^\omega_{p^k(n)} = \mathbf{n_{p^k(n)}} $$

- Joint $p^k(n)$ is a translation:

$$ {}^n\mathbf{J}^v_{p^k(n)} = \mathbf{n_{p^k(n)}},\ {}^n\mathbf{J}^\omega_{p^k(n)} = 0 $$

- Joint $p^k(n)$ is a $SO(3)$:

$$ {}^n\mathbf{J}^v_{p^k(n)} = \left[\mathbf{O_n O_{p^k(n)}}\right]_\times,\ {}^n\mathbf{J}^\omega_{p^k(n)} = I_3 $$

Thus, when $p^k(n)$ is a rotation, we have $||{}^n\mathbf{J}^v_{p^k(n)}|| \leq L$ and $||{}^n\mathbf{J}^v_{p^k(n)}|| \leq 1$. Otherwise, we have $||{}^n\mathbf{J}^{v,w}_{p^k(n)}|| \leq 1$.

A.3 Hessian

The Hessian matrix is defined by ${}^n\mathbf{H}_{i,j,k} = \frac{\partial {}^n\mathbf{J}_{i,j}}{\partial \dot{\mathbf{q}}_k}$. Similarly to ${}^n\mathbf{J}^{v,\omega}_j$, we denote ${}^n\mathbf{H}^{v,\omega}_{j,k} = \frac{\partial {}^n\mathbf{J}^{v,\omega}_j}{\partial \dot{\mathbf{q}}_k}$.

A.3.1 Element of the Hessian matrix

- Joint $p^j(n)$ is a rotation[2]:

$$ \begin{aligned} j > k, {}^n\mathbf{H}^v_{p^j(n),p^k(n)} &= \left[\mathbf{n_{p^j(n)}}\right]_\times {}^n\mathbf{J}^v_{p^k(n)} \\ {}^n\mathbf{H}^\omega_{p^j(n),p^k(n)} &= 0 \\ j \leq k, {}^n\mathbf{H}^v_{p^j(n),p^k(n)} &= \left[\mathbf{n_{p^j(n)}}\right]_\times \left({}^n\mathbf{J}^v_{p^k(n)} - {}^{p^j(n)}\mathbf{J}^v_{p^k(n)}\right) \\ &\quad - \left[\mathbf{O_n O_{p^j(n)}}\right]_\times \left(\left[\mathbf{n_{p^j(n)}}\right]_\times {}^{p^j(n)}\mathbf{J}^\omega_{p^k(n)}\right) \\ {}^n\mathbf{H}^\omega_{p^j(n),p^k(n)} &= -\left[\mathbf{n_{p^j(n)}}\right]_\times {}^{p^j(n)}\mathbf{J}^\omega_{p^k(n)} \end{aligned} $$

- Joint $p^j(n)$ is a translation:

$$ {}^n\mathbf{H}^v_{p^j(n),p^k(n)} = -\left[\mathbf{n_{p^j(n)}}\right]_\times {}^{p^j(n)}\mathbf{J}^\omega_{p^k(n)} $$

$$ {}^n\mathbf{H}^\omega_{p^j(n),p^k(n)} = 0 $$

[2]The case $j = k$ can be covered by both expression. This can be shown by using $\left[\mathbf{n_{p^j(n)}}\right]_\times \left[\mathbf{n_{p^k(n)}}\right]_\times = 0$ and ${}^{p^j(n)}\mathbf{J}^v_{p^k(n)} = 0$.

- Joint $p^j(n)$ is a $SO(3)$:

$$
\begin{aligned}
{}^n\mathbf{H}^v_{p^j(n),p^k(n)} &= \left[\left[{}^n\mathbf{J}^v_{p^k(n)} - {}^{p^j(n)}\mathbf{J}^v_{p^k(n)}\right]\right]_\times \\
{}^n\mathbf{H}^\omega_{p^j(n),p^k(n)} &= 0
\end{aligned}
$$

A.3.2 Bounds

By the mean value theorem, an upper bound of $||||{}^n\mathbf{H}(\mathbf{q})||||_F$ on $\mathcal{CS}$ is a suitable Lipschitz constant for ${}^n\mathbf{J}$. An explicit upper bound is computed in this section.

The Hessian norm can be written as follows.

$$
||||{}^n\mathbf{H}||||_F^2 = \sum_{j\in I(n),k\in I(n)} ||{}^n\mathbf{H}^v_{j,k}||_2^2 + ||{}^n\mathbf{H}^\omega_{j,k}||_2^2 \tag{A.2}
$$

I consider in joint trees so the Hessian matrix is sparse. Joints not in $I(n)$ do not influence the placement of joint n. It gives the following bound.

$$
||||{}^n\mathbf{H}||||_F^2 \leq |I(n)|^2(\max(9L^2,(L+2)^2)+1) \tag{A.3}
$$

For convenience, I introduce an intermediate variable $\sigma(m,\chi,\kappa)$ and rewrite (A.2).

$$
\begin{aligned}
\sigma(m,\chi,\kappa) &= \sum_{j\in I_\chi(n),k\in I_\kappa(n)} ||{}^n\mathbf{H}^m_{j,k}||_2^2 \\
||||{}^n\mathbf{H}||||_F^2 &= \sum_{m\in\{v,\omega\},(\chi,\kappa)\in\{R,T,SO(3)\}^2} \sigma(m,\chi,\kappa)
\end{aligned}
$$

Table A.1 and A.2 below summarizes the upper bound of σ.

A.3.2.1 Upper bound for $\sigma(v,\chi,\kappa)$

$\chi\backslash\kappa$	R	T	$SO(3)$
R	$\lvert I_R\rvert^2L^2$	$4\lvert I_R\rvert\lvert I_T\rvert$	$2\lvert I_R\rvert\lvert I_{SO(3)}\rvert L^2$
T	$2\lvert I_R\rvert\lvert I_T\rvert$	0	$2\lvert I_T\rvert\lvert I_{SO(3)}\rvert$
$SO(3)$	$4\lvert I_R\rvert\lvert I_{SO(3)}\rvert L^2$	$4\lvert I_T\rvert\lvert I_{SO(3)}\rvert$	$4\lvert I_{SO(3)}\rvert(\lvert I_{SO(3)}\rvert-1)L^2$

Table A.1: Upper bound for $\sigma(v,\chi,\kappa)$.

I know give a proof of the values in Table A.1.

Proof. From the element of the Hessian matrix given in Section A.3.1, we get:

$$\begin{aligned}\sigma(v,T,T) &= 0\\ \sigma(v,R,T) &\leq 4|I_R(n)||I_T(n)|\\ \sigma(v,T,R) &\leq 2|I_T(n)||I_R(n)|\\ \sigma(v,T,SO(3)) &\leq 2|I_T(n)||I_{SO(3)}(n)|\end{aligned}$$

and $\forall j \in I_{SO(3)}$:

$$\forall j \in I_{SO(3)}, ||{}^n\mathbf{H}^v_{j,k}||_2^2 = 2(||{}^n\mathbf{J}^v_k - {}^j\mathbf{J}^v_k||^2) \leq \begin{cases} 2L^2 & \text{if } k \in I_R(n)\\ 2 & \text{if } k \in I_T(n)\\ 2L^2 & \text{if } k \in I_{SO(3)}(n)\end{cases}$$

So we get:

$$\sigma(v, SO(3), \kappa) \leq 2|I_{SO(3)}(n)| \times \begin{cases} 2L^2|I_R(n)| & \text{if } \kappa = R\\ 2|I_T(n)| & \text{if } \kappa = T\\ 2L^2(|I_{SO(3)}(n)| - 1) & \text{if } \kappa = SO(3)\end{cases}$$

The Jacobi identity of cross product on ${}^n\mathbf{H}^v_{j,k}$ where $j \in I_R(n)$ gives:

$$\begin{aligned}\sigma(v,R,R) &\leq |I_R(n)|^2 L^2\\ \sigma(v,R,SO(3)) &\leq 2|I_R(n)||I_{SO(3)}(n)|L^2\end{aligned}$$ □

A.3.2.2 Upper bound for $\sigma(\omega, \chi, \kappa)$

$\chi \backslash \kappa$	R	$SO(3)$
R	$\|I_R\|\left(\|I_R\| - 1\right)/2$	$\|I_R\|\|I_{SO(3)}\|$

Table A.2: Upper bound for $\sigma(\omega, \chi, \kappa)$. Omitted combination are null.

I know give a proof of the values in Table A.2.

Proof. From the element of the Hessian matrix given in Section A.3.1, we get:

$$\begin{aligned}\forall \kappa \in \{R, T, SO(3)\}, \quad \sigma(\omega, T, \kappa) &= 0\\ \forall \kappa \in \{R, T, SO(3)\}, \quad \sigma(\omega, SO(3), \kappa) &= 0\\ \sigma(\omega, R, T) &= 0\\ \sigma(\omega, R, SO(3)) &\leq |I_R(n)||I_{SO(3)}(n)|\end{aligned}$$

Moreover, as $\forall (j,k) \in I_R(n) \times I(n) \mid j \geq k$, ${}^n\mathbf{H}^\omega_{j,k} = 0$,

$$\sigma(\omega, R, R) \leq \sum_{j,k \in I_R(n)^2, j<k} 1 = \frac{|I_R(n)|(|I_R(n)| - 1)}{2}$$ □

A.3.2.3 Upper bound for the Hessian

The above inequalities put together gives the following bound.

$$
\begin{aligned}
||||^n\mathbf{H}||||_F^2 \leq \Big(|I_R|^2 + 6|I_R||I_{SO(3)}| + 4|I_{SO(3)}|(|I_{SO(3)}| - 1)\Big) \times L^2 \\
+6|I_T||I_R| + 6|I_T||I_{SO(3)}| + \frac{|I_R|(|I_R| - 1)}{2} + |I_R||I_{SO(3)}|
\end{aligned}
\tag{A.4}
$$

Appendix B

Static stability constraints

The following paragraphs detail static stability criterion for a humanoid robot with one or two co-planar horizontal contacts.

Center of Mass (CoM) above the foot Using notation of Figure B.1, the constraint is simply:

$$(\mathbf{r}_{com}(\mathbf{q}) - \mathbf{r}_{foot}(\mathbf{q})) \times \mathbf{z}_{world} = \mathbf{0}$$

CoM above the line segment linking the feet center The motivation is to define a criterion which make it possible to switch from left single support to right single support. Figure B.1 schematically represents a humanoid robot. $\mathbf{r}_r(\mathbf{q})$ denotes the right foot position and $\mathbf{r}_l(\mathbf{q})$ the left one. $\mathbf{r}_{com}(\mathbf{q})$ denotes the position of the CoM.

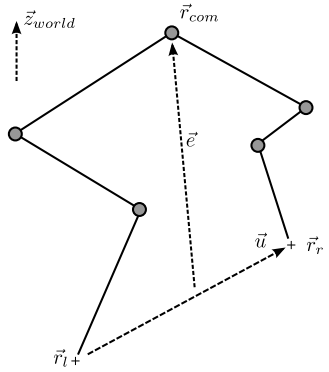


Figure B.1: Schematic model of a legged robot.

$\mathbf{u} = \mathbf{r}_r(\mathbf{q}) - \mathbf{r}_l(\mathbf{q})$ denotes the vector from left to right foot and $\mathbf{e} = \mathbf{r}_{com}(\mathbf{q}) - \frac{\mathbf{r}_l + \mathbf{r}_r}{2}$ denotes the vector from the middle point between the feet and the CoM. The constraint is defined as follows.

$$\langle \mathbf{e} \times \mathbf{u}, \mathbf{z}_{world} \rangle = \mathbf{0} \tag{B.1}$$

$$\langle \mathbf{r}_{com}(\mathbf{q}) - \mathbf{r}_l(\mathbf{q}), \mathbf{u} \rangle \geq 0 \tag{B.2}$$

$$\langle \mathbf{r}_r(\mathbf{q}) - \mathbf{r}_{com}(\mathbf{q}), \mathbf{u} \rangle \geq 0 \tag{B.3}$$

Equation (B.1) imposes $\mathbf{r}_{com}$ to be on the vertical plane passing by points $\mathbf{r}_r$ and $\mathbf{r}_l$. Equations (B.2) and (B.3) reduces the plane to a slice "between" the feet. Overall, the constraint is of dimension 3.

Note the $\mathbf{u}$ is not normalized for numerical reasons. First, the Jacobian of $\mathbf{u}$ is much cheaper to evaluate than of $\frac{\mathbf{u}}{||\mathbf{u}||_2}$. Second, it avoids numerical issues around $||\mathbf{u}||_2 = 0$. This favours to configuration with smaller $||\mathbf{u}||_2$ as they have smaller error. For the extreme case $\mathbf{u} = \mathbf{0}$, the criterion does not ensure stability. Fortunately, such configuration are in collisions.

Height of the CoM To the previous constraints, I also add a constraint on the height of the center of mass, of the form $\langle (\mathbf{r}_{com}(\mathbf{q}) - \mathbf{r}_{foot}(\mathbf{q})), \mathbf{z}_{world} \rangle = h_{ref}$

Bibliography

Ioan A. Şucan and Sachin Chitta. "moveit!", 2013. URL `http://moveit.ros.org`. 82

Rachid Alami, Thierry Siméon, and Jean-Paul Laumond. A geometrical approach to planning manipulation tasks. the case of discrete placements and grasps. In *5th International Symposium on Robotics Research*, Tokyo, Japan, 1989. 14

Rachid Alami, Jean-Paul Laumond, and Thierry Siméon. Two manipulation planning algorithms. In *Algorithmic Foundations of Robotics (WAFR)*, pages 109–125. AK Peters, Ltd. Natick, MA, USA, 1994. 14

Jennifer Barry, Leslie Kaelbling, and Tomás Lozano-Pérez. A hierarchical approach to manipulation with diverse actions. In *IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany, 2013. 15, 18

Jennifer Lynn Barry. *Manipulation with diverse actions*. PhD thesis, Massachusetts Institute of Technology, 2013. 94

Dmitry Berenson, Siddhartha S Srinivasa, Dave Ferguson, and James J Kuffner. Manipulation planning on constraint manifolds. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 625–632. IEEE, 2009. 9

Dmitry Berenson, Siddhartha S Srinivasa, and James Kuffner. Task space regions: A framework for pose-constrained manipulation planning. *The International Journal of Robotics Research*, page 0278364910396389, 2011. 3, 15

Stanislas Brossette, Adrien Escande, Grégoire Duchemin, Benjamin Chrétien, and Abderrahmane Kheddar. Humanoid posture generation on non-euclidean manifolds. In *IEEE International Conference on Humanoid Robots (Humanoids)* , 2015. URL `https://sites.google.com/site/adrienescandehomepage/publications/2015_Humanoids_Brossette.pdf`. 39

Sebastian Brunner, Rico Steinmetz, Franz; Belder, and Andreas Dömel. Rafcon: A graphical tool for engineering complex, robotic tasks. In *IEEE/RSJ Intelligent Robots and Systems (IROS)*. IEEE, 2016. 82

Stéphane Cambon, Rachid Alami, and Fabien Gravot. A hybrid approach to intricate motion, manipulation and task planning. *The International Journal of Robotics Research*, 28(1):104–126, 2009. doi: 10.1177/0278364908097884. URL `http://ijr.sagepub.com/content/28/1/104.abstract`. 15, 16

John Canny. *The complexity of robot motion planning*. MIT press, 1988. 6

Justin Carpentier, Steve Tonneau, Maximilien Naveau, Olivier Stasse, and Nicolas Mansard. A Versatile and Efficient Pattern Generator for Generalized Legged Locomotion. In *IEEE International Conference on Robotics and Automation (ICRA)*, Stockholm, Sweden, May 2016. URL `https://hal.archives-ouvertes.fr/hal-01203507`. 91

Howie M Choset. Principles of robot motion: theory, algorithms, and implementation, 2005. 6

David Coleman, Ioan A. Şucan, Sachin Chitta, and Nikolaus Correll. Reducing the barrier to entry of complex robotic software: a MoveIt! case study. *Journal of Sofware Engineering for Robotics*, 5(1):3–16, May 2014. `http://moveit.ros.org`. 82

Benoit Dacre-Wright, Jean-Paul Laumond, and Rachid Alami. Motion planning for a robot and a movable object amidst polygonal obstacles. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2474–2480. IEEE, 1992. 14, 17, 57, 60

Sébastien Dalibard and Jean-Paul Laumond. Linear dimensionality reduction in random motion planning. *The International Journal of Robotics Research*, 30 (12):pp. 1461–1476, 2011. 7

Sébastien Dalibard, Alireza Nakhaei, Florent Lamiraux, and Jean-Paul Laumond. Manipulation of documented objects by a walking humanoid robot. In *IEEE International Conference on Humanoid Robots (Humanoids)* , pages 518–523. IEEE, 2010. 3, 14, 15

Sébastien Dalibard, Antonio El Khoury, Florent Lamiraux, Alireza Nakhaei, Michel Taïx, and Jean-Paul Laumond. Dynamic walking and whole-body motion planning for humanoid robots: an integrated approach. *The International Journal of Robotics Research*, 32(9-10):1089–1103, 2013. URL `http://hal.archives-ouvertes.fr/hal-00654175`. 2, 9, 12, 30, 34, 38, 91

Rosen Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute, August 2010. URL `http://www.programmingvision.com/rosen_diankov_thesis.pdf`. 80

Andrew Dobson and Kostas Bekris. Planning representations and algorithms for prehensile multi-arm manipulation. In *IEEE/RSJ Intelligent Robots and Systems (IROS)*, 2015. 15

Adrien Escande, Sylvain Miossec, Mehdi Benallegue, and Abderrahmane Kheddar. A Strictly Convex Hull for Computing Proximity Distances With Continuous Gradients. *IEEE Transactions on Robotics*, 30(3):666–678, 2014. doi: 10.1109/TRO.2013.2296332. 73

Roy Featherstone. *Rigid body dynamics algorithms*. Springer, 2008. 59

Hans Joachim Ferreau, Christian Kirches, Andreas Potschka, Hans Georg Bock, and Moritz Diehl. qpOASES: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6(4):327–363, 2014. 38

Richard E Fikes and Nils J Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971. 13

Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Ffrob: An efficient heuristic for task and motion planning. In *Algorithmic Foundations of Robotics (WAFR)*, pages 179–195. Springer, 2015. 15, 18

Mamoun Gharbi, Juan Cortés, and Thierry Siméon. Roadmap composition for multi-arm systems path planning. In *IEEE/RSJ Intelligent Robots and Systems (IROS)*, Saint-Louis, USA, 2009. 15

Dylan Hadfield-Menell, Christopher Lin, Rohan Chitnis, Stuart Russell, and Pieter Abbeel. Sequential quadratic programming for task plan optimization. In *IEEE/RSJ Intelligent Robots and Systems (IROS)*, pages 5040–5047. IEEE, 2016. 93

André Haefliger. Feuilletages sur les variétés ouvertes. *Topology*, 9:183–194, 1970. ISSN 0040-9383. 17

Kensuke Harada, Tokuo Tsuji, and Jean-Paul Laumond. A manipulation motion planner for dual-arm industrial manipulators. in proceedings of. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 928—934, Hongkong, China, 2014. 15, 51, 57

Kris Hauser. Fast interpolation and time-optimization on implicit contact submanifolds. In *Proceedings of Robotics: Science and Systems*, Berlin, Germany, June 2013. doi: 10.15607/RSS.2013.IX.022. 2, 9, 10, 31

Kris Hauser and Victor Ng-Thow-Hing. Randomized multi-modal motion planning for a humanoid robot manipulation task. *The International Journal of Robotics Research*, 30(6):678–698, 2011. 3, 15

Giray Havur, Guchan Ozbilgin, Esra Erdem, and Volkan Patoglu. Hybrid reasoning for geometric rearrangement of multiple movable objects on cluttered surfaces. In *IEEE International Conference on Robotics and Automation (ICRA)*, Hong Kong, China, 2014. 15

Jörg Hoffmann and Bernhard Nebel. The ff planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001. 13, 14

David Hsu, Tingting Jiang, John Reif, and Zheng Sun. The bridge test for sampling narrow passages with probabilistic roadmap planners. In *IEEE International*

Conference on Robotics and Automation (ICRA), volume 3, pages 4420–4426. IEEE, 2003. 7

Piotr Indyk and Jiri Matousek. Low-distortion embeddings of finite metric spaces, 2004. 6

Sören Jentzsch, Andre Gaschler, Oussama Khatib, and Alois Knoll. MOPL: A multi-modal path planner for generic manipulation tasks. In *IEEE/RSJ Intelligent Robots and Systems (IROS)*, September 2015. `http://youtu.be/1QRvjBw58bU`. 3, 15

Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal. Stomp: Stochastic trajectory optimization for motion planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4569–4574. IEEE, 2011. 8

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

Lydia E Kavraki, Petr Svestka, Jean-Claude Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996. 7, 14

Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The international journal of robotics research*, 5(1):90–98, 1986. 6

Alexander A Kirillov. *An introduction to Lie groups and Lie algebras*, volume 113. Cambridge University Press Cambridge, 2008. 20

Jana Koehler and Jörg Hoffmann. On reasonable and forced goal orderings and their use in an agenda-driven planning algorithm. *Journal of Artificial Intelligence Research*, 12:338–386, 2000. 14

Athanasios Krontiris and Kostas Bekris. Dealing with difficult instances of object rearrangement. In *Robotics Science and Systems*, Roma, Italy, 2015. 15

Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991. 6

Jean-Paul Laumond. Kineo cam: a success story of motion planning algorithms. *IEEE Robotics Automation Magazine*, 13(2):90–93, June 2006. ISSN 1070-9932. doi: 10.1109/MRA.2006.1638020. 5, 82

Steven M Lavalle. Rapidly-exploring random trees: A new tool for path planning. Technical report, 1998. 7

Steven M LaValle. Planning algorithms. *Methods*, 2006:842, 2006. doi: 10.1017/CBO9780511546877. URL `http://ebooks.cambridge.org/ref/id/CBO9780511546877`. 6

Puttichai Lertkultanon and Quang-Cuong Pham. A single-query manipulation planner. *IEEE Robotics and Automation Letters*, 1(1):198–205, 2015. 15, 51, 57, 74

Andrew D Lewis. Semicontinuity of rank and nullity and some consequences, 2009. 26

Tomas Lozano-Perez. Spatial planning: A configuration space approach. *IEEE transactions on computers*, 100(2):108–120, 1983. 5

Nicolas Mansard and François Chaumette. Task sequencing for high-level sensor-based control. *IEEE Transactions on Robotics*, 23(1):60–72, 2007. 37

Nicolas Mansard, Florian Valenza, and Justin Carpentier. Pinocchio: Fast forward inverse dynamic for multibody systems , 2014. URL `http://stack-of-tasks.github.io/pinocchio/index.html`. 80

Pat Marion, Maurice Fallon, Robin Deits, Andrés Valenzuela, Claudia Perez-D'Arpino, Greg Izatt, Lucas Manuelli, Matthew Antone, Hongkai Dai, Twan Koolen, John Carter, Scott Kuindersma, and Russ Tedrake. Director: A user interface designed for robot operation with shared autonomy. *To Appear in the Journal of Field Robotics*, 2016. 83

Leon Mirsky. Symmetric gauge functions and unitarily invariant norms. *The quarterly journal of mathematics*, 11(1):50–59, 1960. 26

Dennis Nieuwenhuisen, A Frank van der Stappen, and Mark H Overmars. An effective framework for path planning amidst movable obstacles. In *Algorithmic Foundations of Robotics (WAFR)*, pages 87–102. Springer, 2008. 14

Joseph O'Rourke and Jacob E Goodman. *Handbook of discrete and computational geometry*. CRC Press, 2004. 6

Jun Ota. Rearrangement of multiple movable objects-integration of global and local planning methodology. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 1962–1967. IEEE, 2004. 15

Jia Pan, Sachin Chitta, and Dinesh Manocha. Fcl: A general purpose library for collision and proximity queries. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3859–3866. IEEE, 2012. 80

Erion Plaku and Gregory D Hager. Sampling-based motion and symbolic action planning with geometric and differential constraints. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5002–5008. IEEE, 2010. 18

Stanislav P Ponomarev. Submersions and preimages of sets of measure zero. *Siberian Mathematical Journal*, 28(1):153–163, 1987. ISSN 1573-9260. doi: 10.1007/BF00970225. URL `http://dx.doi.org/10.1007/BF00970225`. 52

Andrea Del Prete, Steve Tonneau, and Nicolas Mansard. Fast Algorithms to Test Robust Static Equilibrium for Legged Robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2016. 39

Vladimir Rakočević. On continuity of the moore-penrose and drazin inverses. *Matematički Vesnik*, 49(209):163–172, 1997. 26

Nathan Ratliff, Matt Zucker, J Andrew Bagnell, and Siddhartha Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 489–494. IEEE, 2009. 8

James Reeds and Lawrence Shepp. Optimal paths for a car that goes both forwards and backwards. *Pacific journal of mathematics*, 145(2):367–393, 1990. 8

Tobias Rodehutskors, Max Schwarz, and Sven Behnke. Intuitive bimanual telemanipulation under communication restrictions by immersive 3d visualization and motion tracking. In *IEEE International Conference on Humanoid Robots (Humanoids)* , pages 276–283. IEEE, 2015. 83

JT Schwartz and M Sharir. Motion planning and related geometric algorithms in robotics. In *Proc. Int. Congress of Mathematicians*, pages 1594–1611, 1986. 5

Thierry Siméon, Jean-Paul Laumond, and Carole Nissoux. Visibility-based probabilistic roadmaps for motion planning. *Advanced Robotics*, 14(6):477–493, 2000. 7

Thierry Siméon, Jean-Paul Laumond, Juan Cortés, and Anis Sahbani. Manipulation planning with probabilistic roadmaps. *The International Journal of Robotics Research*, 23(7-8), July 2004. URL `http://ijr.sagepub.com/content/23/7-8/729.short`. 3, 16, 51, 57

Siddharth Srivastava, Eugene Fang, Lorenzo Riano, Rohan Chitnis, Stuart Russell, and Pieter Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 639–646. IEEE, 2014. 15, 16

Mike Stilman. Global manipulation planning in robot joint space with task constraints. *IEEE Transactions on Robotics*, 26(3):576–584, 2010. 9, 10

Mike Stilman and James Kuffner. Planning among movable obstacles with artificial constraints. *The International Journal of Robotics Research*, 27(11-12):1295–1307, 2008. 14

Mike Stilman and James J Kuffner. Navigation among movable obstacles: Real-time reasoning in complex environments. *IEEE International Conference on Humanoid Robots (Humanoids)* , 2(04):479–503, 2005. 16

Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. *IEEE Robotics and Automation Magazine*, 19(4):72–82, December 2012. doi: 10.1109/MRA.2012.2205651. `http://ompl.kavrakilab.org`. 79

Steve Tonneau, Nicolas Mansard, Chonhyon Park, Dinesh Manocha, Franck Multon, and Julien Pettré. A reachability-based planner for sequences of acyclic contacts in cluttered environments. In *Int. Symp. Robotics Research (ISRR), (Sestri Levante, Italy), September 2015*, 2015. 12

Pierre Tournassoud, Tomas Lozano-Perez, and Emmanuel Mazer. Regrasping. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 4, pages 1924–1928, 1987. 74

Gordon Wilfong. Motion planning in the presence of movable obstacles. In *Proceedings of the fourth annual symposium on Computational geometry*, pages 279–288. ACM, 1988. 14

Chongjie Zhang and Julie A Shah. Co-optimizing task and motion planning. In *IEEE/RSJ Intelligent Robots and Systems (IROS)*, pages 4750–4756. IEEE, 2016. 93

Glossary

CHOMP Covariant Hamiltonian Optimization for Motion Planning. 8

CoM Center of Mass. 12, 28, 38, 88, 89, 101, 102

DA-RRT Diverse Action RRT. 18

DoF degree of freedom. 5, 22, 39, 43, 44, 50, 57, 58, 62, 63, 69, 95

FF Fast Forward. 14, 16

Graph of Constraint Graph of states and transitions encoding manipulation rules. ix, xiii, 2, 3, 41–43, 45–47, 49, 56, 63, 64, 66, 69, 73, 74, 80, 83, 87, 93, 94

HPP Humanoid Path Planner. xi, 2, 4, 31, 79, 81–83, 93

IP interpolation point. 11, 27–32, 34, 35

M-RRT Manipulation-RRT. ix, 3, 42, 47, 48, 52, 54, 55, 79, 80, 93, 94

NAMO Navigation Among Movable Obstacles. 14

NR Newton-Raphson. 9, 10, 24–27, 29, 30, 34, 37, 52, 81

OMPL Open Motion Planning Library. 79–82

OpenRAVE Open Robotics Automation Virtual Environment. 79, 80, 82

PRM Probabilistic RoadMaps. 7, 8, 17

RGD Randomized Gradient Descent. 9

RHP Recursive Hermite Projection. 10, 11, 31–35

RRT Rapidly exploring Random Tree. 2, 3, 7–9, 11, 12, 18, 30, 31, 34, 47, 49, 50

SSC Small Space Controllability. 58

SSM Small Space Manipulability. ix, 59

STOMP Stochastic Trajectory Optimization for Motion Planning. 8

TS Tangent Space Sampling. 9, 10

V-PRM Visibility PRM. 7

Notations

$\mathcal{CS}$ Configuration space of the robot. 5–12, 16, 20, 22–27, 35, 37, 41–43, 45–47, 52, 54, 55, 57–59, 61, 62, 64, 73, 81, 94, 97

$\mathcal{CS}_{free}$ Free configuration space. 6, 7, 9, 17

$\mathcal{CS}_{obs}$ Obstacle configuration space. 6

$\mathcal{CG}$ Subset of valid grasp of $\mathcal{CS}$. 16, 17, 23, 45, 46, 57, 58, 61–63

$\mathcal{CP}$ Subset of valid placement of $\mathcal{CS}$. 16, 17, 45, 46, 57, 58, 61, 62

$\mathfrak{se}(3)$ Lie algebra of $SE(3)$. 59, 60, 70

$SE(3)$ Special Euclidian group. 43, 57, 59, 70

$SO(2)$ Special orthogonal group. 20, 21

$SO(3)$ Special orthogonal group. 20, 21, 95–99

Résumé en français:

Cette thèse traite du problème de planification de mouvement pour objets documentés. La difficulté du problème réside dans le couplage d'un problème symbolique et d'un problème géométrique. Les approches habituelles combinent la planification de tâche et la planification de mouvement. Elles sont complexes à implémenter et coûteuse en temps de calcul. Notre approche se différencie sur trois aspects.

Le premier aspect est un cadre théorique modélisant les mouvements admissibles du robot et des objets. Ce modèle théorique utilise des contraintes pour lier tâche symbolique et chemins géométriques accomplissant cette tâche. Un graphe de contrainte permet de modéliser les règles de manipulation. Un algorithme de planification utilisant ce graphe est proposé.

Le deuxième aspect est la gestion de chemin contraint. Dans le cadre de la manipulation, un définition abstraite sous forme de contrainte numérique est nécessaire. Un critère de continuité pour les méthodes de type Newton-Raphson est proposé pour assurer la continuité de trajectoire dans des sous-variétés.

Le dernier aspect est la documentation des objets. Certaines informations, facile à définir pour l'être humain, accélère grandement la recherche d'une solution. Cette documentation, spécifique à chaque objet et préhenseur, est utilisée pour générer un graphe de contrainte, facilitant ainsi la spécification et la résolution du problème.

Mots clés: Planification de manipulation, Planification sous contraintes, Génération de trajectoire continue, Affordance, Objets documentés

Abstract:

This thesis tackles the manipulation planning for documented objects. The difficulty of the problem is the coupling of a symbolic and a geometrical problem. Classical approaches combine task and motion planning. They are hard to implement and time consuming. This approach is different on three aspects.

The first aspect is a theoretical framework to model admissible motions of the robot and objects. This model uses constraints to link symbolic task and motions achieving such task. A graph of constraint models the manipulation rules. A planning algorithm using this graph is proposed.

The second aspect is the handling of constrained motion. In manipulation planning, an abstract definition of numerical constraint is necessary. A continuity criterion for Newton-Raphson methods is proposed to ensure the continuity of trajectories in sub-manifolds.

The last aspect is object documentation. Some information, easy to define for human beings, greatly speeds up the search. This documentation, specific to each object and end-effector, is used to generate a graph of constraint, easing the problem specification and resolution.

Key words: Manipulation planning, Constrained planning, Continuous trajectory generation, Affordance, Documented objects