KTH Computer Science
and Communication

# Object manipulation by humanoid robots

Constrained motion planning using a graph of constraints applied to object manipulation by humanoid robots.

JOSEPH MIRABEL

Master's Thesis in Systems, Control and Robotics, at LAAS-CNRS
Supervisor: Danica Kragic
Examiner: Danica Kragic

# Abstract

Humanoid robots have been an active field of research during the past few decades owing to the wide variety of motions and interactions possible via Humanoid. Nevertheless, planning motions is still challenging in many cases. This thesis tackles the problem of object manipulation. A graph of constraints is introduced to transform object manipulation tasks in sequences of constraints. Then, a constrained motion planner, using the graph of constraints, is used to find a suitable statically balanced path for a sliding robot. Affordance is used to provide the required information to build the graph of constraints. An algorithm based on Rapidly exploring Random Tree navigates through the graph of constraints. Basic experiments of this method were performed with the humanoid robot HRP-2.

# Sammanfattning

## Objektmanipulation för humanoida robotar.

Humanoida robotar har varit ett aktivit området för forskning under de senaste årtiondena på grund av deras stora möjligheter för rörelser och interaktioner. Rörelseplanering är fortfarande problematiskt i många fall. Detta examensarbete tar itu med problemet med objektmanipulation. En graf av bivillkor införs för att omvandla objektmanipulationsupgifter till sekvenser av bivillkor. Denna graf änvands av rn rörelseplanerare för att hitta en lämplig statiskt balanserade väg för en glidande robot. Affordance används för att tillhandahålla den information som krävs för att bygga upp grafen och en algoritm baserad på Rapidly exploring Random Tree änvands för att navigera genom graffen. Grundläggande försök med denna metod utförts med den humanoida roboten HRP-2.

# Acknowledgements

I would like to offer my special thanks to Florent Lamiraux, Directeur de Recherche at LAAS-CNRS, Toulouse, France, for his valuable suggestions during this research work and his willingness to offer his time. I really appreciate the opportunity given to me to conduct this work.

My special thanks are extended to the members of GEPETTO team of LAAS-CNRS. To the permanent researchers for their availability and their answers to my questions. To the interns, PhD and Post-Doc students for having made my time at LAAS very enjoyable.

Professor Danica Kragic, both my supervisor and examiner, for her assistance and her availability.

Assistance provided by Patric Jensfelt, my professor and Master program director, Manja Schubert and Magnus Lindqvist, my international coordinators at KTH, is greatly appreciated.

Finally, I would like to acknowledge the support provided by my roommates during this work.

# Contents

# List of Algorithms

# List of Figures

# Chapter 1

# Introduction

## 1.1 Overview

A humanoid robot is, by definition, a system looking like human beings, that is a highly redundant and complex system. In the last few decades, our interest in such systems has raised a lot of new problems. Walking turned out to be much harder than driving a car. Moreover, making a humanoid robot walk is not sufficient to make it useful for humans. Like human being, a humanoid robot needs to be able to perform actions in its environment, by manipulating external objects. Its own motions must remain only a means and not the ultimate goal. In such a context, object manipulation becomes a key problem.

The high redundancy of humanoid robots gives them great capabilities for both navigation and manipulation. Navigation, a means, and manipulation, an requirement, makes for a global and a very challenging problem. Because manipulation involves external objects and the robot itself, it is a harder problem than navigation. Actually, a manipulation problem can be seen as a navigation problem of a more complex system.

A humanoid robot is a system with many actuators and sensors. The actuators are motors, one for each joints, and the sensors give measurements of the angle of each joint. A humanoid robot is usually equipped with more sensors like cameras, inertial measurement unit, force and torque sensors and sonar sensors. These sensors will not have any influence on our work, so we do not consider them.

Motion planning refers to a fundamental need in robotics: algorithms that convert high-level specifications from humans into low-level descriptions of how to move. For example, from a task like *"Put the blue box in the drawer"* to a sequence of configuration the robot can follow. The algorithm must determine how to move the robots active degrees of freedom, i.e. actuated degrees of freedom, in order to move the box, which position is defined by passive degrees of freedom, without hitting obstacles or loosing balance. Motion planning focuses primarily on geometrical planning, considering translations and rotations and ignoring dynamics and other differential constraints. Trajectory planning refers to the problem of taking the solution from a robot motion planning algorithm and determining how to move along the solution in a way that respects the mechanical limitations of the robot.

The position of all the joints are not enough to represent the position of the robot in the 3D world because it they do not give the position of the robot relatively to the world. The position of one body of the robot, called free-flyer, is needed. A configuration of the robot is a vector where the first six values represent the position of the free-flyer and the following values represent the internal degrees of freedom of the robot. A configuration fully represents the robot in the 3D world.

Figure 1.1 represents this system from a control point of view. The right part is the feedback control loop and the left part is planning.

The control loop input is a configuration of the robot, called $q_i$. The control loop corrects the error $e$ so that the measurement $q_m$ of the real configuration, $q$, of the robot tends to $q_i$.

Planning can have different kinds of inputs, depending on the way the tasks are specified. In Figure 1.1, the input is a configuration to be reached. Many other inputs are possible, like a goal position of the hand. Because planning takes too much time, computations are done offline. Planning requires the initial configuration of the robot and a map of the world. In Figure 1.1, planning is divided in two phases: path planning and trajectory planning. A path is a sequence of configuration without the notion of time. A trajectory is a path parameterized in time. It means that we cannot take speed or acceleration into account during the path planning step. The dynamic balance of the robot is thus achieved in the second step, while the first step can ensure no more than a statically-balanced path. Section 3.2 of this report explains why the problem can be split. As it will be explained, this *"two phases"* approach makes a hard dynamic problem into a geometrical problem and an easier dynamic problem, thus simplifying the problem.

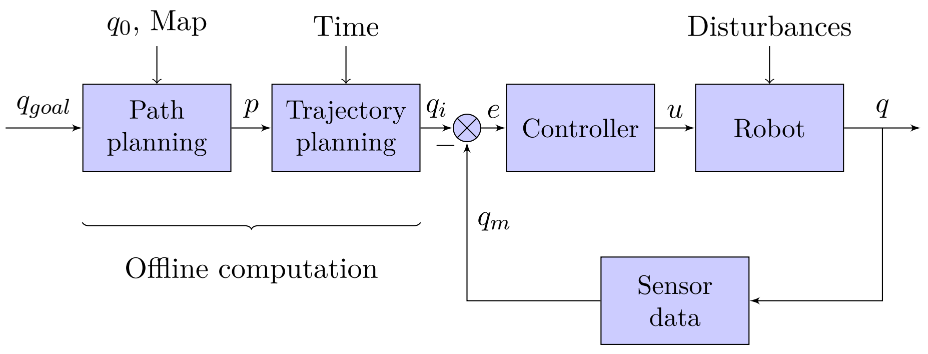

**Figure 1.1.** Robot control loop.

## 1.2 Path planning

Path planning consists in finding a feasible collision-free path between an initial configuration and a goal configuration [15]. A path is feasible if there exists a parameterization in time that allows the robot able to physically execute it. For instance, assuming that the robot does not jump, one foot should always be in contact with the floor. In practice, the contact has to be planar. A path is collision-free if the robot never collides with its environment when executing it.

In a discrete - or discretized - environment, path planning is a graph search. Classical algorithms such as Breadth First Search, Dijkstra's or A* algorithms provide powerful tools for path planning in these environments. These algorithms require an explicit representation of the environment. In a continuous environment, an explicit representation of the entire environment is practically impossible. It is possible to discretize the environment and use a classic graph search but, in practice, as the space of search is high dimensional, this method takes too much time to be considered as a possible solution.

In the last decades, randomized techniques have been developed. The main idea is to randomly sample the environment and try to connect the samples. Section 3.1 gives more details about sampling-based techniques.

In the framework of object manipulation, path planning is subject to constraint. Typically, an object cannot move by itself. The robot must hold it. Such restrictions of motion are called *"constraints"*. A mathematical definition is given in Section 4.1.2. By reducing the number of configurations accessible from a given configuration, constraints make classic random methods fail. A path planner for such problems is called constrained path planner. The difference is that it uses information provided by the constraints inside a classic path planner.

Optimality has not been a focus in this thesis. Finding a feasible path is already hard and a classic approach is to search a feasible path and then optimize it. Optimality for a path depends on a cost function. This cost function is designed according to criteria, such as path length, execution time, energy spent, human-like motion, or a compromise between them.

## 1.3 Problem statement

This thesis tackles the problem of path planning when object manipulation is involved. We focus on integrating objects in a path planning algorithm in order to make task specification easier. A geometric analysis is enough to make the robot capable of completing a task involving secondary action, without explicitly specifying them. Figure 1.2 represents the environment used for experiments. In this apartment, the task *"Put the blue box in the drawer"* requires several secondary tasks: two door openings and eventually manipulation of the drawer. A systematic way of finding this secondary tasks simplifies specification of tasks - in this example, from four tasks to one task.

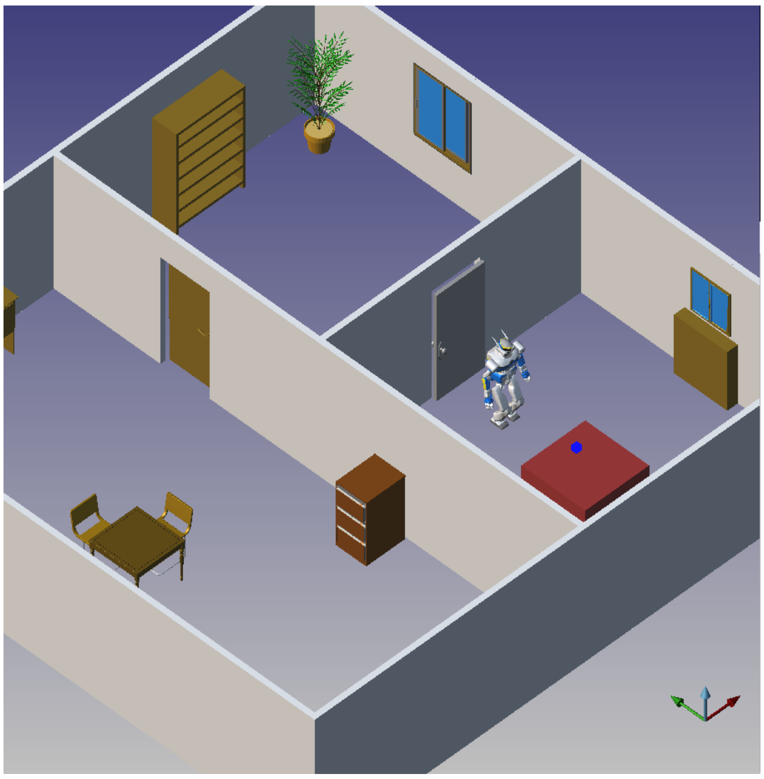

**Figure 1.2.** HRP-2 in an apartment with objects.

In this thesis, we integrate a graph of constraints in a constrained path planner. The graph of constraints can be seen as an interface between a classic constrained path planner and object manipulation. The method is scalable to any number of objects, but has been only tested with one single object.

The related work on this topic are presented in Part I. Then, we introduce in Part II a graph representation of constraints that provides an abstract structure for object manipulation. Then, this representation is integrated in an algorithm based on Rapidly exploring Random Tree. Eventually, the experimental results are shown.

# Chapter 2

# Background

This chapter gives background knowledge in motion planning. The first section describes the model used to represent robots as mechanical systems. Then, two important notions in object manipulation planning are introduced: configuration space and passive degrees of freedom. Eventually, a mathematical formulation of path planning problems is given.

## 2.1 Robot modeling

A robot is a sequence of bodies, linked by joints. It can also be seen as a sequence of joints linked by bodies. For example, *"arm"* and *"forearm"* are bodies linked by *"elbow"*, a joint. *"shoulder"* and *"elbow"* are linked by *"arm"*.

Focusing only on kinematics, the robot is a tree of joints, starting from a chosen root joint. A frame is associated to each joint. The transformation between the frame of a joint and its parent depends only on the parent joint parameter, i.e. its angle for a rotation joint. This joint tree forms a kinematic chain, which will give the position of any point of any body in the world frame.

The joint tree is extended with the bodies in order to allow collision detection. A body is linked to its parent joint. For instance, *"arm"* is linked to *"shoulder"*. Thus, from a configuration, joints are located in the world frame. Then bodies are located in the world frame, and we can check for collision between two bodies of the robot and with a static environment. Bodies have the notion of inertia so that it is possible to compute the center of mass of the robot.

In this context, the object to be manipulated is integrated in the model. The root joint is an anchor joint with two children, one for the robot and one for the object, as shown in Figure 2.1. The latter represents an object with only one joint, but it can have more joints.

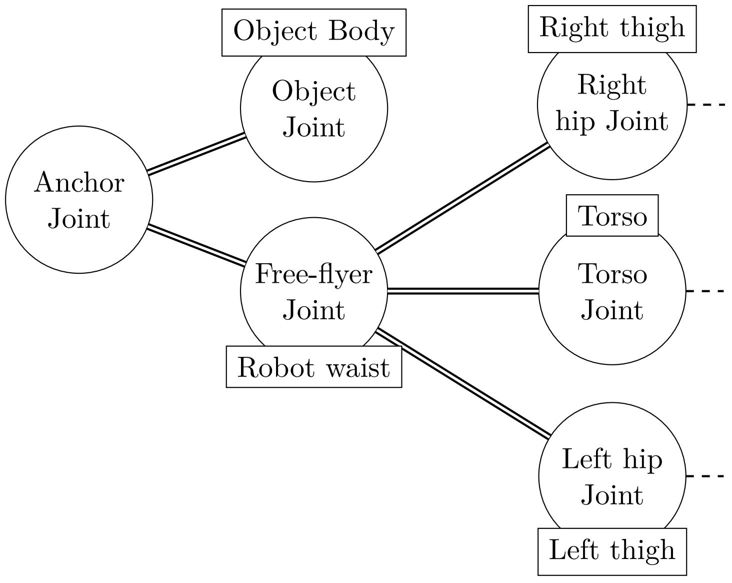

**Figure 2.1.** Tree of joints of the system $\mathcal{R} \times \mathcal{O}$.

## 2.2 Configuration space

Several ways of representing the environment in which a robot evolves have been developed and used. A representation can be explicit or implicit. In mathematics, an implicit definition of a function $f$ is of the form $R(f(x), x) = 0$, where $R(x, y)$ is a function - that can also be implicit, whereas an explicit definition is defined by a formula, $f(x) = R(x)$, where $R$ is known. For instance,

$$x^2 + f(x)^2 - 1 = 0, with f(x) \geq 0$$

is an implicit definition of $f$ and

$$f(x) = \sqrt{1 - x^2}$$

is explicit. A representation of the world is explicit if its elements are known. Typically, with an explicit representation, obstacles are directly represented and collision checking is fast. In general, an explicit representation requires a bigger amount of memory and is easier to deal with. A representation can also be continuous or discrete. It is harder to plan with continuous representation but a good and feasible discretization of the world is not always possible and depends strongly on cases.

In this work, we use a classical representation, continuous and implicit. Let $n$ be the number of joints of the robot. A position of the robot is represented by a configuration

$$q = (x, y, z, \alpha, \beta, \gamma, \theta_1, \cdots, \theta_n) \tag{2.1}$$

where:

- $(x, y, z, \alpha, \beta, \gamma)$ is a 6-dimension transformation (3 translations and 3 rotations). It represents the position of one body of the robot, called free-flyer.
- $(\theta_1, \cdots, \theta_n)$ represents the angular position of each joint of the robot. It also represents the internal degrees of freedom of the robot.

The set of possible configurations for the robot is called the configuration space, and denoted as $\mathcal{CS}$. By construction, $\mathcal{CS} \subset \mathbb{R}^{n+6}$. As the joints motions are usually bounded, $\mathcal{CS}$ is usually not equal to $\mathbb{R}^{n+6}$. The transformation between the 3D world and $\mathcal{CS}$ depends only on the kinematic chain of a robot.

Obstacles of the 3D world can be represented in the configuration space. Thus, we can define $\mathcal{CS}_{free}$, the set of collision-free configurations. It is a subset of $\mathcal{CS}$. It is important to understand that this space is only implicitly defined and not explicitly. To know if a configuration is in the set, collision checking has to be performed. Path planning consists of finding a feasible collision-free curve in $\mathcal{CS}$ from initial to goal configuration, or a feasible curve in $\mathcal{CS}_{free}$. In this thesis, we focus on geometric planning. As time is not taken into account, time-dependent notions, like balance of the robot, are transformed into geometric conditions.

Let us consider a robot $\mathcal{R}$ with $N_{\mathcal{R}}$ degrees of freedom in a 3-dimensional workspace. The workspace contains a movable object $\mathcal{O}$ that has $1 \leq N_{\mathcal{O}} \leq 6$ degrees of freedom. The configuration space of the system $\mathcal{R} \times \mathcal{O}$ is the Cartesian product of both configuration spaces. The subset of valid grasp configurations of $\mathcal{O}$ by $\mathcal{R}$ is denoted $\mathcal{CS}_{grasp}$ and is a sub-manifold of $\mathcal{CS}$. The domain in $\mathcal{CS}$ corresponding to valid placements of $\mathcal{O}$ is denoted $\mathcal{CS}_{plcmt}$.

## 2.3 Passive degrees of freedom

The robot internal degrees of freedom (DOF) are all actuated and their restrictions are basically bounds on their position, speed and/or acceleration. These are the active DOF. In the representation described earlier, the free-flyer joint represents an external and non-actuated DOF, Object joints are also not actuated because they can only be manipulated by the robot through its end-effectors. Hence, they are passive DOF. To modify their values, other actuated DOF are involved. Thus, the variables become coupled. The same holds with Objects too. Section 3.2 explains a two step planning method that transforms the free-flyer passive DOF into an active DOF. Nevertheless, object DOF remain passive.

Movable objects can have constraints on both motion and/or placement:

- Constraints on their motions refer to their kinematic chain. A door, for instance, is stable in every valid configuration but its motions are restricted to a rotation (or a translation for sliding door). Typically, for this kind of object, $N_M \leq 5$.
- Constraints on their placements refer to their stability constraints. We can think about any free flying object, like a cup, for instance. It is not always in a stable configuration but it has no constraint on its motion.

Our work has first addressed objects without constraints on their placement, like a door or a drawer. To extend our approach to more objects, the two following concepts are needed:

– a placement validator integrated into the configuration validator.

– a definition of placement as constraints on which we can project random configurations. This is needed because the sub-manifold of valid placements has no volume so randomly sampling a statically balanced configuration has a zero probability.

The constraints of valid placement can then be integrated in the graph of constraints, explained in Section 5.2.

## 2.4 Motion planning

Before giving more details about sampling-based methods, it is important to understand why these methods are relevant for our problem. The main characteristics of the problem are the continuity of the search space, and its high dimension.

Humanoid robots have many degrees of freedom; so $\mathcal{CS}_{free}$ is a high dimensional space. An explicit representation of such a space is impossible in practice. A discretization of $\mathcal{CS}$ with a good resolution would lead to a gigantic search space, as its size grows exponentially with the dimension of $\mathcal{CS}$. This makes explicit methods intractable for planning in high-dimensional continuous space. In other words, the problem can be solved in theory but, in practice, solving requires a too large an amount of time for the solution to be useful. These types of methods are usually complete. If a solution exists, it will be found in a finite amount of time. If no solution exists, the algorithm will return failure.

As an explicit representation of $\mathcal{CS}$ does not seem feasible, an implicit representation is used. It uses collision detection methods to sample configurations. The samples are generated while a discrete search is running. The search is guided bit randomness. Unlike the previous one, this approach is not complete. Our interest in sampling methods has grown during the past two decades because they can solve problem in continuous search spaces in a relatively short time. Two of the most fundamental sampling-based motion planning algorithms are *Probabilistic Roadmap* (PRM) [13] and *Rapidly exploring Random Trees* (RRT) [14].

Although sampling-based, these techniques are proven to be probabilistically complete [2]. That is, if a solution exists, the algorithm will find it. In manipulation planning, the configuration space is constrained and foliated. A definition and explanations of foliations are given in Section 4.1. The main issue that arises when dealing with constrained environments is that sub-manifolds have no volume. So the probability of randomly sampling a configuration in a sub-manifold is zero. A basic PRM or RRT is not able to solve such planning problem. Works on this matter have led to improvements, like [16], but, to our best knowledge, there has been no general solution to it. In this thesis, we use the RRT based algorithm introduced by Dalibard et al. [5].

As said earlier, planning a path is finding a curve in $\mathcal{CS}$. So a path planning problem can be define as follow:

**Problem.** Given $\mathcal{CS}_{free} \subset \mathbb{R}^n$, $(q_s, q_e) \in \mathcal{CS}^2_{free}$, find $\gamma : [0,1] \to \mathcal{CS}_{free}$ such that $\gamma(0) = q_s$ and $\gamma(1) = q_e$.

where $\mathcal{CS}_{free}$ is given implicitly, that is to say by a collision detection function.

This definition of path planning problems is not well adapted to computer programming. As it is hard to find a continuous unknown function, we provide a definition of the problem that is usually used when planning with random sampling methods.

A path is a sequence of direct paths. A direct path is made of two ending configurations and a method to interpolate configurations between its ends. A steering method builds a direct path between two configurations. For instance, a linear steering method builds direct paths that interpolate linearly between its ends.

A possible definition of a path planning problem is:

**Problem.** Given $\mathcal{CS}_{free} \subset \mathbb{R}^n$, $(q_s, q_e) \in \mathcal{CS}^2_{free}$, find $(q_i)_{0 \leq i \leq l}$ such that:

- $q_0 = q_s$ and $q_l = q_e$;
- a valid direct path between $q_i$ and $q_{i+1}$, where $0 \leq i \leq l-1$, exists.

# Part I

# Related work

# Chapter 3

# Algorithms

## 3.1 Planning algorithms

Humanoid robots are very frequently designed in open kinematic chains. They are easier to design and control this way. Navigation aims at finding a collision-free path in a given environment. Manipulation differs from navigation because it involves external passive objects. They are passive because their joints cannot be controlled directly, with actuators, but only indirectly, through the end-effectors of the robot. As a consequence, a closed kinematic chain may appear. It makes the process of finding a manipulation path more difficult, as explained by LaValle in [15, p. 338] or Cortés and Siméon in [3].

The main focus of the work in this thesis is to address this problem of manipulation using sampling-based motion planning. This method has already shown good results for high dimensional problems, which would be practically impossible to solve using techniques that explicitly represent the configuration space [15, chap. 5]. Several ideas have been implemented to reduce the complexity of navigation, like primitives [10] or PCA analysis [4].

In the following sections, a short review of sampling-based planning algorithms is provided. Some are better for single-query planning, others for multi-query planning. Single-query planning refers to planning a path between two configurations. Multi-query planning refers to several single-query on the same environment. For multi-query planning, it is generally faster to build a graph called *roadmap* that represents well the $\mathcal{CS}_{free}$ and then try to perform the queries. For single-query, a good representation of the environment is not important.

### 3.1.1 Rapidly exploring Random Tree

A short description of the classic algorithm is given here. For a full understanding, it is well presented in [15]. An RRT algorithm grows a tree from an initial configuration by repeating the following steps:

– sample a random configuration $q_{rand}$ in $\mathcal{CS}$.

**Algorithm 3.1.** Randomly exploring Random Tree

***Description:*** *Randomly exploring Random Tree from* $q_0$

```
1: procedure exploreTree($q_0$)
2:     $\mathcal{T}$.init($q_0$)
3:     for $i = 1 \to K$ do
4:         $q_{rand} \leftarrow$ Rand($\mathcal{CS}$)
5:         $q_{near} \leftarrow$ Nearest($q_{rand}$,$\mathcal{T}$)
6:         Extend($\mathcal{T}$, $q_{near}$, $q_{rand}$)
7:     end for
8: end procedure
```

– find the nearest configuration $q_{near}$ of $q_{rand}$ in the tree.
– extend $q_{near}$ toward $q_{rand}$

The vertices of the tree are configurations and the edges are direct paths. Algorithm 3.1 shows these steps. The random sample is taken in $\mathcal{CS}$ and the collision detection, seen as a black box, has to be done before inserting of the extended configuration in the tree. This way, the tree is a subset of $\mathcal{CS}_{free}$. The extension step adds a new configuration by moving from $q_{near}$ in the direction of $q_{rand}$. Several variants exist. A fixed increment can be used to move $q_{near}$, as in [14]. $q_{near}$ can also be moved until there is a collision, as in [15, Sec. 5.5.1]. In Algorithm 3.1, the tree is grown from $q_0$. In a single-query approach, a tree is grown from the initial and goal configuration, thus creating two connected component in the resulting graph. Whenever a node is added to one of the trees, we try to connect it to the other tree. If the trees can be connected, a path has been found.

RRT are well designed for single-query problem [15, sec. 5.5.1]. A variant of this algorithm, detailed in Section 6, has been used in this thesis.

### 3.1.2 Probabilistic Roadmap

Probabilistic Roadmap planners are designed in two phases: a learning phase and a query phase [12]. They are well designed for multi-query planning. During the learning phase, a roadmap is populated such that it represents as well as possible $\mathcal{CS}_{free}$. A roadmap is a set of collision-free configurations and collision-free paths. It is a graph in $\mathcal{CS}_{free}$. A roadmap represents well the $\mathcal{CS}_{free}$ if the nodes are equally distributed in $\mathcal{CS}_{free}$, if every region of $\mathcal{CS}_{free}$ is covered and if the connectivity of the roadmap represents the connectivity of $\mathcal{CS}_{free}$. Indeed, two connected regions of $\mathcal{CS}_{free}$ should be in the same connected component of the roadmap. This may be hard to achieve in case of *narrow passages*, as discussed in Section 4.2.

To build the roadmap, during the learning phases, the two following steps are repeated:

– Pick a random configuration in $\mathcal{CS}$, test it for collision and repeat until it is in $\mathcal{CS}_{free}$.
– Try to connect the former configuration to the roadmap.

During the query phase, a path between an initial and goal configuration is to be found. First, an attempt to connect these configuration to the roadmap is made. Then, a search in the roadmap is performed.

If no path is found, it is possible to improve the roadmap by going back to the learning phase. Thus, the planner can *adapt* the size of the roadmap to the difficulties encountered.

### 3.1.3 Visibility Roadmap

Visibility Roadmap is another useful variation of sampling-based roadmaps [18], [15, sec. 5.6.2]. Like PRM, the idea is to represent $\mathcal{CS}_{free}$ but the representation is kept small. The roadmap contains two kinds of nodes:

**Guard** Guards cannot see each other. Thus, each guard's visibility region is empty of other guards.

**Connectors** Each connector sees at least two guards.

The algorithm is similar to PRM's algorithm. During the learning phase, three cases may happen when a random configuration $q_{rand}$ is collision-free:

– $q_{rand}$ cannot be connected to any guards. Thus, it becomes a guard itself and it is inserted in the roadmap.

– $q_{rand}$ can be connected to guards from at least two connected components of the roadmap. Thus, it becomes a connector.

– In any other situations, $q_{rand}$ is discarded because it does not provide useful information in terms of visibility.

## 3.2 Small-space controllability

In this section, we define two notions that are crucial for the method developed in this work: statically balanced path and small-space controllability.

A configuration $q$ is statically balanced if the corresponding robot can stay in $q$ without moving. The robot does not fall. Using the fundamental principle of static, it is easy to understand that a configuration is statically balanced if the projection of the robot center of mass (COM) onto the ground is inside the support polygon of the robot. The support polygon is the convex hull of its contacts with the ground. A path is statically balanced if every configuration of this path is statically balanced. Let's assume that we have a humanoid robot that has the ability to slide on the floor. This assumption will make sense at the end of this section. In that case, it is possible to express in simple constraints what is a statically balanced path:

– the feet are always in planar contact with the floor;
– the COM is inside the support polygon.

Using the cart-table model, described in Appendix B, legged robots have been proven to be small-space controllable by Dalibard et al. in [7]. To describe this notion, let $Traj(q_f, q_t, T)$ denote the proposition "There exists a trajectory going from $q_f$ to $q_t$ in time $T$". A definition of controllability is:

$$(\forall (q_1, q_2) \in \mathcal{CS}^2), (\exists T > 0), Traj(q_1, q_2, T) \text{ is } True \tag{3.1}$$

From this property, the small-space controllability is derived. It is:

$$(\forall (q, T, \epsilon) \in (\mathcal{CS} \times \mathbb{R}^+ \times \mathbb{R}^+)), (\exists \eta > 0), (\forall q' \in Ball(q, \eta)),$$
$$q' \text{ is reachable in time T by a trajectory included in } Ball(q, \epsilon) \tag{3.2}$$

It typically means that, when the cart-table model is relevant, a collision-free and admissible trajectory can be found between two configurations that can be connected by a collision-free path. As a result, planning for dynamic systems can be reduced to geometric planning. In [7], the author also shows that it is always possible to obtain a collision-free and admissible dynamic trajectory for a humanoid robot from a collision-free statically balanced path for the corresponding sliding robot.

This is a very convenient result. It reduces the complexity of the problem. In this work, the two step trajectory planning algorithm presented in [7] is considered. The first consists of finding a statically balanced path. The second consists of generating a dynamically balanced walk trajectory. They are the two steps that can be seen in the control graph in Figure 1.1.

To ensure that it is always possible to approximate a collision-free statically balanced path for the sliding robot by a collision-free dynamically balanced path for the real robot, the cart-table model must be valid. Thus, two additional constraints are added to make the model valid:

– the COM is at a constant height;
– the robot waist is kept vertical.

Thereafter, the focus is on the first step, that is to say, finding a statically balanced manipulation path for a sliding robot. The constraint on the projection of the COM inside the support polygon is an inequality constraint, whereas the others are equality constraints. For practical reasons, this constraint will be transformed into a stricter constraint: the projection must lie at the center of the support polygon. The robot being more stable if the center of mass is centered with respect to the support polygon, this also helps the trajectory planning, as it gives more stability to the robot.

We will focus on geometric planning with the following stability constraints:

– the right foot is parallel to the floor;
– the right foot is at a constant height;
– the transformation from the right foot to the left foot is constant;
– the projection of the COM on the floor is at a fixed position with respect to the right foot;
– the COM is at constant height;
– the waist is vertical.

## 3.3 Affordance

Affordance is a relation between an object and an organism that affords the opportunity for that organism to perform an action. In our framework, it corresponds to the ability of providing helpful information to the planner. For instance, a door providing information about its handle.

Using affordance, the work of Dalibard et al. in [7] introduces a software architecture to implement a planner for the manipulation planning problem.

The architecture described is based on a graph of possible transitions. Figure 3.1 represents this graph in the context of a door manipulation problem. In this graph, each node of the graph represents a set of constraints on configurations and each edge represents a possible transition. The planning is done in two steps:

– a bounding box of the robot is generated. It encapsulates the object in such a way that a collision-free configuration for the bounding box is necessarily collision-free for the robot. It is very easy to plan for a bounding box as the dimensionality is drastically reduced, from 36 to 3 for the robot HRP-2. A plan for the bounding box is computed using a RRT-based algorithm.

– the path for the bounding box is transformed in a path for the robot. The footsteps are computed using a prioritized inverse kinematics solver[1].

The authors tackle the problem of door opening, and propose a good solution, providing to the planner the graph of possible transitions and the projections corresponding to the constraints. This approach is very interesting and this work aims at generalizing the idea of using constraints to help the planner to find a manipulation path. Nevertheless, although it makes planning very fast, the bounding box model will not be able to solve complex tasks because it hides a part of $\mathcal{CS}_{free}$. A sliding robot assumption is weaker than the bounding box model and is used instead.

[1]The solver is *hpp-gik*. More information about the solver are provided in Section 4.1.2 of this report.

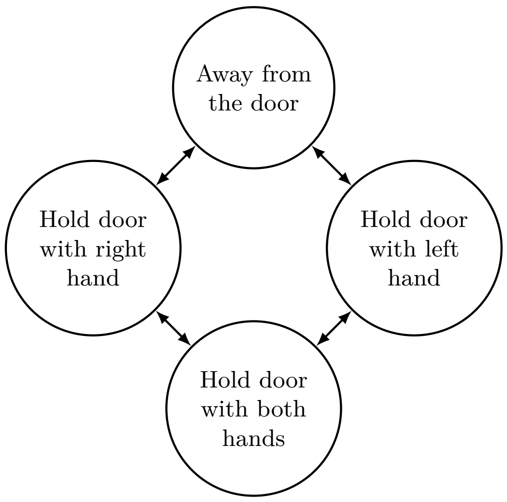

**Figure 3.1.** Graph representing the system $\mathcal{R} \times \mathcal{O}$ seen as a finite state machine, when opening and closing a door using two hands.

# Chapter 4

# Manipulation planning

## 4.1 Foliation and constraint

As manipulation planning involves passive DOF, the configuration space has a foliated structure. As explained in [20] and [19], finding a solution to a manipulation planning problem is finding a sequence of paths in $\mathcal{CS}_{free}$ that are of two kinds:

**Transit paths:** the object $\mathcal{O}$ is in a valid placement and only the robot $\mathcal{R}$ moves. Along this path, the configuration of $\mathcal{O}$ remains constant. A path is a transit path if it is in $\mathcal{CS}_{plcmt}$ and if the constraint of motionless object is respected. So a path in $\mathcal{CS}_{plcmt}$ is generally not a transit path.

**Transfer paths:** both the robot and the object can move, $\mathcal{R}$ holding $\mathcal{O}$. Such paths are in $\mathcal{CS}_{grasp}$.

The set of sub-manifolds of $\mathcal{CS}_{plcmt}$ with a constant configuration of $\mathcal{O}$ makes a foliation. The leaves of this foliation of $\mathcal{O}$ represents also all the configurations reachable from a configuration in $\mathcal{CS}_{plcmt}$ by a transit path. Once $\mathcal{R}$ is in a leaf of this foliation of $\mathcal{CS}_{plcmt}$, it cannot leave it without grasping an object. $\mathcal{CS}_{grasp}$ has the same structure. Given such a structure, a solution path is a sequence of transit/transfer path, that makes $\mathcal{R}$ navigating in various leaves of the two described foliations. Experiments using this structure for planning, in [20] and [19], have shown good results. This structure has been widely used in this work.

### 4.1.1 Foliation definition

Mathematically, a foliation of a $n$-dimensional manifold $M$ is an indexed family $L_\alpha$ 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.

To understand this definition, one can think about a book. In the classic 3D space, it forms the space $M$, and $n = 3$. We assume that it's shape is smooth enough to form a manifold. A foliation of the book is the family formed by all its pages, and the index is the page number. The $i$-th leaf of the foliation is thus the page $i$, a 2D sub-manifold. There is $3 - 2 = 1$ constant coordinate in the frame of the book, along the axis perpendicular to every page.

### 4.1.2 Constraint definition

To implement the constraints, we have used the Generalized Inverse Kinematic software called *hpp-gik*. We provide a quick description of how the constraints are implemented. For a complete description, see [21]. Let $N_{\mathcal{R}}$ (resp. $N_{\mathcal{O}}$) be the number of degrees of freedom of the robot (resp. the object).

A set of $m$ constraints is defined by a non-linear function $f$:

$$f(q) = 0_{\mathbb{R}^m} \tag{4.1}$$

where $f \in \mathcal{C}^1\left(\mathbb{R}^{N_{\mathcal{R}}+N_{\mathcal{O}}+6}, \mathbb{R}^m\right)$ and $m$ is the dimension of the constraint.

Algorithm 4.1 implements a Newton-Raphson method. Using the Moore-Penrose pseudo-inverse of the Jacobian of the constraint, it iteratively updates $q$ to decrease $||f(q)||$ with the update rule in Eq. 4.2. In Algorithm 4.1, $\alpha$ is an adaptive gain that helps the convergence stability.

$$\Delta q = -\alpha \left(\frac{\partial f}{\partial q}(q)\right)^{+} f(q) \tag{4.2}$$

This algorithm projects any configuration on the kernel of $f$. This can be used to generate configurations that satisfy the constraints.

### 4.1.3 Example

To picture the notion of constraint and foliation in our context, consider Figure 4.1. In this example, two constraints need to be defined:

– a grasping constraint: $f_g(q) = GripperPosition(q) - BoxPosition(q) = 0$

– a null velocity constraint of the object: $f_{o,p_{ref}}(q) = BoxPosition(q) - p_{ref} = 0$.

where the function $GripperPosition(q)$ (respectively $BoxPosition(q)$) returns the position of the gripper (resp. the box) in the configuration $q$ and $p_{ref}$ is the reference position of the box during a constrained path. This value is set to the $BoxPosition(q_{from})$.

**Algorithm 4.1.** Constraint solver

***Description:*** *Find $q$ such that $f(q) \leq \epsilon$*

```
1: procedure solveConstraint($q, f, \epsilon$)
2:     i ← 0
3:     while $||f(q)|| > \epsilon$ and $i \leq i_{max}$ do
4:         $q \leftarrow q - \alpha \left(\frac{\partial f}{\partial q}(q)\right)^{+} f(q)$
5:         i ← i+1
6:         $\alpha \leftarrow \alpha_{max} - w(\alpha_{max} - \alpha)$
7:
8:     end while
9:     if $||f(q)|| \leq \epsilon$ then
10:        return $q$
11:    else
12:        return failure
13:    end if
14: end procedure
```

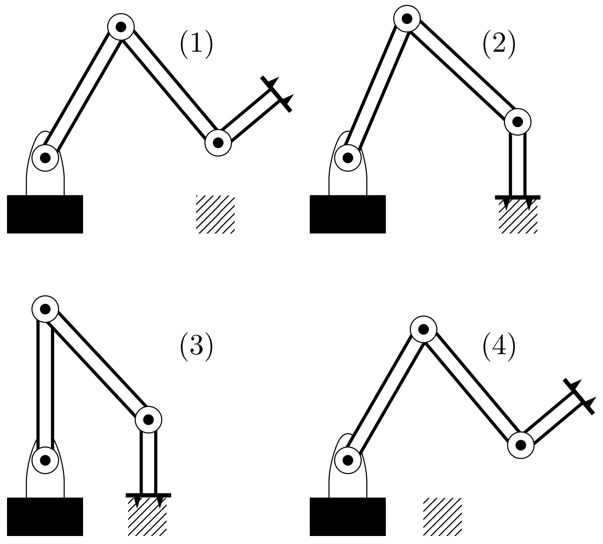

**Figure 4.1.** A 3-axis arm and an object (hatched box).

In configuration $q_2$ and $q_3$, the arm is holding the hatched box, while in configuration $q_1$ and $q_4$, it is not. The box has not moved from $q_1$ to $q_2$, and from $q_3$ to $q_4$. Thus, $q_1$ and $q_2$ are in the same leaf of $\mathcal{CS}_{plcmt}$. The path between these configurations is admissible regarding the $f_o$. This is also true for $q_3$ and $q_4$.

As for $q_1$ and $q_4$, the box has moved without being grasped, they cannot see each other, they are not in the same leaf. Thus a path from $q_1$ to $q_4$ necessarily contains configurations in $\mathcal{CS}_{grasp}$, for instance $q_2$ and $q_3$.

To build a path from $q_1$ to $q_4$, without knowing $q_2$ and $q_3$, one has to project configurations using both $f_o$ and $f_g$. After sampling a configuration $q_{rand}$, its nearest neighbor is found. Let's say $q_1$. To be able to connect $q_1$ and $q_{rand}$, the latter has to be projected in the same leaf of $\mathcal{CS}_{plcmt}$ as $q_1$, i.e. using $f_o$. In order to generate a configuration looking like $q_2$, $f_g$ is also used. A configuration looking like $q_3$ will be generated if the nearest neighbor is $q_4$.

An important remark has to be made at this step. A difference appears between constraints for paths and constraints for configurations. After having sampled $q_{rand}$, both $f_o$ and $f_g$ are required to generate $q_2$. But, a path between $q_1$ and $q_2$ needs only $f_o$.

## 4.2 Cluttered environment

For high dimensional problems, sampling-based algorithms are particularly powerful path planners. In cluttered environments however, classic sampling-based algorithms have poor performances. Two reasons explain the loss of efficiency.

First, the subspace of $\mathcal{CS}_{free}$ containing only feasible configurations may have a lower dimension than $\mathcal{CS}$. This especially happens when there are degrees of freedom that are passive, i.e. not directly actuated. For example, the first six elements of a configuration for a humanoid robot are not directly actuated. To move them, you need to be in contact with the ground. In this case, the subspace of admissible configurations is of a lower dimension than $\mathcal{CS}$ so it has no volume inside $\mathcal{CS}$. The volume of a plane in $\mathbb{R}^3$ is zero. As sampling in a zero-volume space has a zero probability, the problem has a null probability of being solved. This problem is specifically called *constrained motion planning.* Recent works on this topic have led to efficient solutions. This topic is addressed in section 4.3.

Then, the second reason for the loss of efficiency is cluttered environments. Suppose that a good solution to the first problem is used, that is to say the sampling space has the same dimension as the space of collision-free and feasible configurations. In cluttered environments, a *narrow passage* problem appears. These passages do not have a null probability of being sampled, but this probability is small. Thus, sampling-based method takes a long time to find *narrow passages.* For example in Figure 4.2, the green ball has to go through the channel to reach the red circle. Sampling inside the channel has a very low probability so it would take very long for classic sampling-based algorithm to solve this very basic problem. Some solutions have been proposed, for instance using principal component analysis in [4]. Constrained motion planning aims at finding paths in a zero volume sub-manifold. It can thus be a solution to the *narrow passage* problem. See [15, chap. 7.4] or [9] for more in-depth knowledge.

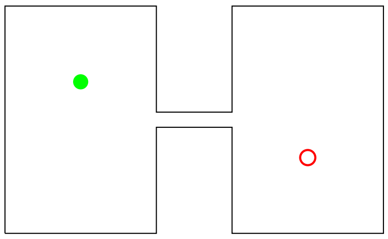

**Figure 4.2.** Basic example of cluttered environment. RRT will not find easily a path from green ball to red circle.

**Algorithm 4.2.** Constrained extend

***Description:*** *Extend $q_{near}$ towards $q_{rand}$ while staying on the manifold $\mathcal{M}$ defined by $f$*

```
1:  procedure ConstrainedExtend($\mathcal{T}$,$q_{near}$,$q_{rand}$,$f$,$\epsilon$)
2:      d ← Distance($q_{near}$,$q_{rand}$)
3:      q ← $q_{near}$
4:      while $d > \epsilon$ do
5:          $q'_{rand}$ ← OrthogonalProject($q_{rand}$, $T_q\mathcal{M}$)
6:                                      ▷ $T_q\mathcal{M}$ : tangent space to $\mathcal{M}$ at $q$
7:          $q''_{rand}$ ← SolveConstraints($q'_{rand}$, $f$, $\epsilon$)
8:                                      ▷ Refers to algorithm 4.1
9:          d ← Distance($q$, $q''_{rand}$)
10:         q ← $q''_{rand}$
11:
12:     end while
13:     $q_{new}$ ← RRT::Extend($\mathcal{T}$,$q_{near}$,$q''_{rand}$)
14: end procedure
```

## 4.3 Constrained RRT

To generate configurations onto the sub-manifold defined by a set of constraints, Algorithm 4.1 can be inserted in the RRT [8]. Dalibard et al. have developed a variant of RRT that projects random configurations on this manifold and extends towards the projected configuration. The result is Algorithm 4.2. This procedure replaces the extension step in the classic RRT algorithm. This ensures that configurations in the tree lie in the constrained sub-manifold. We will integrate the constraint graph so that configurations are extended onto an admissible sub-manifold.

## 4.4 Grasping task

Interactions between the robot and the object considered in this work are very basic. The goal is a high-level manipulation planning. Though it is not the main focus, grasping has several aspects that directly impact on it. It is thus interesting to mention them.

Objects that have constrained motion can apply force on the robot, that does not come from gravity. Our approach does not take these forces into account. For a more realistic planning, it would be important to take them into account because they change the balance of the robot. It may not be possible to go from a statically balanced path of the sliding robot to a dynamically balanced trajectory of the walking humanoid robot. Not considering these forces means that our approach is only valid for manipulation tasks that require negligible efforts compared to the weight of the robot.

Grasping objects is also a challenging task because the robot has to know how to grasp an object and how much grasping force to apply to have a stable manipulation. For an overview of object grasping, see [17]. This particular topic is a field of research that is not covered here but it is possible to integrate more complex grasping. High-level planning means that grasping is part of a larger task that the robot is expected to do, as mentioned in Chapter 2. So goal-oriented grasp, as presented in [1], could be interesting.

Our goal being a high-level manipulation planning, our end-effector will just be a two states gripper, considered as opened or closed.

In our implementation, the grasping constraint is a position constraint. A point in the frame of the hand must match a point in the frame of the object. To grasp an object, the required information is thus limited to a 3D point in its frame. It is possible to give more information to the planner. For instance, a constraint on the orientation of the hand when grasping can be added. On one hand, it would give a more human-like motion and would help the planner as there would be less collision between the hand and the object. On the other hand, adding a not-required constraint reduces the sub-manifold of search and may thus make unreachable some valid paths.

# Part II

# Contributions

# Chapter 5

# Constraint graph

## 5.1 Overview

This chapter explains the structure that has been developed for manipulation planning. After giving the motivations to build such a struture, Section 5.2 defines the constraint graph and the following sections provide more details about the different types of constraint.

In sampling-based algorithms, sampling and connecting are different steps. Remember the remark at the end of Section 4.1. Both steps may require projections, but the set of constraints is different for each projection. Integrating it directly in the planning algorithm would complicate it and a goal of the constraint graph is to decouple the continuous search and the discrete search lying behind manipulation planning.

Algorithm 3.1, a classic RRT algorithm, is very abstract. It has no notion of robots or objects and is simply populating a tree with vectors. It only has the notion of distances. Collision checking is a black box. The goal of the constraint graph is to isolate the issues raised by the foliated configuration space from the planning algorithm. The latter is kept abstract and as simple as possible. It stays at a higher level than the notion of robots, objects and even foliations. These notions are transformed in sets of constraints that the planner uses to project configurations.

Manipulating an object means navigating in a foliation of the configuration space. As we want the planner to be abstract and not aware of object manipulation, it should not be aware of this navigation. For the planner, the actions *"grasp"*, *"move freely"* or *"move while holding"* will all be translated into *"extend $q_{near}$ towards $q_{rand}$ under the set of constraint $f$"*. The difference between the possible actions is hidden in the set $f$. The planner has no understanding of $f$, it does not change its behaviour when $f$ changes. It stays at a more abstract level than object manipulation.

## 5.2 Constraint graph

The solution to a manipulation problem is a sequence of transit path and transfer path, as explained in Section 4.1. A transit path evolves in a leaf of the foliation in which the object does not move. Transfer paths connect these leaves together. The representation of a manipulation problem we want to introduce is based on this foliated structure. Let us consider an end-effector of a robot and an object $\mathcal{O}$ and define six sets of constraints:

- $\mathcal{C}_h^c$ (resp. $\mathcal{C}_{\bar{h}}^c$) represents the set of constraints that a configuration must respect to be in $\mathcal{CS}_{grasp}$ (resp. $\mathcal{CS}_{plcmt}$).
- $\mathcal{C}_g^m$ (resp. $\mathcal{C}_{\bar{g}}^m$) represents the set of constraints that a motion must respect to go from $\mathcal{CS}_{grasp}$ to $\mathcal{CS}_{plcmt}$ (resp. from $\mathcal{CS}_{plcmt}$ to $\mathcal{CS}_{grasp}$).
- $\mathcal{C}_h^m$ (resp. $\mathcal{C}_{\bar{h}}^m$) represents the set of constraints that a motion must respect to stay in a foliation of $\mathcal{CS}_{grasp}$ (resp. in $\mathcal{CS}_{plcmt}$).

Figure 5.1 represents this graph. The two following sections provide more detail about these sets. Before going further, it is interesting to make a few remarks. When the object $\mathcal{O}$ is always in a stable placement, for instance a door, a configuration where the end effector holds $\mathcal{O}$ is both in $\mathcal{CS}_{grasp}$ and $\mathcal{CS}_{plcmt}$. So, in this case, the two sets of constraints $\mathcal{C}_{\bar{g}}^m$ and $\mathcal{C}_{\bar{h}}^m$ must be identical because going from $\mathcal{CS}_{grasp}$ to $\mathcal{CS}_{plcmt}$ is equivalent to going from $\mathcal{CS}_{plcmt}$ to $\mathcal{CS}_{plcmt}$.

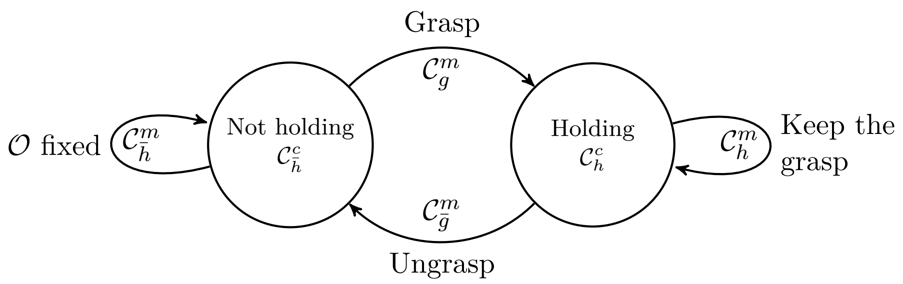

**Figure 5.1.** Graph of constraints for a grasping task.

The graph will help to implement the RRT algorithm to generate both valid and reachable configurations from a given configuration. For the system $\mathcal{R} \times \mathcal{O}$, we first extend a configuration $q_{from}$ towards the projection of a configuration $q_{rand}$ into a leaf of $\mathcal{CS}_{grasp}$ or $\mathcal{CS}_{plcmt}$ (i.e. a leaf defined by a set of constraints). The resulting connection between $q_{from}$ and $q_{extended}$ will be valid regarding the manipulation constraints but may not be collision free. Let's say $q_{from} \in \mathcal{CS}_{plcmt}$ is the closest configuration to a random configuration $q_{rand}$. In both cases, with a graph such as 5.1, there are two options for extending $q_{from}$: toward $\mathcal{CS}_{grasp}$ or toward $\mathcal{CS}_{plcmt}$. For instance, we choose $\mathcal{CS}_{grasp}$. In order to get a configuration reachable from $q_{from}$, $q_{rand}$ will be projected using the union of two sets of constraints:

- $\mathcal{C}_g^m$: the set of possible future paths. This will ensure the existence of a (possibly colliding) path.
- $\mathcal{C}_h^c$: the set of destinations. The path is in $\mathcal{CS}_{plcmt}$ but must end in $\mathcal{CS}_{grasp}$.

To generate the path from $q_{from}$ to $q_{ext}$, the linear interpolation is projected using $\mathcal{C}_g^m$.

## 5.3 Constrained configuration

The two sets of constraints $\mathcal{C}_h^c$ and $\mathcal{C}_{\bar{h}}^c$ (i.e. applied to configurations) contain at least the stability constraints of the robot. The small-space controllability, described in Section 3.2, allows us to consider a sliding robot instead of a humanoid robot. Finding first a path for the arms and the center of mass and extending it to a whole-body trajectory, as done in [8]. Instead of using a bounding box, as in [6], of $\mathcal{R}$, we use a sliding robot. This increases the complexity of the problem to be solved but the bounding box approach would not be able to reach an object on a table. Moreover, the bounding box approach is a very strong assumption regarding collision detection and it invalidates many valid paths.

To summarize, $\mathcal{C}_{\bar{h}}^c$ consists of:

- the stability constraints of the sliding humanoid robot.
- the stability constraints of the objects. For a door, this is empty.

And $\mathcal{C}_h^c$ consists of:

- the stability constraints of the sliding humanoid robot.
- the grasping constraints.

As stated in Section 4.4, planning a grasp was not our focus. However, to plan a high level object manipulation path, planning a low-level grasp is required. The end-effector is a simple gripper which can only be open or closed. The grasping constraint is simply a reaching constraint. It is satisfied if the center of the hand is at a given position in the frame of the object.

Using a simple grasp planning simplifies the implementation but many other grasp planning would be acceptable regarding the graph 5.1, in which the notion of grasping is abstract.

## 5.4 Constrained motion

In the framework of this thesis, the four sets of constraints $\mathcal{C}_g^m$, $\mathcal{C}_{\bar{g}}^m$, $\mathcal{C}_h^m$ and $\mathcal{C}_{\bar{h}}^m$ (i.e. applied to motion) are composed of the two following types of constraint:

- configuration constraints that every configuration of the motion must respect. When $\mathcal{O}$ is a door, these constraints, for $\mathcal{C}_g^m$, $\mathcal{C}_{\bar{h}}^m$ and $\mathcal{C}_{\bar{g}}^m$, are $\mathcal{C}_{\bar{h}}^c$ and only $\mathcal{C}_h^c$ for $\mathcal{C}_{\bar{h}}^m$.

- a zero velocity constraint on $\mathcal{O}$, to stay in the same foliation during all the path.

More complex velocity constraints may be needed in a different framework but it is not our focus here.

It is important to realize that these sets of constraints are sufficient to interpolate valid configurations between the two ends of a constrained path. Indeed, they are sufficient to project the path in the corresponding leaf, when the two ends are in the same leaf. Yet the extension step builds a path from $q_{near}$, by first projecting $q_{rand}$ into a reachable leaf. One end of the desired path is not known. We must ensure that the projected configuration is in a reachable leaf but is also in the targeted state (the destination of the corresponding edge). Thus these sets of constraints are not sufficient to build a path, the targeted state set is also required. An example of this is given in Section 4.1.

Chapter 6

# Constrained RRT using a constraint graph

## 6.1 Constrained manifold selector

To integrate our graph of constraints to the algorithm, the latter needs to know how to select the set of constraints to be used by Algorithm 4.2. Analysing the configuration $q_{near}$ gives the list of constraints sets that can be selected to generate a valid direct path.

**Ordering the possible states** is the first step. Taking the basic example of one end effector and an object with only one possible grasp, the two possible states are *holding* and *not holding*. If the object is always in a valid placement, for instance a door, then $\mathcal{CS}_{grasp}$ is a submanifold of $\mathcal{CS}_{plcmt}$. The state *holding* can then always be considered as *not holding*. Regarding the constraint graph, it means that one state become useless and the planning is not done properly. *holding* must have a higher priority than *not holding*, that is to say that if the conditions for *holding* are verified, then it is *holding* and cannot be *not holding*.

Let's now think about $n_e \geq 1$ end-effectors and the same object. There is one constraint graph for each end-effector and states are independents. If we extend the number of possible grasps to $n_g \geq 1$, on one or more objects, then there are $n_e$ constraint graphs with $n_g + 1$ nodes (one for each grasp plus one when the end-effector is free). The graph 6.1 represents the case $n_g = 4$. Each edge is a set of constraints.

**Choose the leaf** in which the configuration will be extended. This is done by selecting, for each end-effector, an edge going from the considered end effector state. This step corresponds to the discrete part of the search, selecting a new set of constraints. To our best knowledge, there are no good ways of knowing which are the foliations likely to be useful. The choice is done randomly. The main advantage of randomness is its generality. Very likely, there are special cases admitting far better strategies, but they would not work in the general cases. The main inconvenience of randomness is the necessity to set transition probabilities. Their influence will be

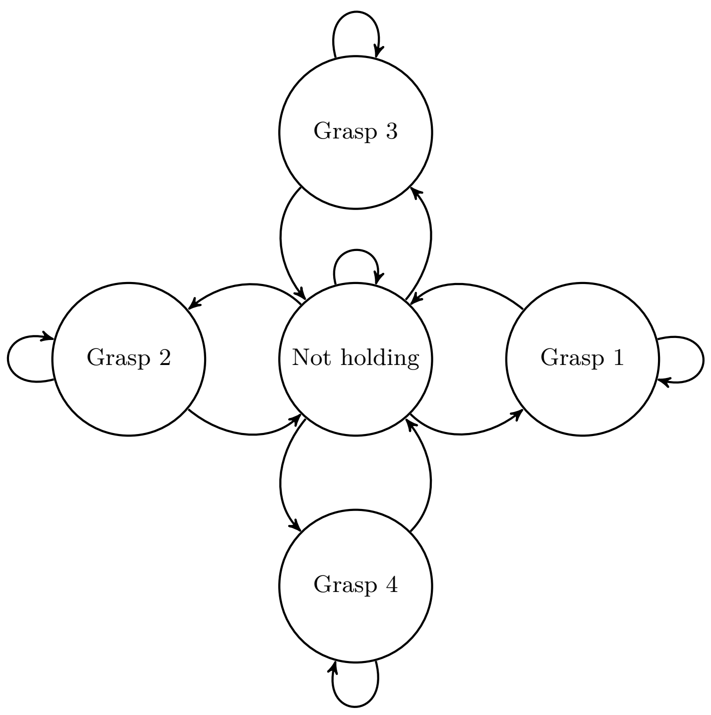

**Figure 6.1.** Graph of constraints for one end-effector and 4 possible grasps.

discussed in details in Section 7.3. It is possible to tune this step in different ways:

– disable impossible simultaneous grasp. It may be impossible for geometrical reasons to grasp both a door and a drawer, because they are too far from each other for instance.

– set a higher probability of staying in the same state, since we prefer to manipulate an object without releasing it.

**Build the sets of constraints** needed by the planner. There are two sets to build: one for the extension, one for the built path. An extension generates an end configuration and a path, so the constraint set for the extension is the union of:

– the set that applies to the targeted state (destination of the randomly chosen edge);
– the set that applies to the path (the randomly chosen edge).

Once the extension is done, the set for extension is forgotten and the set for path is inserted in the generated path. In other words, a generated path remembers the edge of the constraint graph it corresponds to so that it knows how to interpolate configuration between its two ends.

**Algorithm 6.1.** Select a set of constraints

***Description:*** *Select an admissible sets of constraint to build a path from $q$.*

```
1: procedure selectConstraintSet($q$)
2:     $extendSet \leftarrow$ EmptySet()
3:     $pathSet \leftarrow$ EmptySet()
4:     for all end-effector as $e$ do
5:         $state \leftarrow$ State($e, q$)
6:         $edge \leftarrow$ SelectOneOutgoingEdge($state$)
7:                 ▷ GetDestinationConstraints return the constraints of the targeted state.
8:         $extendSet$.add($edge$.GetDestinationConstraints())
9:         $extendSet$.add($edge$.GetConstraints())
10:        $pathSet$.add($edge$.GetConstraints())
11:    end for
12:    return $motionSet$,$stateSet$
13: end procedure
```

Algorithm 6.1 integrates these steps and returns two sets of constraints. From these sets, a projector for the extension step is built, typically the function $f$ in Algorithm 4.2, and another to project a path in the leaf it belongs to.

## 6.2 RRT using a constraint graph

The navigation between leaves of the foliated configuration space is done by the constraint graph. The latter is inserted in the RRT-based algorithm, as shown in Algorithm 6.2. After a classic sampling, the closest neighbor $q_{near}$ of $q_{rand}$ is found. From $q_{near}$ are deduced the sets of constraints for extension, and for the generated path. Finally, a constrained extension is performed from $q_{near}$ to the projection of $q_{rand}$.

It is important to remark that, if the tree is grown only from an initial configuration, the algorithm has a zero probability of reaching any goal configuration that lies in a leaf different from the initial configuration.

## 6.3 Distance evaluator

An RRT algorithm needs to compute distances between configurations. Exact distance can be computationally expensive so a rough and cheap estimation is used instead, as distances are very frequently computed. The distance should reflect the fact that the configuration space is foliated. Distance between two configurations lying in different leaves of $\mathcal{CS}_{plcmt}$ (i.e. the object has moved without being grasped)

**Algorithm 6.2.** Constrained RRT

***Description:*** *Randomly exploring Random Tree from $q_0$ using the constraint graph*

```
1: procedure EXPLORETREE($q_0$)
2:     $\mathcal{T}$.init($q_0$)
3:     for $i = 1 \to K$ do
4:         $q_{rand} \leftarrow$ Rand($\mathcal{CS}$)
5:         $q_{near} \leftarrow$ Nearest($q_{rand}$,$\mathcal{T}$)
6:         $f_{extend}, f_{path} \leftarrow$ selectConstraintSet($q$)
7:         constrainedExtend($\mathcal{T}$, $q_{near}$, $q_{rand}$, $f_{extend}$, $f_{path}$)
8:     end for
9: end procedure
```

must reflect the fact that they do not *see* each other directly. We have decided to implement the same as in [6].

This distance evaluator is very basic. It just adds a high penalty for configurations that are not in the same leaf of a foliation. It would be possible to find an estimate of the distance with projections. In Figure 4.1 for instance, to estimate the distance between $q_1$ and $q_3$, a configuration like $q_2$ is needed since there is no other way $q_1$ and $q_3$ can actually *see* each other. With our distance evaluator, $Distance(q_1, q_3)$ and $Distance(q_1, q_4)$ include the high penalty, but $q_3$ appears further than $q_4$ from $q_1$, which is not true. The distance evaluator is a weakness and would be a useful extension of this thesis.

# Chapter 7

# Experimental results

This chapter presents the results of our experimental evaluation. After a description of the practical problem that we solve, we introduce the estimators used to analyse the results. The last section presents our results.

## 7.1 Problem and implementation

Experimentations were done on HRP-2 and PR2 and we have reused the simulated environment from a previous work [5], shown in Figure 1.2. The environment is an apartment with basic objects. We have focused on the manipulation of a door and a drawer. The goal of our experiments was to validate our approach by showing that it is possible to find a statically balanced manipulation path in a 3D environment, using constraints to guide the planner. We provide to the solver an initial and a final configuration for the system $\mathcal{R} \times \mathcal{O}$, the object having moved between the two. And the planner has to find a path respecting the manipulation rules. Each object has only one possible grasp and objects are considered separately, so we consider two similar problems: door opening and drawer opening.

The experimental evaluation has two goals. First, we wish to validate our approach to solve the manipulation problem. Second, we wish to understand better the influence of the transition probabilities on the planner.

Given this framework and the method we intended to implement, the following problems have been faced:

– Section 4.2 presents the difficulties to find a path in a narrow environment. Grasping the door is a narrow path because HRP-2 hands are of a comparable size with the handles of the door. In our experiments, HRP-2 hand stays at 5 cm before the handle (at a fixed position with respect to the door).

– there is no obvious way of choosing the transition probabilities in the graph of constraints. Experiments with different values have been conducted in order to evaluate their influence.

– A good collision checking algorithm is essential for fast planning. During an initialization phase, a list of all the possible pairs of bodies is generated. To check for collision, we can simply loop over this list to check if one pair is colliding. To speed it up, the pairs of bodies of the robot that will never collide can be removed from the list. For instance, an arm and the corresponding forearm will never collide due to joint bounds. It has been reduced to 407 for PR2 and 317 for HRP-2[1].

## 7.2 Estimators

To compare two runs of the same experiment, several estimators are used. The most obvious ones are the global time to solve the problem and the number of timeouts. Yet, they do not tell us how well the configuration space has been explored. Let $N_f$ be the number of explored foliations and $n_i, 1 \leq i \leq N_f$ the number of node in the $i$-th explored foliation. The total number of nodes is then $N_T = \sum_{1 \leq i \leq N_f} n_i$. Let the *"distribution"* of a roadmap be the sequence $(n_i/N_T)_{1 \leq i \leq N_f}$. It is important to remark that the distribution is scalable regarding $N_T$ so that it is possible to compare roadmaps of different size. A high variance of the distribution of a roadmap means that some foliations have been more explored than others so there were preferred foliations. A low variance means each foliation has about the same number of nodes so there were no preferred foliations. Thus an estimator of the distribution is

$$e_{distri} = Var(\frac{n_i}{N_T}) \tag{7.1}$$

The lower $e_{distri}$, the better.

Nevertheless, a very low variance does not mean that the nodes are well spread over the foliations. Indeed, if only two foliations are equally explored, the variance is zero but the exploration may not have been very efficient. It is hard to know how many foliations should be explored but it is interesting to compare this number for different runs. To have a scalable estimator, the chosen estimator is

$$e_{foliation} = \frac{N_f}{N_T} \tag{7.2}$$

Obviously, we have $0 < e_{foliation} \leq 1$. There is no ideal value for this estimator, but extrema are to be avoided preferably. A value close to zero means many foliations have been explored, but very poorly. A value close to one means a few foliations have been explored a lot.

[1] The list of non-colliding pairs has been automatically generated using the ROS package *MoveIt!* available at `http://moveit.ros.org/`.

## 7.3 Results

### 7.3.1 Overview

Figures 7.1 and 7.2 show a sequence of configurations found by the planner, for respectively the door opening problem and the drawer opening problem. The planner has been able to find a path using the constraint graph. Our first goal is thus achieved.

No path optimizer was used, so we can observe a side effect of random sampling. The motion is not looking like human motion. As the left hand is not considered as an end effector, the only constraint that can modify the left arm DOF is the constraint on the center of mass. As HRP-2 left arm does not weight much compared to the whole robot, its configuration is almost random.

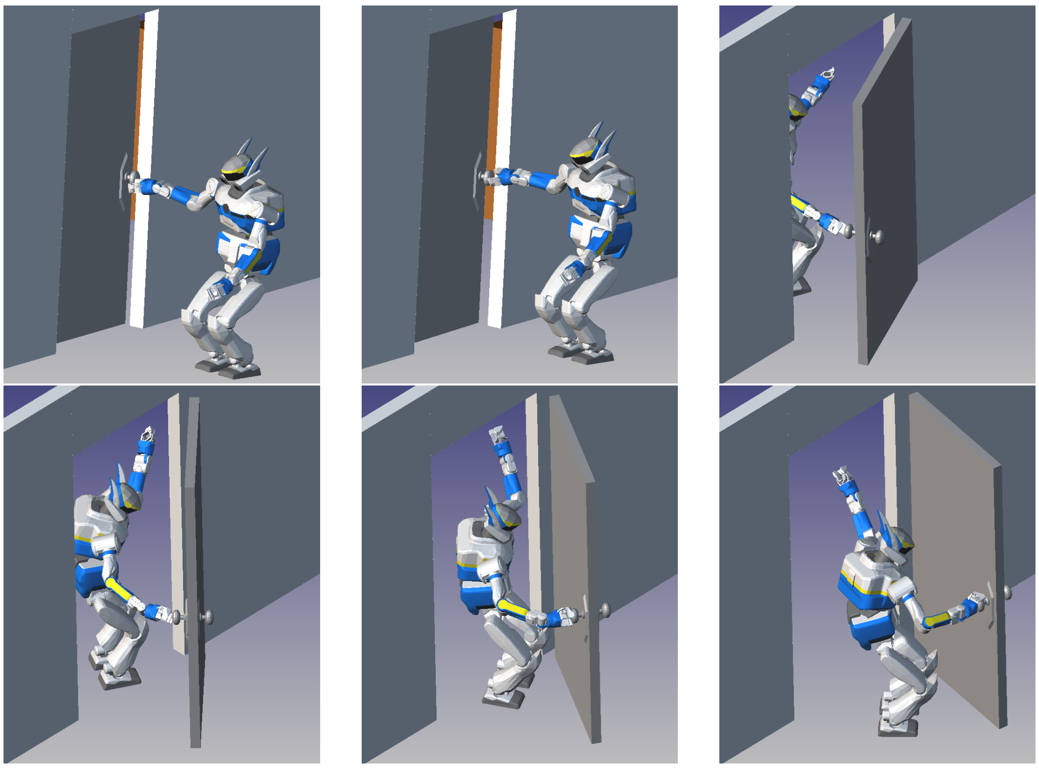

**Figure 7.1.** HRP-2 - Sequence of configurations found by the planner to open a door.

Collision checking is a bit slower for PR2 because our PR2 model has more collision pairs to check than our HRP-2 model. As most of the time is spent checking whether a configuration or a path is in collision, we can expect the planner to be slower with PR2. This can be observed by comparing Figure 7.5 and Figure 7.8.

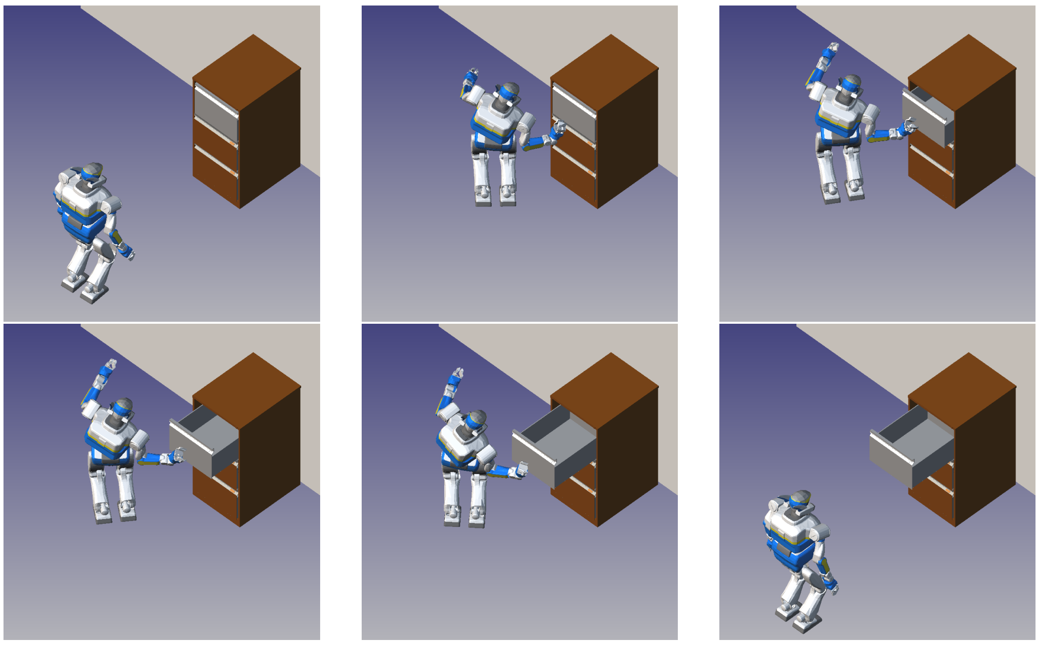

**Figure 7.2.** HRP-2 - Sequence of configurations found by the planner to open a drawer.

### 7.3.2 Analysis

Planning has been done using different values for the transition probabilities $P_{NH \to H}$ and $P_{H \to NH}$, where $NH$ stands for *"Not Holding"* and $H$ for *"Holding"*. For each set of parameters, the same problem has been run twenty times, with a different random seed for each run. A problem is considered unsolved if no solution is found in less than an hour, that is to say a problem is considered unsolved when the planner times out.

Figures 7.3 and 7.4 (resp. 7.6 and 7.7) show the influence of the transition probabilities for HRP-2 (resp. PR2). The following points can be drawn from the graphs:

– Very roughly, we can observe that curves are monotonous and not crossing, so it seems that $P_{H \to NH}$ has more influence than $P_{NH \to H}$.

– A high value for $P_{H \to NH}$ tends to have a lower ratio of unsolved problem and less nodes are required to solve the problem, which is better.

– The variance is decreasing when $P_{H \to NH}$ decreases and $P_{NH \to H}$ increases.

– The ratio $\frac{N_f}{N_T}$ evolves in a inverse manner, it decreases when $P_{H \to NH}$ increases and $P_{NH \to H}$ decreases. The former relationship can be understood as follows: a new leaf (the object does move) is explored when you stay in the state

*"Holding"*, which happens more often when $P_{NH\rightarrow H}$ is high and $P_{H\rightarrow NH}$ is low. Indeed, the higher $P_{NH\rightarrow H}$ and the lower $P_{H\rightarrow NH}$, the more likely configurations are projected in state *"Holding"*.

– Performances for PR2 and HRP-2 are alike. The time to solve is generally longer for PR2 because of collision detection. The door problem is harder because the door leaves make a narrow passage. The passage does not seem to penalize more one robot than the other, probably because they have similar sizes.

Figures 7.5 and 7.8 show the average time to solve a problem. Though it is not really clear, high values of $P_{H\rightarrow NH}$ tends to give a solution faster. It seems hard to draw any conclusion regarding the influence of $P_{NH\rightarrow H}$ on the average time to solve.

To summarize, it seems better to choose a high value for both $P_{H\rightarrow NH}$ and $P_{NH\rightarrow H}$ because less timeouts were observed and exploration of the configuration space is better distributed for these settings.

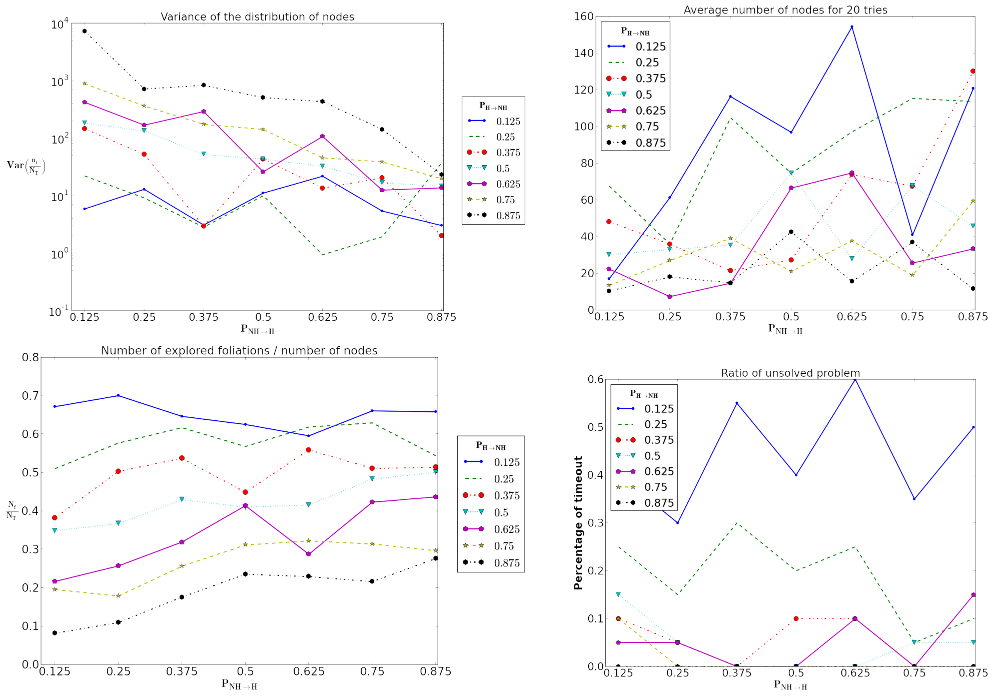

**Figure 7.3.** HRP-2 - Influence of the transition propabilities on the door problem.

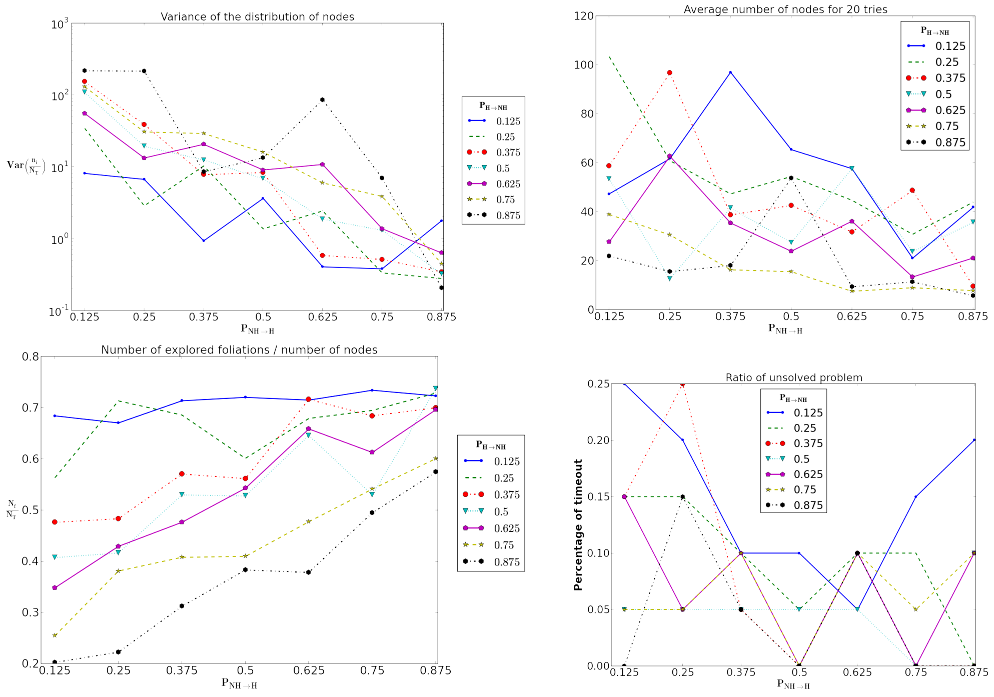

**Figure 7.4.** HRP-2 - Influence of the transition propabilities on the drawer problem.

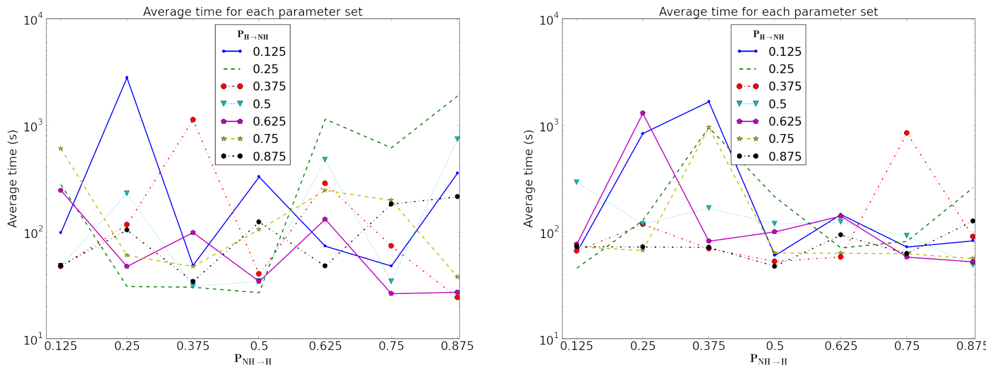

**Figure 7.5.** HRP-2 - Average time to solve for the door (left) and the drawer (right) problems.

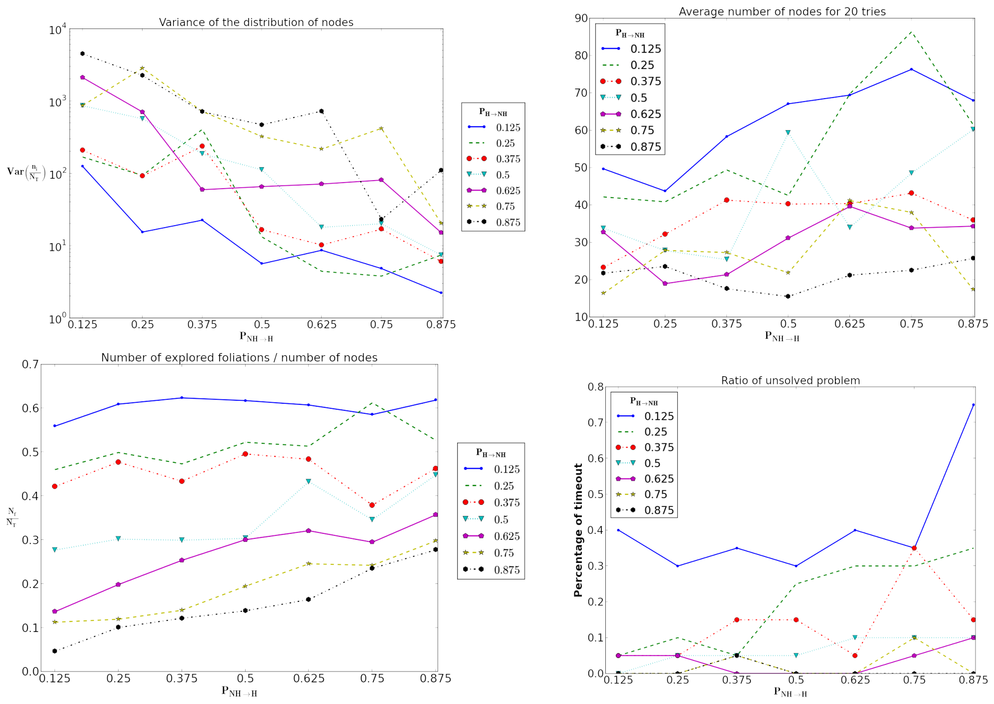

**Figure 7.6.** PR2 - Influence of the transition propabilities on the door problem.

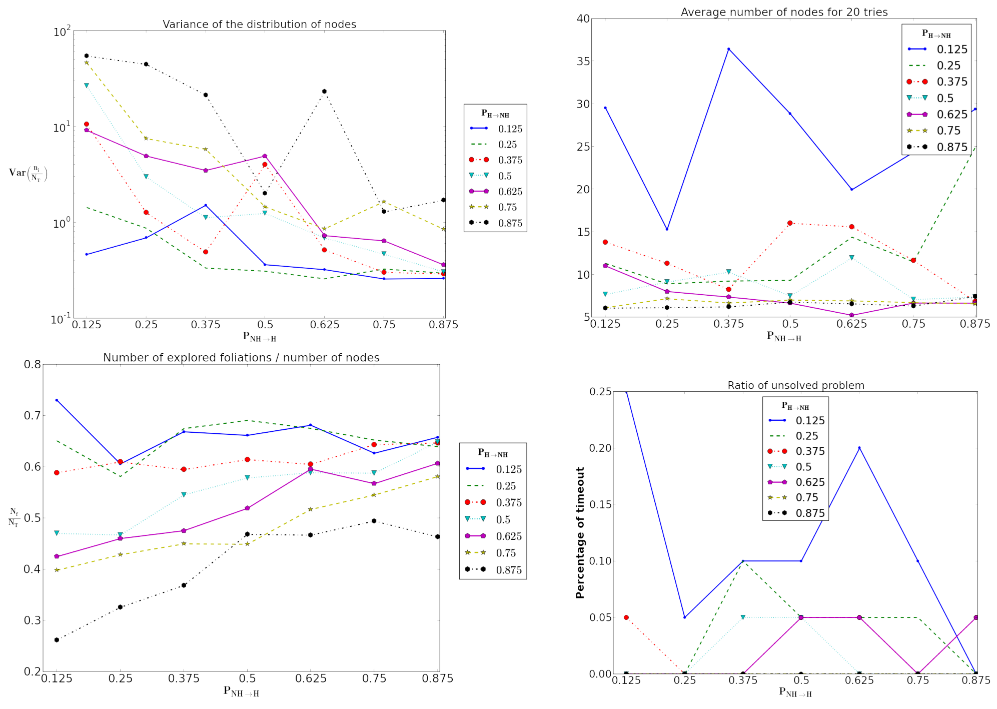

**Figure 7.7.** PR2 - Influence of the transition propabilities on the drawer problem.

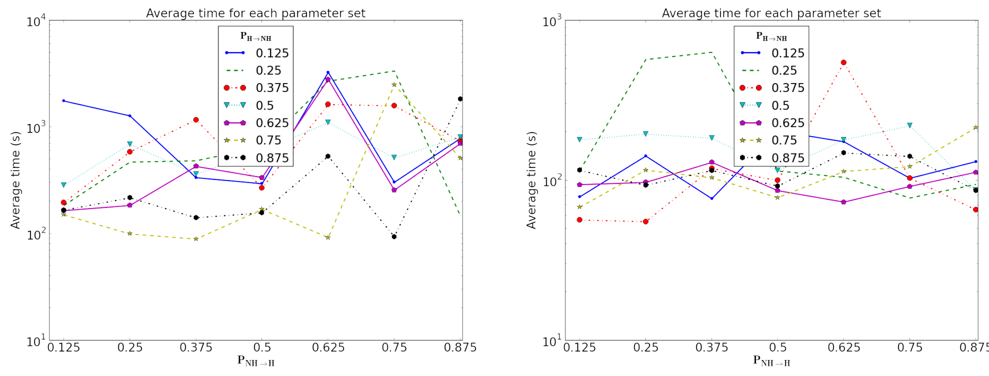

**Figure 7.8.** PR2 - Average time to solve for the door (left) and the drawer (right) problems.

# Chapter 8

# Conclusion

## 8.1 Achievements

This thesis tackles the problem of high level manipulation planning. Manipulation planning is seen as finding a path between two configurations in a foliated configuration space. A graph of numerical constraints represents interactions between robot end-effectors and objects. To find a path respecting manipulation rules, the presented planner uses the constraint graph. The planner and the constraint graph are kept general so that it is theoretically possible to solve any problem that can be specified with constraints. In practice, planning may become extremely long.

The main issue faced along this thesis, as in any manipulation planning work, is the introduction of passive degrees of freedom. They have had two undesirable effects: they introduce discrete planning and they foliate the configuration space.

**Discrete planning** has been hidden from the path planner by the constraint graph. Indeed, the general idea of RRT algorithm remains the same: sample a configuration and extend towards it. But the discrete search is hidden behind discrete choices done during the extension step. This raises two problems. First, it means that we are solving a discrete problem with a random sampling method. It is not well adapted to discrete search. Second, the discrete search, that leads the continuous search, is ruled by transition probabilities that are hard to set.

**The foliation** of the configuration is explored using constraints and their corresponding projectors. The latter are discretely chosen and direct the search towards a leaf. This helps the sampling of configurations that are reachable from another configuration. The problem with the foliated structure is that there is no easy way to know how to lead the exploration. When to start exploring a new leaf? When a leaf can be considered as explored?

In a basic case, we have successfully shown that planning a path using the constraint graph is possible. Nevertheless, no experimentations on complex case were done and a deeper analysis is required to fully validate the developed concepts. As many tasks can be specified using constraints, constrained planning could achieve complex tasks. The constraint graph provides a convenient way of giving the required information to the planner.

Path planning using constraints has thus a promising future in terms of task specification.

## 8.2 Future work

This work has mainly been practical and a theoretical analysis would provide a better understanding of what is happening. The graph of constraints adds discrete planning to an RRT, which is a continuous planner. As we have seen earlier, the transition probabilities have an influence on the performance of the planner. A theoretical analysis, similar to [13], could help to choose good values or even to dynamically adapt their values during the search. Adapting this probabilities dynamically could be a good improvement of the discrete search because it would mean forcing the continuous search towards a direction that is thought good.

Another idea that could be worth exploring is to decouple the planning. To picture this idea, one can think about two leaves of a foliation: leaf A and leaf B. In a leaf, the object is at a constant position but this position is different in leaf A and leaf B. Let's say that leaf A has been explored but not leaf B. If the object has moved only a little between leaf A and leaf B, then from most of the configuration in leaf A, a collision-free configuration in leaf B could be generated by simply changing the object DOF. This could be seen as transporting the exploration of a leaf into a neighboring leaf. This may be efficient when building a visibility roadmap for example, as most of the guards would be transported from one leaf to another. Nevertheless, a way of connecting the leaf is needed and it is the hardest part of manipulation planning.

The problem tackled is high level manipulation planning. To make this kind of planning useful, low level planning is required. Very basically, it is possible to give all the information to the robot, i.e. hand position, grasping procedure, etc. Another possibility is to provide the planner the ability to plan the grasp. Part of this planning could be done with the graph of constraints.

Experiment has been conducted only with one object, and with only equality constraints. As explained earlier, this method is theoretically valid for more than one object and one end-effector. An obvious extension of this work is to implement and test the graph of constraints to support more objects. With this approach, one could think of generating walking motions on a flat ground with a constrained path planner. Indeed, the free-flyer DOF are passive DOF and their interaction can be modeled with constraints. For such a planning, there would be two end-effectors, the feet, and one object, the ground. Constraints on configuration between a foot and the ground are: foot is parallel to the ground, constant height of the foot. It would be possible to find a statically balanced path, but it can always be followed as slowly as necessary to be transformed in a dynamically balanced trajectory.

Appendix A

# Notations

$\mathcal{R}$: the humanoid robot

$\mathcal{O}$: an object

$\mathcal{CS}$: configuration space

$\mathcal{CS}_{grasp}$: sub-manifold of grasping configuration of $\mathcal{CS}$

$\mathcal{CS}_{plcmt}$: sub-manifold of valid placement of $\mathcal{CS}$

$\mathcal{CS}_{free}$: collision-free configuration space

*For the next notations, see Figure 5.1, page 25. "hold" refers to a state of the robot, i.e. a configuration, "grasp" to an action, i.e. a motion.*

$\mathcal{C}^m_g$, $\mathcal{C}^m_{\bar{g}}$, $\mathcal{C}^m_h$, $\mathcal{C}^m_{\bar{h}}$, $\mathcal{C}^c_h$, $\mathcal{C}^c_{\bar{h}}$: Constraints sets. $c$, $m$, $h$, $g$ stands respectively for Configuration, Motion, Holding and Grasping.

Appendix B

# Cart-Table Model

The cart-table model consists in a running cart of mass $m$ on a mass-less table. It has been introduced by [11]. The cart represents the robot center of mass. The table foot represents the supporting foot. This model supposes that the center of mass is at constant height.

The distributed floor reaction force can be replaced by a single force acting on the Zero-momentum point (ZMP). Using the notations defined in Figure B.1, the two momentums that applies to the table at ZMP are:

– $M \times g \times (x_M - x_{ZMP})$, the momentum induced by the gravitational forces.

– $M \times \ddot{x} \times z_M$, the momentum induced by the inertia forces.

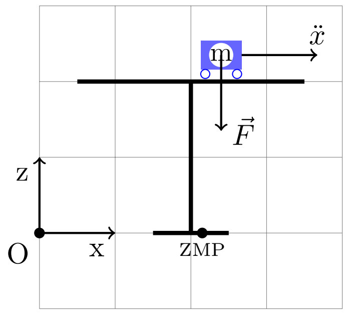

**Figure B.1.** Cart table model.

Thus, the ZMP coordinates $(x_{ZMP}, 0)$ satisfy the zero-momentum equation $M \times g \times (x_M - x_{ZMP}) - M \times \ddot{x} \times z_M = 0$. Finally, we have:

$$x_{ZMP} = x_M - \frac{z_M}{g}\ddot{x} \tag{B.1}$$

Using the ZMP, it is very easy to have a criteria for the table balance. If the ZMP is outside the table foot, the table falls. This criteria is an inequality constraint on $x_{ZMP}$. This model is used for generation of walking trajectories. The basic result obtained above can be expressed as follows, for a humanoid robot: a walking motion is balanced as long as the ZMP stays inside the support polygon.

# Bibliography

[1] Y. Bekiroglu, D. Song, L. Wang, and D. Kragic. A probabilistic framework for task-oriented grasp stability assessment. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 3040–3047, May 2013. doi: 10.1109/ICRA.2013.6630999.

[2] D. Berenson, S. Srinivasa, and J. Kuffner. Task Space Regions: A framework for pose-constrained manipulation planning, 2011. ISSN 0278-3649.

[3] J. Cortés and T. Siméon. Sampling-Based Motion Planning under Kinematic Loop-Closure Constraints. In Michael Erdmann, Mark Overmars, David Hsu, and Frank der Stappen, editors, *Algorithmic Foundations of Robotics VI*, volume 17 of *Springer Tracts in Advanced Robotics*, pages 75–90. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-25728-8. doi: 10.1007/10991541_7. URL `http://dx.doi.org/10.1007/10991541\_7`.

[4] S. Dalibard and J.-P. Laumond. Linear dimensionality reduction in random motion planning. *International Journal of Robotics Research*, 30(12):1461–1476, 2011. ISSN 0278-3649. doi: 10.1177/0278364911403335.

[5] S. Dalibard, A. Nakhaei, F. Lamiraux, and J.-P. Laumond. Whole-body task planning for a humanoid robot: a way to integrate collision avoidance. In *IEEE International Conference on Humanoid Robots*, pages 355–360, 2009. ISBN 978-1-4244-4597-4. doi: 10.1109/ICHR.2009.5379547.

[6] S. Dalibard, A. Nakhaei, F. Lamiraux, and J.-P. Laumond. Manipulation of documented objects by a walking humanoid robot. *2010 10th IEEE-RAS International Conference on Humanoid Robots*, pages 518–523, December 2010. doi: 10.1109/ICHR.2010.5686827. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5686827`.

[7] S. Dalibard, A. El Khoury, F. Lamiraux, M. Taïx, and J.-P. Laumond. Small-space controllability of a walking humanoid robot. *2011 11th IEEE-RAS International Conference on Humanoid Robots*, pages 739–744, 2011. ISSN 2164-0572. doi: 10.1109/Humanoids.2011.6100807. URL `http://hal.archives-ouvertes.fr/docs/00/60/23/84/PDF/dalibard-humanoids11.pdf`.

[8] S. Dalibard, A. El Khoury, F. Lamiraux, A. Nakhaei, M. Taïx, and J.-P. Laumond. Dynamic Walking and Whole-Body Motion Planning for Humanoid Robots: an Integrated Approach. *The International Journal of Robotics Research*, 32(9-10):1–30, 2013. URL `http://hal.archives-ouvertes.fr/hal-00654175`.

[9] A. El Khoury, F. Lamiraux, and M. Taïx. Optimal motion planning for humanoid robots. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, volume 0, pages 3136–3141, May 2013. doi: 10.1109/ICRA.2013.6631013. URL `http://hal.archives-ouvertes.fr/hal-00715419`.

[10] K. Hauser, T. Bretl, J.-C. Latombe, K. Harada, and B. Wilcox. Motion Planning for Legged Robots on Varied Terrain, 2008. ISSN 0278-3649.

[11] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi, and H. Hirukawa. Biped walking pattern generation by using preview control of zero-moment point. *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, 2, 2003. ISSN 1050-4729. doi: 10.1109/ROBOT.2003.1241826.

[12] L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580, August 1996. ISSN 1042-296X. doi: 10.1109/70.508439.

[13] L.E. Kavraki, M.N. Kolountzakis, and J.-C. Latombe. Analysis of probabilistic roadmaps for path planning. *IEEE Transactions on Robotics and Automation*, 14, 1998. ISSN 1042-296X. doi: 10.1109/70.660866.

[14] Jr. Kuffner, J.J. and S.M. LaValle. RRT-connect: An efficient approach to single-query path planning. *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, 2, 2000. ISSN 1050-4729. doi: 10.1109/ROBOT.2000.844730.

[15] S.M. LaValle. Planning Algorithms. *Methods*, 2006:842, 2006. doi: 10.1017/CBO9780511546877. URL `http://ebooks.cambridge.org/ref/id/CBO9780511546877`.

[16] J.M. Porta, L. Jaillet, and O. Bohigas. Randomized path planning on manifolds based on higher-dimensional continuation, 2012. ISSN 0278-3649.

[17] A. Sahbani, S. El-Khoury, and P. Bidaud. An overview of 3D object grasp synthesis algorithms. *Robotics and Autonomous Systems*, 60:326–336, 2012. ISSN 09218890. doi: 10.1016/j.robot.2011.07.016.

[18] T. Siméon, J.-P. Laumond, and C. Nissoux. Visibility-based probabilistic roadmaps for motion planning, 2000. ISSN 0169-1864.

[19] T. Siméon, J. Cortes, A. Sahbani, and J.-P. Laumond. A manipulation planner for pick and place operations under continuous grasps and placements. In *icra02*, volume 2, 2002. ISBN 0-7803-7272-7. doi: 10.1109/ROBOT.2002.1014838.

[20] T. Siméon, J.-P. Laumond, J. Cortes, and A. Sahbani. Manipulation planning with probabilistic roadmaps. *International Journal of Robotics Research*, 23(7-8), July 2004. URL `http://ijr.sagepub.com/content/23/7-8/729.short`.

[21] E. Yoshida, O. Kanoun, C. Esteves, and J.-P. Laumond. Task-driven Support Polygon Reshaping for Humanoids. *2006 6th IEEE-RAS International Conference on Humanoid Robots*, 2006. doi: 10.1109/ICHR.2006.321386.