Footstep planning for humanoid robots: discrete and continuous approaches

Nicolas Perrin

Footstep Planning for Humanoid Robots: Discrete and Continuous Approaches

PhD Dissertation

Nicolas Perrin

October 24, 2011

Contents

Remerciements

Je tiens tout d'abord à remercier mes directeurs de thèse, Florent Lamiraux et Olivier Stasse. Ils ont été une source constante d'idées et de motivation et m'ont guidé dans mon travail tout en me laissant une grande liberté dans mes recherches. C'est un réel privilège d'avoir été leur élève et d'avoir pu profiter de leurs hautes compétences scientifiques et techniques ainsi que de leur grande connaissance de la robotique. Je suis très reconnaissant de ce qu'ils m'ont appris.

C'est un grand honneur pour moi d'avoir dans mon jury Satoshi Kagami, John Reif, Christine Chevallereau, Jean-Paul Laumond et Hubert Comon-Lundh, et je les remercie sincèrement. Je remercie tout particulièrement Satoshi Kagami, John Reif et Christine Chevallereau pour leur relecture attentive de mon manuscrit, et pour leurs remarques et commentaires pertinents.

Je remercie Abderrahmane Kheddar et Eiichi Yoshida de m'avoir permis de travailler dans des conditions exceptionnelles lors de mon long séjour au Joint Robotics Laboratory à Tsukuba.

Je remercie également Young J. Kim et Dinesh Manocha pour leurs conseils précieux et pour leur encadrement lors de mes quelques mois passés à l'Université de Caroline du Nord à Chapel Hill.

Je remercie Olivier Sigaud pour son accueil chaleureux et sa disponibilité lors de mon très bref passage à l'ISIR, et je remercie Vincent Padois de m'avoir invité à donner un exposé à CLAWAR 2011.

Je remercie Rogelio Lozano, Jorge Torres et Juan-Antonio Escareño de m'avoir donné l'opportunité de venir présenter mes travaux au LAFMIA à Mexico.

Pour leur travail, leur code, les discussions scientifiques ou simplement les moments de détente, je tiens à remercier tous les chercheurs et étudiants que j'ai cotoyés durant mes trois années de thèse. En particulier, je remercie Martin Battaglia, Léo Baudouin, Mehdi Benallègue, Karim Bouyarmane, Antoine Bussy, Sébastien Dalibard, Duong Dang, Anthony David, Claire Dune, Adrien Escande, Paul Evrard, Toréa Foissotte, Pierre Gergondet, Sovannara Hak, Andrei Herdt, François Keith, Sujeong Kim, Antonio El Khoury, Nosan Kwak, Jungeun Lee, Sébastien Lengagne, Nicolas Mansard, Mitsuharu Morisawa, Thomas Moulard, Jia Pan, Layale Saab, Wael Suleiman et Pierre-Brice Wieber.

Je remercie également mes amis et ma famille, et j'exprime enfin toute ma gratitude à mes parents et à mon frère pour leur soutien constant.

Chapter 1

Introduction

"Step with care and great tact
and remember that Life's
a Great Balancing Act.
Just never forget to be dexterous and deft.
And *never* mix up your right foot with your left."
— Dr. Seuss (1904 - 1991), *Oh, the Places You'll Go!*

During the last 40 years humanoid robots, and more generally legged robots, have been built and studied for various purposes. Among other things, they are expected to move about comfortably in environments designed for humans, and in fact, the intrinsic ability of humanoid robots to step over (or step on) obstacles on the ground should give them a significant advantage over their wheeled counterparts in a wide range of environments. However, legged locomotion involves trajectories that are much harder to generate and control than the ones for wheeled locomotion, and the problem of generating efficient and reliable walking motions has been challenging researchers since the beginning of humanoid robotics.

Footstep planning is the motion planning problem associated with walking motion generation: it is the process of computing walking motions that bring the robot from its initial state to a goal location while avoiding obstacles. In this thesis, we will always assume that the floor is flat and horizontal, with no height variation (no stairs), and the robot will not be allowed to step *on* the obstacles. These assumptions are restrictive, but considering the complexity of the problem, they are a reasonable starting point. Besides, they hold for many indoor environments. We will also focus on the motions of the legs more than on the rest of the robot, and the reason is that the most critical collisions are the ones involving the lower part of the robot: for example it is more likely that the robot will fall if it walks on a ball rather than if it knocks its shoulder against a shelf. Hence, we will pay a particular attention to the legs.

Since humanoid robots combine high dimensionality with underactuation, two

properties that tend to drastically increase the complexity of motion planning, footstep planning is not an easy task. It has been the object of extensive research, and although there is no completely satisfying solution so far, a lot of promising techniques and tools have been introduced over the past decade. The biggest challenge is probably the introduction of good simplifying hypotheses so that to make the problem computationally affordable (reactive walk is a major need for future humanoid robots), while still exploiting well the advantages of legged locomotion, and producing efficient and fast walking. A good tradeoff is very hard to obtain, and this thesis will explore various strategies in that direction. For example, it is easier to plan and control statically stable walk motions (motions that involve only balanced postures; see [Kuffner et al., 2001]), but these motions are usually slow, not energy-efficient, and better obstacle clearance can be obtained with more dynamic motions. Another common method for footstep planning is called the "bounding box method" (see [Yoshida et al., 2008]): it first plans the continuous motion of a big box that contains the whole robot, and then a sequence of steps that follows the box trajectory. If the bounding box trajectory is collision-free, then the robot trajectory is *a fortiori* collision-free, but the drawback of this method is obvious: since the bounding box must circumvent all the obstacles on the floor – even the smallest ones that could easily be stepped over –, it becomes quite unnecessary for the robot to have legs, and thus this method is better suited for wheeled robots such as PR2 or Robonaut. These were two examples of tradeoffs that are too restrictive. Probably the most successful approaches for footstep planning are based on the use of the A* algorithm (or variants such as D*, see [Stentz, 1994]) with a finite transition model, i.e. a relatively small set of possible steps decided in advance (see for example [Kuffner et al., 2001], [Bourgeot et al., 2002], [Chestnutt et al., 2003], [Chestnutt et al., 2005], [Gutmann et al., 2005]). Because the complexity of the A* search quickly increases with the size of the transition model, this size is limited and so are the stepping capabilities. Thus this approach is not always satisfying for it leads to walking motions with little flexibility, and combined with planning it often results in the robot making many steps to perform tasks for which only one or two steps would have arguably been enough. A lot of *ad hoc* methods have been proposed to extend this approach and perform better footstep planning, such as for instance local footstep adjustments [Chestnutt et al., 2007] or human-like strategies [Ayaz et al., 2006], or tiered planning combining different low-level and high-level planners [Chestnutt and Kuffner, 2004]. There are also really continuous methods that rely on the capacity of the robot to make infinitesimally small steps: sequences of steps are found by first trying to "slide" the robot on the ground (but with these methods stepping over obstacles is not possible; see [Kanoun et al., 2011], [Dalibard et al., 2011]).

In fact, a full spectrum of strategies is emerging, ranging from "discrete approaches" on one end to "continuous approaches" on the other, with very different associated planning techniques. In this thesis, we explore both ends of this spectrum: discrete and continuous approaches, examine the pros and cons, and finally give two original, efficient and coherent frameworks for footstep planning, one based on a discrete approach, and the other on a continuous one.

1.1 Overview of thesis

In chapter 2, we consider the discrete approach to footstep planning, and more specifically to footprint planning which is the important subproblem that consists in planning

sequences of footprints without taking into account the actual robot trajectory. Usually this problem is solved by some graph search algorithm (A*, etc.) in a discrete graph of steps whose size grows exponentially with the maximum length of the sequences of steps considered. By choosing a slightly unusual discretization, it is also possible to obtain a grid whose growth is polynomial in this length (and depends on a resolution parameter). We study related problems of 2D discrete planning on a grid from a theoretical point of view, and wonder if there would exist alternatives to the traditional graph search algorithms, such as for instance algorithms that would directly reason on the description of the obstacles or the transition model definition, instead of taking them only implicitely into account. We show some links with automata theory, but do not exhibit any efficient algorithm: we mostly prove negative results, the most interesting one being probably the NP-hardness result of Section 2.4.

In Chapter 3 we start focusing on more practical algorithms. In order to obtain fast footstep planning, we consider the idea of precomputing data structures that, once embedded in the robot, should replace computationally costly tests by approximation functions that are extremely quick to evaluate. We introduce a simple but original approximation algorithm aimed at maximizing its efficiency by taking advantage of the specificities of the context. We use it to approximate feasibility tests for the robot HRP-2, and show results in experiments requiring real-time step corrections.

In Chapter 4, noticing some drawbacks of the combination between offline precomputations and state-of-the-art walking pattern generators, we introduce a new walking pattern generator which provides features that can be advantageously exploited by approximation and planning algorithms, such as a low dimensional input space and a computationally efficient homotopy for trajectory smoothing. We also obtain and discuss theoretical bounds on the sensitivity of this walking pattern generator.

In Chapter 5, we describe an original and efficient framework for discrete footstep planning where the algorithm of chapter 3 is used to build many swept volume approximations that help to significantly reduce the time spent in online collision checks. We show the results of several implementations on HRP-2.

In Chapter 6, we introduce a new equivalence between some discrete and continuous 2D motion planning problems. We then use this result as the foundation of a novel algorithm for footstep plannings which, as a consequence of the equivalence, can be based on any classical rigid body motion planning algorithm such as RRT [LaValle and Kuffner, 2000], PRM [Geraerts and Overmars, 2002], SBL [Sanchez and Latombe, 2001], KPIECE [Sucan and Kavraki, 2008], etc. We use simulations to test this algorithm which also combines the features of the walking pattern generator of Chapter 3 with an original hybrid bounding box.

Finally, Chapter 7 summarizes the results of this thesis, and presents some directions to extend this work.

1.2 Walking robots and footstep planning: a brief history

More than forty years ago, a seminal method for biped gait synthesis was proposed in [Vukobratovic and Juricic, 1969]. The ideas introduced in this paper lead to the notion of Zero Moment Point (ZMP), a fundamental concept for both biped gait synthesis and biped control. And even three years before that, the robot series WL started to be developped in Ichiro Kato's laboratory at Waseda University. In 1973, WL-5 was used as the lower limbs of the first full-scale anthropomorphic walking robot (WABOT-

1), and in 1984, the concept of ZMP was used in the first realization of dynamically balanced[1] biped gait (it was done with the WL-10D robot). In the 1980s the Leg Lab founded by Marc Raibert at Carnegie Mellon University (but later moved to the Massachusetts Institute of Technology) also had a large influence on the research in robotic locomotion and especially in the consideration of dynamic balance. In the 1990s there were several active humanoid robot projects, and great progress was made in control schemes for biped locomotion (we can mention [Kajita and Tani, 1991] which introduced the first version of the linear inverted pendulum model whose equations are now used in the gait generation algorithms of most humanoid robots). In 2000 was unveiled the humanoid robot ASIMO, result of more than 15 years of research at Honda Motor Co., Ltd. Between 2000 and 2003 several other advanced humanoid robots were completed, such as H6 and H7 that were developed at Tokyo University, QRIO made by Sony Corp., HRP-2 by Kawada Industries, Inc., JOHNNIE built at the Technical University of Munich, or HUBO prototypes developed by the Korea Advanced Institute of Science and Technology (KAIST). With reliable platforms and increasing computational power, research in footstep planning really started during this period, notably in a joint effort between Carnegie Mellon University, Tokyo University and then the Digital Human Research Center of the Japanese institute of Advanced Industrial Science and Technology (AIST). In a decade, researchers have been able to deal with increasingly complex problems such as planning quasi-static walking motions in a known environment [Kuffner et al., 2001] or in an environment acquired by vision [Kuffner et al., 2003], planning dynamically stable walking motions among moving obstacles [Chestnutt et al., 2005] or movable obstacles [Stilman et al., 2006], or using footstep planning [Chestnutt et al., 2009] or advanced balance control [Nishiwaki and Kagami, 2010] for biped navigation in rough environments. Of course the field of footstep planning is not limited to bipeds, and in particular interesting achievements have been reached with quadrupeds such as the Boston Dynamics robots BigDog and LittleDog (see for example [Kalakrishnan et al., 2010], [Byl et al., 2009]), but in this thesis we will focus exclusively on humanoid robots, even though some of our results could be transposed to other types of legged robots.

1.3 Summary of contributions

The main contributions of this thesis are:

- in Chapter 2, some complexity results, and especially the NP-hardness result of Section 2.4, proved in Appendix D;
- in Chapter 3, the approximation algorithm and the proof of its convergence (Section 3.3);
- in Chapter 4, the walking pattern generator based on half-steps with a smoothing homotopy (Section 4.2), and the proof of the sensitivity bound (Section 4.3);
- in Chapter 5, the footstep planning framework obtained by combining offline swept volume approximations and the walking pattern generator of Chapter 4;

[1] In this thesis we interchangeably use the expressions "statically balanced", "statically stable", "quasi-static" and "static" to denote gaits that involve only balanced postures of the robot (i.e. postures such that the center of mass is above the convex hull of the contact points with the ground). We also interchangeably use the expressions "dynamically balanced" and "dynamically stable" to denote gaits that are *not* statically stable but such that the robot does not fall.

- in Chapter 6, the study of the "flea motion planning problem" (Section 6.1), the equivalence result of Section 6.2 to Section 6.5, and the footstep planning algorithm based on a hybrid bounding box (Section 6.7).

Chapter 2

Discrete Motion Planning in the Plane

Footstep planning is *not* a problem of motion planning in the plane. Indeed, even if footprints can often be described as elements of $SE(2) = \mathbb{R}^2 \times SO(2)$ (the Euclidean group of rigid motions of the plane), what needs to be planned are trajectories in the configuration space of a humanoid robot or at least its lower body, i.e. in a space with more than 12 degrees of freedom (most of the advanced humanoid robots have a lower body with at least 6 degrees of freedom per leg). Besides, although footprints must avoid 2D obstacles, the robot itself must avoid 3D obstacles. For these reasons footstep planning is much more complex than motion planning in the plane. Nevertheless, generating trajectories with non-trivial dynamic constraints in a 12-dimensional space being an extremely difficult task, the efficient way to perform footstep planning is to separate the generation of the footprint sequence from the generation of the robot trajectory in the configuration space. The risk with this separation is to produce footprint sequences that don't lead to feasible trajectories, or on the contrary be overconservative and frequently miss solutions. When the first issue is avoided we say that the separation is sound, and when solutions are not missed we say it is complete. Completeness, or at least "almost completeness", is desirable, but soundness is crucial. We will see throughout the following chapters that sound separations which are not too overconservative can effectively be achieved. As a consequence the problem of planning sequences of footprints, which can be considered as a problem of motion planning in the plane (we recall that we assume the ground to be flat and horizontal) is a very important component of footstep planning. In this thesis we will call this subproblem *footprint planning* while footstep planning will refer to the global problem. Footprint planning has a "hybrid nature" in the sense that it has both discrete and continuous aspects: on one hand, the sequence of footprints to plan is obviously discrete, and on the other hand next footprints can be chosen inside continuous feasibility regions. Please note that the hybrid nature of footprint planning is very different from the global hybrid nature of humanoid robot motion planning, which is characteristic of robotic systems with contacts in the sense that they continusously move between

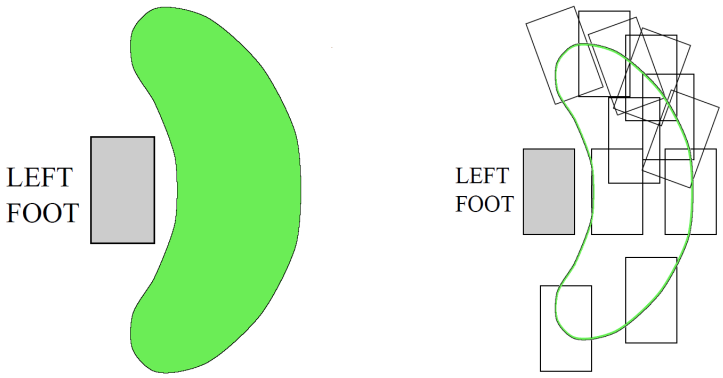


Fig. 2.1: This figure taken from [Kuffner et al., 2001] illustrates continuous and discrete approaches. On the left, a continuous feasibility region is considered, and on the right, only a finite set of possible steps is allowed.

modes where the set of contacts (e.g. footprints) is fixed. In each mode, the allowable motions lie on a lower dimensional submanifold of the configuration space, and the set of contacts can be changed at the intersection between different modes. Finding a discrete sequence of modes together with a continuous motion that follows the sequence is called "multi-modal planning". In this thesis, we don't study multi-modal planning (for a good introduction see [Hauser, 2008]); instead we focus on footstep planning which is a specific problem of multi-modal planning, but for which we show specific approaches that clearly separate the search for a discrete sequence of modes (the footprints) from the generation of a continuous walking motion. In footstep planning the problem of finding the discrete sequence of modes (footprint planning) is hybrid in itself and one can choose to use the continuous aspect of the feasibility regions for steps, or use only a finite number of possible steps so that to obtain a fully discrete planning problem. Fig. 2.1 illustrates these two strategies.

We use the term *discrete approach* to refer to footprint planning methods where only a finite number of steps is considered, and the term *continuous approach* to refer to methods that actually use continuous regions of feasible steps. We also simply call these approaches *discrete footprint planning* and *continuous footprint planning*, respectively, and by extension we use the terms *discrete footstep planning* and *continuous footstep planning* according to the approach used for footprint planning. The discrete approach is actually the current state-of-the-art method for efficient footprint planning: a finite set of steps is fixed in advance, and then variants of the A* search algorithm are used to plan sequences of steps. This method amounts to a search in an infinite but discrete graph of steps; it is convenient because easy to implement but it completely overshadows the continuous aspect of the problem which is very important in approaches such as [Kanoun et al., 2011] and [Dalibard et al., 2011]. What's more, the finite set of steps must remain relatively small (around 15 to 30 steps is standard) for the performance of the A* algorithm quickly dwindles when the number of outgoing edges per node increases (i.e. the number of possible steps), and this has a negative impact on the flexibility of the robot paths.

When the task is to go to a certain goal, we can usually evaluate a rough upper bound of the number of steps required. Let's say for instance that we assume the goal to be reachable in n steps. In that case, with only a k possible steps in the transition

model we know that the solution can be searched in a finite graph whose size grows as k^n. In such a graph an A* search can often perform in time polynomial in n, but a large k still has a strong impact on the performances. As we will see, if discretization is made *up to a resolution*, the graph gets a grid structure which ensures its size to be polynomial in n, but since the number of outgoing edges per node is unchanged it does not necessarily make search algorithms faster in average.

A particularity of the state-of-the-art search techniques for (discrete and continuous) motion planning is that they only implicitely take obstacles into account: indeed obstacles only appear in the main loop of the algorithm when they happen to make some motion or configuration unfeasible. However, for some simple continuous motion planning problems in the plane, there exist exact solutions directly reasoning on the description of the obstacles given in the input. For example algorithms using visibility graphs (see [de Berg et al., 2000], ch. 15), Voronoi diagrams or algebraic representations (see [Schwartz et al., 1986], ch. 6 and ch. 2). Although not always efficient in practice, these algorithms are polynomial in the size of the obstacles description. In this section, we consider problems of discrete motion planning in the plane that are related to footprint planning, and we wonder whether there would exist some algorithms that could efficiently exploit the structure of the transition model as well as a compact description of the obstacles rather than using a heuristic search. While the choice of the finite set of steps is usually made by an expert user, such algorithms would be especially useful when a large number of possible steps is automatically generated (which we will do in Chapter 5): in that case we do not necessarily control the properties of the finite set of steps, and its large cardinality makes classical algorithms such as A* inefficient. The goal of the present chapter is to explore the possibility for such techniques, but we will mainly show negative results. Our discussion will reveal some connections with automata theory, we will present some NP-complete and polynomial problems, an undecidability result, an open problem and a decidability result, and we will conclude with probably the most important result of this chapter: a proof of NP-hardness for a basic problem of discrete motion planning in the plane (Section 2.4). This chapter is essentially independent from the remainder of this thesis, and since it does not contain practical results, the reader in a hurry can safely skip it and directly start at Chapter 3.

2.1 From discrete stepping capabilities to 2-counter machines

We suppose that a biped robot is walking on a flat ground, free from any obstacle. By discrete stepping capabilities, we mean that only a finite set of possible steps is allowed. A step is a transition between two postures of the feet, a posture of the feet being defined by the position and orientation of the right foot *relatively* to the left foot. For mathematical convenience, *in this section* we represent the stepping capabilities of the robot by the finite set of possible sequences of two steps (instead of just one), and always take the left foot as a reference. We use a simple model of discrete stepping capabilities where we can concatenate two sequences of two steps if and only if the initial posture of the second sequence is the same as the final posture of the first sequence (sometimes this concatenation is subject to dynamic constraints, but not here).

Let us provide the flat infinite ground with a Cartesian coordinate system of two

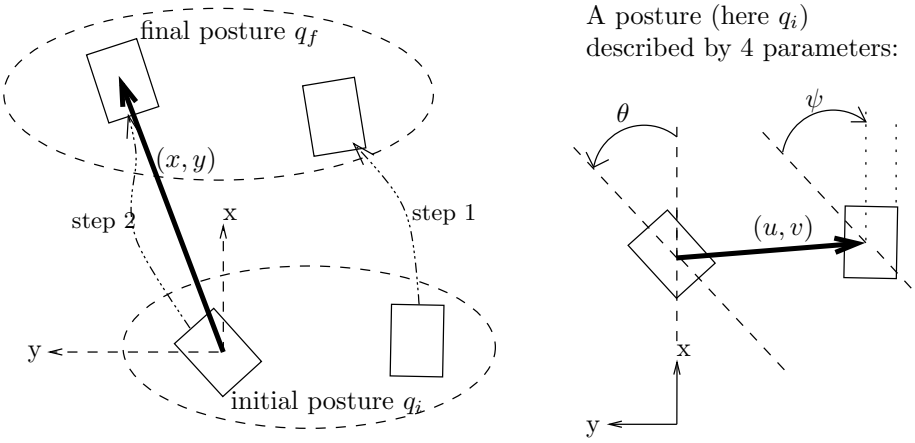


Fig. 2.2: (*Left*) The sequence of two steps from posture q_i to q_f, with translation parameters (x, y). (*Right*) A posture is fully described by the absolute orientation of the left foot, and the relative position and orientation of the right foot.

axes x and y, and let the x axis define the zero orientation. The configuration of the robot (feet) in the free environment is then completely defined by the position and orientation of the left foot, and the current posture of the feet.

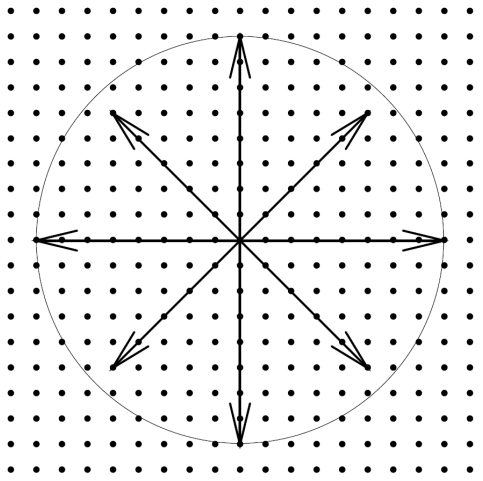


Fig. 2.3: An example of grid construction

A sequence of two steps is completely defined by the initial and final posture, and three additional parameters x, y and θ: the position and orientation of the left foot final placement relatively to its initial placement. If we suppose that this orientation change is a rational number multiplied by π, it turns out that only a finite set of absolute orientations of the left foot are reachable. From now on (in this Chapter), what we call a posture will be defined not only by the position and orientation of the right foot relatively to the left foot, but also by the left foot absolute orientation (so a posture is a quadruple (u, v, ψ, θ): see Fig. 2.2 on the right). Let us denote by $\{p_1, \ldots, p_N\}$ the set of all the reachable postures, which are in finite number. As shown on Fig. 2.2, a sequence of two steps is now completely defined by the initial and final posture plus a vector of two parameters. Similarly, a configuration of the robot is a posture and a vector of two parameters (position of the left foot). These vectors should intuitively have

real-valued coordinates, since orientation changes usually induce irrational numbers that make the set of reachable positions dense in $\mathbb{R}^2$. However, we know that the feasibility regions are actually continuous, and thus we can decide to slightly modify the irrational positions in order to make them belong to a grid of fixed resolution that we can decide in advance (a similar "trick" for grid construction is described in [Tate, 1991], section 4.3, where it is illustrated with Fig. 2.3). As a result, we can assume that all the vectors are couples of rational numbers, and then, adapting the scale, couples of integers. It follows that the whole stepping capabilities can be represented by a finite state machine (or automaton) with two integer-valued counters, where $Q = \{p_1, \ldots, p_N\}$ is the set of states, and each sequence of two steps is a transition $(q_a, x, y, q_b) \in Q \times \mathbb{Z}^2 \times Q$. A configuration of the machine is a triple $(q, z_1, z_2) \in Q \times \mathbb{Z}^2$, which exactly corresponds to a configuration of the robot (posture q and left foot at (z_1, z_2)) in the free environment. If (q_a, z_1, z_2) is the current configuration of the machine, after transition (q_a, x, y, q_b) its configuration is $(q_b, z_1 + x, z_2 + y)$. The set $Access(q_0, x_0, y_0)$ of all the reachable configurations from an initial configuration (q_0, x_0, y_0) is the set of configurations (q, x, y) such that there exists a finite sequence (q_0, x_1, y_1, q_1), (q_1, x_2, y_2, q_2),..., (q_{k-1}, x_k, y_k, q_k) such that $q_k = q$, $\sum_{j=0}^{k} x_j = x$ and $\sum_{j=0}^{k} y_j = y$. This set corresponds to the set of configurations that are actually reachable by the robot walking in the free environment, when its initial configuration is (q_0, x_0, y_0). One simple but important property of $Access()$ is:

$$Access(q_0, x_0, y_0) = \{(q, x, y) | (q, x - x_0, y - y_0) \in Access(q_0, 0, 0)\} \qquad (2.1)$$

Fig. 2.4 illustrates the conversion of a simple example of discrete stepping capabilities into a 2-counter machine.

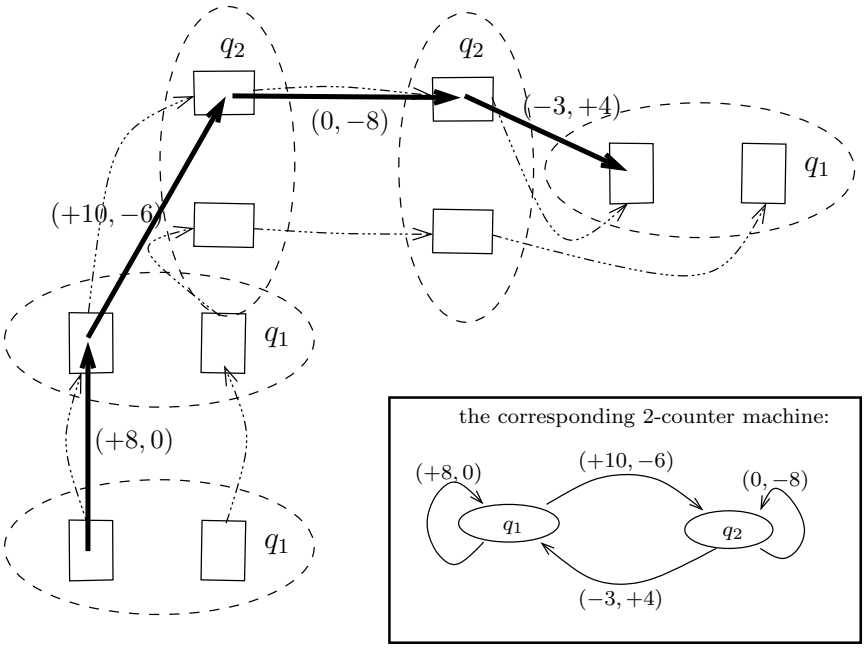


Fig. 2.4: A simple set of possible sequences of 2 steps converted into a 2-counter machine.

So, discrete stepping capabilities can be seen as 2-counter finite state machines. The converse is not true because stepping capabilities of biped robots have natural

properties that are not shared by all 2-counter machines. However, if we find efficient planning techniques for general 2-counter machines, it will be possible to apply them to footprint planning, and with this hope we study general 2-counter machines in sections 2.2 and 2.3. Besides, 2-counter machines can be used to model discrete motion planning problems other than discrete footprint planning, so an efficient algorithm could have several applications. Our 2-counter machines are closely related to what was first called blind multicounter machines (see [Greibach, 1978], [Kambites, 2006]) and now known under various names such as valence automata ([Render and Kambites, 2009]), extended finite automata ([Mitrana and Stiebe, 2001], [Mitrana and Stiebe, 1997]) or integer weighted automata ([Halava and Harju, 1999], [Halava and Harju, 1998], [Boichut et al., 2009]). These automata have been studied in the context of model checking and they also provide algebraic methods to characterize important language classes such as the context-free, recursively enumerable and blind counter languages, whose accepting power has been thoroughly studied, leading to a plethora of results that might be useful in our context, but which would help to characterize classes of paths rather than solve planning problems. Planning problems are close to reachability problems, and thus model checking techniques might be more directly applicable.

2.2 The reachability and universal reachability problems in a free unbounded environment

As we have seen in the previous section, the reachability of a configuration, for a biped robot walking with discrete stepping capabilities on an infinite flat ground free from obstacles, can be expressed as the reachability of a configuration in a 2-counter machine. If (Q, T) is a 2-counter machine (Q is the set of states and T the set of transitions), we denote by $|(Q, T)|_B$ the size of a classical binary encoding of (Q, T).

The problem of reachability can be expressed this way:

Definition 2.2.1 (Reachability problem)**.** *Given a 2-counter machine (Q, T), and two configurations (q_i, x, y) and (q_f, x', y'), is there a finite sequence of transitions that goes from (q_i, x, y) to (q_f, x', y'), i.e. do we have $(q_f, x', y') \in Access(q_i, x, y)$?*

Because of the equality (2.1), the reachability of (q_f, x', y') from (q_i, x, y) is equivalent to the reachability of $(q_f, x' - x, y' - y)$ from $(q_i, 0, 0)$. Thus when $(q_f, x', y') \in Access(q_i, x, y)$, we simply write $q_i \xrightarrow[(x'-x,y'-y)]{*} q_f$.

We will also consider the problem of universal reachability which is slightly different:

Definition 2.2.2 (Universal reachability problem)**.** *Given a 2-counter machine (Q, T), and an initial configuration (q_0, x, y), is any configuration reachable from it, i.e. do we have $Access(q_0, x, y) = Q \times \mathbb{Z}^2$?*

These two problems correspond to very natural questions in automata theory, and in our case they are related to motion planning, even if there are no obstacles so far. As we mentionned previously, the new techniques we are looking for might be useful when discrete stepping capabilities with a large number of steps are automatically generated, and thus it is interesting to evaluate, for some basic problems, the complexity in the size of the transition model, i.e. in $|(Q, T)|_B$.

The complexity of the reachability problem is clearly at least linear in $|(Q, T)|_B$, and it is also equivalent to or more than the complexity of the universal reachability

problem, because universal reachability can be essentially reduced to the conjunction of a few reachability problems. Indeed, if $(q_0, x+u, y+v) \in Access(q_0, x, y)$ for all $(u, v) \in \{-1, 0, 1\}^2$ (this is the conjunction of 9 reachability problems), then we have $q_0 \xrightarrow[(a,b)]{*} q_0$ for all $(a, b) \in \mathbb{Z}^2$, and if on top of that all states can be reached from q_0 (this can be verified with a simple graph search), it follows that $Access(q_0, x, y) = Q \times \mathbb{Z}^2$. The converse implication is also true: if $Access(q_0, x, y) = Q \times \mathbb{Z}^2$, then all states can be reached from q_0, and $(q_0, x+u, y+v) \in Access(q_0, x, y)$ is verified for all $(u, v) \in \{-1, 0, 1\}^2$.

In the next section, we prove that the reachability problem is actually NP-complete. In section 2.2.2, we prove that with a natural additional assumption (the ability to make U-turns in any posture), the universal reachability problem is in PTIME.

2.2.1 NP-completeness of the reachability problem

The reachability problem is equivalent to the "fixed distance problem" in finite weighted graphs, in the case of weights belonging to $\mathbb{Z}^2$. In [Nykänen and Ukkonen, 1999], this problem is studied and proven NP-complete. The membership to NP follows from a reduction to integer linear programming (i.e. linear optimization problems in which the variables are restricted to be integers). Here, we prove again the NP-hardness using a reduction from SUBSET-SUM (see [Garey and Johnson, 1990] p.223) similar to the one done in [Laroussinie et al., 2004], where the problem of reachability for timed automata with 2 clocks is considered.

Definition 2.2.3 (SUBSET-SUM)**.** *Assume we are given a set $\{a_1, \ldots, a_p\}$ of integers and a goal b, one asks whether there exists a subset $J \subset \{1, \ldots, p\}$ such that $\sum_{j \in J} a_j = b$.*

This problem is known to be NP-complete, and Fig. 2.5 shows a (polynomial) reduction of an instance $(\{a_1, \ldots, a_p\}, b)$ of SUBSET-SUM to the reachability problem for a 2-counter machine (actually only one counter is used in the reduction). The existence of a solution for SUBSET-SUM is clearly equivalent to the reachability of configuration $(q_p, b, 0)$ from $(q_0, 0, 0)$.

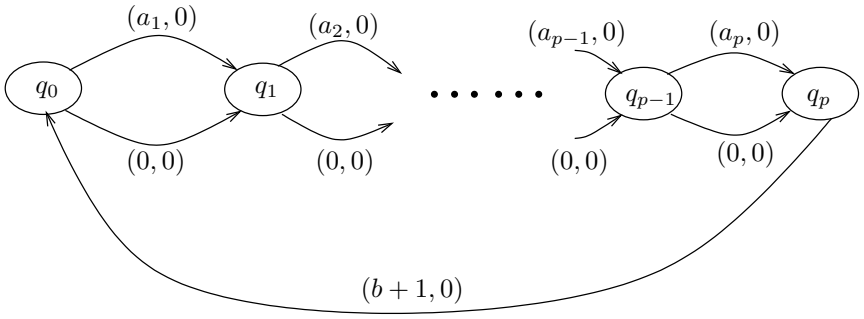


Fig. 2.5: The configuration $(q_p, b, 0)$ is reachable if and only if the corresponding instance of SUBSET-SUM has a solution.

We can thus state the first theorem:

Theorem 2.2.1. *In a free environment, the reachability problem is NP-complete.*

As well as an immediate corollary:

Corollary 2.2.1. *In a free environment, the universal reachability problem belongs to NP.*

Surprisingly, proving that the universal reachability problem is NP-hard (and thus NP-complete) seems much more difficult. We let the NP-hardness of the universal reachability problem as an open question, but in the next section we show that some natural property (natural for biped robots) can lead to a linear-time decision algorithm.

2.2.2 The U-turn property

Definition 2.2.4 (U-turn property)**.** *A 2-counter machine (Q,T) is said to verify the "U-turn property" if there exists a permutation $(\cdot)^\top : Q \to Q$ such that:*

1. $\forall q \in Q,\ (q^\top)^\top = q$
2. $\forall (q,x,y,q') \in T,\ (q^\top, -x, -y, (q')^\top) \in T$
3. $\forall q \in Q,\ q \xrightarrow[(0,0)]{*} q^\top$

If (Q,T) is viewed as stepping capabilities, the U-turn property can be translated as follows: in any configuration (q,x,y), the robot can perform a sequence of steps to reach the configuration $(q^\top,x,y)$ which is obtained by applying on (q,x,y) a rotation of angle 180° around the center of the left foot. Of course, taking the left foot as a reference was arbitrary, and the result we are going to prove holds if we decide to take for example the right foot, or a position between the feet, as a reference.

Theorem 2.2.2. *For the class of 2-counter machines effectively verifying the U-turn property, the universal reachability problem can be solved in time linear in $|(Q,T)|_B$.*

The proof of this theorem is to be found in appendix A.

2.3 With finitely described obstacles

We first show that a natural way to define obstacles is to use transition guards, i.e. logical constraints that restrict the use of transitions.

Automata with transition guards are of prime importance in the field of model checking (see for example [Vardi and Wolper, 1986]). Usually, guards are defined by a particular logic with the hope that some natural questions (e.g. the emptiness of the language recognized, the reachability of some configuration, etc.) can be proven decidable. For example, by reducing them (the natural questions) to formula expressed in a decidable theory, such as the additive theory of $\mathbb{Z}$ (the Presburger arithmetic), they can be decided in doubly exponential time ([Fischer and Rabin, 1974]). More particularly, a lot of work has been done on counter machines with transition guards, with both positive and negative results. For example, if the machine is flat (i.e. its transition graph has no nested loop), then the reachability problem stays decidable for an expressive fragment of logic (see [Comon and Jurski, 1998]). However, flatness is not really a natural property for the 2-couter machines corresponding to walking robots.

Besides, in order to describe obstacles, it is more natural to put guards on the configurations rather than on the transitions. Therefore we only consider guards described by formulas of the form $\phi_q(x,y)$ and such that a configuration (q,x,y) is allowed if

and only if the corresponding formula $\phi_q(x, y)$ is evaluated to `true`, and forbidden otherwise (because the configuration intersects an obstacle).

In the next section we show a simple class of guards which leads to the undecidability of the reachability problem, and in Section 2.3.2, we consider even simpler guards corresponding to a finite number of finite obstacles in the environment. With no assumption on the 2-counter machine, whether the reachability is decidable or not is let as an open problem, but we show that it becomes decidable at least when the U-turn property and two other minor constraints are verified.

2.3.1 A result of undecidability

In [Hopcroft et al., 2006] (theorem 7.9, p.172), it is shown that if a 2-counter machine is allowed to test whether the value of any of its counters is zero, then it can simulate an arbitrary Turing machine, and therefore the corresponding reachability problem is undecidable. Let (Q, T) be such a 2-counter machine. The zero-tests are naturally represented by guards on the transition that can be written:

$$(q, x, y) \xrightarrow{\phi(x,y)} (q', x + a, y + b),$$

where $\phi(x, y)$ is a formula constructed according to the following grammar:

$$\phi(x, y) ::= \texttt{true} \quad | \quad x = 0 \quad | \quad y = 0 \quad | \quad \neg\phi_1(x, y) \quad | \quad \phi_1(x, y) \wedge \phi_2(x, y)$$

Definition 2.3.1 (Zero-test guards)**.** *We call "zero-test guards" the guards constructed according to the above grammar.*

The above transition can be fired if and only if $\phi(x, y)$ is evaluated to `true`.

Let us show how to convert any such machine into a 2-counter machine with guards *on the configurations.*

We start with the graph Q with no edge. For each transition $t = (q, x, y) \xrightarrow{\phi(x,y)} (q', x + a, y + b)$ of the initial machine, we create a new state q_t on which we put the guard $\phi_{q_t}(x, y) = \phi(x, y)$: a configuration (q_t, x, y) is allowed if and only if $\phi(x, y)$ is evaluated to `true`. We add also the transitions $(q, 0, 0, q_t)$ and (q_t, a, b, q'). It is clear that the reachability in the initial machine of a configuration (q, x, y) is equivalent to the reachability of the same configuration in the 2-counter machine just created. Therefore, we can conclude, stating the following theorem:

Theorem 2.3.1. *The problem of reachability for 2-counter machines with zero-test guards on the configurations is undecidable.*

On Fig. 2.6, we show how would look like a real environment associated with a 2-counter machine with zero-test guards. Of course, the Turing machine simulation used in the proof of undecidability is not likely to correspond to reasonable discrete stepping capabilities, and therefore using some characteristic properties of realistic stepping capabilities (such as the U-turn property for instance) might make it possible to obtain the decidability of this reachability problem.

We tried to use model checking tools to solve the problem of motion planning among obstacles defined as transition guards, with very simple discrete stepping capabilities. Our hope was that despite the general undecidability result, model checking tools might be able to discover and exploit the intrinsic simplifying properties of our stepping capabilities. Unfortunately it was not the case, and the tools that we used: BRAIN

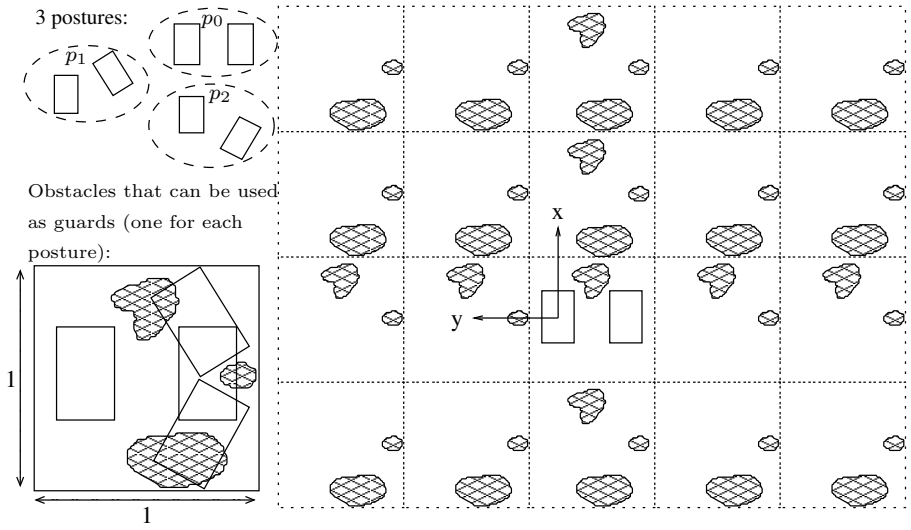


Fig. 2.6: We consider a 2-counter machine with 3 states p_0, p_1 and p_2, and guards $\phi_{p_0}(x,y) = (x = 0 \wedge y = 0)$, $\phi_{p_1}(x,y) = (x \neq 0 \wedge y \neq 0)$, and $\phi_{p_2}(x,y) = (x = 0)$. On this figure we show how obstacles can be put in the whole environment in order to encode these guards for 3 postures corresponding to the 3 states of the 2-counter machine.

([Rybina and Voronkov, 2002]) and FAST ([Bardin et al., 2003]), were much slower than an A* search, which is not very surprising given that they were not designed to tackle motion planning problems. They can however be used to prove properties for which an A* search is useless, such as for example proving that some configuration is unreachable, or that all obstacle-free configurations are reachable, etc. But the time required to obtain such results is prohibitive for any practical application.

2.3.2 An open problem and a result of decidability

Let (Q, T) be a 2-counter machine, with guards of the form $\phi_q(x,y) = ((x,y) \notin S_q)$, where S_q are finite subsets of $\mathbb{Z}^2$. This kind of guards, called "finite guards", correspond to environments with a finite number of finite obstacles, as shown on Fig 2.7.

We are interested in the decidability of the reachability problem in this context. Although there are a lot of tools in the litterature that have been successfully used to tackle similar questions, we let it as an open problem. Let us nevertheless present here some related work.

Let $\mathcal{L}$ be the language recognized by a Pushdown Finite Automata on alphabet $\Sigma = a_1, \ldots, a_n$. We denote by $\#_{a_i}(w)$ the number of occurences of a_i in the word w. The theorem of Parikh states that $\{(\#_{a_1}(w), \#_{a_2}(w), \ldots, \#_{a_n}(w)), w \in \mathcal{L}\}$ (the image of Parikh of the automaton), is an effectively computable semi-linear set ([Parikh, 1966]). Besides, semi-linear sets are exactly the sets that can be expressed in the decidable Presburger Arithmetic. If we could show that even in presence of (finite) obstacles, the sets of reachable configurations are semi-linear, it would result that the reachability problem is decidable. To do that, using the theorem of Parikh could be a solution, and indeed Parikh's construction has been used in many works of great interest (see for example [Seidl et al., 2004]). Pushdown Finite Automata allow the use of one counter, which could be used to represent the obstacles. Nevertheless, the

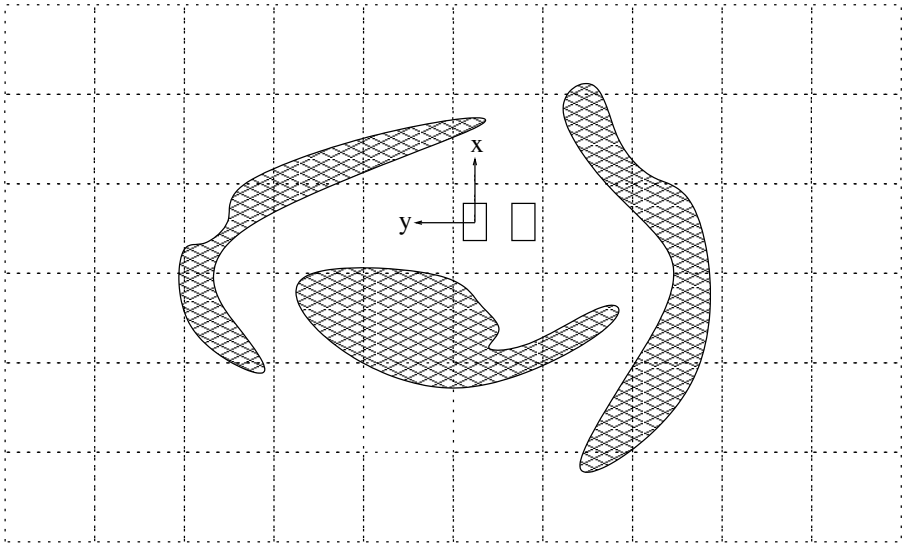


Fig. 2.7: An infinite environment with finite obstacles.

difficulty here is that, to "remember" the position of the obstacles, we need 2 counters, which leads to the same expressiveness as Turing Machines.

A problem very similar to our reachability problem is the reachability problem for 2-dimensional Vector Addition Systems with States (2D-VASS). Actually, if we consider no obstacle but instead the constraint that the robot can only move in the quadrant of the plane corresponding to the couples of non-negative integers, then we exactly obtain the problem of reachability for 2D-VASS. Hopcroft and Pansiot [Hopcroft and Pansiot, 1976] proved that the set of reachable vectors is effectively semi-linear for 2D-VASS, and they designed a decision algorithm for the problem of reachability. Their proof cannot be directly used with the four quadrants of the plane and obstacles, but it might be possible to adapt the idea of constructing a tree which can generate the whole set of reachable configurations, and whose finiteness is a consequence of König's Lemma (such a construction is often called a Karp-Miller tree, see [Karp and Miller, 1969]). In [Verma and Goubault-Larrecq, 2004], Karp-Miller trees are used to prove the decidability of some problems on BVASS, an extension of VASS. Yet, the decidability of reachability in BVASS is equivalent to the decidability of provability in MELL (Multiplicative Exponential Linear Logic), which is an open problem. One other property of the 2D-VASS exposed in [Hopcroft and Pansiot, 1976] is that they can be simulated by 5-dimensional VAS (Vector Addition Systems with only one state, which are equivalent to Petri Nets). With only one state our problem of reachability would be probably easier to prove decidable, but unfortunately without the constraints of the VAS (all the counters must stay positive), we cannot apply the trick used in [Hopcroft and Pansiot, 1976]. Finally, over the past decades, a very large amount of work has been done on Petri Nets and VAS or VASS, leading to new proof techniques (see for example [Leroux, 2009]) and promising extensions (see for example [Reinhardt, 2008], [Finkel and Sangnier, 2010]). At first, our problem of reachability seems easier than some similar problems that have been proven decidable, and actually the vast litterature might already contain tools that could be adapted to validate or invalidate the following conjecture:

Conjecture 2.3.1. *In an infinite environment with finite obstacles, the reachability*

problem is decidable.

Yet, the proof of this conjecture (if valid) probably requires advanced arguments, whereas we show now that a rather natural property (natural for our context) leads to a quite straightforward decidability result.

Property 2.3.1 (Four-Vectors property)**.** *We say that a 2-counter machine (Q,T) verifies the "Four-Vectors property" if there exist two non-colinear vectors $u \in \mathbb{Z}^2$ and $v \in \mathbb{Z}^2$ such that, when ignoring the guards, for each $q \in Q$ we have $q \xrightarrow[u]{*} q$, $q \xrightarrow[-u]{*} q$, $q \xrightarrow[v]{*} q$, and $q \xrightarrow[-v]{*} q$.*

Soon we will see that with a few more assumptions, this property is implied by the U-turn property (now we say that a 2-counter machine with guards verifies the U-turn property if ignoring the guards leads to a 2-counter machine verifying the U-turn property). Besides, the Four-Vectors property implies the decidability of the reachability problem in an infinite environment with finite obstacles:

Theorem 2.3.2. *For 2-counter machines with finite guards effectively verifying the Four-Vectors property, the reachability problem is decidable.*

The proof of Theorem 2.3.2 is to be found in appendix B.
Additionally, we prove the following theorem in appendix C:

Theorem 2.3.3. *If (Q,T) is a 2-counter machine such that there exists at least two transitions that are not colinear, and such that the graph (Q,T) is strongly connected, then:*

(Q,T) verifies the U-turn property $\Rightarrow$ (Q,T) verifies the Four-Vectors property

Using Theorem 2.3.2, we obtain an immediate corollary:

Corollary 2.3.1. *If (Q,T) is a 2-counter machine with finite guards such that there exist at least two transitions that are not colinear, and such that the graph (Q,T) is strongly connected, then:*

(Q,T) effectively verifies the U-turn property $\Rightarrow$ the reachability problem is decidable

We would like to stress here the fact that for a 2-counter machine representing a walking robot, the two first conditions can almost be taken for granted: first, if all the transitions are colinear, it means that the robot can only move along a line (which is quite useless), and second, being always able to come back to some standard posture is a very natural expectation for a humanoid robot. Thus in most cases the graph (Q,T) should be strongly connected.

2.4 A NP-hard 2D discrete shortest path problem

The previous sections of this chapter give the idea that there is little hope for an efficient discrete planning algorithm based on a 2-counter machine representation, unless maybe if we find specific simplifying hypotheses that make planning and model checking problems easier. To see if there is a chance for such simplifications to actually lead to practical algorithms, a natural preliminary step is to study problems with very simple discrete stepping capabilities. If we can show that some problems are hard even in that

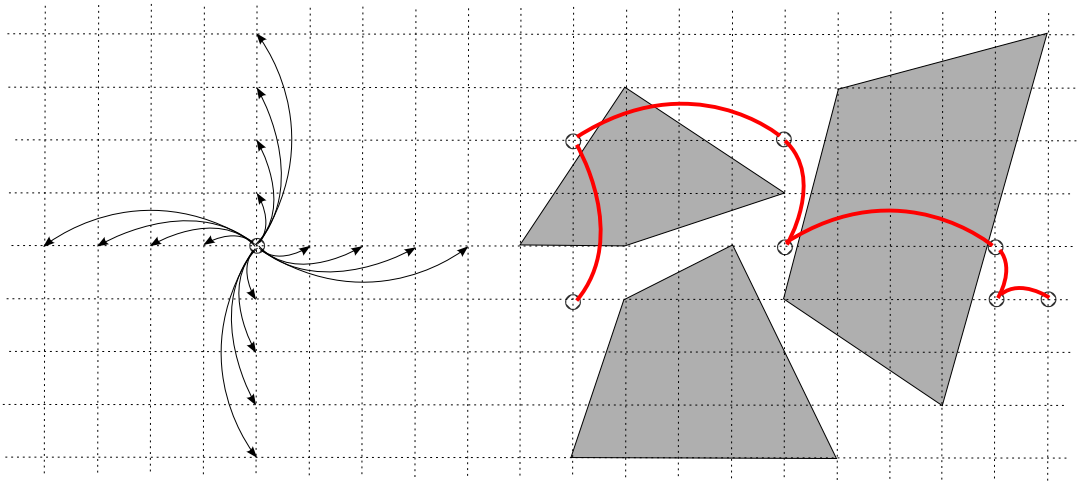


Fig. 2.8: On the left, the "jumping capabilites" of the point robot, and on the right, a valid sequence of jumps among disjoint polygonal obstacles.

case, then we will be in a position to believe that even with simplifying hypotheses the hope for algorithms with good worst-case complexity is thin. This is exactly what we do in this section: we consider a very basic discrete shortest path problem in the plane, and show that it is NP-hard.

The problem definition is the following: we consider a point robot that moves in $\mathbb{Z}^2$ (it has no orientation), and can jump (rather than walk) in the 4 main directions. Its jumps can be of size 1 to 4. We call the displacements of this point robot "jumps" because they can be feasible even if there is an obstacle between the initial configuration and the one after the displacement (but both initial and final configurations must be outside the obstacles). These "jumping capabilites" are described on Fig. 2.8. The input consists of an initial position in $\mathbb{Z}^2$ and a goal position, and a list of disjoint polygonal obstacles whose vertices belong to $\mathbb{Z}^2$. The problem is to find the minimum number of jumps that can bring the robot from its initial position to the goal while avoiding the obstacles (and we arbitrarily decide that being on the edges of an obstacle is a collision). Of course, we can set up a Dijsktra search to run in pseudo-polynomial time, so it means that our NP-hardness result will require the obstacles to occupy an exponentially large region of the plane (exponential in n, the total number of vertices of the obstacles). This can seem somewhat unrealistic, but it is related to the 3D *continuous* shortest path problem, which is NP-hard [Canny and Reif, 1987], but for which discretization leads to approximation algorithms that are polynomial in the resolution parameter (see [Papadimitriou, 1985], [Sellen et al., 1995]). In our case, the grid unit size corresponds to a resolution parameter, and the exponential space occupied by the obstacles can also be seen as an exponentially precise resolution of the grid.

To prove that the problem is NP-hard, we follow the strategy introduced in [Canny and Reif, 1987]: we obtain a reduction from 3-SAT with gadgets implementing operations of "path splitting" and "literal filtering". Similar reductions have been used to prove the NP-hardness of several motion planning problems [Reif and Wang, 1998; Asano et al., 1996]. An interesting point is that 2D continuous shortest path problem is in PTIME [Sharir and Schorr, 1986; Clarkson et al., 1987; Storer et al., 1994], and thus we can somehow relate our NP-hardness result to the gap of complexity between

linear programming with real-valued variables (in PTIME) and integer linear programming (NP-complete, see [Schrijver, 1986] ch. 18). This comparison must be interpreted with caution, however, because the ability to jump over obstacles is essential in the NP-hardness proof, so we are not really comparing a discrete and a continuous version of the same problem. In fact in Chapter 6 we will introduce a problem which is a more natural "continuous version" of the 2D discrete shortest path problem considered in this section. Along the same lines, it is not clear whether our NP-hardness result shows that discretization potentially has negative effects on the computational complexity of motion planning problems, but this question–which we will not answer– is interesting in the context of this thesis.

The proof of NP-hardness being quite involved, we defer it in Appendix D.

2.5 Conclusion

In this chapter, we saw that the natural discretization of the stepping capabilites of a biped robot can be slightly modified to obtain a grid of configurations rather than an unorganized discrete graph. Yet, we proved negative results which give the intuition that this approach will not lead to particularly efficient motion planning techniques. In the next chapter, we start considering continuous stepping capabilites, but since it is quite complicated to know in advance what the continuous feasibility regions look like (e.g. the green area on the left of Fig. 2.1), we investigate the possibility of approximating them through extensive offline precomputations.

Chapter 3

Offline Precomputations

The high number of degrees of freedom of humanoid robots has a strong impact on the computation cost of numerical methods used to produce motions as well as on the complexity of collision detection [Kuffner et al., 2002]. The number of potentially colliding pairs of bodies grows as the square of the number of bodies. What's more, humanoid robots are subject to dynamic constraints that make most motions unstable. One way to cope with this complexity is to precompute data structures that depend only on the robot geometry and dynamics, and then use these data structures to speed up online computations. We explore this idea in the present chapter.

As we will see later, the footstep planning problem can be solved by growing large trees of steps with a bias towards the goal. The growing tree of steps should obviously only consider feasible steps, and if the robot has no specific prior knowledge, the only solution to test the feasibility of a step is to actually generate the trajectory, and then check some properties (or run a dynamic simulation) to verify that it is valid. Unfortunately, the current trajectory generation and verification techniques only allow the verification of a step in a couple of hundreds of milliseconds (on a standard computer). This fact drastically limits the size of the tree of steps, and makes this method unsuitable for reactive walk.

Yet, reactive walk is a major requirement for humanoid robots, because it is needed in any potentially changing environment, and *a fortiori* in any task involving cooperation with humans. As we mentionned in the introduction, the current solution is to only allow a small set of steps for the robot: in that case the generation and verification phases are useless since all the trajectories can be memorized and verified offline [Kuffner et al., 2001; Chestnutt et al., 2003], but with this method the lack of flexibility of the robot motions is usually a major drawback (in [Chestnutt et al., 2007] local adjustments are used to cope with this issue).

Instead of limiting the possible steps, in this chapter we base our work on the following remark: even if a humanoid robot has a great number of degrees of freedom, the set of all possible steps on a flat ground has 6 dimensions (if the steps can be entirely defined by the initial and final relative configurations of feet). So, if we add some restrictions ensuring that to one step definition corresponds a unique trajectory, then the number of parameters used to describe a trajectory could be small enough to

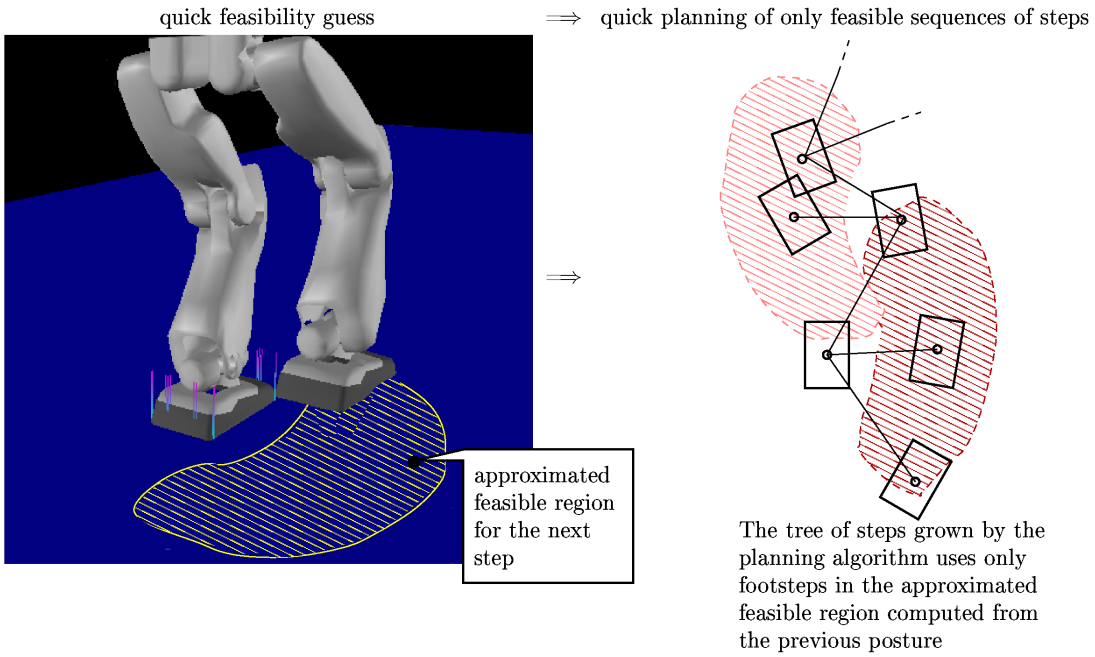


Fig. 3.1: From quick feasibility guess to fast and safe footsteps planning.

make machine learning or approximation techniques of practical interest. With such restrictions, we attempt to approximate feasibility regions (in the space of all possible steps) through extensive offline computations.

Should a satisfying approximation be reached, the robot would be able to use it online to correctly and quickly guess whether a given step is feasible or not. It would cut down the time consumption of that phase, and let the planning algorithm grow online a large tree of feasible steps in the blink of an eye, as it can be seen on Fig. 3.1. Hence, it would enable reactive walk.

In this chapter, we present an original approximation algorithm (Section 3.3) aimed at being particularly efficient in our specific context. We show how we used it and obtained an approximation that helped the robot HRP-2 to perform experiments where reactivity is constantly needed. The same approximation algorithm will be used to build swept volume approximations in Chapter 5.

3.1 Problem statement

Let us denote by $\mathbf{S}$ the state space of the robot. A state of the robot is represented by a vector $\mathbf{x} \in \mathbf{S}$ containing the configuration of the robot together with the first and second derivatives:

$$\mathbf{x} = (\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) \tag{3.1}$$

Let γ be a trajectory in the state space defined over an interval $[0, T]$, and corresponding to one step of the robot.

In various applications, the trajectories are constrained by some real valued functions defined over the state space. In our work, we took into account four specific constraints:

1. The distance to self-collision: $SC(\mathbf{x})$. This is the minimum distance between two bodies of the robot that could collide.

Since the geometry of the robot is not taken into account during the creation of the trajectory, we need to verify that no self-collision occurs along γ. Thanks to V-clip [Mirtich, 1998], we dispose of a very efficient algorithm to calculate the distances between the robot bodies along the trajectory. We keep the minimum of these values and add a margin (σ_{SC}) to take into account the possible tracking errors of the motor PID controllers or the modifications that can be introduced by the stabilization control law (we use a stabilizer to control the dynamic balance of the robot).

2. The distance to joint limits: $JL(\mathbf{x})$.

   The joint values of a humanoid robot are limited into ranges that are not taken into account during trajectory generation. So, we check along the trajectory the distance to limits for each joint (negative when beyond the limit), and keep the minimal value; again a margin (σ_{JL}) is added.

3. The Zero Moment Point (ZMP) deviation: $ZD(\mathbf{x})$. This is the distance between the ZMP reference used by the pattern generator and the ZMP corresponding to the trajectory actually produced. The ZMP position depends on $\mathbf{q}$ and its second derivative.

   Most of the current real-time dynamic walking pattern generators are based upon the linear inverted pendulum model [Kajita et al., 2003], and use as main criterion for balance the belonging of the ZMP to the polygon of support (we will explain more details about this in Chapter 4). These algorithms first generate a reference trajectory for the ZMP and then for the Center of Mass (CoM). Finally the configuration space trajectory is generated so that to realize the CoM reference (this can be done for example through inverse kinematics). However, due to the simplified equations of the linear inverted pendulum model, a precise tracking of the CoM reference does not necessarily correspond to a precise tracking of the ZMP reference. Once the whole configuration space trajectory has been generated, we can compute the ZMP trajectory for the robot multibody model and verify that it stays close to the ZMP reference. We set a threshold σ_{ZD} to decide whether the maximum deviation of the multibody ZMP[1] is acceptable or not.

4. The variance of the multibody ZMP:

$$\frac{1}{T}\int_0^T \big(ZD(\gamma(t))\big)^2 dt \tag{3.2}$$

   Through experiments, we noticed that in some cases if the ZMP goes out of the polygon of support for a very short time and with a relatively small amplitude, the robot might not fall. That's why we took a relatively high value for σ_{ZD}, and instead decided to strongly reject the cases where the multibody ZMP is regularly far from the reference along the trajectory. Thus we take into account the variance of the multibody ZMP relatively to the ZMP reference, and set a maximum value σ_{VZ} which ensures that the multibody ZMP stays mostly inside the polygon of support.

[1] Note that here the term ZMP is an abuse of language since the motion is not necessarily feasible; one of the terms virtual ZMP, Fictitious ZMP or Foot-Rotation Indicator should be used instead (see [Vukobratovic and Borovac, 2004], [Goswami, 1999]). However this slight ambiguity is not a major issue in this thesis, and we will simply keep using the term ZMP.

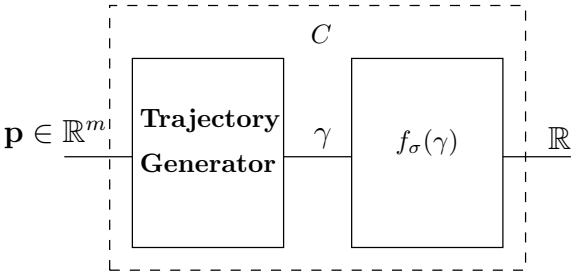


Fig. 3.2: The mapping to approximate. The Trajectory Generator takes in input a vector of parameters defining a step, and must return a *unique* trajectory. The value returned by the mapping is positive if the trajectory is acceptable (feasible), negative otherwise.

We desire to characterize the actual feasibility of a given trajectory with a unique real number, so finally, the formula we use returns the minimum of the four constraints considered:

$$\begin{aligned} f_\sigma(\gamma) = \min \Big(& -\sigma_{SC} + \min_{t\in[0,T]} \big(SC(\gamma(t))\big), \\ & -\sigma_{JL} + \min_{t\in[0,T]} \big(JL(\gamma(t)\big), \\ & \sigma_{ZD} - \max_{t\in[0,T]} \big(ZD(\gamma(t)\big), \\ & \sigma_{VZ} - \frac{1}{T}\int_{t=0}^{T} \big(ZD(\gamma(t))\big)^2 dt \Big), \end{aligned} \tag{3.3}$$

where σ is the quadruple $(\sigma_{SC}, \sigma_{JL}, \sigma_{ZD}, \sigma_{VZ}) \in \mathbb{R}^4$. The way we obtained the four margins σ_{SC}, σ_{JL}, σ_{ZD} and σ_{VZ} is empirical. We chose a margin of 2.5 centimeters for the self-collisions, 3 degrees for the joint limits, and, as for the margins related to the ZMP trajectories, we calibrated them through tests.

In the end, the desired property is the following: if the result $f_\sigma(\gamma)$, which depends on the minimum distance to self-collisions and joint limits along the trajectory, the maximum of the ZMP deviation, as well as its "variance", is positive, then the trajectory γ (i.e. the step) should be feasible on the real robot. The margins σ_{SC}, σ_{JL}, σ_{ZD} and σ_{VZ} are chosen so that so converse property is also not far from being true: if $f_\sigma(\gamma) < 0$, then with high probability the trajectory is not feasible on the real robot.

It is straightforward to approach $f_\sigma(\gamma)$ when the trajectory is discretized, but it is quite time consuming for it requires a high frequency sampling. What we would like to do is to build offline an approximation helping us to guess the value and more importantly the sign of $f_\sigma(\gamma)$ without even having to generate the trajectory.

Since we can only approximate functions from $\mathbb{R}^m$ to $\mathbb{R}$, we need to be able to produce trajectories from vectors of parameters in $\mathbb{R}^m$. This would give us a mapping C as shown on Fig. 3.2, which can entirely be computed and approximated offline.

We show in section 3.4 how we describe steps with a vector of $\mathbb{R}^m$, and how we map a vector of parameters to a unique trajectory. This will complete the definition of the mapping C. Before that, in section 3.3, we present the original approximation algorithm which will be used to approximate C. But first, we present some related work.

3.2 Related work

This idea of precomputing robot dependent data structures has been exploited in path planning for multibody robots in the past [Leven and Hutchinson, 2000; Kallmann and Mataric, 2004; Nakhaei and Lamiraux, 2008]. In these papers a roadmap is computed for a multibody robot without obstacles. Once the robot is placed in an environment with obstacles, the precomputed roadmap is pruned by removing edges in collision with the obstacles. The remaining roadmap is then used to plan paths.

In robot motor control, Bayesian and kernel-based methods have been studied extensively for value function approximation or locally weighted regressions. For instance in [Ting et al., 2008], a Bayesian formulation of spatially local adaptive kernels for locally weighted regression is proposed to estimate the Jacobian of the SARCOS anthropomorphic arm. In [Sugiyama et al., 2007], a method for value function approximation based on "geodesic Gaussian kernels" is proposed and evaluated in the contexts of robot arm control and robot navigation.

Closer to our application, in [Takubo et al., 2007] a 2-dimensional map is built which returns the time necessary to change the length of a step of HRP-2 in order to realize an emergency stop. The keypoint of this work is to build a map which verifies that the ZMP realized by the robot stays in the polygon of support for a given step-length modification done at a given time while walking. Indeed walking pattern generators such as the ones proposed by Kajita et al. [Kajita et al., 2002] or Morisawa [Morisawa et al., 2007] do not guarantee that the robot ZMP stays in the polygon of support. The main difference between this work and our approach is that we consider more constraints, and propose an adaptive partition of the input space well suited for higher dimensions. Indeed our work, taking into account free steps (in [Takubo et al., 2007] only forward walking is considered), aims at dealing with higher dimensions.

Concerning our approximation algorithm, we focused on techniques that reduce the required number of samples. The main specificity of our problem is that we are only interested in the sign of the function to approximate. Therefore, we naturally chose to use a method for which the sampling is adaptive and focuses on the regions where the mapping changes its sign. We adapted the concept of Recursive Stratified Sampling, which has been extensively used for Monte Carlo integration and image reconstruction (see [Schürer, 2002], [Hachisuka et al., 2008] and section 7.8 of [Press et al., 1992]). Locally, we approximate by using a classical Quadratic Programming problem minimizing under some constraints a distance between a linear combination of basis functions and the samples. Similar optimization problems are used for basic regression techniques (see the introduction of [Smola and Schölkopf, 2004]).

We also use in our algorithm two techniques (one global, and one local) of approached Farthest Point Sampling, a method which has been shown to lead to high data acquisition rates (see [Moenning and Dodgson, 2003] for recent developments on Farthest Point Sampling).

3.3 Mapping approximation

The approximation algorithm presented in this section is also described and studied in [Perrin et al., 2010a].

Let C be a mapping from a bounded hyper-rectangle $B_0 \subset \mathbb{R}^m$ to $\mathbb{R}$ (bounded hyper-rectangles will be called *boxes* from now on). We suppose that the zero-level

surface $\{\mathbf{x} \in B_0 | C(\mathbf{x}) = 0\}$ (the "frontier") is of Lebesgue measure zero. We also assume that C is continuous, although our approximation algorithm works well in practice if C is only piecewise continuous.

Our goal is to approximate C through sampling (since we know nothing about C we can see this task as nonparametric learning), focusing on the correctness of the sign of the result, and reducing as much as possible the number of samples needed for a satisfying approximation (because evaluating one sample is quite time consuming). These objectives come from the specificities of our problem, but the algorithm we present can be used in various cases. Typically, C can be an important function that needs to be frequently called by a real-time application, with decisions made depending on the sign of the results returned by C. Yet, if C is originally implemented with a quite time consuming algorithm, the computation cost makes it unsuitable for real-time applications, and therefore it would be interesting to approximate the sign of $C(\mathbf{x})$ offline in order to then be able to quickly guess it online, without having to actually go through any lengthy computation. This is the kind of context in which the algorithm presented here is most useful. This context corresponds to our problem of approximation of the mapping defined in Section 3.1, but as we will see later, the algorithm can also be used in order to build implicit swept volume approximations that can be evaluated very fast (in that case the result obtained is a bit similar to an Adaptive Distance Field, see [Frisken et al., 2000]). Indeed, if collisions have to be checked over and over for the same trajectory of an object in different environments, it might be very useful to approximate the sign of the corresponding distance field (i.e. the distance to the volume swept by the object along the trajectory) minus a fixed margin. If the result of the approximation is then stored in a very efficient data structure, a lot of time can be subsequently saved during collision checks. Fig. 3.3 shows the corresponding mapping C for a trajectory followed by a cube; it also displays the result of our approximation algorithm on this case example.

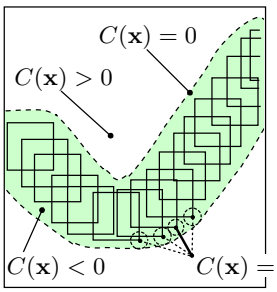
Back to the general case, the goal of the algorithm is to learn the sign of C through adaptive sampling. The algorithm is articulated around three principles:

1. Recursive Stratified Sampling: the initial input space B_0 is recursively partitioned into small boxes (a tree structure keeps the trace of all splittings; the current partition is formed by the "leaf-boxes"). Basically, a box will be split when the local approximation process fails, which can happen only when both positive and negative samples are contained in the box. Thus, the boxes will be split a lot and will become small especially near the frontier, i.e. the zone where C changes its sign. This property will enable an effective implementation of adaptive sampling focusing on the frontier.

2. Farthest Point-like Sampling in two different scales:

   - While selecting a leaf-box for sampling: we roughly estimate for each leaf-box our "confidence in the sign" of the local approximation, and select among boxes of lowest confidence.
   - While sampling inside a leaf-box: since there will be a limited number of samples inside a leaf-box, we can use a naive technique for an approached farthest point-like sampling.

3. Solving small Quadratic Programming problems (i.e. optimizing quadratic functions of several variables that are subject to linear constraints) to obtain local approximations with correct sign on the training data.



On the left, a 2D representation of a cube moving along a discretized trajectory. We denote its successive configurations by c_0, c_1, ..., c_q. For a point $\mathbf{x}$ of the Euclidean space, $C(\mathbf{x})$ is defined as the minimum distance from $\mathbf{x}$ to any configuration of the cube, minus a fixed margin τ. The margin is important to avoid errors due to the discretization, and besides, it makes the level set $\{\mathbf{x} \in \mathbb{R}^3 \mid C(\mathbf{x}) = 0\}$ smoother, thus easier to approximate.

The 2D plot on the bottom-left shows how the approximation algorithm recursively divides the Euclidean space into small boxes in order to adaptively approximate the surface $C(\mathbf{x}) = 0$. The approximated surface defines an approximation of the volume swept by the cube. A view of this swept volume approximation is displayed on the 3D plot on the bottom right.

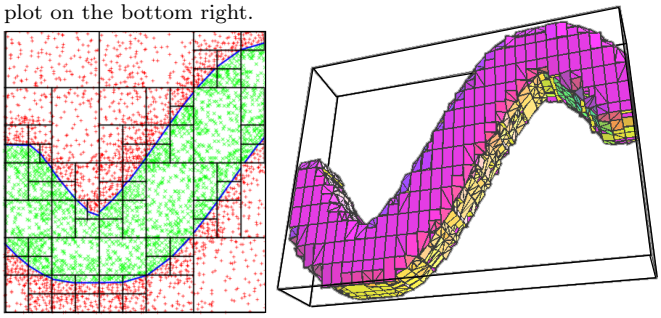


Fig. 3.3: Case example: implicit approximation of the volume swept by a cube.

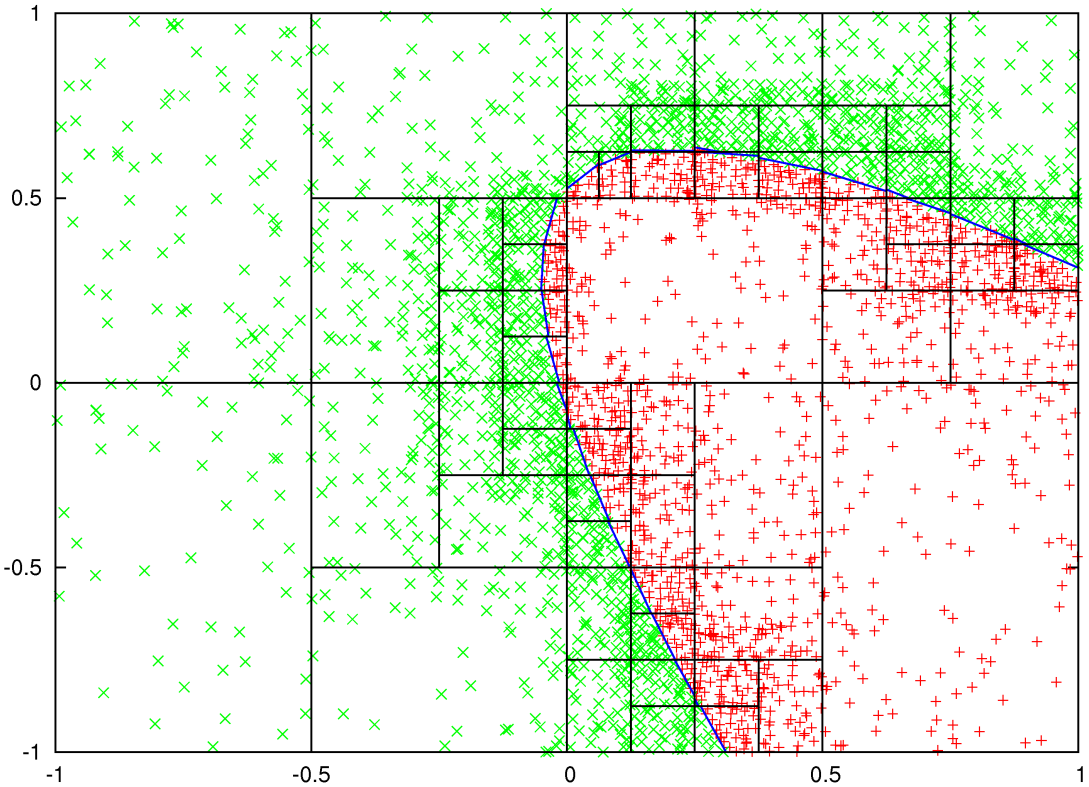


Fig. 3.4: An example run of our approximation algorithm on a continuous function from $\mathbb{R}^2$ to $\mathbb{R}$.

Figure 3.4 shows the result of our approximation algorithm on a function from $[-1,1]^2$ to $\mathbb{R}$. We can see that the boxes are split adaptively, and that it leads to an adaptive sampling focusing on the region where the function changes its sign. Let us call f the result of the approximation. f is constantly updated during the execution of the algorithm, and like C, it is a function from $B_0 \subset \mathbb{R}^m$ to $\mathbb{R}$. We denote by $f_{|B}$ the restriction of f to a box B contained in B_0. Below, Algorithm 1 describes the algorithm structure, and leaves just four questions, which we answer respectively in the four next sections: "how to pick a new leaf-box?", "how to get new samples?", "how to split the boxes?", and "how to locally approximate?".

Algorithm 1 The algorithm structure.

$f \leftarrow$ the constant function 0.
$Set\_of\_Leaves \leftarrow \{B_0\}$
while (1) **do**
  Pick (using Algorithm 2) a leaf-box B out of Set_of_Leaves (and remove it from Set_of_Leaves).
  Get new samples in B.
  if there are only positive samples in B **then**
    Define $f_{|B}$ as any positive constant on B, for example choosing the minimum value of C on the samples in B.
  else if there are only negative samples in B **then**
    Define $f_{|B}$ as any negative constant on B, for example choosing the maximum value of C on the samples in B.
  else
    $Temporary\_Stack \leftarrow \{B\}$
    while $Temporary\_Stack \neq \emptyset$ **do**
      Pop a leaf-box $B_{current}$ out of $Temporary\_Stack$
      Try to locally approximate C on $B_{current}$, i.e. try to define a new value of $f_{|B_{current}}$, the main constraint being that $f_{|B_{current}}$ must have a correct sign on all the samples in $B_{current}$.
      if this attempt does not succeed **then**
        Split $B_{current}$ into two son boxes of equal dimensions: B_{left} and B_{right}, and push them both in $Temporary\_Stack$.
      else
        Put $B_{current}$ in Set_of_Leaves.
      end if
    end while
  end if
end while

3.3.1 How to pick a new leaf-box

It is very important to find a good method to pick new leaf-boxes, because this choice is the core of the adaptiveness of the sampling. We want to focus on the boxes where we know that C changes its sign, but without forgetting to sample on other boxes where a change of sign might still have to be discovered. It is very difficult to decide at which rate one should sample near the frontier and at which rate one should sample in regions where only positive (or only negative) samples have been seen so far.

In the first version of our algorithm (see [Perrin et al., 2010b]), the mapping C was assumed to be $K-$Lipschitz for some $K \in \mathbb{R}$, and a formula for the "confidence in the sign" of a local approximation was defined, with the aim of picking only leaf-boxes with lowest confidence. This formula works well in general, but causes the algorithm to fail to converge with some particular mappings C. In the variant presented here, we use a different heuristic which this time comes with a proof of convergence (see section 3.3.5).

The set of leaf-boxes – let us call it Set_of_Leaves – is divided into three disjoint subsets: the set of positive leaf-boxes (containing only samples on which C is positive), the set of negative leaf-boxes (containing only samples on which C is negative), and the set of "frontier" leaf-boxes (containing either no sample or both positive and negative samples). We denote these subsets by $Set\_of\_PositiveLeaves$, $Set\_of\_NegativeLeaves$, and $Set\_of\_FrontierLeaves$. We also denote by max_value the maximum absolute value of C seen on the samples collected so far. For a leaf-box B containing k samples $(\mathbf{s}_1, C(\mathbf{s}_1)), \ldots, (\mathbf{s}_k, C(\mathbf{s}_k))$, with $k \geq 1$, we define $co(B)$, which is inspired by the value $conf(B)$ used in [Perrin et al., 2010b] (the "confidence in the sign"), but leads to better convergence properties. The definition is divided into two cases:

1. If B is a leaf-box with at least one sample on it, $co(B)$ is defined as follows (the measure of Lebesgue, or volume of B, denoted by volume(B), is simply the product of its m lengths):

$$co(B) = \frac{k}{\text{volume}(B)} + \frac{1}{max\_value} \times \frac{\sum_{i=1}^{k} |C(\mathbf{s}_i)|}{\text{volume}(B)} \tag{3.4}$$

2. If B is a leaf-box with no sample, it has necessarily just been created, and in that case either B is the input space, B_0, and we pose $co(B) = (\text{volume}(B_0))^{-1}$, or B has been created by splitting its parent box B', and we pose:

$$co(B) = \frac{co(B')}{2}, \tag{3.5}$$

   where $co(B')$ is the value just before the split.

The important properties of $co(B)$ are that first, it increases when the density of samples in the leaf-box (i.e. $\frac{k}{\text{volume}(B)}$) increases, and second, $co(B)$ has a component which depends on the values of C on the samples: this helps to drive the sampling towards regions where C takes relatively small values. Indeed, considering only the density of samples is not enough: if B_1 is a box containing 10 samples on which the value of C is about 5, and B_2 a box of same size with 9 samples on which the value of C is about 1, even though the density of samples is higher in B_1, it seems natural to sample first in B_2, because intuitively C has more chances to change its sign in B_2. $co(B)$ gives us an arbitrary rule to know when to stop sampling on B_2 and start sampling on B_1. The third interesting property comes from the use of max_value and is that if we simply multiply C by a non-zero constant, the behaviour of the sampling would remain unchanged: without randomness it would lead exactly to the same samples and the same leaf-boxes. Finally, one of the main advantages of this definition of $co(B)$ is that it is extremely easy to compute.

Algorithm 2 shows how $co(B)$ is used to pick a new leaf-box. We see that by default, this algorithm gives the same importance to the two following tasks:

Algorithm 2 The default algorithm for picking a new leaf-box.

Flip a coin.
if the flip is heads AND $Set\_of\_FrontierLeaves$ is not empty **then**
  Choose a leaf-box $B_{new} \in Set\_of\_FrontierLeaves$ such that $co(B_{new}) = \min\{co(B)|B \in Set\_of\_FrontierLeaves\}$.
else
  Flip another coin.
  if the flip is heads AND $Set\_of\_PositiveLeaves$ is not empty **then**
    Choose a leaf-box $B_{new} \in Set\_of\_PositiveLeaves$ such that $co(B_{new}) = \min\{co(B)|B \in Set\_of\_PositiveLeaves\}$.
  else
    Choose a leaf-box $B_{new} \in Set\_of\_NegativeLeaves$ such that $co(B_{new}) = \min\{co(B)|B \in Set\_of\_NegativeLeaves\}$.
  end if
end if

1. Select a leaf-box among $Set\_of\_FrontierLeaves$, i.e. try to make the approximation more precise on the known parts of the frontier of C.
2. Select a leaf-box among $Set\_of\_PositiveLeaves \cup Set\_of\_NegativeLeaves$, i.e. try to discover new parts of the frontier of C.

Two remarks:

- In some cases, it is better to customize the Algorithm 2. For example, when C is a distance minus a small threshold, the positive and negative regions are not symmetric and should not be treated so. C will mainly be equal to $-\tau$ (τ being the threshold) on the negative region, which gives few information; thus in that case it seems appropriate to focus more on the positive region than the negative one. Another example is when the frontier is known to have a simple shape: in that case it is arguably better to pick "frontier" leaf-boxes with a probability greater than 50%, thus boosting the focus on the frontier.
- We call "task 1)" selecting a leaf-box in $Set\_of\_FrontierLeaves$, and we call "task 2)" selecting a leaf-box in $Set\_of\_PositiveLeaves \cup Set\_of\_NegativeLeaves$. Looking for a good tradeoff between tasks 1) and 2) is very similar to some issues raised by the so-called multi-armed bandit problem, where a gambler is using a slot machine with multiple levers. When pulled, each lever provides a reward drawn from a distribution associated with that specific lever. The objective of the gambler is to maximize the sum of rewards earned through a sequence of lever pulls. The crucial tradeoff the gambler faces at each trial is between "exploitation" of the lever that has the highest expected payoff and "exploration" to get more information about the expected payoffs of the other levers. In our problem, the task 1) can be seen as the "exploitation", and the task 2) as the "exploration". This similarity could be the starting point of a more theoretical foundation for the Algorithm 2, and indeed a few examples of adaptive sampling techniques based on Bayesian or frequentist analyses of the bandit problem exist in the litterature, such as in [Hardwick and Stout, 1998]. Yet in our case the payoff is not very clear, or at least not known in advance (it should quantify the improvement of the frontier approximation), and furthermore one of our priorities, computational efficiency, limits the range of possible methods.

3.3.2 How to get new samples

Let us assume that new samples must be chosen inside the leaf-box B (see line 5 of Algorithm 1).

In the default version of the algorithm, N new samples are simply chosen independently and uniformly inside B, N being an integer fixed in advance.

In a more elaborate version, we can first pick more than N samples (e.g. $10N$), and then select a group of samples that are far from each other and far from the samples already in B. This version can be chosen if the evaluation of C is much longer than the computation needed to obtain such samples.

In an even more elaborate version, we allow the samples to be drawn also a bit outside of B. This technique has already been efficiently used in several adaptive algorithms, such as the one presented in [Hachisuka et al., 2008]; it allows samples to spill into neighboring boxes and enables the adaptive sampling method to "crawl" along the frontier of C.

3.3.3 How to split the boxes

We simply split the boxes cyclically. We give an order to the dimensions: *dim.* 1, *dim.* 2,..., *dim.* m. The initial box B_0 is split in half orthogonally to *dim.* 1, and its children boxes are split orthogonally to *dim.* 2, then *dim.* 3 for the grandchildren boxes, etc. If a box is split orthogonally to *dim.* m, its children boxes will be split orthogonally to *dim.* 1, and so on...

If the input space has 2 or 3 dimensions, specific implementations with quadtrees or octrees can be advantageously used.

3.3.4 How to locally approximate

Here is the context: a leaf-box $B_{current}$ contains a finite number of samples, and we want to find a local approximation which has a correct sign on these samples, or return "fail".

The three first cases are simple: if there is no sample in $B_{current}$, we don't modify $f_{|B_{current}}$; if there are only positive samples in $B_{current}$, we define $f_{|B_{current}}$ as any positive constant on $B_{current}$ (e.g. the minimum value of C on the samples in $B_{current}$); if there are only negative samples in $B_{current}$, we define $f_{|B_{current}}$ as any negative constant on $B_{current}$ (e.g. the maximum value of C on the samples in $B_{current}$).

In the last possible case, $B_{current}$ contains both positive and negative samples. We use another parameter $N_{max} > N$ chosen in advance, and if the box contains more than N_{max} samples, we return "fail" (and the box will be split, which will lead to two smaller boxes with hopefully less samples, and local approximations will be attempted again on each of them). This parameter N_{max} should be chosen so that only a small portion of the computation time is spent on local approximations (whose complexity depends on the number of samples); most of the computation time should be spent in evaluations of C.

We present a simple technique for local approximation based on Quadratic Programming (see the introduction of [Smola and Schölkopf, 2004] for details on regression techniques of that sort). The convergence (Theorem 3.3.1) of the algorithm does not depend on local approximations, but good local approximations will speed up the convergence rate.

Let us call $\{(\mathbf{s}_1, C(\mathbf{s}_1)), \ldots, (\mathbf{s}_l, C(\mathbf{s}_l))\}$ the samples in $B_{current}$.

In the technique presented here, $f_{|B_{current}}$ is being searched among the elements of a finite dimension vector space chosen by the user (this vector space must contain constant functions). Let us describe the basic case of the affine functions which take the following form:

$$g(\mathbf{x}) = g(x_1, x_2, \ldots, x_m) = \langle w, (x_1, x_2, \ldots, x_m, 1) \rangle, \tag{3.6}$$

with $w \in \mathbb{R}^{m+1}$ and where $\langle \cdot, \cdot \rangle$ denotes the dot product on $\mathbb{R}^{m+1}$. We solve the following Quadratic Programming problem over the real numbers:

$$\begin{aligned} &\text{minimize} \sum_i (g(\mathbf{s}_i) - C(\mathbf{s}_i))^2 = \langle w, \mathbf{M}w \rangle + \langle d, w \rangle + \sum_i C(\mathbf{s}_i)^2 \\ &\text{subject to} \begin{cases} \forall i \mid C(\mathbf{s}_i) > 0, & g(\mathbf{s}_i) > 0 \\ \forall i \mid C(\mathbf{s}_i) \leq 0, & g(\mathbf{s}_i) < 0 \end{cases} \end{aligned} \tag{3.7}$$

where $\mathbf{M}$ is a symmetric positive $(m+1) \times (m+1)$ matrix, and $d \in \mathbb{R}^{m+1}$.

The solution g found defines the new value for $f_{|B_{current}}$. If no solution is found, "fail" is returned.

Our implementation uses the solver QL [Schittkowski, 2005].

3.3.5 Convergence result

Each leaf-box than can appear during the execution of the algorithm can be defined by a finite list of booleans coding a path in the infinite tree. Therefore there is a bijection between theses boxes and $\mathbb{N}$, and we can denote them by $B^{(1)}, B^{(2)}, B^{(3)}, \ldots$

The execution of the algorithm is entirely determined by an infinite sequence of independent coin flips randomly drawn, and for each box $B^{(i)}$, an infinite sequence of independent samples uniformly drawn in it (when new samples are needed in a box B, we just follow the sequence of samples corresponding to this box). Let us call $\mathcal{R}_{flips}$ the sequence of coin flips, and $\mathcal{R}^{(i)}_{samples}$ the sequence of samples in the box $B^{(i)}$. Let us denote by $\mathcal{E}(\mathcal{R}_{flips}, (\mathcal{R}^{(i)}_{samples})_{i\in\mathbb{N}})$ the execution of the algorithm with random sequences $\mathcal{R}_{flips}$ and $(\mathcal{R}^{(i)}_{samples})_{i\in\mathbb{N}}$, and let us also denote by f_i the approximation obtained after i iterations of the while loop (line 3 of Algorithm 1).

With probability 1, both of the two following properties are verified (a key argument to prove this is that any countable union of sets of Lebesgue measure 0 has Lebesgue measure 0):

1. In the sequence of coin flips there are infinitely many "tails" and infinitely many "heads", and what's more, any finite sequence appears infinitely many times (e.g. "tails-heads", or "heads-heads-tails", etc.).

2. For each box $B^{(i)}$, $\mathcal{R}^{(i)}_{samples}$ contains infinitely many points in any non-empty open subset of $B^{(i)}$.

Under these assumptions, we have the following:

Theorem 3.3.1. *For all compact subsets $X \subset B_0$ such that $\forall \mathbf{x} \in X,\ C(\mathbf{x}) \neq 0$, there exists $n_0 \in \mathbb{N}$ such that:*

$$\forall n > n_0,\ \forall \mathbf{x} \in X,\ \begin{cases} C(\mathbf{x}) > 0 \Rightarrow f_n(\mathbf{x}) > 0 \\ C(\mathbf{x}) < 0 \Rightarrow f_n(\mathbf{x}) < 0 \end{cases}$$

Since the frontier is assumed to be of Lebesgue measure zero, we have the immediate corollary:

Corollary 3.3.1. *The Lebesgue measure of Err_n, the set of points of B_0 on which f_n does not have a correct sign, tends to zero when n tends to $+\infty$.*

We will demonstrate Theorem 3.3.1 with the default version of the algorithm, the one where groups of samples are simply drawn uniformly, but with very small changes in the proof we obtain the same theorem with the algorithm variants. Before proving the theorem, we demonstrate three preliminary lemmas:

Lemma 3.3.1. *For any leaf-box B, we always have:*

$$co(B) \geq \frac{k+1}{4(\text{volume}(B))},$$

where k is the number of samples in B.

Proof. It is true for the initial value of $co(B_0)$, and when the equation (3.4) is used, we have $co(B) \geq \frac{k}{(\text{volume}(B'))}$, with $k \geq \frac{k+1}{4}$.

When the equation (3.5) is applied, it means that B contains no sample ($k = 0$), but the parent box B' necessarily contained samples (otherwise it wouldn't have been split), so we had $co(B') \geq \frac{1}{(\text{volume}(B'))}$ (by eq. (3.4)). Since the volume of B is half the volume of B', we obtain indeed $co(B) \geq \frac{0+1}{4(\text{volume}(B))}$. □

Lemma 3.3.2. *Let B be a leaf-box after iteration i of Algorithm 1.*
There exists $j > i$ such that B will be selected by Algorithm 2 at iteration j.

Proof. We prove it by contradiction. Without loss of generality we assume that B belongs to $Set\_of\_PositiveLeaves$; the reasoning would be the same with $B \in Set\_of\_NegativeLeaves$ or $B \in Set\_of\_FrontierLeaves$.

Assuming that B is never to be selected, the value $co(B)$ will never change.

Because of Lemma 3.3.1, the number of boxes $B^{(i)}$ which could verify $co(B_i) \leq co(B)$ if they were leaf-boxes is finite (for the boxes $B^{(i)}$ that are too small, namely such that $co(B^{(i)}) < (4(\text{volume}(B_0)))^{-1}$, this inequality cannot be verified). Let us call them $B^{(i_1)}, B^{(i_2)}, \ldots, B^{(i_K)}$.

Thanks to the assumption made on the sequence of coin flips, it can be shown that $Set\_of\_PositiveLeaves$ will be selected infinitely many times by Algorithm 2. Each time, some samples will be put in one leaf-box $B^{(i)}$ which belongs to the set $\{B^{(i_1)}, B^{(i_2)}, \ldots, B^{(i_K)}\}$, and must verify $co(B_i) \leq co(B)$. At least one of the boxes $B^{(i_j)}$ must be chosen an infinite number of times, but because of the inequality $co(B^{(i_j)}) \geq \frac{1+\text{number of samples in } B^{(i_j)}}{4\times\text{volume}(B^{(i_j)})}$ the value $co(B^{(i_j)})$ will eventually exceed $co(B)$, and it will become impossible to choose $B^{(i_j)}$: contradiction. We can thus conclude that the box B will be selected again. □

Lemma 3.3.3. *Let B be a leaf-box that will eventually be split. Then the intersection between B and the frontier is not empty.*

Proof. If the intersection was empty, since C is continuous, it would either be negative on all points of B, or positive on all points of B. In that case $f_{|B}$ would always be defined as a constant on B, and B would never be split. □

Now we can prove the convergence theorem:

Proof of Theorem 3.3.1. Let X be a compact subset of B_0 such that $\forall \mathbf{x} \in X,\ C(\mathbf{x}) \neq 0$, and let us call $d > 0$ the distance between X and the frontier. We call $X_{d/2} \supset X$ the open set of points $x \in B_0$ such that the distance between x and the frontier is greater than $d/2$. Thanks to Lemma 3.3.3 we know that after some iteration, all the leaf-boxes that have a non-empty intersection with $X_{d/2}$ will never be split, because in the opposite case, as a consequence of König's lemma the minimum size of the boxes that will be split and have a non-empty intersection with $X_{d/2}$ should endlessly decrease and tend to zero, which is impossible because since the distance between $X_{d/2}$ and the frontier is $d/2 > 0$, no box too small can at the same time contain points of $X_{d/2}$ and points of the frontier.

So there exists $n_0 \in \mathbb{N}^+$ such that after iteration n_0, all the leaf-boxes having a non-empty intersection with $X_{d/2}$ are fixed. Let us consider one such leaf-box $B^{(i)}$. Thanks to lemma 3.3.2, we know that B will be picked an infinite number of times, and as a result will receive an infinite number of samples. Besides, $B^{(i)}$ will never belong to $Set\_of\_FrontierLeaves$, because otherwise it would be eventually split (see section 3.3.4: with more than N_{max} samples, leaf-boxes of $Set\_of\_FrontierLeaves$ are automatically split). As a consequence, $f_{|B^{(i)}}$ will always have the same constant sign. Moreover, since $X_{d/2}$ is an open set and $B^{(i)}$ the closure of an open set, the intersection between $B^{(i)}$ and $X_{d/2}$ contains an open subset, and since $\mathcal{R}^{(i)}_{samples}$ contains infinitely many points in any non-empty open subset of $B^{(i)}$, we know that a sample of $X_{d/2}$ will eventually be drawn. It follows that after iteration n_0 the sign of $f_{|B^{(i)}}$ is equal to the sign of C on some point of $X_{d/2} \cap B^{(i)}$. But since C is continuous, it can only have a constant sign on $X_{d/2} \cap B^{(i)}$; otherwise we could easily obtain a contradiction by considering two open subsets of $B^{(i)}$, one on which C is positive, and one on which C is negative. Since we know that samples will eventually be drawn in these two open subsets, $B^{(i)}$ would be transfered to $Set\_of\_FrontierLeaves$, which as we have proved cannot happen.

Hence, the sign of $f_{|B^{(i)}}$ is equal to the sign of C on *every* point of $X_{d/2} \cap B^{(i)}$. By extending this result to all the leaf-boxes having a non-empty intersection with $X_{d/2}$, we deduce that after iteration n_0, f has a correct sign on the entire set $X_{d/2}$, and *a fortiori* on X:

$$\forall n > n_0,\ \forall \mathbf{x} \in X,\ \begin{cases} C(\mathbf{x}) > 0 \Rightarrow f_n(\mathbf{x}) > 0 \\ C(\mathbf{x}) < 0 \Rightarrow f_n(\mathbf{x}) < 0 \end{cases}$$

□

3.4 Reducing the dimensionality of the parameter space

So, now that we presented our approximation algorithm, and showed how to get from a trajectory γ a real number whose sign characterizes the feasibility of γ, the only thing left to do (before applying the algorithm to approximate feasibility tests for the robot HRP-2) is the definition of a parameter space from which unique trajectories can be generated. It is the purpose of this section.

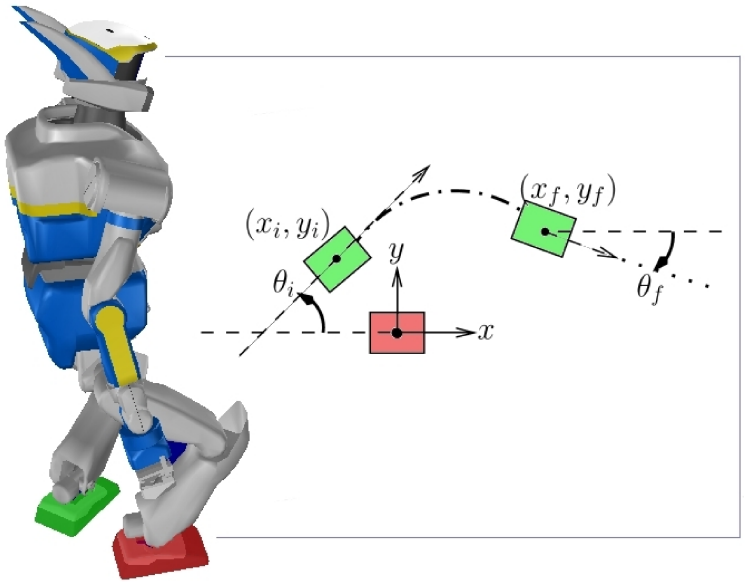


Fig. 3.5: Parameterization of a step for the humanoid robot HRP-2. The right foot defines the origin of the coordinate frame, and the left foot moves from (x_i, y_i, θ_i) to (x_f, y_f, θ_f). The robot has 36 degree of freedom. 6 for the waist position and orientation (the "free-flyer"), and 30 revolute joints.

3.4.1 Unique trajectories from 6 parameters

As shown on Fig. 3.5, 6 parameters can fully describe a step: 3 for the initial position and orientation of the swing foot (relatively to the stable foot), and 3 for its final position and orientation. Nevertheless, this geometric description usually doesn't correspond to a unique trajectory: recent walking pattern generators produce fully dynamic walks, and thus take into account the initial speed of the robot's Center of Mass (CoM), as well as the next few steps that are going to be performed (see [Kajita et al., 2003]).

Our approximation only considers isolated single steps: the initial and final speeds of the CoM are zero. This corresponds to a conservative approach, since preliminary work showed that in most cases, a non-zero initial speed only expands the feasible set of steps.

Moreover, a fixed CoM height is given, as well as the initial and final horizontal positions of the CoM during any step: namely, right in between the centers of the feet. We also set the initial and final orientations of the robot (i.e. of its waist): its initial orientation is the one of the stable foot while its final orientation corresponds to the final orientation of the swing foot.

With all these restrictions, we can obtain a unique trajectory from the 6 parameters describing a step geometry.

3.4.2 From 6 to 4 parameters

In a 6-dimensional space, 10 values in each dimension correspond to a total of 1 million samples.

Being able to treat 1 sample in about 0.4s, we were not able to obtain a satisfying approximation in a reasonable time with the computational power we dispose of.

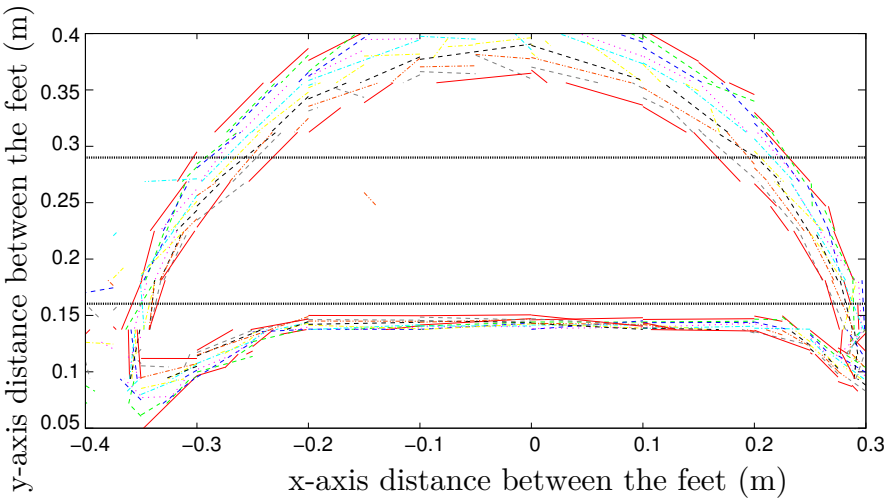


Fig. 3.6: Approximation of the feasibility region for the landing position of the left foot from a fixed initial position, for several values taken in $[-5\,^\circ, +5\,^\circ] \times [-5\,^\circ, +5\,^\circ]$ setting its initial and final orientations. The line segments of a same color define the contour (for one couple of orientations) of the set of positions on which the swing foot can land after a valid step. We can see that if the initial and final orientations of the swing foot are modified a little bit, the feasibility region is slightly changed, but we can find a safe zone inside the feasibility region so that to ignore these modifications.

Therefore, a further reduction of dimensionality was still necessary.

We noticed that if we oblige the feet to stay at a reasonable distance from each other, and if the initial and final orientations of the swing foot remain close to the support foot orientation (e.g. $\pm 5\,^\circ$), then orientations do not have much impact on the feasibility region from a given initial position.

Fig. 3.6 shows some feasibility regions with different initial and final orientations of the swing foot. The results are similar in shape, with differences when the lateral displacement is large. To avoid significant differences we set an upper bound limit (29cm) for the lateral distance between the feet of the robot to 29cm, and also add a lower bound limit at 16cm, so that the feet stay relatively far from each other. With these restrictions, we limit the orientation of the feet to the interval $[-5\,^\circ, +5\,^\circ]$ and ignore the orientation parameters when guessing the feasibility of a step. Fig. 3.6 motivates this heuristic.

3.5 Experimental results of online footstep correction

3.5.1 Analysis of the approximation

It took us 11 days with an Intel(R) Pentium(R) CPU 3.40GHz to build the complete approximation, and the total number of points sampled was 2,672,928.

The resulting function is instanciated in 9μs: it means that with the same computer we divided the verification time for the feasibility of a step by about 40,000.

We tested the approximation on 349,559 points randomly drawn according to a uniform distribution over the input space. Among them, 8,607 corresponded to steps declared feasible by our verification process, 340,952 were declared not acceptable

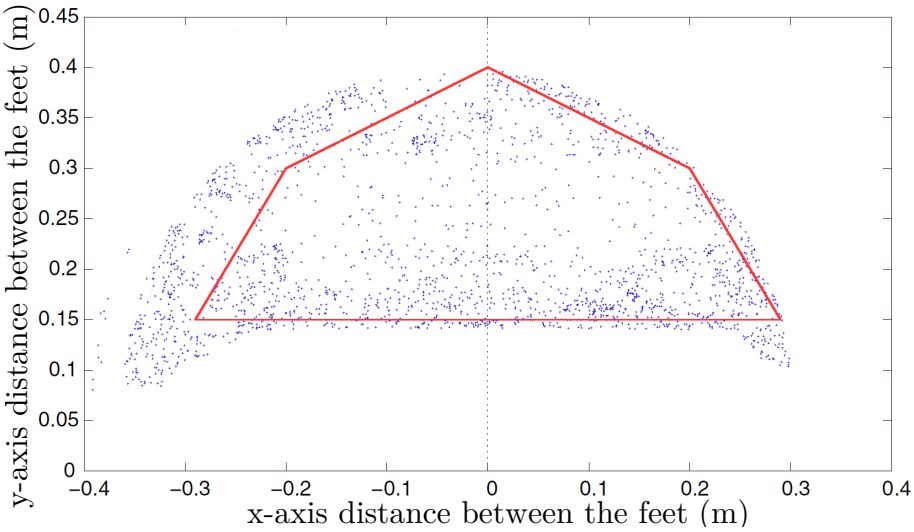


Fig. 3.7: Construction of a simple feasibility region inside a zone defined by positive samples. This region is also chosen symmetric with respect to the y-axis so that to always keep the same reachable zone no matter what the previous step was.

(which, because of the margins, does not necessarily mean that the trajectory would actually fail). On the 8,607 positive points, the approximation was positive 8,278 times. On the 340,952 negative points, the approximation was negative 340,862 times. So for these random samples, if the approximation returns a positive result, it is correct (i.e. the trajectory generated would be considered feasible by our verification process) at 98.8%. If the returned result is negative, it is correct at 99.9%.

3.5.2 Preliminary experiments

First, instead of using the approximation obtained, we constructed a simpler region of feasibility, as shown on Fig. 3.7. The advantage of such a simple region is that it can be written as a conjunction of linear constraints, and thus it is possible to use it in a Walking Pattern Generator which computes foot placements as part of an optimization problem. It was used in [Herdt et al., 2010] where a reference speed is sent to the robot which then tries to find the best foot placements and leg trajectories to follow the reference.

We also used similar simple regions to correct footsteps in an experiment where the robot is guided by the hands of the user: see Fig. 3.8 and [Perrin et al., 2009], [Stasse et al., 2009a,b].

3.5.3 Experiment: steering HRP-2 with a gamepad

We used the real approximation obtained through offline computations to correct footstep placements in the following steering experiment: through a gamepad with 2 axes the user controls simple requests of steps that are repeatedly sent to the robot HRP-2. This experiment is described in [Perrin et al., 2010b]. Each request has two components: a position of the objective footprint relatively to the support foot, and an orientation change. The mechanism of treatment of the requests of steps is described on Fig. 3.9.

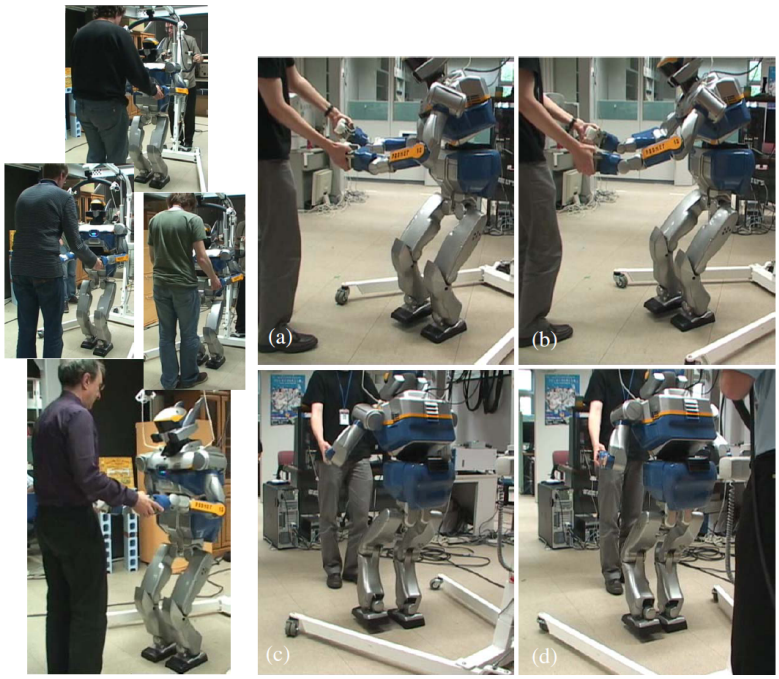


Fig. 3.8: A human-humanoid interaction experiment.

Every 20ms, the step currently set by the gamepad is sent to our approximation, which guesses whether from the current posture it will be a feasible step or not. As long as the step is not feasible, it is discarded. But when a step is declared feasible, the software saves it and gets ready to send it to the robot. When another step is guessed feasible, the previous candidate step is overwritten: therefore, when the robot is ready to perform a new step, it is always given the last feasible step.

Fig. 3.9: Steering HRP-2 with a gamepad.

With this simple approach, we have been able to repeatedly teleoperate the robot for several minutes without making it fall on the ground. the steering is very intuitive, and the user naturally avoids keeping the axes in a configuration that would correspond to unfeasible steps.

Nevertheless, in some cases, the robot, pushed to its limits, falls down. This is due to discrepancies between the model on which we tested steps, and the real conditions. First, in our experiments the waist and arms of the robot are not fully constrained by tasks, and thus don't have exactly the same behaviour as the one fixed for our offline computations. Second, our verification process of trajectories neither includes torque limits nor dynamic simulations of contact forces, and does not take into account the robot ankle compliance. Third, we use slightly different models ([Morisawa et al., 2007] and [Kajita et al., 2002]) for online pattern generation and offline tests. But the fourth and main discrepancy between the offline approximation and the real conditions is that we approximated the results of tests on *isolated* steps. However, during the

online execution, with a state-of-the-art walking pattern generator like the one we used ([Morisawa et al., 2007]), two consecutive steps are *not* independent: the trajectory of a step depends on the previous one, and even on the next one (because of the use of preview control). To avoid this discrepancy we would have to use a new parameter space where sequences of 3 steps are considered, but this would significantly increase dimensionality, and would cause an explosion of the approximation time.

3.6 Conclusion

In this chapter we presented an original approximation algorithm, and showed how, with some restrictions, we were able to reduce dimensionality and perform an offline approximation of feasibility regions for a state-of-the-art walking pattern generator. HRP-2 was then able to guess the feasibility of a step 40,000 times faster than with the normal verification process, and that helped us realize some successful experiments of reactive walk.

Chapter 4

Walking Pattern Generation

In the previous chapter, we used a state-of-the-art walking pattern generator, and tried to approximate the set of feasible steps through extensive offline computations. One of the main problems encountered was the relatively high dimensionality of the input space: indeed one step is geometrically defined by 6 parameters, and this is enough to lead to slow approximations. We added debatable restrictions to reduce the input space down to a 4-dimensional space, and finally obtained results that were successfully applied to HRP-2. Nevertheless, as we mentioned there are some unavoidable discrepancies between the steps considered during the approximation process and the steps actually performed by the robot: with a state-of-the-art walking pattern generator the connections between steps are not done at zero speed, and therefore consecutive steps are not independent. This means that for a sound result we would be obliged to increase even further the dimensionality of the input space.

In section 4.2, we introduce a new walking pattern generator based on half-steps, and show how to use it in order to obtain a 3-dimensional input space, without having to give up on the dependency between consecutive steps. This dimensionality reduction enables better offline approximations, and a smoothing process ensures the soundness of the approach: the initial trajectories correspond exactly to a concatenation of the steps that are evaluated by the approximation algorithm. This walking pattern generator is also described in [Perrin et al., 2011b] and [Perrin et al., 2011a]. In chapters 5 and 6, we will see how to use the interesting properties of this walking pattern generator in fast footstep planning frameworks.

In section 4.3, we conduct a theoretical study of the sensitivity of this new walking pattern generator. But first, we present the related work.

4.1 Related work

Our approach for walking pattern generation shares a lot of similarities with the article [Kuffner et al., 2001], which presents an algorithm for planning safe navigation strategies for biped robots moving in obstacle-cluttered environments. In [Kuffner et al., 2001], in order to reduce the number of transition trajectories between two consecu-

tive footstep placements, the authors introduce two intermediate postures Q_{right} and Q_{left} that serve as via points for all footstep transitions. Q_{right} and Q_{left} correspond to default postures in which the robot is respectively balanced entirely on the right or left foot, with the other foot raised high above the walking surface. We use the same intermediate postures as extremities for what we call half-steps: an upward half-step goes from a double support configuration to the posture Q_{left} or Q_{right}, whereas a downward half-step goes from either Q_{left} or Q_{right} to a double support configuration. This divides by 2 the dimensionality of the input space.

The goal of [Kuffner et al., 2001] is to quickly find statically stable walking motions from an initial position to a goal location. The problem with considering only statically stable motions is that it considerably reduces the set of possible steps. For example large steps can usually not be obtained with statically stable motions. Indeed, during a statically stable motion the center of mass (CoM) must always be above the polygon of support (defined as the convex hull of the set of points of the robot in contact with the walking surface). Therefore, the CoM has to shift completely from one foot to the other before the swing foot can be raised. This often causes an overstretching of the leg, while large steps can easily be produced with dynamic motions where the swing foot can be raised while the CoM is not above the polygon of support.

On the contrary our approach directly uses a low dimensional space of dynamic motions (the half-steps). Thanks to its low dimensionality, we will be able to replace tests on this space by approximation functions that are computed offline, following the method introduced in chapter 3. Although dynamic, the half-steps we consider start and finish with zero speed, so they are still statically stable at their extremities. This enables us to concatenate them freely in order to produce "raw" sequences of half-steps, but it also makes these sequences contain strong speed variations (they frequently reach zero speed) and unnecessary sway motions of the CoM. For this reason even if those raw sequences are better than statically stable motions, they are still very poor compared to trajectories generated by state-of-the-art walking pattern generators. We show that we can cope with this issue by using a very simple homotopy which continuously deforms a raw sequence of half-steps into a smoother and more dynamic sequence where the zero speed configurations have totally disappeared.

4.2 A walking pattern generator based on half-steps and a smoothing homotopy

We use a classical simplified model of the robot dynamics: the linear inverted pendulum model (see [Kajita et al., 2003]). In this model the mass of the robot is assumed to be concentrated in its CoM which is supposed to be rigidly linked to the robot waist and directly above it at all time. Besides, the robot is supposed to have only coplanar point contacts with the horizontal walking surface. An analysis of the subsequent equations leads to a further approximation which enables the decoupling of the dynamic differential equations for the x-axis and y-axis. They can be written as follows:

$$p_x = Z(x) \tag{4.1}$$

$$p_y = Z(y) \tag{4.2}$$

$$\text{with } Z \triangleq Id - \frac{z_c}{g}\frac{d}{dt^2} \tag{4.3}$$

(x, y) are the (x-axis,y-axis) coordinates of the CoM of the robot, z_c is the height of the robot center of mass which is supposed constant during the steps, and (p_x, p_y) are the (x-axis,y-axis) coordinates of the virtual Zero Moment Point (ZMP). A classical dynamic balance criterion for biped walking is that the ZMP should stay at all time inside the polygon of support (see [Vukobratovic and Borovac, 2004]). An important thing to notice in these equations is that Z is a linear operator.

In the article [Harada et al., 2006], Harada et al. show how analytical trajectories for both the CoM and the ZMP can be derived from these equations. The ZMP trajectory is a polynomial of the time variable t, and the CoM trajectory $\begin{pmatrix} x(t) \\ y(t) \end{pmatrix}$ has the general following form:

$$cosh(\sqrt{\frac{g}{z_c}} \cdot t) \begin{pmatrix} V_x \\ V_y \end{pmatrix} + sinh(\sqrt{\frac{g}{z_c}} \cdot t) \begin{pmatrix} W_x \\ W_y \end{pmatrix} + \begin{pmatrix} r_x(t) \\ r_y(t) \end{pmatrix} \tag{4.4}$$

where $r_x(t)$ and $r_y(t)$ are polynomials entirely determined by $p_x(t)$ and $p_y(t)$, respectively.

From this equation we see that for a given ZMP profile, there are just enough free parameters (V_x, V_y, W_x, W_y) to set the initial horizontal position and speed of the CoM:

$$\begin{pmatrix} x(0) \\ y(0) \end{pmatrix} = \begin{pmatrix} V_x + r_x(0) \\ V_y + r_y(0) \end{pmatrix} \tag{4.5}$$

$$\begin{pmatrix} \dot{x}(0) \\ \dot{y}(0) \end{pmatrix} = \begin{pmatrix} \sqrt{\frac{g}{z_c}} \cdot W_x + \dot{r_x}(0) \\ \sqrt{\frac{g}{z_c}} \cdot W_y + \dot{r_y}(0) \end{pmatrix} \tag{4.6}$$

Using these equations, next we show how to produce the C-space (configuration space) trajectory corresponding to an isolated half-step. We just need to obtain a unique C-space trajectory from a small number of half-step parameters (as we will see, in our case it will be 3 parameters). If each of the robot legs has 6 degrees of freedom or more (the redundancy can be treated using generalized inverse kinematics, see [Nakamura and Hanafusa, 1987]), then this problem can be reduced to the generation of trajectories for the waist and the feet. Besides the compulsory constant waist height, we also make a few arbitrary and convenient restrictions (which reduce the number of parameters): the pitch and roll parameters of the waist orientation will stay at zero, and similarly the swing foot will always stay parallel to the walking surface. Thus, the lower body trajectory is entirely defined by 7 functions of the time:

- the waist horizontal position: $x(t)$, $y(t)$ (we recall that the waist and CoM are rigidly fixed)
- the waist orientation: $\theta(t)$
- the swing foot position: $SF_x(t)$, $SF_y(t)$, $SF_z(t)$
- the swing foot orientation $SF_\theta(t)$

4.2.1 Producing isolated half-steps

There are two types of half-steps: upward half-steps which start in double support configuration and end up with the swing foot at maximum height, and downward half-steps which start with the swing foot raised and end up in double support configuration.

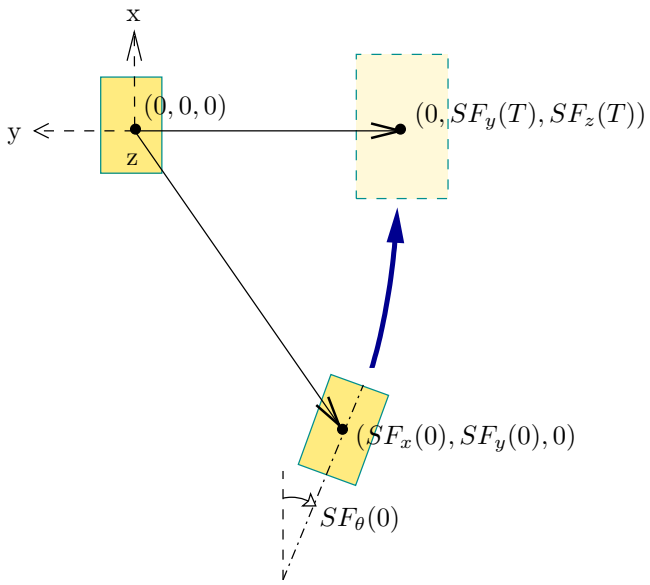


Fig. 4.1: Here we show an upward half-step from above. It is fully determined by the 5 parameters $SF_x(0)$, $SF_y(0)$, $SF_\theta(0)$, $SF_y(T)$ and $SF_z(T)$. A downward half-step is also fully determined by 5 parameters.

In this section we only consider upward half-steps, but the method for the generation of downward half-steps trajectories is similar.

So, let us consider an upward half-step. In order to reduce the dimensionality of the parameter space, we make several assumptions. First, we fix and denote by T the duration of any half-step. Then, we assume that the initial and final speed of the ZMP and swing foot are 0, but we *do not* assume that the CoM initial and final speed are zero.

$$\dot{p_x}(0) = \dot{p_y}(0) = \dot{p_x}(T) = \dot{p_y}(T) = 0 \tag{4.7}$$

$$\dot{\theta}(0) = \dot{\theta}(T) = 0 \tag{4.8}$$

$$\dot{SF_x}(0) = \dot{SF_y}(0) = \dot{SF_z}(0) = \dot{SF_\theta}(0) = 0 \tag{4.9}$$

$$\dot{SF_x}(T) = \dot{SF_y}(T) = \dot{SF_z}(T) = \dot{SF_\theta}(T) = 0 \tag{4.10}$$

Second, the initial vertical projection on the ground of the CoM is equal to the ZMP initial position, right in between the centers of the feet. Taking the center of the support foot as the origin of the Euclidean space, it gives us:

$$x(0) = p_x(0) = \frac{SF_x(0)}{2} \tag{4.11}$$

$$y(0) = p_y(0) = \frac{SF_y(0)}{2} \tag{4.12}$$

We also assume that the final horizontal position of the CoM and ZMP coincide at the center of the support foot, and that the final orientation of the swing foot and the initial and final orientation of the waist are equal to the orientation of the support foot (as a consequence during downward half-steps the initial orientation of the waist is equal to the orientation of the support foot, but the final orientation of the waist is

equal to the final orientation of the swing foot). Besides, the line passing through the centers of the final positions of the feet is orthogonal to this final orientation:

$$x(T) = p_x(T) = 0 \tag{4.13}$$

$$y(T) = p_y(T) = 0 \tag{4.14}$$

$$\theta(0) = \theta(T) = SF_\theta(T) = 0 \tag{4.15}$$

$$SF_x(T) = 0 \tag{4.16}$$

As a consequence of these equations, the final and initial configurations are entirely determined by 5 parameters (as shown on Fig. 4.1):

$$SF_x(0),\ SF_y(0),\ SF_\theta(0),\ SF_y(T) \text{ and } SF_z(T).$$

Besides, concerning the derivatives at the boundaries, the only free parameters are $\dot{x}(0)$, $\dot{x}(T)$, $\dot{y}(0)$, and $\dot{y}(T)$. This adds up to a total of 9 free parameters.

Now, we show how the ZMP trajectory is defined. An upward half-step is divided into 3 phases: during the first one, of duration t_1, the ZMP stays right in between the centers of the feet (and the feet keep their positions as well), so we have $p_x(t) = \frac{SF_x(0)}{2}$, $p_y(t) = \frac{SF_y(0)}{2}$, and $\dot{p_x}(t) = \dot{p_y}(t) = 0$. Then there is the "shift" phase, during which the ZMP quickly shifts from its initial position to its final position, reached at time t_2. Then, from t_2 to T, the ZMP stays at its final position, so we have $p_x(t) = p_y(t) = \dot{p_x}(t) = \dot{p_y}(t) = 0$. During the "shift" phase we set p_x and p_y as third-degree polynomials determined by the respective boundary conditions $p_x(t_1) = \frac{SF_x(0)}{2}$, $p_x(t_2) = \dot{p_x}(t_1) = \dot{p_x}(t_2) = 0$, and $p_y(t_1) = \frac{SF_y(0)}{2}$, $p_y(t_2) = \dot{p_y}(t_1) = \dot{p_y}(t_2) = 0$. For the downward half-step, even if the phase of double support and single support are inverted, we keep the same durations: the ZMP shift occurs between time t_1 and t_2. In practice, we set $t_1 = T - t_2$.

By application of eq. (4.4), if we fix $SF_x(0)$, $SF_y(0)$, $\dot{x}(0)$, and $\dot{y}(0)$, we can get an analytical expression of the unique C^2 solution for $x(t)$ and $y(t)$ over $[0, T]$. The solution is unique because during the first phase, V_x, V_y, W_x and W_y are fixed by the following equations (obtained from eq. (4.5) and eq. (4.6)):

$$V_x = \frac{SF_x(0)}{2} - r_x(0) \tag{4.17}$$

$$V_y = \frac{SF_y(0)}{2} - r_y(0) \tag{4.18}$$

$$W_x = \sqrt{\frac{z_c}{g}}\,(\dot{x}(0) - \dot{r_x}(0)) \tag{4.19}$$

$$W_y = \sqrt{\frac{z_c}{g}}\,(\dot{y}(0) - \dot{r_y}(0)) \tag{4.20}$$

Moreover, the unique solution during the first phase leads to unique values for $x(t_1)$, $y(t_1)$, $\dot{x}(t_1)$, and $\dot{y}(t_1)$. These values fix the free parameters of the unique C^2 extension of the solution on $[t_1, t_2]$, and subsequently the free parameters of the unique C^2 extension over $[t_2, T]$. Nevertheless, those two unique C^2 solutions might violate the constraints $x(T) = 0$ and $y(T) = 0$ (eq. (4.13) and eq. (4.14)). Analyzing the impact of $\dot{x}(0)$ and $\dot{y}(0)$ on the anayltical solutions, we can see that they have a monotonic influence over respectively $x(T)$ and $y(T)$, and that to one value of $x(T)$ (resp.

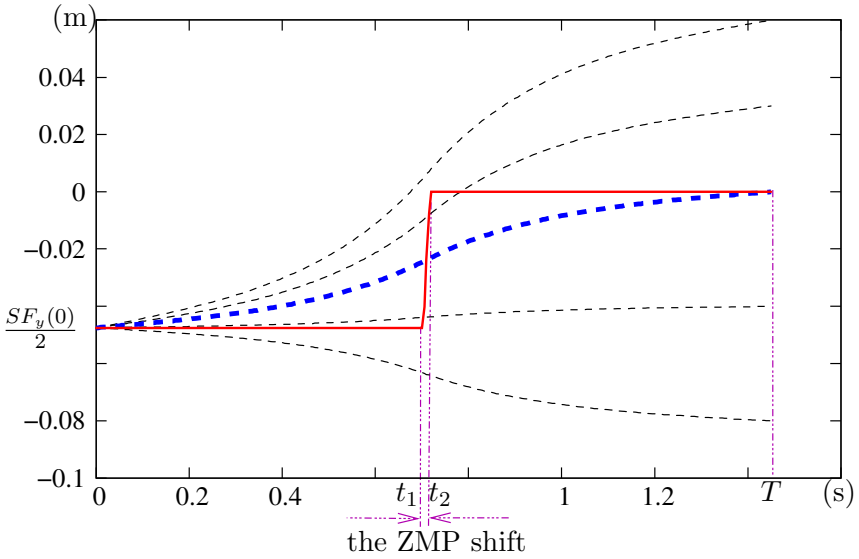


Fig. 4.2: We consider the upward half-step of Fig. 4.1, and show the corresponding ZMP trajectory along the y-axis: $p_y(t)$ (solid line). To this trajectory corresponds an infinity of C^2 solutions for $y(t)$ which all verify $y(0) = p_y(0) = \frac{SF_y(0)}{2}$, each of them being fully defined by the derivative of y at 0. We show several such C^2 solutions (dotted lines); the thick dotted line is the solution retained: it is the unique one verifying $y(T) = 0$.

$y(T)$) corresponds a unique value $\dot{x}(0)$ (resp. $\dot{x}(0)$). We implemented a dichotomic search for those values, and with simple methods avoided problems of numerical unstability (using the fact that with only one ZMP shift and the boundary conditions $CoM(0) = ZMP(0)$ and $CoM(T) = ZMP(T)$, the solution CoM trajectories x and y are necessarily monotone).

Fig. 4.2 considers the half-step of Fig. 4.1, and it shows the trajectory of the ZMP along the y-axis as well as several C^2 solutions for $y(t)$, for different values of $\dot{y}(0)$. Only one solution is retained, the one with $y(T) = 0$. If the durations t_1 and $T - t_2$ are long enough, the values obtained for $\dot{x}(0)$, $\dot{x}(T)$, $\dot{y}(0)$ and $\dot{y}(T)$ can be neglected, and thus the CoM trajectories obtained can be assumed C^2 continuous over $(-\infty, \infty)$. Performing tests on a real humanoid robot empirically validated this asumption: time discretization of the control law itself makes the neglected velocity unnoticeable. For the trajectories other than $x(t)$ and $y(t)$ ($\theta(t)$, $SF_x(t)$, $SF_y(t)$, $SF_z(t)$, $SF_\theta(t)$), we simply use polynomials of degree 3 that ensure C^2 smoothness and satisfying profiles, with a few specific constraints (e.g. in our implementation the swing foot always leaves the ground and lands vertically). So, we can completely define a half-step with 5 parameters (whether it is an upward half-step or a downward half-step). In our application, we decided to fix the maximum height of the swing foot ($SF_z(T)$), and the horizontal distance between the feet when the maximum height is reached (which fixes $SF_y(T)$). This means that now the configuration when the swing foot reaches its maximum height (i.e. the configuration "right in the middle" of a full step trajectory) is completely fixed. This puts us in the conditions of [Kuffner et al., 2001] where two statically stable, intermediate postures Q_{right} and Q_{left} are fixed, and serve as via points for all footstep trajectories (see Fig. 4.3). In [Kuffner et al., 2001] these "via

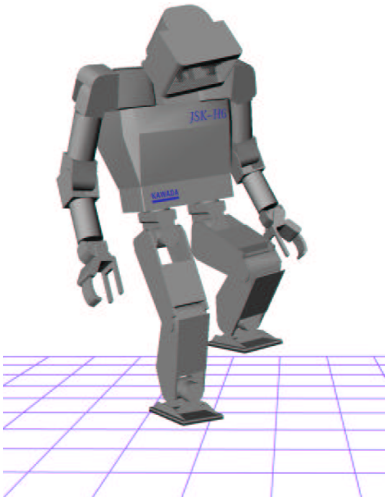


Fig. 4.3: The intermediate posture Q_{right} used in [Kuffner et al., 2001] for transitioning between left leg footstep placements.

point configurations" are used to reduce the number of transition trajectories, and in our case they have the similar effect of reducing the dimensionality of the parameter space[1]. Indeed, with these constraints only 3 parameters are needed to completely define a half-step. Once these parameters are set, we are capable of generating *unique* analytical solutions for the 7 functions of time that are required to produce the lower body trajectory in the C-space.

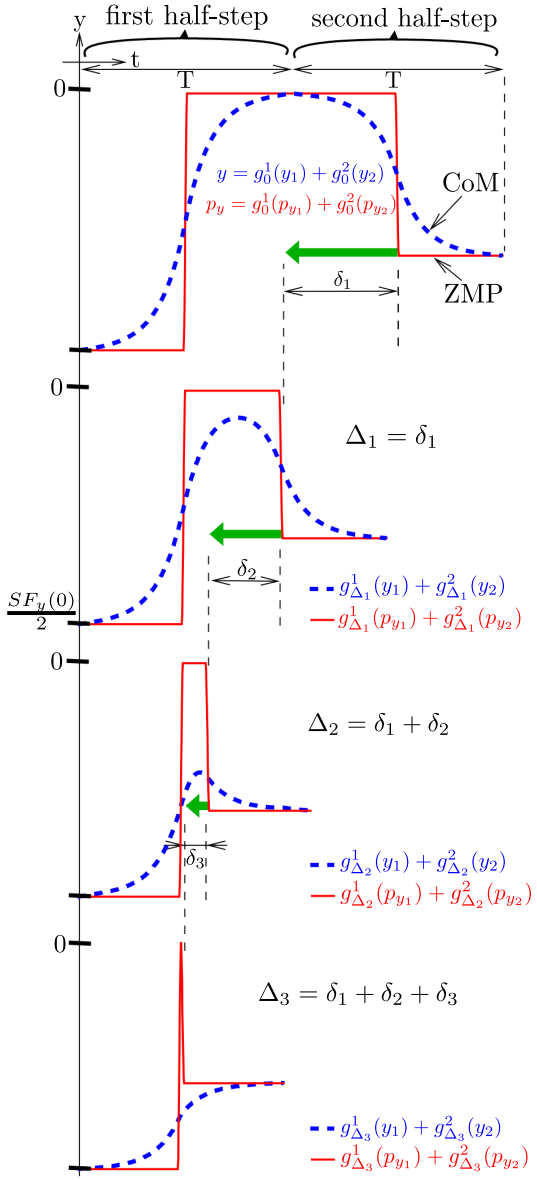
4.2.2 Smoothing a sequence of half-steps

Using the results of the previous section, we can generate C-space trajectories for isolated half-steps. Since they start and finish with zero speed, we can simply join them to produce sequences of half-steps. Alternating upward and downward half-steps will produce a walking motion. During each half-step, the motion is dynamically stable (i.e. not quasi-static, but the robot does not fall), but at the boundary of each half-step motion, the configuration is statically stable (i.e. it is a balanced posture). This is not a satisfactory result because between each half-step the robot comes to a stop, so the walk motion is not visually smooth, and rather slow. Recent walking pattern generators achieve much better results by using preview control (see [Kajita et al., 2003]). In this section, we show how to continuously modify a sequence of half-steps using a simple homotopy, in order to make it faster and smoother along the same footstep sequence. We first show how to do so for a sequence of two half-steps, and start with the case of an upward half-step followed by a downward half-step.

Upward then downward

We consider an upward half-step followed by a downward half-step. Together the two half-steps make a classical full step: double support phase, then single support phase, and then double support phase again.

[1] A remark: we use dynamic trajectories while in [Kuffner et al., 2001], only statically stable trajectories are considered.



The plot on the top shows the trajectories $y(t)$ and $p_y(t)$ for a raw sequence of two half-steps (no overlap), the first half-step being the one of Fig. 4.1, whose CoM and ZMP trajectories are shown on Fig. 4.2. Notice that the CoM reaches the ZMP between the half-steps. On the other plots, we show the effect of progressively increasing the overlap, using the operators g_Δ^1 and g_Δ^2. We can see that the CoM trajectory becomes more natural: it does not need to reach the top of the ZMP curve between the two ZMP shifts anymore. Indeed, the overlap works a bit like a preview control: the first CoM trajectory is influenced by the second one during the overlap, so it is as if it already "knew" that there will be another ZMP shift, and adapted consequently.

Fig. 4.4: Progressively increasing the overlap between two half-steps.

We recall that the whole C-space trajectory of the lower body during one half-step is generated by inverse geometry from 7 functions of the time. Since here we are dealing with two consecutive half-steps (with the same support foot), we have to consider 14 functions. Let us first consider for example the position of the waist along the y-axis, respectively for the upward half-step: $y_1(t)$, and the downward half-step: $y_2(t)$. We have $y_1(T) = y_2(0) = 0$. Let us define two operators g^1_Δ and g^2_Δ such that:

$$g^1_\Delta(f)(t) = \begin{cases} f(t) \text{ for } t \in (0, T) \\ f(T) \text{ for } t \in (T, 2T - \Delta) \end{cases} \tag{4.21}$$

$$g^2_\Delta(f)(t) = \begin{cases} 0 \text{ for } t \in (0, T - \Delta) \\ f(t - T + \Delta) - f(0) \text{ for } t \in (T - \Delta, 2T - \Delta) \end{cases} \tag{4.22}$$

$g^1_0(y_1) + g^2_0(y_2)$ corresponds to the simple concatenation of y_1 and y_2 without overlap. Knowing that $p_{y_1} = Z(y_1)$, $p_{y_2} = Z(y_2)$, and $y_1(T) = y_2(0) = 0$, it is quite easy to verify that for any $0 \leq \Delta \leq T$, $g^1_\Delta(p_{y_1}) = Z(g^1_\Delta(y_1))$ and $g^2_\Delta(p_{y_2}) = Z(g^2_\Delta(y_2))$. And, since Z is a linear operator:

$$g^1_\Delta(p_{y_1}) + g^2_\Delta(p_{y_2}) = Z(g^1_\Delta(y_1) + g^2_\Delta(y_2)) \tag{4.23}$$

It follows that operators g^1_Δ and g^2_Δ enable us to obtain new combined CoM and ZMP trajectories that still verify the Linear Inverted Pendulum equations (eq. (4.1) and eq. (4.2)). Starting with $\Delta = 0$ and progressively increasing the value of Δ continuously modifies the CoM trajectory (starting from the initial trajectory $g^1_0(y_1) + g^2_0(y_2)$) to make the second ZMP shift (the one of p_{y_2}) happen earlier, creating an overlap of duration Δ between the two trajectories y_1 and y_2. Fig. 4.4 illustrates this effect. When we increase the value of Δ we can see for example that the position of the CoM does not need to reach the center of the support foot anymore.

We use the same operators, g^1_Δ and g^2_Δ, to produce an overlap between the functions of time corresponding to the waist orientation and swing foot position and orientation. Since the inverse geometry for the legs is a continuous function as long as we stay inside the joint limits, these operators used on the bodies trajectories actually implement a simple homotopy that continuously deforms the initial C-space trajectory into a smoother, more dynamic trajectory. The linearity of simplified differential equations has already been used in a similar way to produce mixtures of motions ([Nishiwaki et al., 1999] and [Nishiwaki et al., 2001]), but the purpose was to create new steps, not to smooth them nor speed them up.

In the case of an upward half-step followed by a downward half-step, increasing Δ reduces the duration of the single support phase, and therefore it increases the speed of the swing foot. To limit this effect we must bound Δ. Besides, if Δ is too large undesirable phenomena can occur, such as a negative swing foot height. To avoid these problems we set an upper bound such that the maximum overlap results in a moderately fast gait.

Downward then upward

We can apply the same technique to produce an overlap in the case of a downward half-step followed by an upward half-step. Since the last phase of the downward half-step and the first phase of the upward half-step are double support phases, the constraint on the swing foot motion disappears and the maximum bound on Δ becomes simply T.

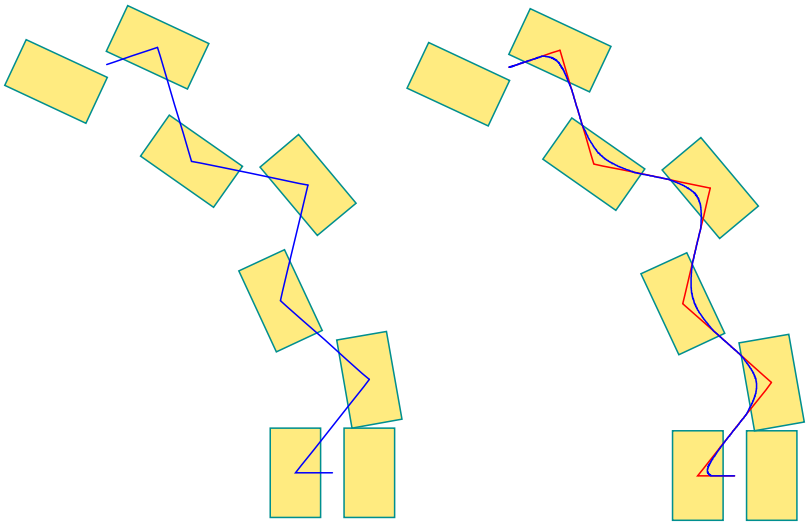


Fig. 4.5: We illustrate the "smoothing" of a raw sequence of half-steps. On the initial raw sequence (on the left), the support paths of the ZMP and CoM trajectories are superimposed. Then, after adjusting the overlaps, the ZMP support path stays the same but the CoM support path becomes smoother (on the right). We can smooth even more, but it reduces the duration of the single support phase that is directly linked with the swing foot speed. Therefore limitations on the swing foot speed constrain the smoothing process.

For longer sequences of half-steps, we can simply repeat the procedure to smooth the whole trajectory, setting the overlaps one by one. Fig. 4.5 shows the results obtained with an example of raw sequence. After the smoothing, the CoM trajectory is visually smoother and besides, the new trajectory is much faster (about 3 times faster). Changing overlaps inside a sequence of half-steps modifies the whole C-space trajectory: not only the CoM and ZMP, but also the swing foot trajectory. When the overlap is increased, the swing foot tends to move faster and closer to the ground. If one property must be preserved (for instance the absence of collision), it must be checked after every modification. Since the smoothing by overlap is a continuous operator, we can use dichotomies to quickly find large acceptable values of overlap. Let us consider an example for two consecutive half-steps. We predefine a maximum overlap Δ_{max} and, first, we simulate the part of the trajectory modified by the overlap Δ_{max}, and check for collisions, self-collisions and joint limit violations. If none of these events occur, we set the overlap to Δ_{max}. Otherwise, we use a dichotomy starting at $\Delta_{max}/2$ to quickly converge towards the largest "good" overlap value below Δ_{max}. Fig. 4.6 shows the effect of the smoothing process on the swing foot trajectory: with the dichotomy we can quickly find a large overlap that keeps the trajectory collision-free.

4.3 On the sensitivity of the walking pattern generator

In Chapter 5, we will take advantage of the low dimensionality of our pattern generator to go further in terms of offline precomputation by building a large (but finite) number of swept volume approximations for the leg trajectories and then use them online to speed up collision checks. But instead of only a finite number of swept volume

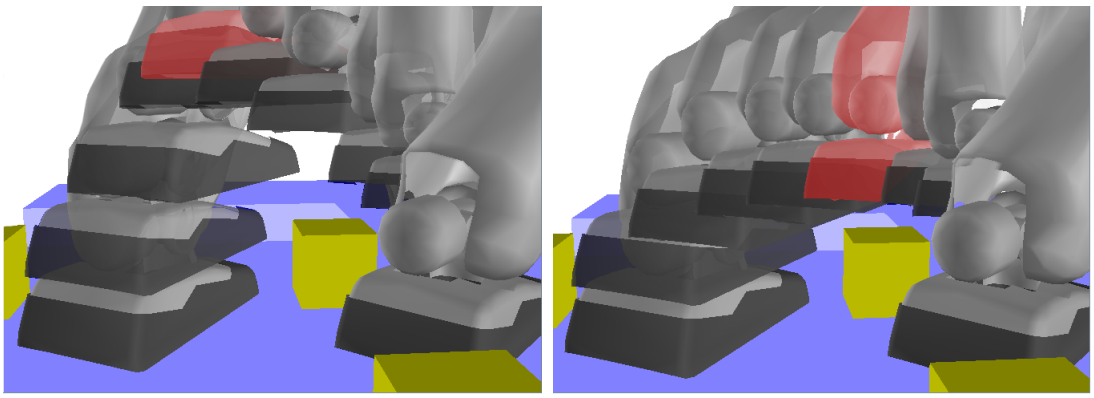


Fig. 4.6: *On the left*: a raw sequence of two half-steps avoiding a box on the ground. We can see that the swing foot reaches an unnecessarily high position. After smoothing (*on the right*), the trajectory has been modified so that the foot moves very close to the obstacle.

approximations, it would be even better to extrapolate swept volume approximations for continuous regions of the 3-dimensional input spaces defining half-steps. In the rest of the present chapter, we follow this simple idea: if we would expand the swept volumes by a fixed margin, then they could cover neighborhoods of half-steps definitions instead of single half-step definitions, and we could use a finite number of swept volumes in order to cover the whole continuous input space, while keeping the soundness of collision checks. Before doing that it is fundamental to know the relation between the size of the margin and the size of the neighborhoods. In other words, we need to know how many swept volumes are needed to cover the whole input space when the margin has a fixed size ϵ. Theorem 4.3.2 gives an answer to this question, but which, as we will see in Section 4.3.5, is unfortunately impractical.

4.3.1 Problem statement

We recall that through inverse geometry, both upward and downward half-steps are completely defined by the Center of Mass (CoM) horizontal trajectory (it remains at a fixed height), the waist orientation and the swing foot position and orientation trajectories, and all these trajectories are generated from only 3 input parameters: the relative position and orientation of the swing foot when on the walking surface.

In the following sections (4.3.2 to 4.3.5) we make a new convenient assumption which is just a slightly different convention than the one followed so far: we suppose that at all time, the orientation of the waist is equal to the mean value of the orientations of the feet (up to now we assumed that during a full step the initial orientation of the waist is equal to the orientation of the support foot while the final orientation of the waist is equal to the final orientation of the swing foot). This new convention gives more symmetry to the half-steps, and actually, because of another symmetry property imposed on the functions used to produce the trajectories (equations (4.29)), upward half-steps and downward half-steps become completely symmetric in the sense that a downward half-step is just an upward half-step with time going backwards. As a result, we don't need to consider both downward and upward half-steps: the bound obtained for upward half-steps will directly apply to downward half-steps. Besides, the half-

steps with left and right support foot are also symmetric, so in the end we only need to consider upward half-steps with left support foot. We recall that they are defined as shown on Fig. 4.1: at the end of the half-step, when $t = T$, the robot is in the fixed "via-point configuration" in which the swing foot coordinates are $(0, SF_y(T), SF_z(T))$, and its orientation zero. The input parameters are $SF_x(0)$, $SF_y(0)$, and $SF_\theta(0)$.

We define the robot lower body as the union of the two legs: the two thighs, calfs and feet. For a given vector of input parameters $(SF_x(0),\ SF_y(0), SF_\theta(0))$, we denote by $\mathcal{SV}\big((SF_x(0),\ SF_y(0), SF_\theta(0))\big)$ the volume swept by the robot lower body during the corresponding half-step. We also denote by $\mathcal{E}$ the continuous transition model, i.e. the set of input vectors $\vec{v}$ such that the corresponding half-step is feasible on a flat ground without obstacles (no self-collision, no joint limits violations –we mention in section 4.3.3 some of the restrictions imposed on the joint limits–, ...).

The question we want to answer is related to the following one:
Given an input vector $(SF_x(0), SF_y(0), SF_\theta(0)) \in \mathcal{E}$, and a small variation $(\Delta x, \Delta y, \Delta\theta)$ such that $(SF_x(0) + \Delta x, SF_y(0) + \Delta y, SF_\theta(0) + \Delta\theta)$ also belongs to $\mathcal{E}$, how can we bound, with respect to $(\Delta x, \Delta y, \Delta\theta)$, the variation that it implies for the swept volume?

In other words:
Given two input vectors $(SF_x(0), SF_y(0), SF_\theta(0)) \in \mathcal{E}$ and $(SF_x(0) + \Delta x, SF_y(0) + \Delta y, SF_\theta(0) + \Delta\theta) \in \mathcal{E}$, how can we bound, in function of $(\Delta x, \Delta y, \Delta\theta)$, the Hausdorff distance between $\mathcal{SV}\big((SF_x(0),\ SF_y(0), SF_\theta(0))\big)$ and $\mathcal{SV}\big((SF_x(0)+\Delta x, SF_y(0)+\Delta y, SF_\theta(0) + \Delta\theta)\big)$?

We answer this question by dividing the problem into three parts: first, in section 4.3.2, we bound the variation induced on the waist trajectory and the trajectories of the feet by a slight modification of the input parameters. Then, in section 4.3.3, we bound the displacement of the physical points of the robot leg induced by a slight modification of the end effector (the foot) configuration in the workspace. Finally we combine the two results in section 4.3.4 and obtain Theorem 4.3.2. In section 4.3.5, we present numerical results and conclude.

4.3.2 Sensitivity of the trajectory generation

Let us consider an upward half-step defined by the 3 parameters $SF_x(0)$, $SF_y(0)$ and $SF_\theta(0)$. As explained in Section 4.2, these parameters are used to set the following functions of the time (plus the waist orientation):

- the waist horizontal position, which is equal to the CoM horizontal position (the waist and CoM are rigidly fixed) : $x(t)$, $y(t)$
- the swing foot position: $SF_x(t)$, $SF_y(t)$, $SF_z(t)$
- the swing foot orientation $SF_\theta(t)$

For all these trajectories except $x(t)$ and $y(t)$, C^2 cubic splines are used. $SF_z(t)$ does not depend on the input parameters. For the other functions $SF_x(t)$, $SF_y(t)$ and $SF_\theta(t)$, we use a single operator $\mathcal{F}$ which maps a couple of real numbers (a, b) to a C^2 cubic spline $s(t)$ that smoothly goes from a at $t = 0$ to b at $t = T$ with constraints such as zero speed at $t = 0$ and $t = T$:

$$SF_x(t) = \mathcal{F}(SF_x(0), 0)(t) \tag{4.24}$$

$$SF_y(t) = \mathcal{F}(SF_y(0), SF_y(T))(t) \tag{4.25}$$

$$SF_\theta(t) = \mathcal{F}(SF_\theta(0), 0)(t) \tag{4.26}$$

$\mathcal{F}$ has three important properties:

$$\mathcal{F}(a+c, b+d) = \mathcal{F}(a,b) + \mathcal{F}(c,d) \tag{4.27}$$

$$\forall t \in [0,T],\ \min(a,b) \leq \mathcal{F}(a,b)(t) \leq \max(a,b) \tag{4.28}$$

$$\forall t \in [0,T],\ \mathcal{F}(a,b)(t) = \mathcal{F}(b,a)(T-t) \tag{4.29}$$

If we call x', y', SF'_x, SF'_y and SF'_θ the trajectories associated to the input parameters $(SF_x(0)+\Delta x, SF_y(0)+\Delta y, SF_\theta(0)+\Delta\theta)$, then from the above properties of $\mathcal{F}$ follows, for all $t \in [0,T]$:

$$\min(0,\Delta x) \leq SF'_x(t) - SF_x(t) \leq \max(0,\Delta x) \tag{4.30}$$

$$\min(0,\Delta y) \leq SF'_y(t) - SF_y(t) \leq \max(0,\Delta y) \tag{4.31}$$

$$\min(0,\Delta\theta) \leq SF'_\theta(t) - SF_\theta(t) \leq \max(0,\Delta\theta) \tag{4.32}$$

Now we will try to obtain similar inequalities for x and y. For the sake of clarity, we rewrite the equations linking the ZMP trajectories p_x, p_y and the CoM trajectories x, y:

$$p_x = Z(x) \tag{4.33}$$

$$p_y = Z(y) \tag{4.34}$$

$$\text{with } Z \triangleq Id - \frac{z_c}{g}\frac{d}{dt^2} \tag{4.35}$$

We recall that $p_x(0) = x(0) = \frac{SF_x(0)}{2}$ and $p_y(0) = y(0) = \frac{SF_y(0)}{2}$, and $p_x(T) = x(T) = 0$ and $p_y(T) = y(T) = 0$. It follows that if p_x and p_y are fixed as C^2 cubic splines, there are unique C^4 solutions for x and y over $[0,T]$ (in which $\dot{x}(0)$, $\dot{y}(0)$ and $\dot{x}(T)$, $\dot{y}(T)$ are not zero, but, as mentionned in section 4.2, can be neglected if T is large enough and p_x and p_y well chosen) that we denote respectively by $Z^{-1}(p_x)$ and $Z^{-1}(p_y)$.

p_x and p_y can be seen as the result of an operator $\mathcal{F}_Z$ which maps a couple of reals (a,b) to a C^2 cubic spline that quickly shifts from a to b around time $t = T/2$, and which verifies the same properties (4.27), (4.28) and (4.29) as $\mathcal{F}$. We have then:

$$p'_x = \mathcal{F}_Z\left(\frac{SF_x(0)+\Delta x}{2}, 0\right) = p_x + \mathcal{F}_Z\left(\frac{\Delta x}{2}, 0\right) \tag{4.36}$$

$$p'_y = \mathcal{F}_Z\left(\frac{SF_y(0)+\Delta y}{2}, 0\right) = p_y + \mathcal{F}_Z\left(\frac{\Delta y}{2}, 0\right) \tag{4.37}$$

From the linearity of the operator Z we deduce the linearity of the operator Z^{-1}, and thus we have:

$$x' = Z^{-1}(p'_x) = x + Z^{-1}(\mathcal{F}_Z(\Delta x/2, 0)) \tag{4.38}$$

$$y' = Z^{-1}(p'_y) = y + Z^{-1}(\mathcal{F}_Z(\Delta y/2, 0)) \tag{4.39}$$

Now let's have look on the properties of this operator Z^{-1}. Since the result is a continuous function, as shown on Fig. 4.7, the restrictions $Z^{-1}(p)(0) = p(0)$, $Z^{-1}(p)(T) = p(T)$ imply that for a C^2 cubic spline p over $[0,T]$, we necessarily have:

$$\forall t \in [0,T],\ \min_{\tau\in[0,T]}(p(\tau)) \leq Z^{-1}(p)(t) \leq \max_{\tau\in[0,T]}(p(\tau)) \tag{4.40}$$

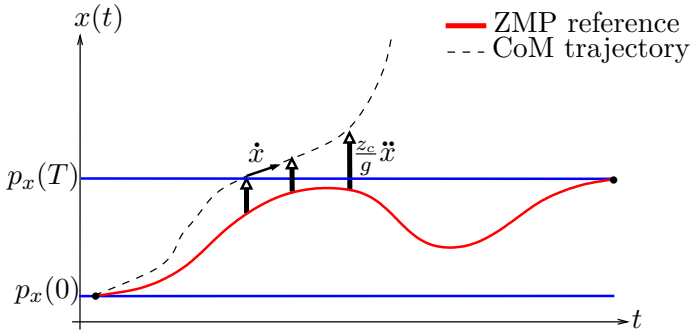


Fig. 4.7: If one of the components of the CoM position, say for example x, goes outside the stripe $[p_x(0), p_x(T)]$ defined by the ZMP reference, then the second derivative of x, obtained with the equation (4.33), can only make x diverge. Therefore $x(t)$ must stay inside the stripe for all $t \in [0, T]$. Remark: this figure is only illustrative, the curves do *not* correspond to real ZMP/CoM curves.

Since we also have for all $t \in [0, T]$, $\mathcal{F}_Z\left(\Delta x/2, 0\right)(t) \leq \max(0, \Delta x/2)$, $\mathcal{F}_Z\left(\Delta x/2, 0\right)(t) \geq \min(0, \Delta x/2)$ and the same for $\mathcal{F}_Z\left(\Delta y/2, 0\right)(t)$, we finally obtain the inequalities we were looking for: for all $t \in [0, T]$,

$$\min\left(0, \frac{\Delta x}{2}\right) \leq x'(t) - x(t) \leq \max\left(0, \frac{\Delta x}{2}\right) \tag{4.41}$$

$$\min\left(0, \frac{\Delta y}{2}\right) \leq y'(t) - y(t) \leq \max\left(0, \frac{\Delta y}{2}\right) \tag{4.42}$$

4.3.3 Sensitivity of the inverse kinematics of a simple humanoid robot leg

In this section we consider a very generic robot leg with 6 degrees of freedom. Its geometry is exactly similar or almost similar to the geometry of the legs of several humanoid robots, including HRP-2, ASIMO, HRP-4, QRIO, HUBO, WABIAN, and LOLA. For slightly different geometries, the proofs presented in this section can be adapted, and lead to bounds of the same order of magnitude. The robot leg consists in 3 joints: the hip (yaw + pitch + roll), the knee (only pitch), and the ankle (pitch + roll), the foot being the end effector. The main axis of the foot is supposed to be always included in the plane defined by the hip, knee and ankle. The center of the hip joint is the origin (0,0,0) of the workspace, and a horizontal reference orientation is fixed. Since we assume that the foot always stays parallel to the ground, as shown on Fig. 4.8 the configuration of the leg is entirely defined by two vectors: $\vec{L}$ (the vector from the hip joint to the ankle joint), and $\vec{F}$, a unit vector defining the orientation of the foot. $\vec{F}$ is also completely defined by a single value θ which does not necessarily correspond to the value of the hip yaw. The leg is composed of three bodies: the thigh, the calf, and the foot. For the thigh and the calf, we assume that all the physical points of the robot are contained in a volume defined by 2 spheres and a cylinder: see Fig. 4.8. The spheres and cylinder of the thigh have radius r_T, while the spheres and cylinder of the calf have radius r_C. The two links and cylinders have the same length l. We also assume that the foot is entirely contained in a sphere of radius r_F and center the ankle joint. We impose a few restrictions on the joint values: let us

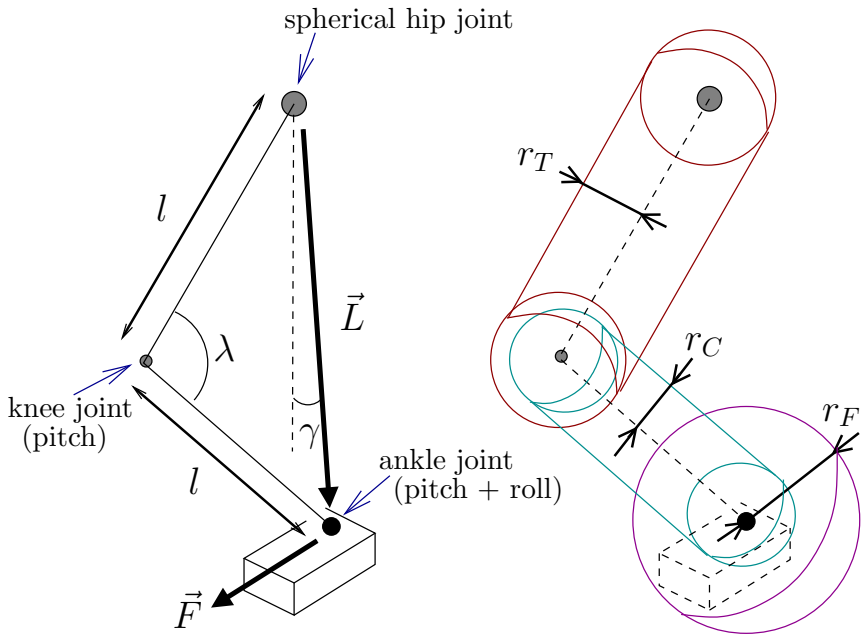


Fig. 4.8: A simple 6-DOF robot leg.

call ν the vertical axis passing through (0,0,0), γ the angle between ν and $\vec{L}$, and λ the value of the knee joint (see Fig. 4.8). The restrictions are: $0 \leq \gamma \leq \gamma_{max} < \pi/3$, and $0 < \lambda_{min} \leq \lambda \leq \lambda_{max} < \pi$. Here is the context of our problem: the current configuration of the leg is $(\vec{L}, \theta)$ (where θ is the orientation of $\vec{F}$), and we modify it to $(\vec{L} + \Delta\vec{L}, \theta + \Delta\theta)$, with $\Delta\vec{L}$ and $\Delta\theta$ "relatively small" (we will discuss about this notion later). We ask the following question: how can we bound, in function of $\Delta\vec{L}$ and $\Delta\theta$ only, the distance between any physical point of the leg in configuration $(\vec{L}, \theta)$, and the same physical point when the configuration is $(\vec{L} + \Delta\vec{L}, \theta + \Delta\theta)$?

Even a closed-form solution of the inverse kinematics of the leg (see [Siciliano and Khatib, 2008], section 1.7.1) cannot be straightforwardly used to obtain a tight bound, and the same is true for an approach based on the inverse Jacobian, so instead we choose to use purely geometrical considerations in an attempt to obtain a tighter bound, which will lead us to the following theorem:

Theorem 4.3.1. *For any point of the robot leg, the displacement resulting from the motion from configuration $(\vec{L}, \theta)$ to configuration $(\vec{L} + \Delta\vec{L}, \theta + \Delta\theta)$ is bounded by the maximum of the three following expressions:*

$$
\begin{array}{l}
1) \quad 2\sin\left(\frac{\alpha_{max}}{2}\right)(l + \max(r_T, r_C)) + \sin\left(\frac{1}{2}\arcsin\left(\frac{2\sin(\alpha_{max}/2)}{\cos\gamma_{max}}\right)\right)(l + \max(r_T, r_C)) \\
+2\sin(\beta_{max}/2)(l + \max(r_T, r_C)) + 2\sin\left(\frac{\cos\gamma_{max}\cdot|\Delta\theta|}{4\cos\gamma_{max} - 2}\right)(l\cdot\cos\left(\frac{\lambda_{min}}{2}\right) + \max(r_T, r_C)) \\
2) \quad \|\Delta\vec{L}\| + 2\sin(\alpha_{max}/2)r_C + 2\sin\left(\frac{1}{2}\arcsin\left(\frac{2\sin(\alpha_{max}/2)}{\cos\gamma_{max}}\right)\right)r_C \\
+2\sin(\beta_{max}/2)r_T + 2\sin\left(\frac{\cos\gamma_{max}\cdot|\Delta\theta|}{4\cos\gamma_{max} - 2}\right)r_C \\
3) \quad \|\Delta\vec{L}\| + 2\sin(\frac{|\Delta\theta|}{2})r_F
\end{array}
$$

with:

$$
\begin{cases}
\alpha_{max} = \arcsin\left(\frac{\|\Delta\vec{L}\|}{2l\sin(\frac{\lambda_{min}}{2})}\right) \\
\beta_{max} = \frac{\lambda_{max}}{2} - \arcsin\left(\sin(\frac{\lambda_{max}}{2}) - \frac{\|\Delta\vec{L}\|}{2l}\right)
\end{cases}
$$

We denote by $\mathcal{B}(\|\vec{\Delta L}\|, |\Delta\theta|)$ *this bound, and it should be noted that it tends to zero when* $\|\vec{\Delta L}\|$ *and* $|\Delta\theta|$ *tend to zero.*

We will decompose the demonstration of this theorem in several steps, and start with a preliminary lemma which shows that if we can bound the displacement undergone by the points inside the 5 spheres previously defined (2 spheres for thigh, 2 spheres for the calf, and 1 for the foot), we will have obtained a global bound that applies to any physical point of the robot leg (in other words, no need to consider the cylinders). The proofs of these steps use only basic mathematics but can be quite complex, so most of the elements of the proofs are in appendix. During the demonstrations we will make some assumptions that have not been stated before. We don't prove it here, but it can be verified that all those assumptions are true when $\|\vec{\Delta L}\|$ and $\vec{\Delta\theta}$ are relatively small. We don't give more details because as $\|\vec{\Delta L}\|$ and $\vec{\Delta\theta}$ are always very small in the potential applications, these assumptions always hold in practice.

Lemma 4.3.1. *Let's consider the robot leg whose bodies are exactly the unions of the spheres and cylinders previously defined. For any motion of this leg, the maximum displacement undergone by its physical points is obtained for a point inside one of the spheres.*

Proof. Thanks to Chasles' Screw displacement theorem (see [Beatty, 1986], section 3.11), we know that the transformation undergone by any rigid body of the robot leg can be written as the commutative composition of a rotation about an axis and a translation along the same axis. Since the displacements resulting from the rotation and the translation are orthogonal, we can take them into account separately. The translation results in the same displacement for all the points, so in order to find the maximum displacement we just need to consider the rotation. Let us project the rigid body onto a plane orthogonal to the rotation axis, and call O the intersection point between the axis and the plane. We use O as the origin of this plane and use vectors to represent projections of the rigid body points. Let $\vec{p}$ be such a projection; it lies on a segment line whose extremities $\vec{a}$ and $\vec{b}$ are projections of points inside the spheres at the extremities of the link. We thus have $\vec{p} = \mu\vec{a} + (1-\mu)\vec{b}$, with $\mu \in [0,1]$. If we denote by Rot the rotation, the displacement $d(\vec{p})$ undergone by the point projected on $\vec{p}$ can be written:

$$\begin{aligned} d(\vec{p}) &= \|Rot(\vec{p}) - \vec{p}\| \\ &= \|\mu(Rot(\vec{a}) - \vec{a}) + (1-\mu)(Rot(\vec{b}) - \vec{b})\| \end{aligned} \tag{4.43}$$

Since the maximum distance from a point to a line segment is always obtained at one of its extremities, we have:

$$d(\vec{p}) \leq \max(\|Rot(\vec{a}) - \vec{a}\|, \|Rot(\vec{b}) - \vec{b}\|) \tag{4.44}$$

$$d(\vec{p}) \leq \max(d(\vec{a}), d(\vec{b})) \tag{4.45}$$

We conclude that the maximum displacement is obtained for a point inside a sphere. □

We can now start the demonstration of theorem 4.3.1.

Proof. Thanks to Lemma 4.3.1, we know that we only need to bound the displacements for points inside the spheres, and we will naturally derive 5 bounds, one for each sphere. But first, in order to calculate these bounds, let us describe the transformations that are applied to the leg bodies.

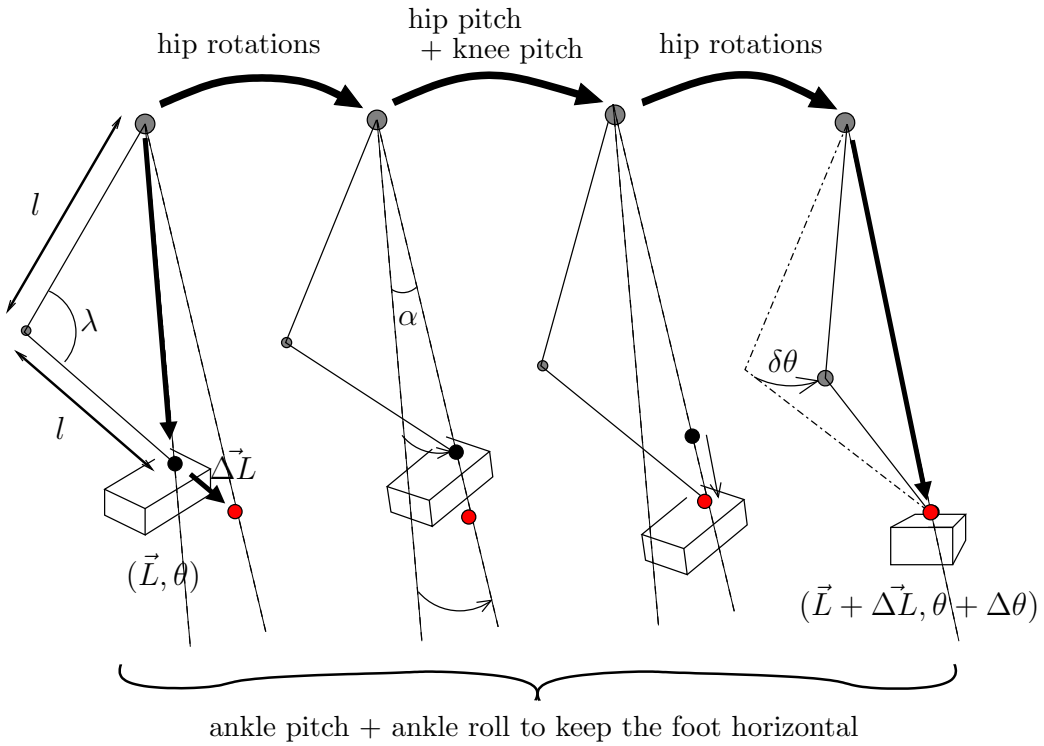


Fig. 4.9: The three phases to go from $(\vec{L}, \theta)$ to $(\vec{L} + \Delta\vec{L}, \theta + \Delta\theta)$.

The motion changing $(\vec{L}, \theta)$ into $(\vec{L} + \Delta\vec{L}, \theta + \Delta\theta)$ can be decomposed into three parts (see Fig. 4.9): first, using hip rotations, we obtain the correct *hip joint - ankle joint* axis while keeping $\vec{F}$ unchanged thanks to ankle rotations; then, sliding along this axis, using the hip and knee pitch, we obtain $(\vec{L} + \Delta\vec{L}, \theta)$ (ankle rotations are again used to keep $\vec{F}$ unchanged). Finally, we use the hip joint to rotate about the axis defined by $\vec{L} + \Delta\vec{L}$, keeping the foot parallel to the walking surface by modifying the ankle pitch and roll.

In the first phase, the correct axis can be obtained by two consecutive rotations about orthogonal axes. The first rotation of angle α_1 uses only the hip pitch, and during it the foot is kept horizontal by modifying the ankle pitch. Then follows a rotation of angle α_2 about the axis passing through the origin and with direction the vector $\vec{F}$. During this rotation the ankle roll is used to keep the foot horizontal. So far $\vec{F}$ remains unchanged. To obtain bounds on the displacements resulting from these rotations, we will need to bound the angles α_1 and α_2.

In the second phase, three rotations are combined to slide the ankle along the *hip joint - ankle joint* axis: a rotation of angle β (hip pitch), a rotation of angle -2β (knee pitch), and a rotation of angle β (ankle pitch) to keep $\vec{F}$ unchanged (see Fig. F.1 in appendix F). We will need to bound the angle β.

In the third phase, the angle $\delta\theta$ of the rotation (about the axis defined by $\vec{L} + \Delta\vec{L}$) is not necessarily equal to $\Delta\theta$ and needs to be bounded as well.

Since for a rotation of angle ψ on a point p at distance 1 of the rotation axis, the resulting displacement of p is $|2\sin(\psi/2)|$, we will rather try to bound this expression instead of bounding directly the angle value. Here are the bounds obtained for the 4 angles (with proofs in appendix):

The positive angles α_1 and α_2 are such that:

$$|2\sin(\alpha_1/2)| \leq 2\sin(\alpha_{max}/2), \tag{4.46}$$

$$|2\sin(\alpha_2/2)| \leq 2\sin\left(\frac{1}{2}\arcsin\left(\frac{2\sin(\alpha_{max}/2)}{\cos\gamma_{max}}\right)\right), \tag{4.47}$$

with $\alpha_{max} = \arcsin\left(\frac{\|\Delta\vec{L}\|}{2l\sin(\frac{\lambda_{min}}{2})}\right)$.

The demonstration is in appendix E.

The angle β is such that:

$$|2\sin(|\beta|/2)| \leq 2\sin(\beta_{max}/2) \tag{4.48}$$

with $\beta_{max} = \frac{\lambda_{max}}{2} - \arcsin\left(\sin\left(\frac{\lambda_{max}}{2}\right) - \frac{\|\Delta\vec{L}\|}{2l}\right)$.

The demonstration is in appendix F.

The angle $\delta\theta$ is such that:

$$|2\sin(|\delta\theta|/2)| \leq 2\sin\left(\frac{\cos\gamma_{max}}{4\cos\gamma_{max} - 2}\cdot|\Delta\theta|\right) \tag{4.49}$$

The demonstration is in appendix G.

Now we can bound the displacements undergone by the points of each sphere.

The sphere of the thigh at the hip joint

During the first, second and third phase of the motion of the leg, this sphere undergoes 4 successive rotations of angles α_1, α_2, β and $\delta\theta$, and the points of the sphere are at a distance from the rotations axes which is bounded by r_T. It follows that the total displacement undergone by any point of the sphere is bounded by the following sum:

$$\begin{array}{l} 2\sin(\alpha_{max}/2)r_T + 2\sin\left(\frac{1}{2}\arcsin\left(\frac{2\sin(\alpha_{max}/2)}{\cos\gamma_{max}}\right)\right)r_T \\ +2\sin(\beta_{max}/2)r_T + 2\sin\left(\frac{\cos\gamma_{max}}{4\cos\gamma_{max}-2}\cdot|\Delta\theta|\right)r_T \end{array} \tag{4.50}$$

The sphere of the thigh at the knee joint

This sphere undergoes the same rotations as the other sphere of the thigh. During the first and second phases the distance of its points to the rotation axes is bounded by $l + r_T$, while during the third phase, this distance is bounded by $l\cdot\cos(\lambda_{min}/2) + r_T$ ($l\cdot\cos(\lambda_{min}/2)$ is the maximum distance between the knee joint and the *hip joint* - *ankle joint* axis). It results in the following bound:

$$\begin{array}{l} 2\left(\sin\left(\frac{\alpha_{max}}{2}\right) + \sin\left(\frac{1}{2}\arcsin\left(\frac{2\sin(\alpha_{max}/2)}{\cos\gamma_{max}}\right)\right)\right)(r_T + l) \\ +2\sin(\beta_{max}/2)(l + r_T) \\ +2\sin\left(\frac{\cos\gamma_{max}}{4\cos\gamma_{max}-2}\cdot|\Delta\theta|\right)(l\cdot\cos(\lambda_{min}/2) + r_T) \end{array} \tag{4.51}$$

The sphere of the calf at the knee joint

This sphere undergoes the same rotations as the sphere of the thigh at the knee joint, except the rotation of angle β: while the sphere of the thigh undergoes a rotation of angle β about an axis at distance l from its center (the axis passes through the hip joint), the sphere of the calf undergoes a rotation of angle $-\beta$ about a virtual axis also at distance l (see Fig. F.1 in appendix F). It follows that the same expressions of displacement bounds can be used for all the four rotations, except that r_T must be replaced by r_C; hence the bound is:

$$\begin{aligned} &2\left(\sin\left(\tfrac{\alpha_{max}}{2}\right)+\sin\left(\tfrac{1}{2}\arcsin\left(\tfrac{2\sin(\alpha_{max}/2)}{\cos\gamma_{max}}\right)\right)\right)(r_C+l)\\ &+2\sin(\beta_{max}/2)(l+r_C)\\ &+2\sin\left(\tfrac{\cos\gamma_{max}}{4\cos\gamma_{max}-2}\cdot|\Delta\theta|\right)(l\cdot\cos(\lambda_{min}/2)+r_C) \end{aligned} \tag{4.52}$$

The sphere of the calf at the ankle joint

This sphere undergoes the same rotations as the other sphere of the calf, but since we know that the whole transformation results in a translation of vector $\vec{\Delta L}$ of the center of the sphere, we can decompose the transformation into two motions: first a translation of vector $\vec{\Delta L}$, and then 4 consecutive rotations (of angles α_1, α_2, β and $\delta\theta$) about axes passing through the center of the sphere. During any of these 4 rotations the distance of the points of the sphere to the rotation axis is at most r_C, and thus the following bound on the total displacement undergone by the points of the sphere follows:

$$\begin{aligned} &\|\vec{\Delta L}\|+2\sin(\alpha_{max}/2)r_C\\ &+2\sin\left(\tfrac{1}{2}\arcsin\left(\tfrac{2\sin(\alpha_{max}/2)}{\cos\gamma_{max}}\right)\right)r_C\\ &+2\sin(\beta_{max}/2)r_T+2\sin\left(\tfrac{\cos\gamma_{max}}{4\cos\gamma_{max}-2}\cdot|\Delta\theta|\right)r_C \end{aligned} \tag{4.53}$$

The sphere of the foot

Since this sphere is attached to the end effector (the foot), the global transformation undergone is simple: a translation of vector $\vec{\Delta L}$ followed by a rotation of angle $\Delta\theta$ about a vertical axis passing through the center of the sphere. As a result there is a simple upper bound for the displacements undergone by the points inside the sphere of the foot:

$$\|\vec{\Delta L}\|+2\sin(|\Delta\theta|)r_F \tag{4.54}$$

Finally, by regrouping the bounds (4.50), (4.51), (4.52), (4.53), and (4.54), noticing that the bound (4.51) is greater than the bound (4.50), we obtain the global bound of theorem 4.3.1, and that concludes the demonstration. □

4.3.4 Global bound

We now combine the bounds obtained in the two previous sections. At an instant $t \in [0, T]$, we can describe the configuration of the robot lower body with the position of the CoM and orientation of the waist, and the position and orientation of both feet relatively to the CoM position and waist orientation. For the half-step with input parameters $(SF_x(0), SF_y(0), SF_\theta(0))$, it gives us the following values:

- For the CoM position and the waist orientation: $x(t)$, $y(t)$, and $\frac{SF_\theta(t)}{2}$.

- For the swing foot relative position:

$$\underbrace{\begin{pmatrix} \cos(-SF_\theta(t)/2) & -\sin(-SF_\theta(t)/2) \\ \sin(-SF_\theta(t)/2) & \cos(-SF_\theta(t)/2) \end{pmatrix}}_{R_{-SF_\theta(t)/2}} \begin{pmatrix} SF_x(t) - x(t) \\ SF_y(t) - y(t) \end{pmatrix}$$

- $\frac{SF_\theta(t)}{2}$ for the swing foot relative orientation.
- For the support foot relative position:

$$R_{-SF_\theta(t)/2} \begin{pmatrix} -x(t) \\ -y(t) \end{pmatrix}$$

- $\frac{-SF_\theta(t)}{2}$ for the support foot relative orientation.

Using the notations of section 4.3.2, for the half-step with input parameters $(SF_x(0) + \Delta x, SF_y(0) + \Delta y, SF_\theta(0) + \Delta\theta)$ those values are:

- For the CoM position and the waist orientation: $x'(t)$, $y'(t)$, and $\frac{SF'_\theta(t)}{2}$.
- For the swing foot relative position:

$$R_{(-SF'_\theta(t)+SF_\theta(t))/2} R_{-SF_\theta(t)/2} \begin{pmatrix} SF'_x(t) - x'(t) \\ SF'_y(t) - y'(t) \end{pmatrix}$$

- $\frac{SF'_\theta(t)}{2}$ for the swing foot relative orientation.
- For the support foot relative position:

$$R_{(-SF'_\theta(t)+SF_\theta(t))/2} R_{-SF_\theta(t)/2} \begin{pmatrix} -x'(t) \\ -y'(t) \end{pmatrix}$$

- $\frac{-SF'_\theta(t)}{2}$ for the support foot relative orientation.

As in section 4.3.3, we call ν the vertical axis passing through the CoM. Let r_{max} be a bound of the distance between ν and any point of the robot lower body at all time: the robot lower body is always completely included in a cylinder of axis ν and radius r_{max}. It follows that the displacements resulting from a rotation R_ψ of axis ν are bounded by $2\sin(|\psi|/2)r_{max}$.

By application of the inequalities (4.41) and (4.42), at any instant t the distance between the CoM position in the trajectory of parameters $(SF_x(0), SF_y(0), SF_\theta(0))$, and the CoM position in the trajectory of parameters $(SF_x(0)+\Delta x, SF_y(0)+\Delta y, SF_\theta(0)+\Delta\theta)$, is at most $\frac{\sqrt{\Delta x^2+\Delta y^2}}{2}$. Similarly, the difference between the two waist orientations is bounded by $|\Delta\theta|/2$. Applying also the inequalities (4.30), (4.31), and (4.32), we obtain that for each foot its position relatively to the corresponding hip is changed by at most $\sqrt{\Delta x^2 + \Delta y^2} + 2\sin(|\Delta\theta|/4)r_{max}$ (the term $2\sin(|\Delta\theta|/4)r_{max}$ is due to the change of orientation of the waist, bounded by $|\Delta\theta|/2$). The relative orientations of the feet are changed by at most $|\Delta\theta|/2$. A bound on the total displacement undergone by the physical points of the robot lower body is given by the sum of the bounds for the two following displacements: first the displacement resulting from the modification of the CoM position and waist orientation, and then the displacement resulting from the modification of the relative configurations of the legs. The latter can be bounded using Theorem 4.3.1, and that leads us to the following theorem:

Theorem 4.3.2. *Assuming the previously defined restrictions concerning the values* r_T, r_C, r_F, γ_{max}, λ_{min}, λ_{max}, r_{max}, *we have the following result: given two input vectors* $(SF_x(0), SF_y(0), SF_\theta(0)) \in \mathcal{E}$ *and* $(SF_x(0)+\Delta x, SF_y(0)+\Delta y, SF_\theta(0)+\Delta\theta) \in \mathcal{E}$, *the Hausdorff distance between* $\mathcal{SV}\big((SF_x(0),\ SF_y(0), SF_\theta(0))\big)$ *and* $\mathcal{SV}\big((SF_x(0)+\Delta x, SF_y(0)+\Delta y, SF_\theta(0)+\Delta\theta)\big)$ *is bounded by:*

$$\frac{\sqrt{\Delta x^2+\Delta y^2}}{2} + \Omega + \mathcal{B}\left(\sqrt{\Delta x^2+\Delta y^2}+\Omega, \frac{|\Delta\theta|}{2}\right), \tag{4.55}$$

where $\Omega = 2\sin\left(\frac{|\Delta\theta|}{4}\right) r_{max}$.

4.3.5 Numerical estimates and potential applications

We defined standard values that should approximately correspond to the normal use of a humanoid robot with almost human dimensions: $r_T = r_C = 15cm$, $r_F = 20cm$, $l = 30cm$, $r_{max} = 50cm$, $\lambda_{min} = \frac{\pi}{3}$, $\lambda_{max} = \frac{\pi}{3}$, $\gamma_{max} = \frac{\pi}{4}$. We also fixed the limits of the input space: $SF_x(0) \in [-0.30m, +0.30m]$, $SF_y(0) \in [-0.30m, 0m]$, and $SF_\theta(0) \in [-\frac{\pi}{6}, \frac{\pi}{6}]$. With those values, we used theorem 4.3.2 to estimate, for different margins on the swept volumes, the total number of points needed to obtain a covering of the whole input space: see table 4.1. We also tried with bounds specifically calculated for the robot HRP-2, and obtained comparable results. As shown in Table 4.1, for a margin of 10cm, about 1,000,000 points are needed.

This means that the following method would be theoretically possible for safe and fast footstep planning: 1) for each input of a (3D) grid of 1,000,000 regularly spaced points, precompute the corresponding swept volume expanded by $10cm$, i.e. the set of points at distance at most $10cm$ from the actual swept volume; 2) use the whole continuous input space for footstep planning, and when collisions must be checked, instead of using the trajectory of the half-step $(SF_x(0), SF_y(0), SF_y(0))$ actually executed, use the precomputed expanded swept volume of the nearest neighbor of $(SF_x(0), SF_y(0), SF_y(0))$ among the 1 million points.

The theoretical bounds given by theorem 4.3.2 ensure the soundness of the approach (modulo the drift at execution time): if no obstacle intersects the precomputed expanded swept volume, then the actual half-step $(SF_x(0), SF_y(0), SF_y(0))$ also avoids the obstacles. Unfortunately with a standard computer, typical swept volume approximations take about 20 to 40s (see [Kim et al., 2003]) and would lead to unreasonable computation time for 1 million points. As a result using a finite number of swept volume approximations in a continuous footstep planning framework will not be made possible by the theorems proved in this section.

Nevertheless, by quantifying the relation between the uncertainty on the input parameters and the uncertainty on the distance to collision, we obtained theoretical upper bounds on how much a small perturbation of the input parameters can affect the trajectories of the physical points of the robot in the worst case. Such bounds have never been used in humanoid robotics, and they could be useful in scenarios that are different from the ones originally planned: for instance it is not compulsory to cover the whole input space; in [Chestnutt et al., 2007] and [Chestnutt et al., 2006], a discrete transition model (the authors call it action set) is used with the possibility to slightly modify any of the fixed transitions (actions). This corresponds to a finite cloud of small neighborhoods in the input spaces, and our approach could enable to deal with it with guarantees that do not exist yet in the literature.

Table 4.1: Numerical bounds with standard parameters

Number of points needed in order to cover the the whole input space:	Corresponding margin:
1,000	54.1cm
10,000	29.3cm
100,000	16.5cm
1,000,000	9.4cm
10,000,000	5.4cm
100,000,000	3.0cm

With further analyses, it might also be possible to tighten the bounds (e.g. taking into account the fact that the modifications of the position of the feet are horizontal), therefore reducing the necessary number of points. In particular, if instead of the whole input space we choose to study only trajectories in the vicinity of a reference trajectory, we can easily find better values for λ_{min}, λ_{max}, γ_{max}. As a result the bound obtained is likely to be much tighter.

We could also use the functions that appear in our results to decide of a good basis of primitive functions and then use them in a learning process whose objective would be to build a model of the walking pattern generator sensitivity. This could lead to new bounds, with no certainty but tighter than the theoretical ones and with probably higher practical utility.

4.4 Conclusion

In this chapter we presented a new walking pattern generator based on half-steps, and analyzed its sensitivity. Since 3 parameters entirely define a half-step, unlike with state-of-the-art walking pattern generators it will be possible to easily approximate (through offline simulations) the domains of feasibility of the half-steps. Our walking pattern generator also benefits from concatenation operators that can continuously produce an overlap between consecutive half-steps. Starting from a concatenation of half-steps with zero speed at both extremities, by using these operators we progressively obtain a much smoother and more satisfying trajectory. As we will see in the next chapter, the continuity of this process can bring more coherence to footstep planning based on offline approximations.

Chapter 5

Discrete Footstep Planning

In this chapter we start tackling the problem of footstep planning itself, and give an efficient and coherent solution based on a discrete approach. It corresponds to the method described in [Perrin et al., 2011a].

As explained previously, the state-of-the-art approaches for footstep planning are based on the use of the A* algorithm with a finite set of possible steps (see [Kuffner et al., 2001], [Bourgeot et al., 2002], [Chestnutt et al., 2003], [Chestnutt et al., 2005]) called the transition model, and the A* algorithm strongly constrains the size of this transition model (because it performs poorly with many transitions). As a result the stepping capabilities actually used are often much less expressive than the actual stepping capabilities of the biped robot. In this chapter, we keep a discrete approach but we replace the A* search by a sampling-based algorithm in order to directly deal with a large transition model, and we add several other improvements to the standard $\{A^* + \textit{finite transition model}\}$ approach. Here are the key points of our approach:

- The walking pattern generator introduced in Chapter 4 has a low-dimensional search space which can be densely covered by relatively few points. With an automatically generated finite transition model of about 300 points in this search space, we are able to obtain very expressive stepping capabilities. To deal with such a large transition model, we use, instead of the classical A* search, a specific Rapidly-exploring Random Tree (RRT) algorithm.

- Each point in the transition model corresponds to a configuration space trajectory of the robot. Through extensive offline computations, for each of them we approximate (using the algorithm introduced in Chapter 3) the volume swept in the workspace by a part of the robot lower body (from the knees down) during the execution of the trajectory, and store it in an efficient data structure. It helps to drastically reduce the time consumed by the online planning phase when checking for collisions with the environment.

- Finally, with the homotopy provided by the walking pattern generator, we quickly smooth and accelerate the trajectories obtained after the planning phase, and as a result the final motions produced are fully dynamic. On top of that, there is no incoherence between the planning phase and the smoothing phase, so we

have the guarantee that if the planner returns a collision free solution, then the robot will execute a sequence which will also be collision free (this guarantee is up to some details –discrepancies between simulation and real world, errors of approximation, errors due to discretization, etc.–).

Let us give more details about the transition model and the specific RRT algorithm:

Finite transition model and swept volume approximations
We use about 300 points to cover an input space of 3 dimensions. Here each point corresponds to a sequence of two half-steps. For each point of this grid we first simulate the sequence of half-steps and check that it is feasible, i.e. that it contains no self-collision and does not violate the joints limits. The points which correspond to feasible trajectories will be the elements of our transition model. In section 5.1 we explain the construction of this transition model and show how, for each of its elements, we approximate the volume swept by the robot lower body during the execution of the corresponding trajectory. These approximations will enable us to save a considerable computation time online.

We use the walking pattern generator of Chapter 4 and at first it might seem strange to combine precomputed swept volumes and a smoothing homotopy that modifies trajectories, but in fact in the whole process the homotopy is only applied to one *feasible* trajectory returned by the planning process during which the swept volume approximations are extensively used. When the homotopy is applied we do not use precomputed swept volumes for the collision checks.

Several efficient swept volume approximation algorithms exist, such as for example the ones introduced in [Kim et al., 2003] and in [Himmelstein et al., 2009]. Using such advanced specific algorithms will be part of our future work, but in this chapter we validate our framework with a simpler approach. Since the highest priority is the evaluation time (because approximations are used multiple times at each iteration of the RRT algorithm), the approximation algorithm introduced in Chapter 3 is perfectly suitable for this problem: it stores the swept volume approximations in compact tree structures that can be used to very quickly check for collisions with obstacles of the environment. The use of swept volumes is widespread in robotics, especially for path planning (see [Schwarzer et al., 2002], [Hasegawa et al., 2003]), but relatively absent in the field of humanoid robotics, where, for the sake of computational efficiency, simpler bounding volumes are often preferred ([Yoshida et al., 2005], [Elmogy et al., 2009]).

An RRT variant for footstep planning
The last part of our framework is the planning phase. Since we have a large transition model, the traditional A* search would perform poorly. Alternatives to A* have already been proposed. For example in [Harada, 2010], Harada uses a PRM (Probabilistic Roadmap Method, see [Kavraki et al., 1996]) approach to plan footsteps: a tree of "milestone configurations" is grown from an initial configuration to a goal configuration. At first collisions are checked only at the milestone configurations, and only once a candidate path is found, the full trajectory is verified. An issue of this approach is that even though the milestones are collision-free, collisions might occur in the candidate path. Thus the process might have to be restarted several times, leading to lengthy computations.

The idea of using an RRT algorithm ([LaValle and Kuffner, 2000]) for footstep planning was introduced in [Xia et al., 2009], where single-node-extending and multi-

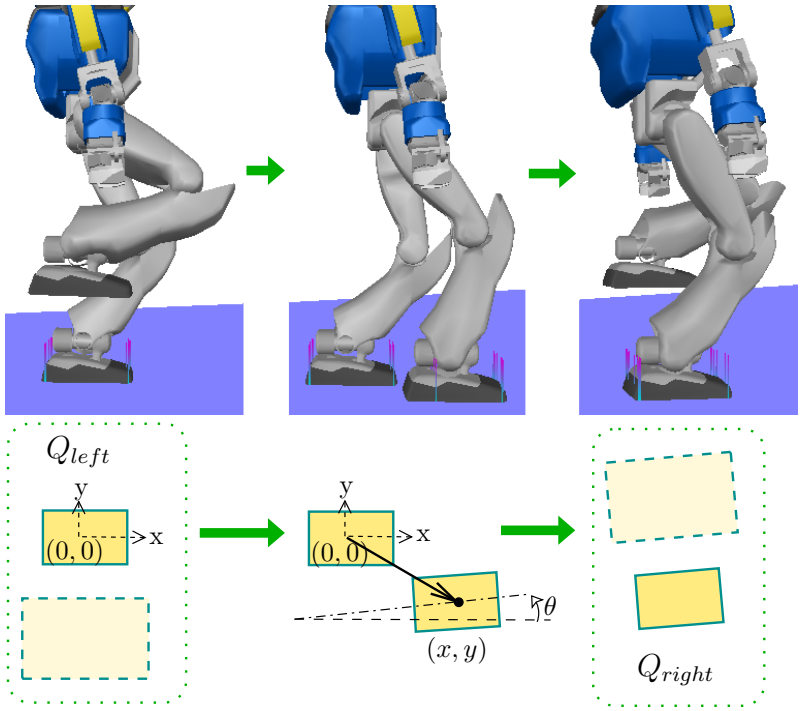


Fig. 5.1: A raw sequence of two half-steps is entirely defined by three parameters: x, y, and θ. The via point configurations Q_{left} and Q_{right} were chosen such that the swing foot is quite high. This provides good obstacle avoidance capabilities to raw sequences of steps and in the absence of obstacles, smoothing significantly reduces the height.

node-extending RRT methods are proposed. In section 5.2 we follow the single-node-extending method and present a new variant of the RRT algorithm for footstep planning, where we deal separately with the sets of left and right footsteps. When a new transition (i.e. a new footstep) is considered by the RRT algorithm, we test the corresponding approximated swept volume against the objects that are close enough, and discard the transition if collisions are found.

The whole framework was used on the robot HRP-2 to quickly solve (in less than 30s) complicated problems of footstep planning in an environment cluttered with 3D obstacles. The first experimental results are presented in Section 5.3. Then, in Section 5.4, we present a more advanced integration of this framework in experiments of online replanning where the position of the robot and the obstacles are acquired through motion capture. This precedes a brief discussion on the extension of our framework to a continuous transition model (Section 5.5).

5.1 Building the transition model and the swept volume approximations

5.1.1 The transition model

Thanks to the walking pattern generator described in the previous chapter, we can produce isolated half-steps with only three parameters. If we join a downward half-

step with the corresponding upward half-step, we obtain a trajectory that goes from Q_{left} to Q_{right} or Q_{right} to Q_{left}, and which is entirely defined by only three parameters, as shown on Fig. 5.1. We denote such a trajectory (expressed in the frame of the left foot) by $\langle Q_{left}, (x, y, \theta), Q_{right} \rangle$ or (expressed in the frame of the right foot) $\langle Q_{right}, (x, y, \theta), Q_{left} \rangle$. We also pose:

$$\mathcal{T}_l = \{\langle Q_{left}, (x, y, \theta), Q_{right} \rangle \mid (x, y, \theta) \in \mathbb{R}^2 \times SO(2)\},$$

and:

$$\mathcal{T}_r = \{\langle Q_{right}, (x, y, \theta), Q_{left} \rangle \mid (x, y, \theta) \in \mathbb{R}^2 \times SO(2)\}.$$

We will interchangeably call the elements of $\mathcal{T}_l$ or $\mathcal{T}_r$ points, transitions (because the transition model will be a finite set of elements of $\mathcal{T}_l$ and $\mathcal{T}_r$), sequences (each element corresponds to a *downward half-step - upward half-step* sequence), or trajectories. By concatenating alternatively trajectories from $\mathcal{T}_l$ and trajectories from $\mathcal{T}_r$, we obtain walk motions. With a symmetric robot (like HRP-2), $\mathcal{T}_l$ and $\mathcal{T}_r$ are symmetric in the sense that the feasibility of a sequence $\langle Q_{left}, (x, y, \theta), Q_{right} \rangle$ is equivalent to the feasibility of the sequence $\langle Q_{right}, (x, -y, -\theta), Q_{left} \rangle$, and that the corresponding swept volumes are symmetric. Therefore, only one transition model was built, on the space $\mathcal{T}_l$, but it can be used by symmetry on $\mathcal{T}_r$. To build it, as explained on Fig. 5.2 we first covered a reasonably large domain of $\mathcal{T}_l$ with regularly spaced points. Considering the robot (HRP-2) dimensions and joint limits, this domain was defined as the following box:

$$\mathcal{B}_l = \{\langle Q_{left}, (x, y, \theta), Q_{right} \rangle \mid x \in [-0.35m, +0.35m],$$
$$y \in [-0.37m, -0.02m], \theta \in [-30^\circ, +30^\circ]\}$$

We covered the box $\mathcal{B}_l$ with 600 points (15 possible values for x, 8 possible values for y, 5 possible values for θ), and for each point, using discretized trajectories –one for each body of the robot legs–, we verified the feasibility of the corresponding *downward half-step - upward half-step* sequence. If any self-collision (which were checked using the algorithm introduced in [Benallegue et al., 2009]) or joint limit violation occurred, the point was discarded.

The 276 remaining points all correspond to feasible sequences, and they form the transition model.

We denote by $\mathcal{M}_l \subset \mathcal{B}_l$ this finite transition model, $\mathcal{M}_r \subset \mathcal{B}_r$ being defined by symmetry. We denote by $\mathcal{S}(\mathcal{M}_l, \mathcal{M}_r)$ the set of finite *feasible* sequences $(s_1, s_2, \dots s_n)$ alternating left and right support foot.

5.1.2 The swept volume approximations

For each of the 276 points of the transition model, we build an approximation of the volume swept by the lower part of the robot (from the knees down) during the corresponding *downward half-step - upward half-step* sequence. The algorithm used is the one introduced in chapter 3: given a transition $z \in \mathcal{M}_l$ it approximates through adaptive sampling the sign of the mapping $C_z(\mathbf{p})$ which returns the distance (minus a fixed margin –1cm in our case–) between a point $\mathbf{p}$ of the Euclidean space and the finite set of polyhedra consisting of all the configurations of the robot legs bodies along their discretized trajectories during the sequence corresponding to z.

The result of the approximation is stored in a tree structure which can be evaluated extremely quickly. The computation time saved as a result is considerable: with the

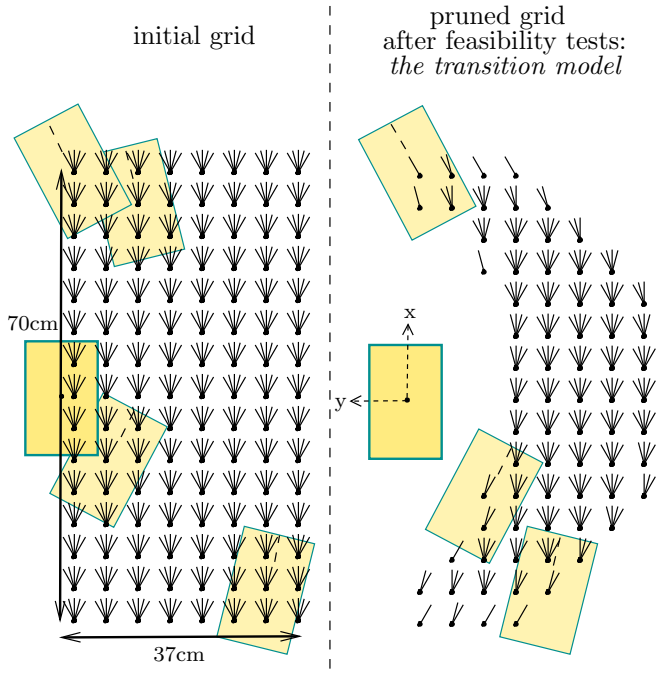


Fig. 5.2: An initial grid of 600 points covers $\mathcal{B}_l$. To each of the 120 values of (x, y) correspond 5 possible orientations. All the corresponding trajectories (generated by the walking pattern generator presented in Chapter 4) are sequences of two half-steps. We test each of them, checking for self-collisions and joint limit violations, and remove all the unfeasible ones. The 276 remaining points form the transition model $\mathcal{M}_l$ used for planning.

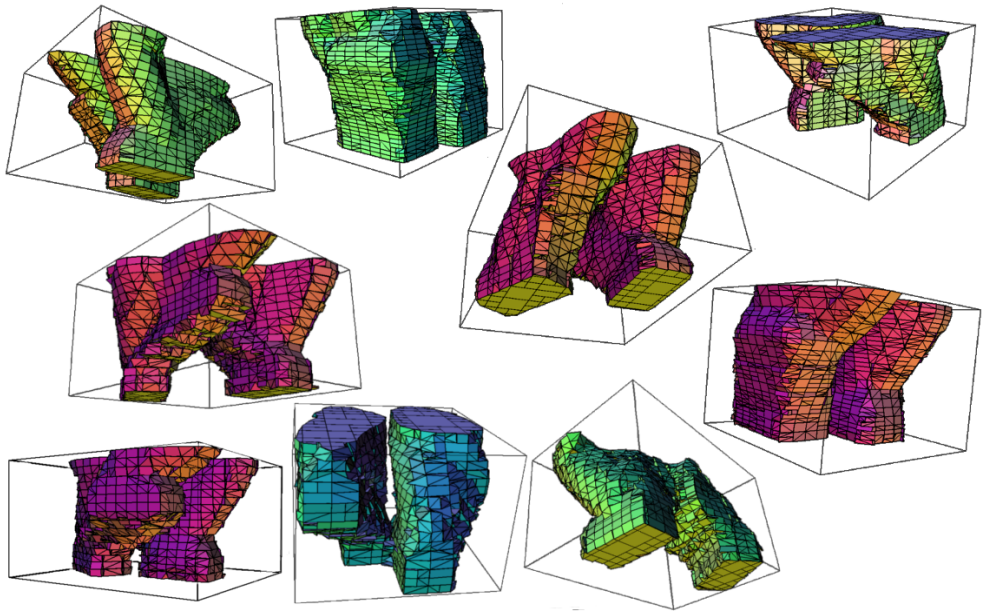


Fig. 5.3: 3D representations of swept volumes approximations (we show the zero-level surfaces of the approximations).

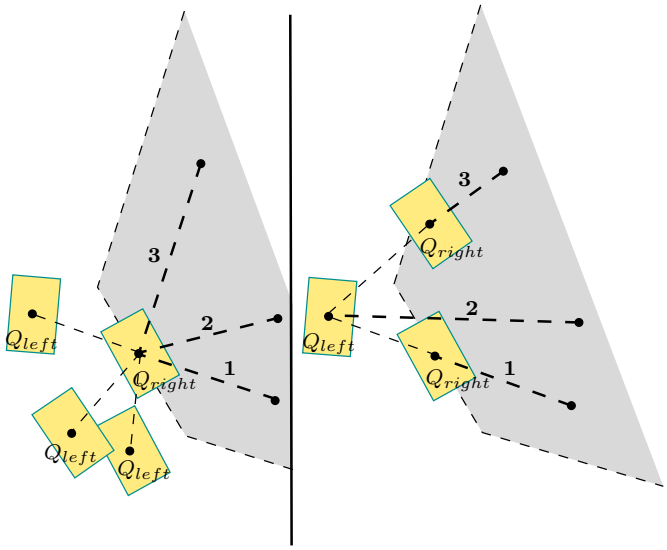


Fig. 5.4: The advantage of separating left and right support feet during nearest neighbor queries.
- *On the left* (global nearest neighbor): all the points in the gray region have the same nearest neighbor $(Q_{right}, x, y, \theta)$, but no successor of $(Q_{right}, x, y, \theta)$ is inside the gray region. Therefore numerous samples are required before expanding the search tree towards the gray region.
- *On the right* (alternate nearest neighbors): when only states with left support foot are considered, the nearest neighbor will not be $(Q_{right}, x, y, \theta)$, but maybe one of its successors. With the alternation strategy, the search tree is more likely to quickly grow inside the gray region.

approximation, checking whether a point is outside or inside one of the swept volumes we consider is done in 4μs. This is about 2,000 times faster than with the normal evaluation of $C_z(\mathbf{p})$.

For a transition $z = \langle Q_{left}, (x, y, \theta), Q_{right} \rangle \in \mathcal{M}_l$, we denote by $V_z(\mathbf{p})$ the corresponding swept volume approximation ($V_z(\mathbf{p}) > 0$ if and only if $\mathbf{p}$ is outside the approximated swept volume). If $z' = \langle Q_{right}, (x, -y, -\theta), Q_{left} \rangle \in \mathcal{M}_r$, we can easily obtain the approximation $V_{z'}$ by applying a symmetry to V_z; thus only 276 swept volume approximations are needed. With an Intel(R) Xeon(R) 2.00Ghz CPU, it took a bit less than 48 hours to generate them all, but we believe that by using state-of-the art swept volume approximation algorithms (and maybe only afterwards apply our algorithm to obtain reapproximations that can be evaluated very fast), we should be able to significantly reduce this offline computation time.

Fig. 5.3 shows some of the 276 swept volume approximations.

5.2 Footstep planning with a variant of RRT

In this section, we present a simple adaptation of the RRT algorithm for footstep planning, quite similar to the one introduced in [Xia et al., 2009].

Let us first define the search space. Since in our formalism we connect single support phases, the search space is $S = \{(q, x, y, \theta) \mid q \in \{Q_{left}, Q_{right}\}, (x, y, \theta) \in \mathbb{R}^2 \times$

Algorithm 3 RRT variant for footstep planning

$\mathbb{T}$.init($x_{init} \in S_{|\mathcal{E}}$)
$i \leftarrow 0$
stop_condition $\leftarrow$ false
while $\neg$*stop_condition* **do**
  Pick a random state $x_{rand} \in S_{|\mathcal{E}}$
  i++
  if $i == 0 \bmod 2$ **then**
    $x_{near} \leftarrow \{$ among states with **left** support foot, nearest neighbor of x_{rand} in the tree $\mathbb{T}$
    Pick a random transition $s_{rand} \in \mathcal{M}_l$.
  else
    $x_{near} \leftarrow \{$ among states with **right** support foot, nearest neighbor of x_{rand} in the tree $\mathbb{T}$
    Pick a random transition $s_{rand} \in \mathcal{M}_r$.
  end if
  Using the approximated swept volumes, verify that starting from state x_{near}, the transition s_{rand} does not collide with any point of the obstacle point clouds.
  if NO COLLISION **then**
    $\mathbb{T}$.add_node($\delta(x_{near}, s_{rand})$)
    $\mathbb{T}$.add_edge(x_{near}, s_{rand}, $\delta(x_{near}, s_{rand})$)
    if $\delta(x_{near}, s_{rand})$ is close enough to the goal and the path to $\delta(x_{near}, s_{rand})$ is short enough **then**
      stop_condition $\leftarrow$ true
    end if
  end if
end while

$SO(2)\}$, where q is the support foot, (x, y) the position of the support foot (relatively to a fixed reference), and θ its orientation (relatively to a fixed reference). We pose $S_l = \{(Q_{left}, x, y, \theta) \mid (x, y, \theta) \in \mathbb{R}^2 \times SO(2)\}$ and $S_r = \{(Q_{right}, x, y, \theta) \mid (x, y, \theta) \in \mathbb{R}^2 \times SO(2)\}$. The transition model being an alternation between $\mathcal{M}_l$ and $\mathcal{M}_r$, we can apply transitions to states of the search space using the operator $\delta : S_l \times \mathcal{M}_l \cup S_r \times \mathcal{M}_r \rightarrow S$:

$$\begin{aligned}&\delta\left((q, x, y, \theta), \langle q, (x', y', \theta'), \overline{q}\rangle\right)\\ &\quad = (\overline{q}, x'cos(\theta) - y'sin(\theta), x'sin(\theta) + y'cos(\theta), \theta + \theta'),\end{aligned}$$

where $\overline{Q_{left}} = Q_{right}$ and $\overline{Q_{right}} = Q_{left}$. In practice, we will use only a compact subset of the search space, depending on the environment $\mathcal{E}$. We denote it by $S_{|\mathcal{E}}$. For example, if the robot stays in a $5m \times 5m$ room, we naturally use these dimensions to define $S_{|\mathcal{E}}$ and bound x and y. Considering the classical RRT algorithm (see [LaValle and Kuffner, 2000]), the only operation that cannot be straightforwardly adapted to the context of footstep planning is the extension towards random samples (to find the nearest neighbor we use the Euclidean metric, ignoring the orientations). Let $(q, x, y, \theta) \in S$ be a uniform random sample of the search space, and (q', x', y', θ') the nearest state in the search tree. In [Xia et al., 2009], two options are considered: either add to the tree all the successors of (q', x', y', θ'), or just one random successor. Due to the size of our transition model, we chose to follow the latter strategy. Fig. 5.4 shows one issue of this approach: in some cases, it is difficult to expand the search tree

towards a given region. To cope with this problem, many options are possible. We simply chose to alternatively look for nearest states with left support foot and nearest states with right support foot. It leads to our RRT variant presented in Algorithm 3 (the while loop stops when a *sufficiently short* path to the goal region has been found, "*sufficiently short*" being defined by a simple heuristic). We based our implementation on a fast and modular open-source code by Karaman and Frazzoli which uses kd-trees for fast nearest neighbor queries (this code implements RRT and RRT*, the algorithm introduced in [Karaman and Frazzoli, 2010]).

Further analyses and improvements of the variants of RRT for footstep planning can probably help to obtain faster results, but are out of the scope of this thesis.

5.3 Preliminary experimental results

The framework presented in this chapter was experimentally tested on the robot HRP-2.

We studied the two Experimental Setups described on Fig. 5.5, where 2D obstacles (holes in the ground) are combined with 3D obstacles. The 3D obstacles shown on Fig. 5.5 have the same size as the ones in the real environment (see Fig. 5.6), but are smaller than the ones used for the collision checks (a margin is needed because of the robot drift during the real-world experiments).

The construction of the solution trajectory is divided into two parts: first, during the planning phase, just as explained in the previous section, we use a specific variant of RRT to find a sequence $(s_1, s_2, \ldots, s_n) \in \mathcal{S}(\mathcal{M}_l, \mathcal{M}_r)$ which reaches the goal. Then, we use the homotopy of Section 4.2.2 to smooth the sequence $(s_1, s_2, \ldots, s_n)$, so that to obtain the final fast and dynamic trajectory that will be performed by the robot.

5.3.1 The planning phase: RRT vs. A*

We implemented a classical A* search algorithm and compared it with the RRT variant introduced in the previous section. For the costs required by A* we used a simple heuristic where the estimated remaining cost is derived from the Euclidean distance, and the cost of a path is the sum of each (fixed) transition cost. Better heuristics exist, such as for example heuristics derived from a mobile robot planner that looks for continuous paths between the initial position and the goal, but because they do not take stepping over capabilities into account, such heuristics tend to severly misjudge costs in very constrained environments like the ones we consider here (for a review on the association { A* + heuristic } see [Chestnutt, 2007], ch. 8). Finding a robust heuristic that would perform well in challenging environments is as hard as solving the problem without using A*: that is why we tried to directly apply RRT. Other approaches of interest include planning algorithms based on inflated heuristics (see [Gonzalez and Likhachev, 2011]): they usually find solutions faster than a classical A* search, but they are not as efficient as RRT to avoid local minima. Their main advantage over RRT is that they provide suboptimality bounds; however, due to the particularity of the problem of footstep planning, it is not clear whether such bounds can still be obtained in our context. Finally it might be interesting to try to adapt control-based strategies such as [Sucan and Kavraki, 2008], but the adaptation would be far from straightforward.

In Setup 1 and 2 (cf. Fig. 5.5) we fixed an upper bound, and stopped the execution of RRT or A* as soon as a path of cost smaller than this upper bound was found.

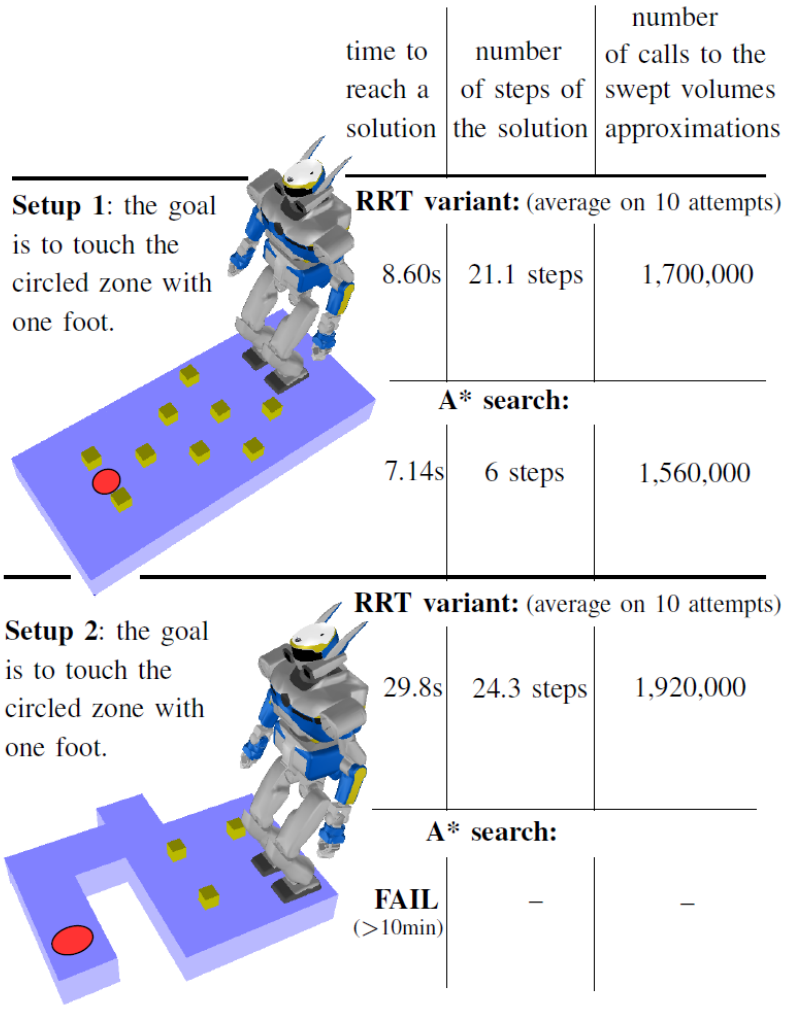


Fig. 5.5: Experimental setups and results of the planning. The computations are made with an Intel(R) Xeon(R) 2.00Ghz CPU. Remark: in Setup 1, the exterior surface of the 3D obstacles (the boxes on the ground) is covered by 250 points. In Setup 2, the exterior surface of the 3D obstacles is covered by 75 points.

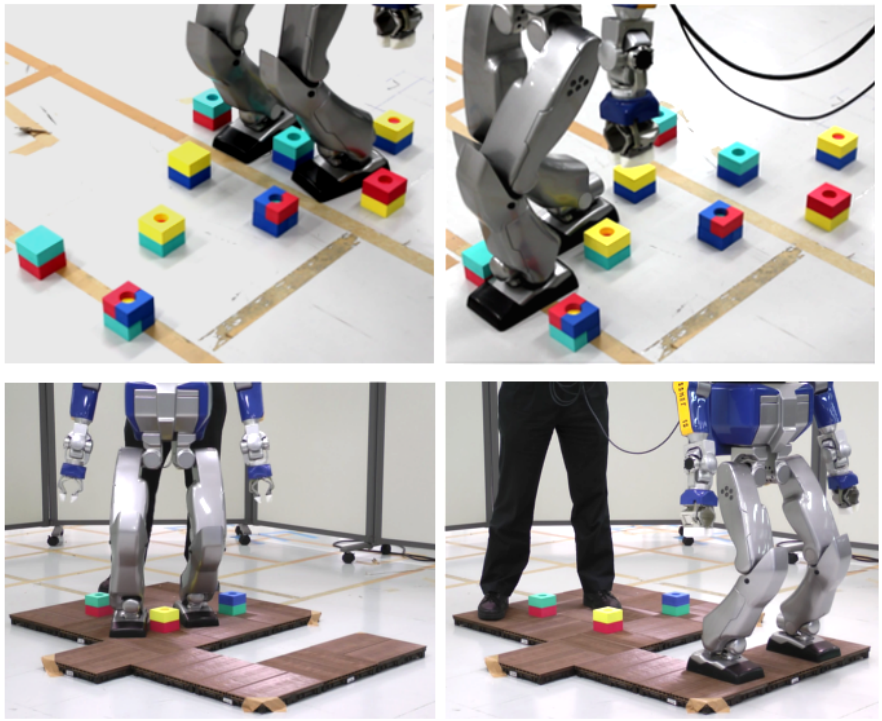


Fig. 5.6: The robot HRP-2 executing planned trajectories. *Above*: the so-called toy problem of walking in a child's bedroom avoiding toys on the ground.

As shown by the results on Setup 1, without strong local minima, the time needed by RRT and A* to find a solution is approximately the same, but A* finds a better trajectory cost (it finds solutions with fewer steps).

On the other hand, we can see with the results on Setup 2 that when the transition model is large, A* seems much more sensitive to local minima than RRT: indeed A* fails to find a solution on Setup 2, whereas the RRT method consistently finds solutions in less than 40 seconds (and 29.8 seconds in average).

This is easily explainable because A* usually has to explore a subtree of fixed height h (which depends on the heuristic costs used) before being able to avoid a local minimum. Therefore it will try about $(|\mathcal{M}|^h - 1)\left(\frac{|\mathcal{M}|}{|\mathcal{M}|-1}\right)$ transitions ($|\mathcal{M}|$ being the size of the transition model) before overcoming the local minimum. This can be done if both $|\mathcal{M}|$ and h are relatively small, but since in our case $|\mathcal{M}| = 276$, the complexity can quickly become prohibitively large.

As a randomized approach, RRT does not have this caveat, and that is why we think it is more suitable than A* when the transition model is large.

A remark on the time saved thanks to the swept volume approximations: on the Setup 1 whose environment contains a lot of points (250), we can see that during their execution both the RRT variant and A* make about 200,000 calls to a swept volume approximation every second. Without the approximations, these 200,000 calls would be replaced by more than 26 minutes spent in collision checking.

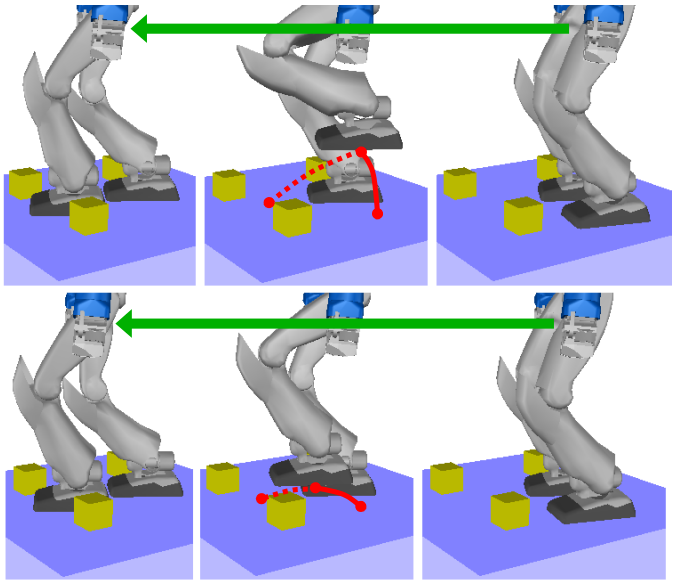


Fig. 5.7: *Above*: a raw sequence of two half-steps. *Below*: the smoothed sequence. When there are no obstacles, the swing foot trajectory of the smoothed sequence depends on the minimum time between two ZMP shifts, which is fixed in advance in order to bound the speed of the feet.

5.3.2 The smoothing phase

Once a trajectory avoiding the obstacles has been found by the planner, since it consists in a concatenation of isolated half-steps, we can use the homotopy described in Section 4.2.2 to smooth it. One overlap parameter has to be set for each pair of consecutive half-steps, and since the overlaps are independent, they can be set sequentially. This means that we can start to execute the trajectory on the robot even if only a few initial overlaps have been set, the next overlaps being computed during the execution of the trajectory. Let us notice that the dichotomy search for the best overlap time is an "any-time process" that can be interrupted if the computation time is too long, the current result being anyway not worse than the initial raw motion. Another important remark: since we cannot know in advance the swept volumes for the trajectories involved in the smoothing processes, we have to use classical collision checks. We measured the overlaps computation time for 10 raw sequences of half-steps obtained in Setup 1, and 10 raw sequences obtained in Setup 2. In all cases, the duration of the smoothing was less than the final trajectory execution time. For the solutions in Setup 1, the average time needed for the smoothing was 14.4s, and the average execution time of the final trajectory was 31.1s. For the solutions in Setup 2, the average duration of the smoothing was 13.4, and the average execution time of the final trajectory was 41.6s. These results validate the possibility of smoothing trajectories while they are being executed. Fig. 5.7 illustrate the effect of the smoothing on the foot trajectories.

5.4 Real-time replanning experiments

In this section, we briefly present how the techniques for fast footstep planning presented throughout this chapter have been implemented and tested in real-time replan-

ning experiments with the robot HRP-2, where the obstacles and the robot position are acquired by motion capture. This implementation and these experiments are also briefly mentioned in [Perrin et al., 2011a], and more precisely described in [Baudouin et al., 2011].

In these experiments, we used two distinct computers to plan and execute the robot motions: one computer for the planning, and one for the control, the latter being equipped with a real-time operating system. A CORBA server organizes and transfers communications between the 4 units of our architecture: the two computers just mentioned, the motion capture system responsible for obstacles and robot localization, and a viewer that shows in real-time the new paths found by the robot.

The control of the robot motion is made with a module called Stack of Tasks (see [Mansard and Chaumette, 2007], [Mansard et al., 2009]) which is a structure managing priorities between the active controllers. In an execution thread, the planned trajectory is progressively sent to the control part, while the localization information is read to check for potential collisions. If a collision is detected along the planned trajectory, a query is sent to the planner to generate a new path that tries to link the part of the current trajectory before the expected collisions to the part of the current trajectory after the expected collisions.

The planner sends four data flows. The first three flows are: 2D obstacles (which are mostly used to simulate artificial obstacles), the current goal position, and the current sequence of steps planned. They are only used by the viewer. The fourth signal is the most important and contains leg joint trajectories as well as CoM and ZMP trajectories.

The main difficulty for the CORBA server is to manage communications between two systems running at different speeds. The control loop reads inputs every 5ms, and the planner is obviously asynchronous. To cope with this issue the controller uses a buffer to handle the large vectors sent by the planner. When the environment is not changing, the planner uses its free time to try to improve parts of the current path, or to smooth the currently planned sequences of steps.

The two-phase approach of the trajectory generation is very convenient for online replanning because we control the independence between half-steps in the sense that the smoothing between two consecutive half-steps can easily be canceled. For instance it is easy to "unsmooth" a part of the currently planned sequence in order to delay a step, or to modify it without modifying the previous steps. Since we approximately know the time required for the smoothing (it does not vary a lot), resmoothing parts of the trajectory can be done while the robot is performing the motion.

The way we dealt with collisions in the experiments of the previous section is clearly not optimal: we represented obstacles by covering them with points on their exterior surface, and all the points were always taken into account. The results showed that the swept volume approximations can be called a great number of times in a short period, proving that significant speed-up can be obtained compared to frequent collision checks along *a priori* unknown trajectories. What is more, in some cases, point clouds are a very natural input, and it would be interesting to see if we can organize them in a good structure so that to use our approximation functions in an efficient way. This is beyond the scope of this thesis, but we can already obtain better results by using state-of-the-art collision detection algorithms. First, we can notice that our swept volume approximations are defined by intersections of small boxes with planes (we chose affine functions for the local approximations). Thus, it is easy to construct meshes that describe the swept volume approximations (we actually use simplified

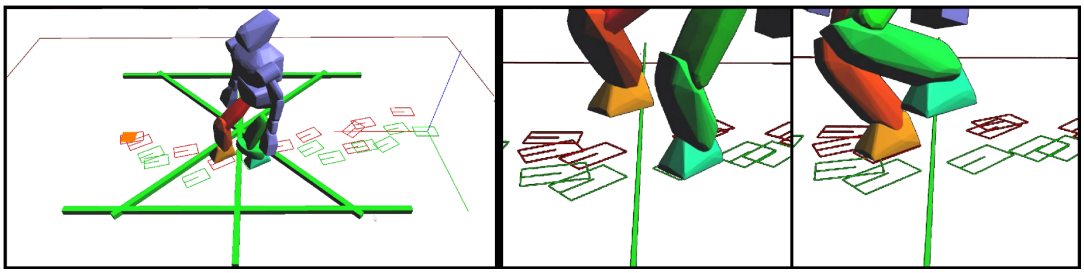


Fig. 5.8: On the left: in simulation, HRP-2 stepping over bars of section 8cm×8cm. On the right: HRP-2 stepping over an horizontal cable 10cm above the ground.

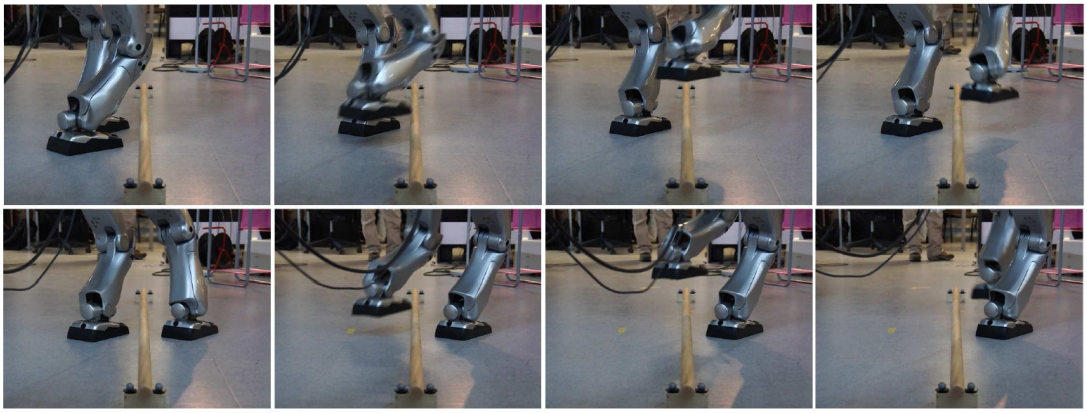


Fig. 5.9: Details on the stepping over of a bar placed 8cm above the ground.

meshes, i.e. they have a slightly simpler geometry than the initially precomputed approximations). With these 276 meshes, we use the PQP algorithm [Larsen et al., 2000] for collision checks, and during the smoothing process we also use PQP to check for collisions between the environment and convex hulls of the robot bodies (using convex hulls is a bit more conservative and slightly reduces the number of triangles). One of the main advantage we obtain by doing so is that PQP stores the obstacles in bounding volume hierarchies that reduce the complexity of collision checks.

With this method a significant speed-up is reached: with the Setup 2 of Fig. 5.5, we performed 1000 trials with a slightly faster CPU (Intel(R) Xeon(R) 2.40GHz) but overall in similar conditions. A solution was always found, and the average time required was only 1.60 seconds, which is almost 20 times faster than the preliminary results. The average number of steps of the solution was 28.5 steps, and in average 18,000 collision checks were needed before finding a solution.

Fig. 5.8, Fig. 5.9, Fig. 5.10 and Fig. 5.11 illustrate some of our results in experiments or simulations. The goal and 3D obstacles can be moved in real-time, and the robot tries to adapt its trajectory consequently. An interesting point of the results is that even when stepping over motions are required, the robot can often replan a trajectory very quickly. However, there is some latency and the robot can typically replan the trajectory only 1 or 2 steps after the current one.

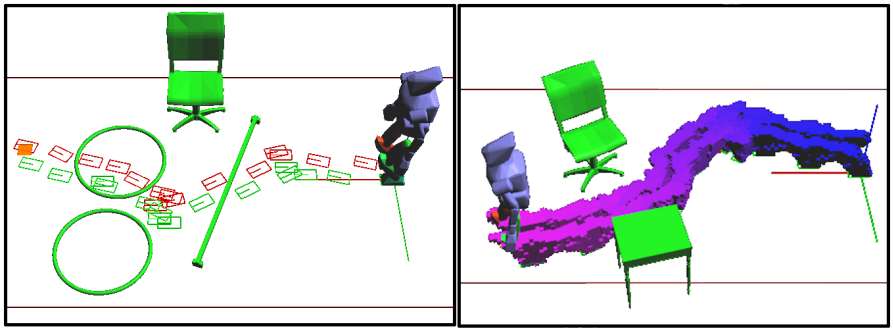


Fig. 5.10: On the left: a sequence of steps found in a complex environment. On the right, we show for one sequence of steps the concatenation of the swept volumes made of simplified meshes obtained from the original swept volume approximations. For the upper body simpler bounding boxes are used for the collision checks.

5.5 Discussion on an extension to continuous transition models

Even if the expressiveness of a continuous transition model can be approached by the one of a large finite transition model, a continuous transition model would still be preferable.

Several useful techniques would indeed be easier to apply with a continuous transition model: local footstep modifications ([Chestnutt et al., 2007], [Chestnutt et al., 2006]), extraction of convex regions in the transition model in order to use optimization techniques to determine foot placements ([Herdt et al., 2010]), path deformation ([Jaillet and Siméon, 2006]), etc.

RRT and other sampling-based algorithms (e.g. PRM, see [Kavraki et al., 1996]) would be easier to adapt with a continuous transition model, so it would cause no problem at the planning phase. Besides, it would not be difficult to approximate the feasibility regions so as to obtain continuous transition models $\mathcal{M}_l$ and $\mathcal{M}_r$ (although it might be hard to obtain the guarantee that all transitions are indeed feasible). But then, the main issue would be the need to approximate swept volumes that depend on a continuous parameter $z \in \mathcal{M}_l$, and as we have seen a bit in Chapter 4, this is likely to be very computationally challenging. In that case rather than trying to compute the swept volumes more efficiently, other collision detection routines should be taken into account, such as continuous collision detection [Zhang et al., 2007], GPU-based approaches [Lauterbach et al., 2010] or other variants (e.g. [Tang et al., 2010], [Schmidl et al., 2004], ...). But instead of exploring this direction, in the next chapter we present a completely different approach for continuous footstep planning (although based on the same walking pattern generator).

5.6 Conclusion and future work

In this chapter, we have described a novel and coherent framework for fast footstep planning based on a finite but large transition model and on precomputed swept volume approximations. The experiments realized in changing environments show some promising results: although planned very quickly the trajectories seem quite natural: no pauses, no exaggerated motions to avoid small obstacles, and a large diversity of

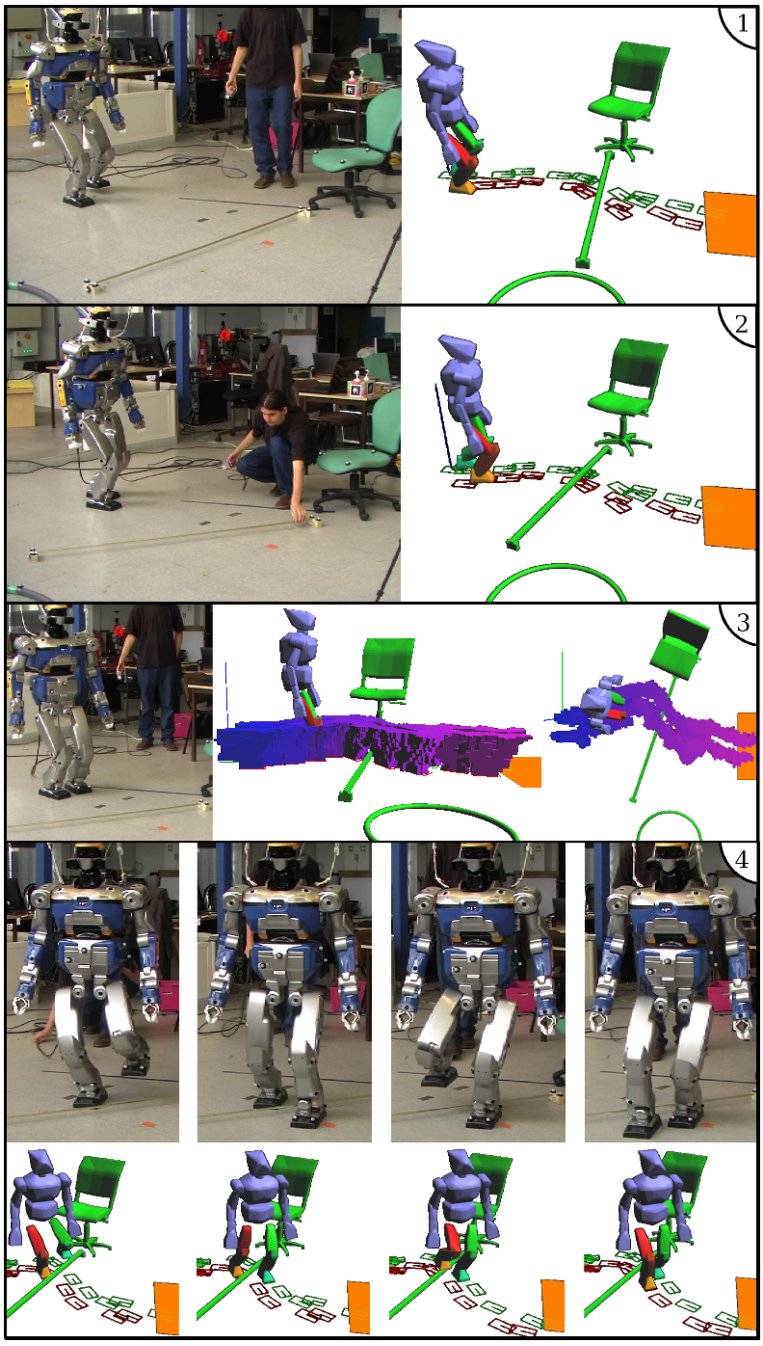


Fig. 5.11: In this experiment a bar placed 5cm above the ground is moved while the robot is executing its initial plan. (1): HRP-2 starts to execute the sequence initially found. (2): the bar is suddenly moved, and the current sequence of step would lead to collisions. (3): while walking, HRP-2 is able to compute a new sequence of steps towards the goal (we show the concatenation of the swept volumes which indeed avoid the bar). (4): the robot finally steps over the bar while at the same time it tries to optimize the rest of the path towards the goal. Remark: due to uncertainty on positions, we use a model of bar that is thicker than the actual one.

foot placements. But they also show some limitations: for instance our approach is not fast enough yet to provide modifications of the current step when obstacles suddenly appear, and the trajectories found are not really optimal in terms of number of steps.

In future work we plan to develop new footprint planning techniques well adapted to large transition models, possibly using parallel computations. Our hope is to obtain a significant speed-up of the online planning phase, but this will not be useful without developping also efficient and safe techniques for real-time short-term trajectory replanning for the swing foot, ZMP and CoM.

Chapter 6

Continuous Footstep Planning

As we have seen, the problem of footstep planning, which has mostly a discrete nature, is usually solved by making it completely discrete (by choosing a finite set of possible steps). In this chapter we do the contrary and make it continuous.

In the previous chapter, one of the difficulties observed was the adaptation of RRT to the problem of footprint planning. In classical rigid body motion planning problems, we look for continuous paths, while in footprint planning we look for discrete "jumps" between configurations. Because of this intrinsic difference the classical methods for rigid body motion planning are rather difficult to apply to the problem of footprint planning (see [O'Kane and LaValle, 2004] which examines the difficulties of applying RRT to discrete search spaces). Yet, there are numerous efficient rigid body motion planning algorithms which advantageously use the connectedness of the configuration space, and it would be interesting to find a way to apply them to footstep planning. It is possible to do it with tiered planning [Chestnutt and Kuffner, 2004], but in that case it can be hard to know exactly what is lost when using several layers of motion planning: indeed, a high level planner might miss existing solutions that would have been found by the lower level planner. It is also possible to do it with the bounding box method [Yoshida et al., 2005]: after planning a continuous motion for the bounding box with a classical rigid body motion planner we can plan a discrete sequence of steps following this trajectory. But as mentioned in the introduction humanoid robots do not use their stepping over abilities with this approach. The same problem tends to occur with the use of A* with a mobile robot heuristic (see [Chestnutt, 2007], section 8.2). In this chapter we will eventually introduce a new bounding box method which can still be used with continuous rigid body motion planners, but which also captures well the stepping over abilities of humanoid robots. Using the OMPL library [OMPL, 2010], we will compare several motion planning algorithms such as KPIECE [Sucan and Kavraki, 2008], RRT-Connect [Kuffner and Lavalle, 2000], PRM [Geraerts and Overmars, 2002], SBL [Sanchez and Latombe, 2001], and will show that real-time footstep planning among 3D obstacles can be obtained with this method. But first we start by introducing and studying in Section 6.1 a very simple 2D planning problem that we call the "flea motion planning problem" and

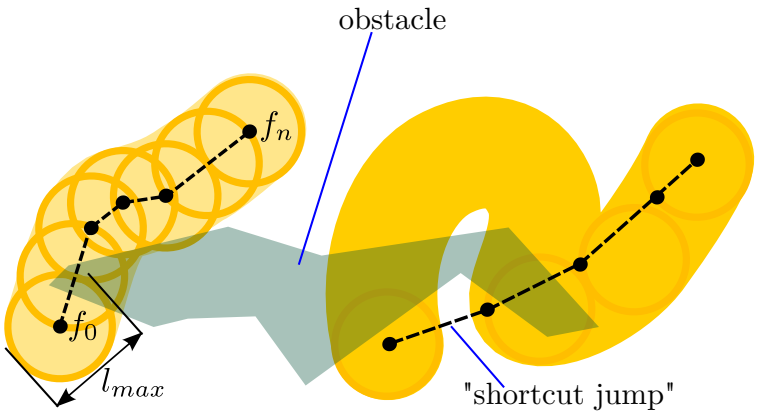


Fig. 6.1: The "flea motion planning problem". On the left: from a collision-free sequence of flea jumps to a continuous weakly collision-free path for the disk. On the right: converting a continuous weakly collision-free path of the disk into a sequence of flea jumps, using a greedy algorithm.

which shares a lot of similarities with footprint planning. In fact, on a conceptual level flea motion planning and footprint planning are essentially the same problem. Before presenting a polynomial-time algorithm which gives close-to-optimal solutions in practice, we show an equivalence between discrete sequences of flea jumps and continuous motions of a disk, using a modified notion of collision. Then in sections 6.2, 6.3, 6.4, 6.5, we adapt this equivalence result to a simple case of footprint planning, and in Section 6.6 we discuss its potential applications. Finally, by introducing a hybrid bounding box (Section 6.7), we show that it is possible to use our equivalence result and the walking pattern generator of Chapter 4 as the foundations of an efficient footstep planning algorithm. We implement this algorithm for HRP-2 and show results of simulations in Section 6.8, and conclude this chapter in Section 6.9.

6.1 The preliminary problem of "flea motion planning"

We consider a flea moving on a flat surface among polygonal obstacles. This flea can make jumps in any direction and of any length comprised between 0 and l_{max}. The goal is to find a sequence of jumps from a location $A(x_A, y_A)$ to a location $B(x_B, y_B)$ while avoiding the obstacles.

A sequence of jumps can be described by the sequence of configurations of the flea on the surface (i.e. elements of $\mathbb{R}^2$). Let us assume that a sequence of jumps has been found, and that it corresponds to the sequence of configurations $f_0 = (x_0, y_0), f_1, f_2, \ldots, f_n = (x_n, y_n)$. We consider the continuous motion of a disk of diameter l_{max} as depicted on Fig. 6.1 (on the left): it starts in position $(x_A, y_A) = (x_0, y_0)$, ends in position $(x_B, y_B) = (x_n, y_n)$, and between two consecutive configurations (x_i, y_i) and (x_{i+1}, y_{i+1}), it moves straight from (x_i, y_i) to (x_{i+1}, y_{i+1}). A time parametrization of this motion gives us a continuous path $(s(t))_{t\in[0,1]} \in (\mathbb{R}^2)^{[0,1]}$. An interesting property of this continuous disk motion is the following (it is a direct consequence of the upper bound l_{max} on jump lengths):

Property 6.1.1. *For all $t \in [0, 1]$, the configuration $s(t)$ of the disk contains at least one of the flea configurations $f_0, f_1, \ldots, f_n$.*

Therefore, if we define a new notion of collision as follows, then the continuous disk motion is collision-free:

Definition 6.1.1. *We say that a disk configuration (x, y) is collision-free if there exists at least one flea configuration inside the disk which is collision-free. We call this new notion of collision-freeness the "weak collision-freeness", and say that the disk configuration is "weakly collision-free". Conversely, if all the flea configurations inside the disk are in collision (i.e. the disk is entirely covered by the obstacles), we say that the disk is in "strong collision". We say that a continuous path is weakly collision-free if all the configurations along the path are weakly collision-free.*

What's even more interesting than Property 6.1.1 is that the converse property is true:

Property 6.1.2. *If there exists a weakly collision-free continuous path of the disk from A to B, then there exists a discrete sequence of jumps from A to B.*

We do not actually demonstrate this property, but somehow it will be done later since the proof is similar to and easier than the proof of Section 6.4. Together, Property 6.1.1 and Property 6.1.2 form an equivalence between the weakly collision-free paths of the disk and the collision-free sequences of jumps. Thus, in order to look for a discrete sequence of jumps, it is equivalent to first look for a continuous disk path, and that can be done with any classical rigid body motion planning algorithm (we just have to implement collision checks according to Definition 6.1.1).

If obstacles are represented by polygons, this equivalence can be used to easily find short sequences of jumps from an initial position to a goal. Indeed, the disk is in strong collision with a polygonal obstacle $\mathcal{P}$ if and only if its center intersects the "shrunk" obstacle $\mathcal{P}'$ whose construction is shown on the top of Fig. 6.2. What is more, the disk is in strong collision with a union of *disjoint* polygonal obstacles if and only if its center intersects the union of the shrunk obstacles. The shrunk obstacles are not necessarily polygonal, but they can be constructed in polynomial time, and because of the concavity of the non-polygonal parts, it is possible to adapt a polynomial-time algorithm for the construction of a visibility graph (see [de Berg et al., 2000], ch. 15) that will lead to an overall polynomial-time computation of the shortest collision-free path for the disk center from the initial position to the goal among the shrunk obstacles, i.e. the shortest *weakly collision-free* path for the disk. We will see later (in Section 6.7.2) an efficient greedy technique to obtain a good sequence of jumps from a weakly collision-free path. We do not have the guarantee that the shortest weakly collision-free path of the disk leads to the shortest path or the smallest number of jumps for the flea, but it is likely that bounds on sub-optimality can be proven (this is however out of the scope of this thesis), and in practice this method, illustrated on Fig. 6.2, is very efficient.

An interesting remark here is that if instead of considering the whole continuous region of jumps we decide to select only a finite number of possible jumps for the flea, then we would obtain a problem of discrete motion planning in a grid which would be very close to the NP-hard problem of Chapter 2, Section 2.4. It is not completely clear whether this fact feeds the intuition that discretizing the stepping capabilities might actually increase the intrinsic complexity of footprint planning, but the parallel between the efficient planning methods based on continuous weakly collision-free paths on one hand, and the NP-hardness of the discrete problem of Section 2.4 on the other,

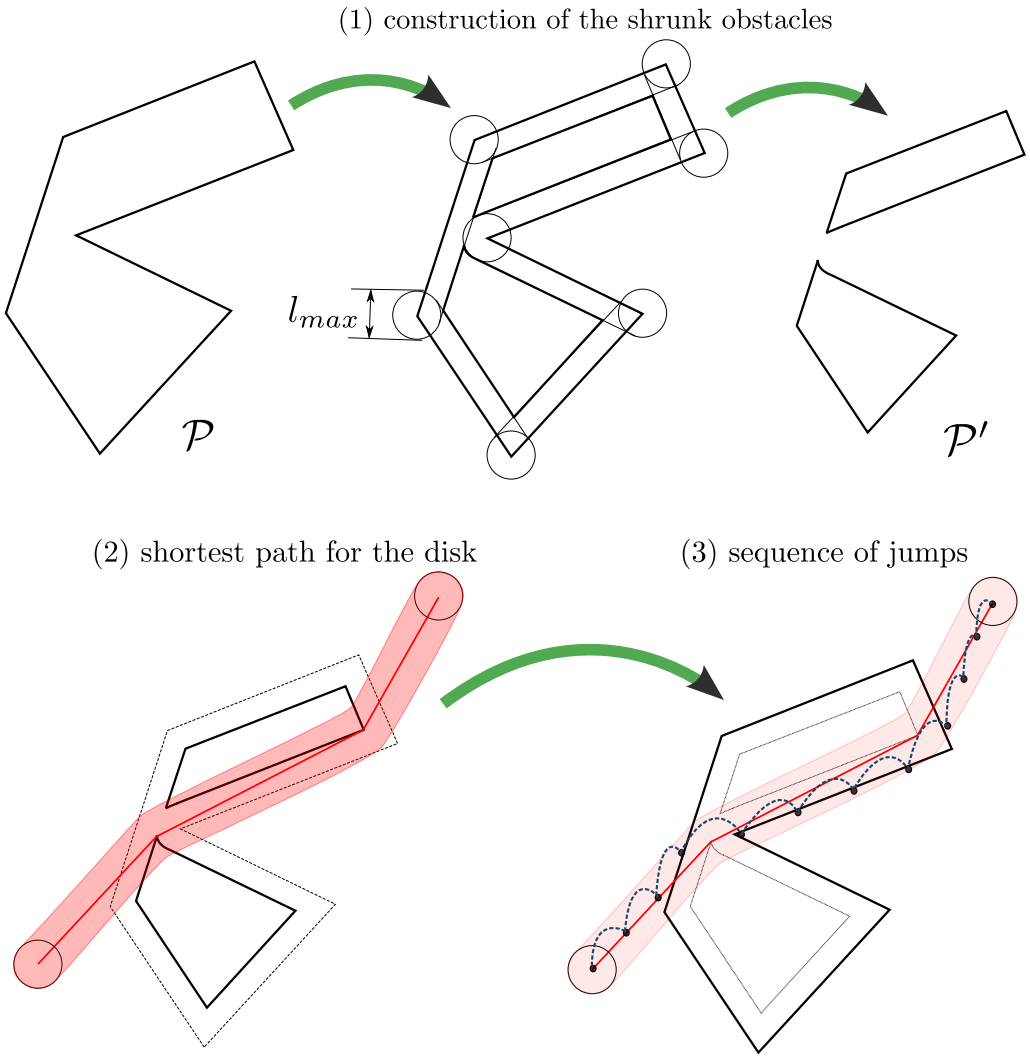


Fig. 6.2: Computing a short sequence of jumps towards a goal.

is interesting nonetheless. A deeper study of this parallel could be fruitful, but is out of the scope of this thesis.

That leaves us with a few interesting open questions related to the complexity of flea motion planning:

- Can we actually bound the sub-optimality of the method illustrated on Fig. 6.2?
- Is there a polynomial algorithm that solves the flea motion planning problem in an optimal way?
- Are all the non-trivial "discretized" versions of flea motion planning NP-hard?
- Are there efficient approximation algorithms for the discretized versions of flea motion planning?

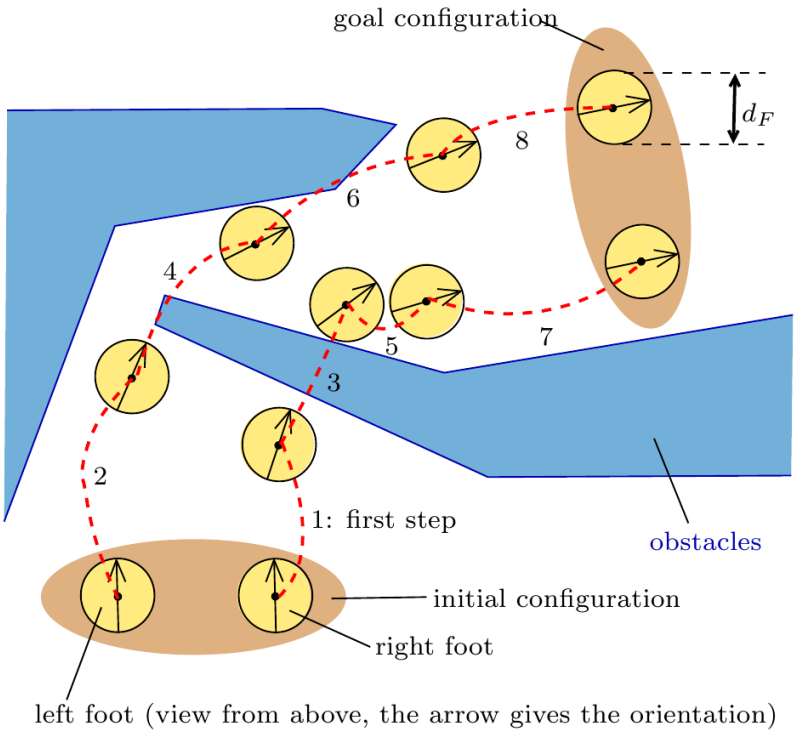


Fig. 6.3: Footprint planning for a robot with circular feet.

6.2 A continuous footprint planning problem

In this section and the three next sections, we study an abstract case of footprint planning (part of this work can be found in [Perrin et al., 2011c]). As part of the related work, we can mention [Boissonnat et al., 1992] where a simple abstract problem of footprint planning for a spider robot is studied (the main difference with our problem is that there is only a fixed finite set of possible footprints for the spider robot).

Let us first describe the stepping capabilites of the robot we consider. As shown on Fig. 6.3, its feet have the shape of a circle of diameter d_F. The problem is the following: the robot must find a discrete sequence of footprints that leads to a goal location without intersecting any obstacle. A footprint (or foot configuration) is defined by three parameters: two for its position and one for its orientation. A stance is defined by two configurations of the feet (left and right foot). Now we start to define the particular stepping capabilites that are needed for the theorem we prove below: for a stance to be acceptable, both feet must be inside a portion of disk whose configuration is fixed relatively to the other foot (see Fig. 6.4). More formally, if we denote by (x, y, θ) the left foot configuration, then the set of acceptable positions for the right foot is the set $\mathcal{E}_{\mathcal{R}}(x, y, \theta) \subset \mathbb{R}^2$ of positions (x', y') such that:

1. $\sin\theta \cdot (x' - x) - \cos\theta \cdot (y' - y) \geq h$
2. $\sqrt{(x' - x)^2 + (y' - y)^2} \leq r$

Similarly, if we denote by (x, y, θ) the configuration of the right foot, then the set $\mathcal{E}_{\mathcal{L}}(x, y, \theta)$ of acceptable positions for the left foot is the set of positions (x', y') such that:

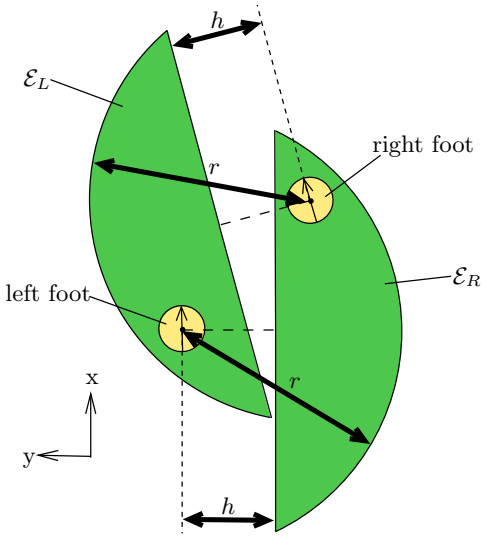


Fig. 6.4: A part of our hypothesis is that for a stance to be acceptable, a necessary condition is that each foot must be inside a region defined by the position and orientation of the other foot.

1. $\sin\theta \cdot (x' - x) + \cos\theta \cdot (y' - y) \geq h$
2. $\sqrt{(x' - x)^2 + (y' - y)^2} \leq r$

So, a necessary condition for a stance $\big((x, y, \theta), (x', y', \theta')\big)$ to be acceptable is that:

$$(x', y') \in \mathcal{E}_{\mathcal{R}}(x, y, \theta) \text{ and } (x, y) \in \mathcal{E}_{\mathcal{L}}(x', y', \theta')$$

This constraint defines also implicit restrictions on the orientations, but additional restrictions can be imposed as well, and it will not change the stepping capabilities, as long as the following property is verified:

Definition 6.2.1. *A constraint $C\big((x, y, \theta), (x', y', \theta')\big)$ on the stances is said to verify the μ-property if there exists $\mu > 0$ such that $|\theta' - \theta| < \mu$ automatically implies that the constraint is satisfied.*

The fact that additional constraints verifying the μ-property do not change the stepping capabilities is surprising: it means that even with very restrictive constraints on the relative orientations of the feet, the same sequence of footprints can still be followed, but possibly with more steps. This is true only because the feet have a circular shape.

Our stepping capabilities are not only defined by the constraints on the set of acceptable stances: an additional constraint that reduces the maximum length of feasible steps is imposed. For this purpose we introduce the crucial 2D object shown on Fig. 6.5: it is composed of two symmetric portions of disk, and can be obtained by extruding an open stripe of width h from a disk of radius $r/2$ (where r and h are the same as the ones used to define the first constraints on the acceptable stances). Its configuration in the plane is defined by three parameters x, y and θ, where (x, y) denote the position of its center and θ its orientation. We call Φ this object, and when in configuration (x, y, θ), we denote by $\Phi(x, y, \theta)$ the set of points it contains. Φ being

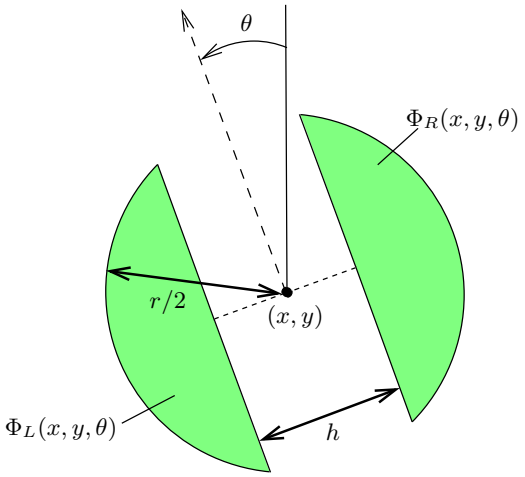


Fig. 6.5: The 2D object Φ.

divided into two parts, its orientation naturally defines a left and a right part. In configuration (x, y, θ), we denote respectively by $\Phi_L(x, y, \theta)$ and $\Phi_R(x, y, \theta)$ the sets of points contained in the left and right part of Φ.

A step of the robot can be described by three triples: the support foot configuration, and the initial and final swing foot configurations. For example, if the left foot is the support foot, we can call (x_L, y_L, θ_L), $(x_R^i, y_R^i, \theta_R^i)$ and $(x_R^f, y_R^f, \theta_R^f)$ these three configurations.

Definition 6.2.2. *A step defined by such configurations is said feasible if and only if the following constraints are verified:*

1. $(x_R^i, y_R^i) \in \mathcal{E}_{\mathcal{R}}(x_L, y_L, \theta_L)$
2. $(x_L, y_L) \in \mathcal{E}_{\mathcal{L}}(x_R^i, y_R^i, \theta_R^i)$
3. $(x_R^f, y_R^f) \in \mathcal{E}_{\mathcal{R}}(x_L, y_L, \theta_L)$
4. $(x_L, y_L) \in \mathcal{E}_{\mathcal{L}}(x_R^f, y_R^f, \theta_R^f)$
5. *Possibly a finite number of additional constraints on the stances, all verifying the μ-property (Definition 6.2.1).*
6. *There exists a configuration (x, y, θ) such that $(x_L, y_L) \in \Phi_L(x, y, \theta)$, $(x_R^i, y_R^i) \in \Phi_R(x, y, \theta)$ and $(x_R^f, y_R^f) \in \Phi_R(x, y, \theta)$.*

These constraints (and, of course, the symmetric constraints with the right support foot) completely describe the stepping capabilities of the robot. Fig. 6.6 shows a few feasible and unfeasible steps.

Now, we precisely formulate the footprint planning problem and state the main theorem of this chapter. The obstacles are defined by a finite number of closed sets of points in the plane. The robot starts with both feet on the ground, the left and right foot being respectively in configuration $(x_L^0, y_L^0, \theta_L^0) = c_L^0$ and $(x_R^0, y_R^0, \theta_R^0) = c_R^0$, which are supposed collision-free (i.e. the disks of radius $\frac{d_F}{2}$ and centers (x_L^0, y_L^0) and (x_R^0, y_R^0) don't intersect with any obstacle, even on their borders). A collision-free goal

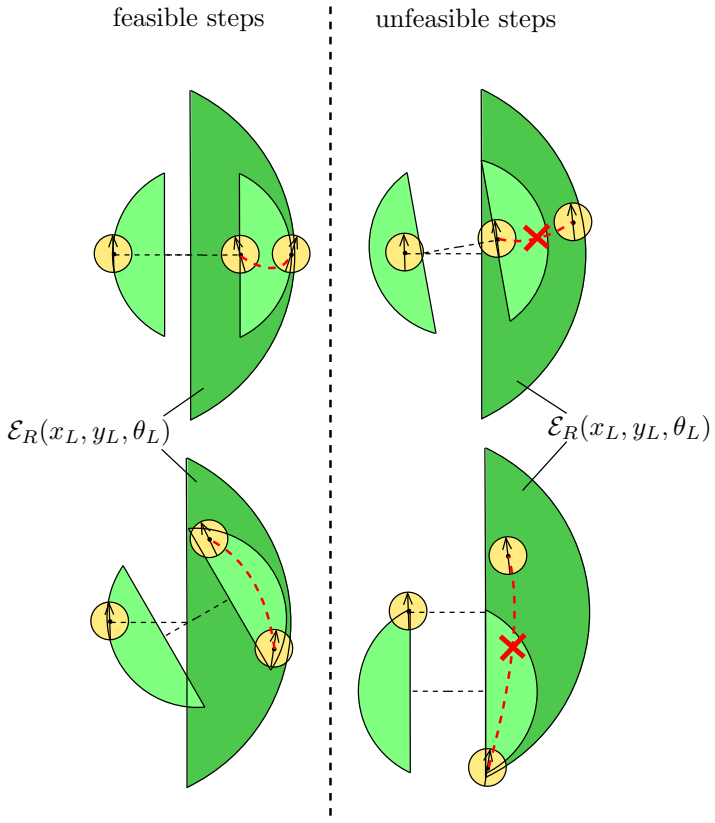


Fig. 6.6: Some feasible and unfeasible steps.

configuration is given for both feet ($(x_L^G, y_L^G, \theta_L^G) = c_L^G$ and $(x_R^G, y_R^G, \theta_R^G) = c_R^G$), and the objective is to find a finite sequence of feasible steps that goes from (c_L^0, c_R^0) to (c_L^G, c_R^G) while avoiding all the obstacles. An example instance of this problem is shown on Fig. 6.3.

Before stating the main theorem, we adapt the notion of "weak collision-freeness" seen in the previous section to the object Φ:

Definition 6.2.3. *A configuration $\Phi(x, y, \theta)$ of Φ is said to be "weakly collision-free" if and only if there exist $(x_l, y_l) \in \Phi_L(x, y, \theta)$ and $(x_r, y_r) \in \Phi_R(x, y, \theta)$ such that there is no intersection between the obstacles and the two disks of radius $\frac{d_F}{2}$ and centers (x_l, y_l) and (x_l, y_l). If it is not the case Φ is said to be in "strong collision".*

Fig. 6.7 shows a weakly collision-free configuration, and a continuous weakly collision-free path, i.e. a continuous path $\mathcal{S} : [0, 1] \mapsto \mathbb{R}^2 \times SO(2)$ such that $\forall t \in [0, 1]$, $\Phi(\mathcal{S}(t))$ is weakly collision-free.

We can now state the main theorem of this chapter:

Theorem 6.2.1. *There exists a collision-free sequence of feasible steps from $\big((x_L^0, y_L^0, \theta_L^0), (x_R^0, y_R^0, \theta_R^0)\big)$ to the goal if and only if there exists a continuous weakly collision-free path $\mathcal{S} : [0, 1] \mapsto \mathbb{R}^2 \times SO(2)$ such that:*

1. $(x_L^0, y_L^0) \in \Phi_L(\mathcal{S}(0))$ *and* $(x_R^0, y_R^0) \in \Phi_R(\mathcal{S}(0))$
2. $(x_L^G, y_L^G) \in \Phi_L(\mathcal{S}(1))$ *and* $(x_R^G, y_R^G) \in \Phi_R(\mathcal{S}(1))$

In the next two sections we prove the two implications of this theorem.

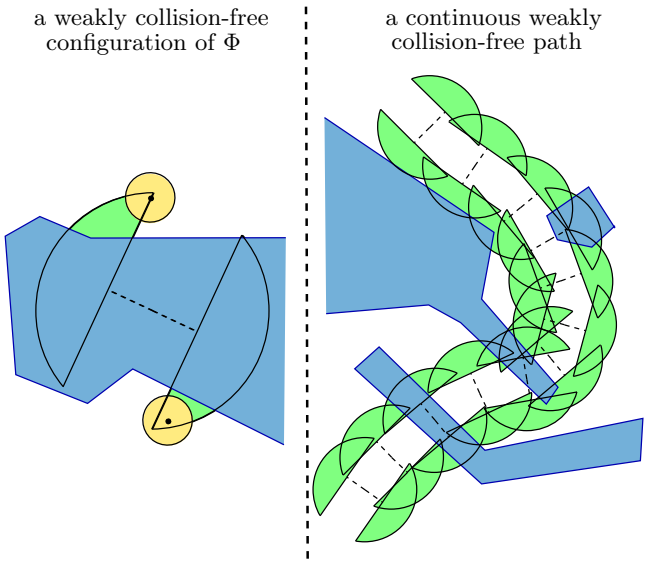


Fig. 6.7: Weak collision-freeness.

6.3 From a solution to a weakly collision-free path

We first show that the existence of a finite sequence of feasible steps from the initial configuration to the goal implies the existence of a continuous weakly collision-free path for Φ.

Proof. Without loss of generality, we suppose that the first support foot is the right one, and that the last support foot is the left one. Let us denote by (c_R^0, c_L^0, c_L^1), (c_L^1, c_R^0, c_R^1), $\ldots$, $(c_L^n, c_R^{n-1}, c_R^n) = (c_L^G, c_R^{n-1}, c_R^G)$ a sequence of feasible steps that goes to the goal while avoiding the obstacles, with $c_R^i = (x_R^i, y_R^i, \theta_R^i)$ and $c_L^i = (x_L^i, y_L^i, \theta_L^i)$ (we recall that (c_R^a, c_L^a, c_L^b) denotes the step from stance (c_L^a, c_R^a) to (c_L^b, c_R^a), and (c_L^a, c_R^a, c_R^b) the step from stance (c_L^a, c_R^a) to (c_L^a, c_R^b)).

For any step $(c_R^i, c_L^i, c_L^{i+1})$, thanks to the constraint 6) in definition 6.2.2, we know that there exists a configuration $\Phi(x_{2i}, y_{2i}, \theta_{2i})$ such that $(x_R^i, y_R^i) \in \Phi_R(x_{2i}, y_{2i}\theta_{2i})$ and $(x_L^i, y_L^i) \in \Phi_L(x_{2i}, y_{2i}, \theta_{2i})$ and $(x_L^{i+1}, y_L^{i+1}) \in \Phi_L(x_{2i}, y_{2i}, \theta_{2i})$. Since the steps avoid the obstacles, this configuration is necessarily weakly collision-free. Similarly, for any step $(c_L^{i+1}, c_R^i, c_R^{i+1})$ there exists a weakly collision-free configuration $\Phi(x_{2i+1}, y_{2i+1}, \theta_{2i+1})$ containing the three points in the corresponding parts of Φ.

We show that there exists a continuous weakly collision-free path $\mathcal{S} : [0,1] \mapsto \mathbb{R}^2 \times SO(2)$ such that for all $k \in \{0, \ldots, 2n-1\}$, $\mathcal{S}(\frac{k}{2n-1}) = (x^k, y^k, \theta^k)$. To obtain this result it is enough to prove that $\forall k \in \{0, \ldots, 2n-2\}$, there is a continuous weakly collision-free path $\mathcal{S}^k : [\frac{k}{2n-1}, \frac{k+1}{2n-1}] \mapsto \mathbb{R}^2 \times SO(2)$ such that $\mathcal{S}^k(\frac{k}{2n-1}) = (x^k, y^k, \theta^k)$ and $\mathcal{S}^k(\frac{k+1}{2n-1}) = (x^{k+1}, y^{k+1}, \theta^{k+1})$. To avoid heavy notations we prove it for $k = 0$, but exactly the same demonstration applies for any $k \in \{1, \ldots, 2n-2\}$. There are 4 foot positions to consider: (x_R^0, y_R^0), (x_L^0, y_L^0), (x_R^1, y_R^1) and (x_L^1, y_L^1). $\Phi(x_0, y_0, \theta_0)$ contains (x_R^0, y_R^0), (x_L^0, y_L^0) and (x_L^1, y_L^1) while $\Phi(x_1, y_1, \theta_1)$ contains (x_L^1, y_L^1), (x_R^0, y_R^0) and (x_R^1, y_R^1). Fig. 6.8 sums up the situation. The key point is that (x_R^0, y_R^0) and (x_L^1, y_L^1) are contained in both $\Phi(x_0, y_0, \theta_0)$ and $\Phi(x_1, y_1, \theta_1)$. Because of the particular

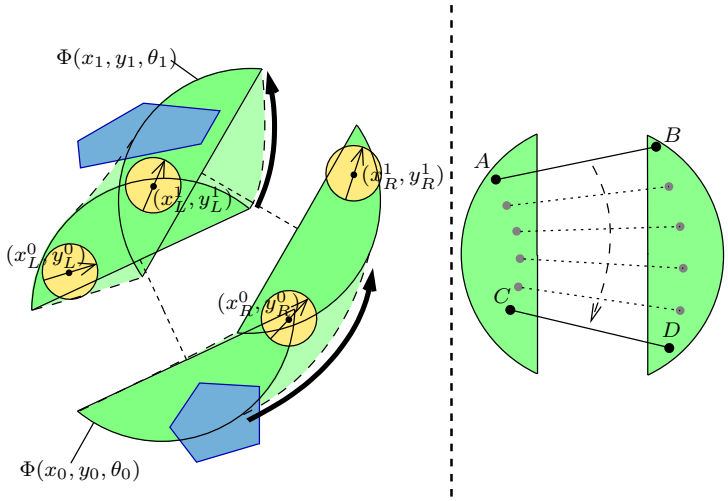


Fig. 6.8: On the left: a continuous weakly collision-free path from $\Phi(x_0, y_0, \theta_0)$ to $\Phi(x_1, y_1, \theta_1)$, keeping (x_L^1, y_L^1) and (x_R^0, y_R^0) in Φ at all time. On the right, a dual transformation: moving continuously the segment line AB towards CD while keeping its extremities inside Φ.

shape of Φ, it is possible to show that we can go continuously from $\Phi(x_0, y_0, \theta_0)$ to $\Phi(x_1, y_1, \theta_1)$ while keeping (x_R^0, y_R^0) in Φ_L and (x_L^1, y_L^1) in Φ_R. We will not explain it in detail but this is a consequence of the following property: considering a fixed configuration of Φ, for any couple of line segments of same length AB and CD, such that $A, C \in \Phi_L$ and $B, D \in \Phi_R$, it is possible to continuously move AB until it coincides with CD, without ever moving A (resp. B) out of Φ_L (resp. Φ_R). On Fig. 6.8 A and C would both correspond to (x_L^1, y_L^1), and B and D would both correspond to (x_R^0, y_R^0). There are other shapes than Φ verifying the same property, but Φ is one of the simplest.

So, the continuous displacement from $\Phi(x_0, y_0, \theta_0)$ to $\Phi(x_1, y_1, \theta_1)$ always contains (x_R^0, y_R^0) and (x_L^1, y_L^1), and thus it is clearly weakly collision-free. Connecting the paths obtained for $k = 0, 1, \ldots, 2n-2$ gives us a continuous weakly collision-free path from (x^0, y^0, θ^0) to $(x^{2n-1}, y^{2n-1}, \theta^{2n-1})$, and that concludes the demonstration. □

6.4 From a weakly collision-free path to a solution

In this section, we prove the converse implication: if there is a continuous weakly collision-free path for Φ, then there exists a finite sequence of feasible steps from the initial configuration to the goal. It might be noted that this property is a bit similar to the reduction property in [Alami et al., 1994], where it is proven that a continuous collision-free solution to a manipulation planning problem can always be converted into a finite sequence of manipulation tasks with fixed grasping configurations.

First, we prove the following lemma:

Lemma 6.4.1. *Let (x, y, θ) be a configuration of Φ, and (x_L^a, y_L^a), (x_L^b, y_L^b) two points in $\Phi_L(x, y, \theta)$, and (x_R^a, y_R^a), (x_R^b, y_R^b) two points in $\Phi_R(x, y, \theta)$. We assume that these four points are at a distance greater than $\frac{d_F}{2}$ from the obstacles. Then from any acceptable stance based on (x_L^a, y_L^a) and (x_R^a, y_R^a) there exists a collision-free sequence of feasible steps to an acceptable stance based on (x_L^b, y_L^b) and (x_R^b, y_R^b).*

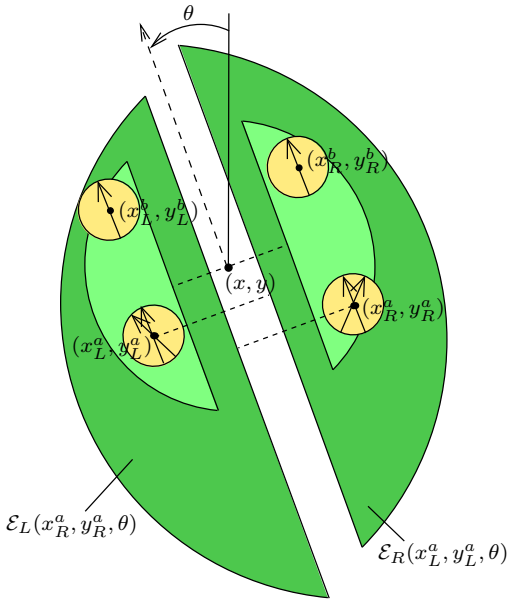


Fig. 6.9: There always exists a sequence of feasible steps from $((x_L^a, y_L^a, \theta_L^a), (x_R^a, y_R^a, \theta_R^a))$ to $((x_L^b, y_L^b, \theta), (x_R^b, y_R^b, \theta))$.

Proof. Let us denote by $(x_L^a, y_L^a, \theta_L^a)$ and $(x_R^a, y_R^a, \theta_R^a)$ the two initial configurations of the feet. These two configurations form an acceptable stance, so we know that $(x_R^a, y_R^a) \in \mathcal{E}_R(x_L^a, y_L^a, \theta_L^a)$ and $(x_L^a, y_L^a) \in \mathcal{E}_L(x_R^a, y_R^a, \theta_R^a)$, with possible additional constraints verified. Because of the symmetry between $\mathcal{E}_L$ and $\mathcal{E}_R$, we also have $(x_L^a, y_L^a) \in \mathcal{E}_L(x_R^a, y_R^a, \theta_L^a)$. Besides, any constraint verifying the μ-property is satisfied by the stance $\big((x_L^a, y_L^a, \theta_L^a), (x_R^a, y_R^a, \theta_L^a)\big)$. Since the point 6) of Definition 6.2.2 is also verified, we deduce that the step $\big((x_L^a, y_L^a, \theta_L^a), (x_R^a, y_R^a, \theta_R^a), (x_R^a, y_R^a, \theta_L^a)\big)$ is feasible. After this step both feet have the same orientation. Let us also consider the configurations (x_L^a, y_L^a, θ) and (x_R^a, y_R^a, θ) (we recall that θ is the orientation of Φ). As shown on Fig. 6.9, we can easily verify that $\mathcal{E}_R(x_L^a, y_L^a, \theta) \supset \Phi_R(x, y, \theta)$, and $\mathcal{E}_L(x_R^a, y_R^a, \theta) \supset \Phi_L(x, y, \theta)$. Therefore we have $(x_R^a, y_R^a) \in \mathcal{E}_R(x_L^a, y_L^a, \theta)$, and $(x_L^a, y_L^a) \in \mathcal{E}_L(x_R^a, y_R^a, \theta)$. Thanks to this and to the μ-property of the possible additional constraints on the stances, it follows that there exists a finite sequence of feasible steps that goes from stance $\big((x_R^a, y_R^a, \theta_R^a), (x_R^a, y_R^a, \theta_L^a)\big)$ to stance $\big((x_R^a, y_R^a, \theta), (x_R^a, y_R^a, \theta)\big)$ (but this might be a long sequence, with each step resulting in just a slight orientation change). Once the left and right foot are respectively in configurations (x_L^a, y_L^a, θ) and (x_R^a, y_R^a, θ), the next 2 steps leading first to $\big((x_L^b, y_L^b, \theta), (x_R^a, y_R^a, \theta)\big)$ or $\big((x_L^a, y_L^a, \theta), (x_R^b, y_R^b, \theta)\big)$, and then to $\big((x_L^b, y_L^b, \theta), (x_R^b, y_R^b, \theta)\big)$, are also feasible (see Fig. 6.9). As a consequence, we obtained a sequence of collision-free feasible steps from $\big((x_L^a, y_L^a, \theta_L^a), (x_R^a, y_R^a, \theta_R^a)\big)$ to $\big((x_L^b, y_L^b, \theta), (x_R^b, y_R^b, \theta)\big)$, which is an acceptable stance, and that concludes the proof of lemma 6.4.1. □

We can now start the main demonstration.

Proof. Let $\mathcal{S} : [0, 1] \mapsto \mathbb{R}^2 \times SO(2)$ be a continuous weakly collision-free path of Φ towards the goal. For a configuration (x, y, θ) of Φ, we denote by $d_{obs}^L(x, y, \theta)$ the maximum distance between any point of $\Phi_L(x, y, \theta)$ and the obstacles, and by $d_{obs}^R(x, y, \theta)$

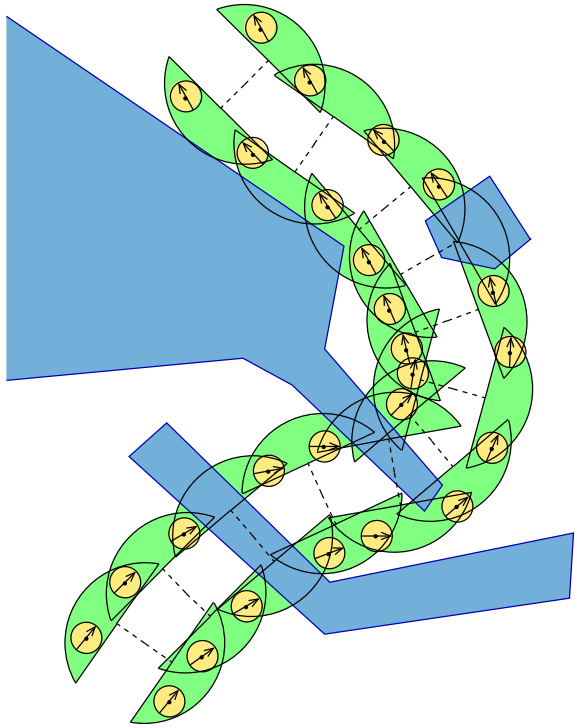


Fig. 6.10: From a continuous weakly collision-free path to a finite sequence of feasible collision-free steps.

the maximum distance between any point of $\Phi_R(x, y, \theta)$ and the obstacles. We also pose:

$$d_{obs}(x, y, \theta) = \min(d^L_{obs}(x, y, \theta), d^R_{obs}(x, y, \theta)),$$

and:

$$d_{min}(\mathcal{S}) = \min_{t\in[0,1]} (d_{obs}(\mathcal{S}(t)))$$

Since $\mathcal{S}$ is weakly collision-free, and since both Φ and the obstacles are represented by closed set of points, we know that:

$$d_{min}(\mathcal{S}) > \tfrac{d_F}{2}$$

Besides, $\mathcal{S}$ is uniformly continuous on $[0, 1]$, and therefore for any $\rho > 0$ there exists $\epsilon > 0$ such that for all $(t_1, t_2) \in [0, 1]^2$, $|t_1 - t_2| < \epsilon \Rightarrow \|\mathcal{S}(t_1) - \mathcal{S}(t_2)\|_\infty < \rho$, where $\|(\Delta x, \Delta y, \Delta\theta)\|_\infty = \max(|\Delta x|, |\Delta y|, |\Delta\theta|)$ (with $|\Delta\theta|$ being the smallest value among $\{|\Delta\theta + k2\pi|,\ k \in \mathbb{Z}\}$). It follows that for any $\eta > 0$, there also exists $\epsilon > 0$ such that $|t_1 - t_2| < \epsilon$ implies that from $\mathcal{S}(t_1)$ to $\mathcal{S}(t_2)$ the points of Φ are moved by at most η. For a given $\eta > 0$, let us denote by $\epsilon(\eta)$ a satisfying value of ϵ, and let us denote by N an integer such that $\frac{1}{N} < \epsilon(\eta)$.

We consider the configurations $\mathcal{S}(\frac{0}{N})$, $\mathcal{S}(\frac{1}{N})$, $\ldots$, $\mathcal{S}(\frac{N}{N})$.

There exists η small enough so that for any $i \in \{0, \ldots N-1\}$, every point in $\Phi_L(\mathcal{S}\left(\frac{i}{N}\right)) \cup \Phi_L(\mathcal{S}\left(\frac{i+1}{N}\right))$ is at distance at most $(d_{min}(\mathcal{S}) - \frac{d_F}{2})/2$ from the non-empty intersection $\Phi_L\left(\mathcal{S}(\frac{i}{N})\right) \cap \Phi_L\left(\mathcal{S}(\frac{i+1}{N})\right)$, with the same property for Φ_R. In that case, for any point $(x, y) \in \Phi_L\left(\mathcal{S}(\frac{i}{N})\right)$ at distance at least $d_{min}(\mathcal{S})$ from the obstacles, there exists a point $(x', y') \in \Phi_L\left(\mathcal{S}(\frac{i}{N})\right) \cap \Phi_L\left(\mathcal{S}(\frac{i+1}{N})\right)$ at distance at least $d_{min}(\mathcal{S}) -$

$(d_{min}(\mathcal{S}) - \frac{d_F}{2})/2 > \frac{d_F}{2}$ from the obstacles. The result also applies for points in Φ_R, and it follows that we can extract two sequence of points $(x_L^1, y_L^1), \ldots, (x_L^N, y_L^N)$ and $(x_R^1, y_R^1), \ldots, (x_R^N, y_R^N)$ that are all at distance greater than $\frac{d_F}{2}$ from the obstacles, and such that:

$$\forall i \in \{1, \ldots, N\},\ (x_L^i, y_L^i) \in \Phi_L\left(\mathcal{S}\left(\tfrac{i-1}{N}\right)\right) \cap \Phi_L\left(\mathcal{S}\left(\tfrac{i}{N}\right)\right),$$

and:

$$\forall i \in \{1, \ldots, N\},\ (x_R^i, y_R^i) \in \Phi_R\left(\mathcal{S}\left(\tfrac{i-1}{N}\right)\right) \cap \Phi_R\left(\mathcal{S}\left(\tfrac{i}{N}\right)\right).$$

As an almost direct consequence of Lemma 6.4.1, these points can be the support of a sequence of feasible steps going from the initial configuration to the goal. That concludes the demonstration, and Fig. 6.10 illustrates the whole construction. □

6.5 Generalization to different stepping capabilities

The stepping capabilites considered so far are realistic, but the condition 6) in Definition 6.2.2 might seem a bit restrictive. Here we consider the stepping capabilites with this constraint removed: in other words, a step is feasible as long as it involves two acceptable stances. Actually, we make another very slight change: the sets $\mathcal{E}_\mathcal{L}$ and $\mathcal{E}_\mathcal{R}$ are replaced by their interior. Thus, the definition of feasibility becomes:

Definition 6.5.1. *The step defined by the 3 configurations* $((x_L, y_L, \theta_L)$, $(x_R^i, y_R^i, \theta_R^i)$, $(x_R^f, y_R^f, \theta_R^f))$ *is feasible if and only if the following constraints are verified:*

1. $(x_R^i, y_R^i) \in \mathcal{E}_\mathcal{R}^\circ(x_L, y_L, \theta_L)$
2. $(x_L, y_L) \in \mathcal{E}_\mathcal{L}^\circ(x_R^i, y_R^i, \theta_R^i)$
3. $(x_R^f, y_R^f) \in \mathcal{E}_\mathcal{R}^\circ(x_L, y_L, \theta_L)$
4. $(x_L, y_L) \in \mathcal{E}_\mathcal{L}^\circ(x_R^f, y_R^f, \theta_R^f)$
5. *Possibly additional constraints verifying the μ-property (Property 6.2.1).*

An equivalence similar to Theorem 6.2.1 can be obtained, but an additional operation on Φ is required. This operation depends on one parameter $\delta \in (-1, 1)$, and is described on Fig. 6.11. It gives a fourth dimension to the configuration space of Φ, which now becomes $\mathbb{R}^2 \times SO(2) \times (-1, 1)$. And the theorem becomes:

Theorem 6.5.1. *There exists a collision-free sequence of feasible steps from* $\big((x_L^0, y_L^0, \theta_L^0)$, $(x_R^0, y_R^0, \theta_R^0)\big)$ *to the goal if and only if there exists a continuous weakly collision-free path* $\mathcal{S} : [0, 1] \mapsto \mathbb{R}^2 \times SO(2) \times (-1, 1)$ *such that:*

1. $(x_L^0, y_L^0) \in \Phi_L(\mathcal{S}(0))$ *and* $(x_R^0, y_R^0) \in \Phi_R(\mathcal{S}(0))$
2. $(x_L^G, y_L^G) \in \Phi_L(\mathcal{S}(1))$ *and* $(x_R^G, y_R^G) \in \Phi_R(\mathcal{S}(1))$

With a 4-dimensional configuration space, motion planning algorithms become a bit slower, but this theorem might be even more useful than the first one because with appropriate additional constraints the stepping capabilities obtained can be very close to the ones of real humanoid robots. To change the stepping capabilities even more, it might also be possible to change the shape of $\mathcal{E}_L$, $\mathcal{E}_R$, and Φ, or even the notion of weak collision-freeness. It should be possible to obtain general criteria defining the set

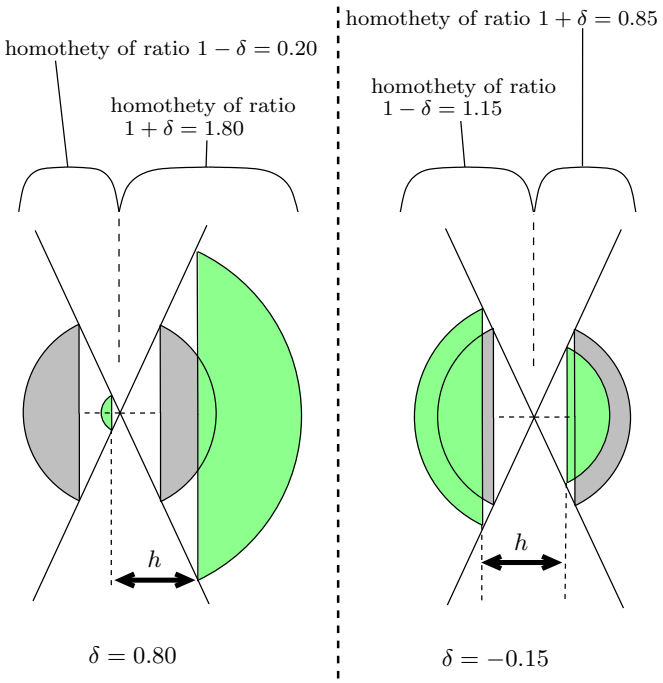


Fig. 6.11: A new operation on Φ: if the first parameters of its configuration are x, y, and θ, then for $\delta \in (-1, 1)$, Φ_L undergoes an homothety of center (x, y) and ratio $1 - \delta$, while Φ_R undergoes an homothety of center (x, y) and ratio $1 + \delta$ (thus Φ is unchanged if $\delta = 0$).

of Φ shapes and weak collision-freeness definitions that can be used while keeping the equivalence result, but this is out of the scope of this thesis. So far, we only considered footprints with a circular shape; when the feet of the robot are rectangular, things are a bit more complicated. The easiest way to deal with it is to be a bit overconservative and use circles that contain the rectangular footprints. With a more precise approach, results similar to Theorem 6.2.1 can be obtained with stepping capabilities based on rectangular footprints, but they require new limitations. In the following sections we present applications with rectangular footprints, but we do not explain how the equivalence results are adapted.

6.6 Potential applications

For the same reason why the reduction property in [Alami et al., 1994] is useful for manipulation planning, theorems 6.2.1 and 6.5.1 are useful for footstep planning, as they turn an *a priori* specific problem of motion planning into an instance of a better-studied and more fundamental problem. Indeed, the problem of searching for continuous collision-free paths (in $\mathbb{R}^2 \times SO(2)$) for a rigid 2D shape that can translate and rotate, is the 2-dimensional version of the most studied problem of motion planning: the classical piano mover's problem. A large number of techniques have been designed to solve efficiently this problem (and most of them also apply if the configuration space is $\mathbb{R}^2 \times SO(2) \times (-1, 1)$), while fewer algorithms exist for footstep planning. For example PRM and RRT are two widespread techniques of motion planning that cannot

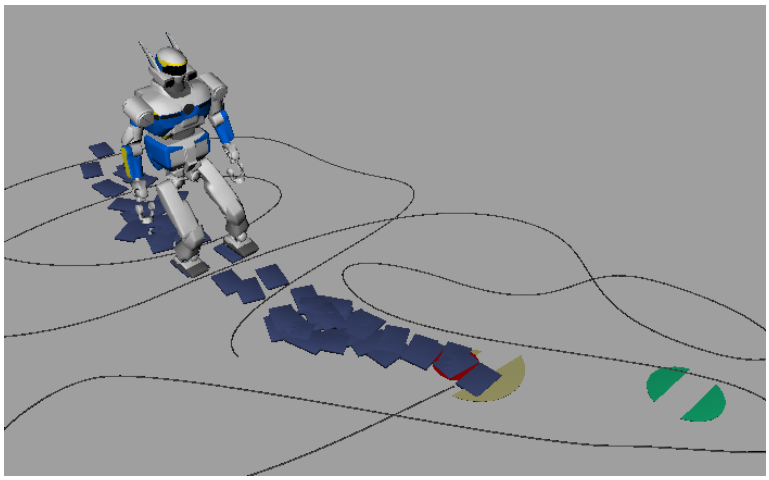


Fig. 6.12: The robot HRP-2 following a weakly collision-free path and avoiding a cable on the ground

be directly applied to footstep planning in a sound way. Thanks to Theorem 6.2.1, it becomes the case: indeed, for a robot with the stepping capabilities described in section 6.2, a sound way to solve the footstep planning problem is to first use PRM or RRT to find a continuous path for Φ, and then convert it into a finite sequence of steps. Once a solution path $\mathcal{S}$ has been found, as we will see in Section 6.7.2 the conversion can be done very efficiently by a greedy algorithm that always tries to put the next stance of the robot in $\Phi(\mathcal{S}(t))$ with t as large as possible.

During the execution of PRM or RRT, only one thing has to be changed: the collision checks must be replaced by strong collision checks, which are a bit more computationally costly, but can be handled efficiently with appropriate data structures, parallel or approximate approaches. It might also be interesting to use a "tuned cost" for the paths lengths because lateral motions of the Φ object can produce many unnecessary lateral steps. Another way to solve this problem could be to apply nonholonomic motion planning algorithms, which for example aim at planning the motion of a car (see [Laumond, 1986], [Barraquand and Latombe, 1993], [Lamiraux et al., 2004]). In our case, artificial constraints or control laws can be used to oblige the robot to favor forward motions rather than backward or lateral walking, or simply to execute maneuvers that look natural. To do motion planning with these control laws an algorithm such as KPIECE (for example) could be applied ([Sucan and Kavraki, 2008]).

Fig. 6.12 shows a preliminary result where the robot HRP-2 finds footprints that avoid a cable on the ground. Unfortunately, so far we can only avoid 2D obstacles, and for useful applications we must be able to take into account 3D obstacles. This is the purpose of the next two sections.

6.7 Footstep planning with a two-level hybrid bounding box

The bounding box used to conveniently plan the walking motions of a humanoid robot is usually a single rigid body. For instance it can be a rectangular box that always contains the whole robot or at least sweeps volumes that entirely contain the robot

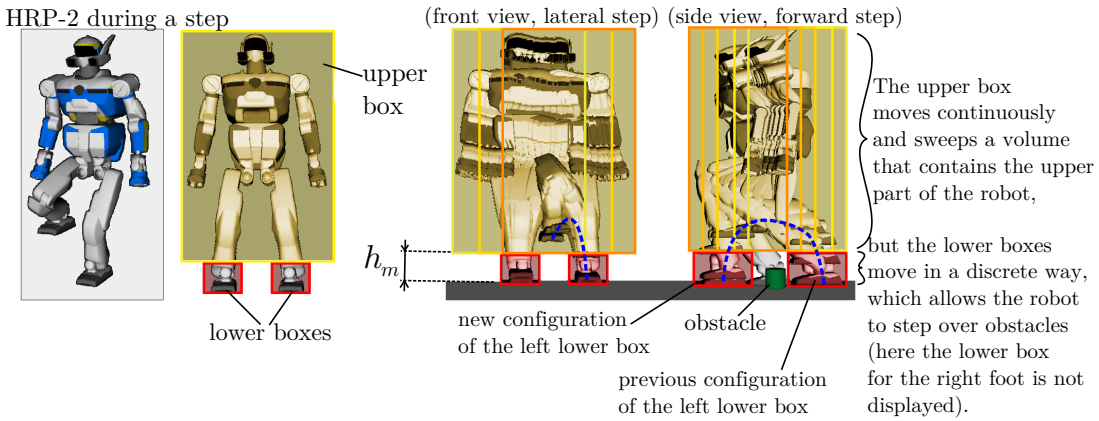


Fig. 6.13: HRP-2 and the improved bounding box, made of three rectangular boxes.

motions (see [Yoshida et al., 2008], [El Khoury et al., 2011]). A cylinder is also sometimes used, for example when motion planning methods for differential drive robots are applied (see [Hayet et al., 2009]). Bounding boxes are also sometimes made of several solid objects rigidly linked (see [Gutmann et al., 2005]).

In this section we introduce an improved bounding box that behaves somehow as a single box but has in fact three parts: two lower rectangular boxes, and one upper rectangular box. The upper box is similar to a classical bounding box in the sense that it moves continuously. The volume swept by the upper box contains every part of the robot above a fixed height h_m, which corresponds to the maximum height of the obstacles that can be stepped over. For the robot HRP-2, this height was chosen to be 20cm, which means that the robot will circumvent all the obstacles of height greater than 20cm. Because of the similarity with the classical bounding box, the upper box is of secondary importance in this section and the next. Instead, we focus on the two lower boxes, which are used to capture the stepping over capabilities of the robot. Fig. 6.13 shows how the discrete motion of the lower boxes enables the robot to step over obstacles (without leaving the volume defined by the boxes union).

In order to use classical motion planning techniques for our two-level bounding box, we present a footstep planning algorithm based on the equivalence proved in the previous sections (Theorem 6.2.1). This algorithm also benefits from the features of a slightly improved version of the walking pattern generator introduced in Chapter 4. Its aim is to make HRP-2 walk from an initial location A to a goal B on a flat ground cluttered with 3D obstacles, and it is divided into 3 phases:

1. Solve a classical rigid body motion planning problem to obtain a weakly collision-free continuous path.

2. Convert the continuous path into a hybrid motion of the two-level bounding box. The bounding box motion directly corresponds to slow and large steps avoiding the obstacles, and we try to plan it so that to minimize the number of steps.

3. Keeping the same footprints, smooth and speed up the steps in order to reach the goal faster and avoid the obstacles in a more efficient way.

These 3 phases are described in the next 3 sections.

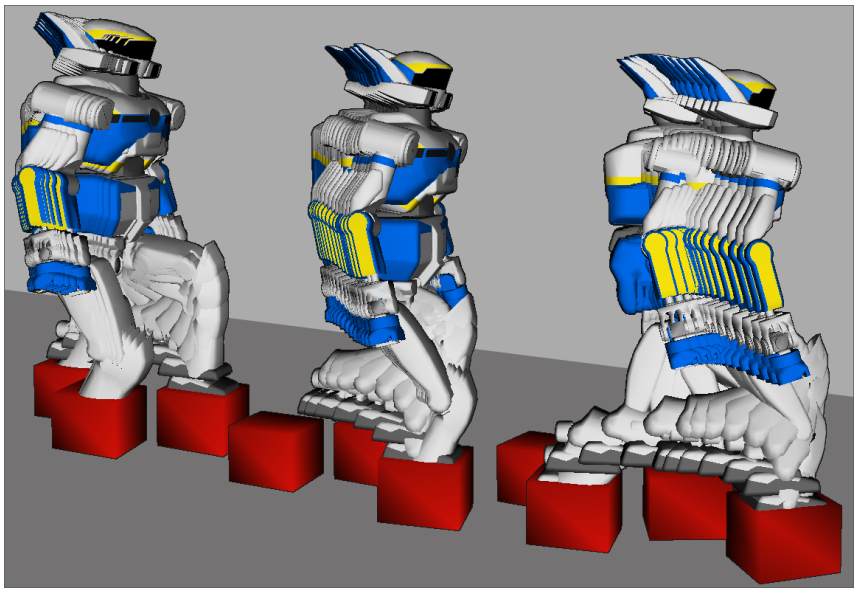


Fig. 6.14: A sequence of (raw) steps performed by HRP-2, with the corresponding configurations of the lower boxes. The feet of the robot always leave and enter the rectangular boxes from above, not from the sides. Thus, for all the obstacles whose height is less than h_m, the lower boxes can be used for conservative collision checks.

6.7.1 Weakly collision-free paths and hybrid bounding box trajectories

Since just like footprints, the configurations of the lower bounding boxes are defined by elements of $\mathbb{R}^2 \times SO(2)$, it is quite straightforward to apply the equivalence result (Theorem 6.2.1) just as we did for the preliminary application illustrated on Fig. 6.12. To obtain a good approximation of the stepping capabilities of HRP-2 we use the operation described on Fig. 6.11, and thus configurations of Φ are described by elements of $\mathbb{R}^2 \times SO(2) \times (-1, 1)$. The configuration of Φ is also used to set the configuration of the upper bounding box which is centered between $\mathcal{E}_\mathcal{L}$ and $\mathcal{E}_\mathcal{R}$, and shares the same orientation as Φ. So, for a configuration $\Phi(x, y, \theta, \delta)$ of Φ, we test the collision with the environment as follows:

- classical collision checks for the upper bounding box,
- strong collision checks for the lower bounding boxes: instead of checking 2D collisions as in sections 6.2 to 6.5 we check collisions between the lower bounding boxes and the lower part of the environment (no need to consider the obstacles above height h_m).

This is somehow an extension of the notion of weak collision-freeness, and the geometry of the bounding boxes and Φ have been chosen so that weakly collision-free paths can always be converted into (a) *a continuous motion of the upper bounding box* and (b) *a discrete sequence of configurations of the lower bounding boxes*, so that the overall hybrid motion of the two-level bounding box can be followed by a walking motion of the robot which stays inside the bounding boxes, as depicted on Fig. 6.13. On Fig. 6.14 we show such a walking motion and display only the lower bounding boxes: we can see that the swing foot always enters or leaves them from above, and thus follows $\cap$-shaped trajectories.

Algorithm 4 SetNextLeftStep($(t,\ (c_L, c_R))$)

Require: An upper bound on $t_{new} - t$: Δ_{max}
Require: A precision parameter $\epsilon > 0$

$t_{min} \leftarrow t$
$t_{max} \leftarrow \min(1,\ t + \Delta_{max})$
while `true` **do**
  $t_{new} \leftarrow \frac{t_{min}+t_{max}}{2}$
  $stepfound \leftarrow$ `false`
  if $c_R \in \Phi_R(s(t_{new}))$ **then**
    Try to find (through sampling) a feasible left step such that the next double support configuration (c_{new}, c_R) is in $\Phi(s(t_{new}))$, and such that the lower box in configuration c_{new} is collision-free.
    if such a step is found **then**
      $stepfound \leftarrow$ `true`
    end if
  end if
  if $stepfound$ **then**
    if $t_{max} - t_{new} < \epsilon$ **then**
      Break.
    end if
    $t_{min} \leftarrow t_{new}$
  else
    $t_{max} \leftarrow t_{new}$
  end if
end while
Return $(t_{new},\ (c_{new}, c_R))$

6.7.2 Reduction to a finite sequence of steps

Now we assume that a continuous weakly collision-free path has been found for Φ, and we show how to convert it into a hybrid motion of the rectangular boxes. First, let us consider again the "flea example" of Section 6.1. We denote by $(s(t))_{t\in[0,1]} \in (\mathbb{R}^2 \times SO(2))^{[0,1]}$ the solution path for the disk. When we convert it into a sequence of jumps, a natural objective is to try to minimize the number of jumps. To do so we use the following greedy algorithm (see Fig. 6.1, on the right): assuming that the current configuration of the flea is (x_0, y_0, θ_0) inside the disk of configuration $s(t_0)$, we search for the largest $t_{new} > t_0$ such that it is possible to jump from (x_0, y_0, θ_0) into the disk of configuration $s(t_{new})$. This greedy algorithm can be straightforwardly adapted to our "hybrid bounding box planning problem" and implemented with a dichotomy (see Algorithm 4 which sets the next *left* step), but attention must be paid to one important detail related to the upper box motion. Basically, during the planning phase we continuously move the upper box along the path $s(t)$, and we use classical collision checks to determine the validity of the path (the weak collision checks are for Φ, i.e. the lower boxes). However, in order to enable the kind of shortcut shown on Fig. 6.1, some non-trivial properties of the upper box must be verified. Roughly, the union of the first and last configurations of the upper box during a step must be enough to encompass the whole motion of the upper part of the robot (i.e. everything above height h_m) during this step. In our current implementation we chose an upper

box big enough to empirically verify this property, but in future work we will consider automated procedures to tune the upper box geometry in order to verify the required properties while not being overconservative.

Another detail must be carefully addressed: if the continuous path $(s(t))_{t\in[0,1]}$ is indeed made of weakly collision-free configurations, then the next step can always be found. However planners usually decide the validity of a whole continuous path based upon the validity of a discretization of this path. Therefore it can happen that the solution path is in fact not entirely weakly collision-free, and in that case the Algorithm 4 might not find a next step. Although there are several techniques to avoid that, one convenient solution is to use a margin for the collision checks when the continuous weakly collision-free solution path is searched for, but no margin (or a smaller one) when the finite sequence of lower box configurations is constructed. For more rigorous results we could try to adapt the method described in [Ferré and Laumond, 2004].

6.7.3 Smoothing

As explained in section 6.7.1 and shown on Fig. 6.13 and Fig. 6.14, the steps have initially $\cap$-shaped swing foot trajectories. Although $\cap$-shaped swing foot trajectories can slightly reduce the stepping capabilities of the robot due to potential joint limits violations, small variations of the CoM height during the steps compensate for this effect (vertical motions of the CoM is one of the slight improvements of the walking pattern generator used in this chapter compared with the original version presented in Chapter 4). However, because of the distance traveled by the swing foot such steps tend to be rather slow, and not energy-efficient. For this reason, we use the smoothing process introduced in Section 4.2.2 to continuously modify the swing foot trajectories at execution time, so that the swing foot will gently avoid the obstacle, stepping over it without an unnecessarily large motion. The effect of this smoothing process is depicted on Fig. 6.15. It can speed up the motion by up to a factor 3, and like the steps construction (section 6.7.2), it is done progressively (i.e. we smooth one step at a time).

6.8 Implementation and simulations

Let us first recall the 3 phases of our algorithm: 1) find a continuous path $s(t)$, 2) convert it into a sequence of steps, and 3) smooth the steps.

Fig. 6.16 illustrates the phases 1) and 2).

Phases 2) and 3) can require a non-negligible amount of time, especially if the sequence of steps is long. However they have the good property of being built progressively. On top of that, in practice we verify that the time needed for the construction and smoothing of one step (when the continuous path is known) is much less than the duration of the step. Thus we can use two threads (that are launched in parallel just after the continuous path has been found): one that actually executes the walking motion on the robot, and the other that progressively sets and smooths the steps. The first thread makes the robot start to move as soon as the first step has been constructed and smoothed by the second thread (it takes less than half a second). Then, while the robot is performing the first step, the second thread sets (based on the continuous path) and smooths the second step which can then be executed without interruption. This parallel process continues further and since the steps are always constructed and

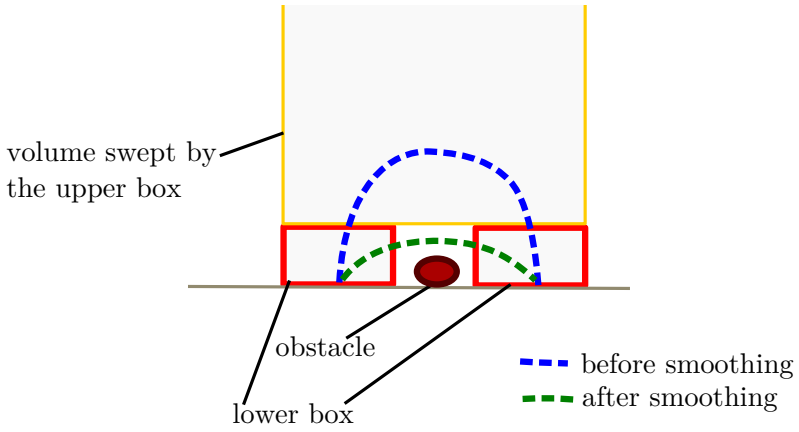


Fig. 6.15: Swing foot trajectory before and after smoothing. We can see that after the smoothing the robot trajectory is not necessarily contained in the boxes anymore.

smoothed faster than they are executed, no problem occurs. Consequently to this use of threads the relevant time-complexity of the algorithm is only the one of phase 1), unless the robot has to perform other online tasks with the same CPU(s).

As explained in previous sections, finding a continuous path $s(t)$ is a classical problem of rigid body motion planning, except for the configuration space and the configuration validity test which are specific (strong collision checks). The library OMPL ([OMPL, 2010]) provides exactly the API we need: we can easily redefine the manifold representing the configuration space, and the configuration validity test (called "state validity test" in OMPL). Then, any rigid body motion planner can be directly used. We tried the algorithms KPIECE ([Sucan and Kavraki, 2008]), SBL ([Sanchez and Latombe, 2001]), PRM ([Geraerts and Overmars, 2002]), and RRT-Connect ([Kuffner and Lavalle, 2000]), all readily available in OMPL, and compared their performances.

We chose the following experimental setup: in a fixed environment (the one of Fig. 6.17, which is a 4m×4m room cluttered with 3D obstacles lying on the floor), the robot is given 8 consecutive random goals (the same random goals for every motion planner). For each goal and each motion planner we execute our algorithm 10 times and record the average time required to find a continuous solution path, as well as the average number of steps of the sequences constructed. The computations are made on two Intel(R) Core(TM) i7 1.60GHz CPUs. The results are shown on Fig. 6.18 and 6.19 where the goals are sorted by increasing difficulty. What we can see is that RRT-Connect and KPIECE are the fastest algorithms (averaging both less than half a second on the eight goals), but that KPIECE produces sensibly longer solution paths that lead to larger sequences of steps.

As collision checking is the bottleneck of most motion planning algorithms, the time costs not only depend on the algorithm chosen but also on the quickness of our configuration validity tests, i.e. our strong collision checks.

In these tests, we have to check whether there exists a collision free configuration of the lower box in Φ_L and in Φ_R. We do this by randomly sampling configurations in Φ_L and Φ_R (which is a conservative approach). If the current Φ_L (resp. Φ_R) has a large overlap with the previously checked Φ'_L (resp. Φ'_R), which happens often, for instance when testing consecutive configurations $s(t_k)$ along a discretized segment line, then it

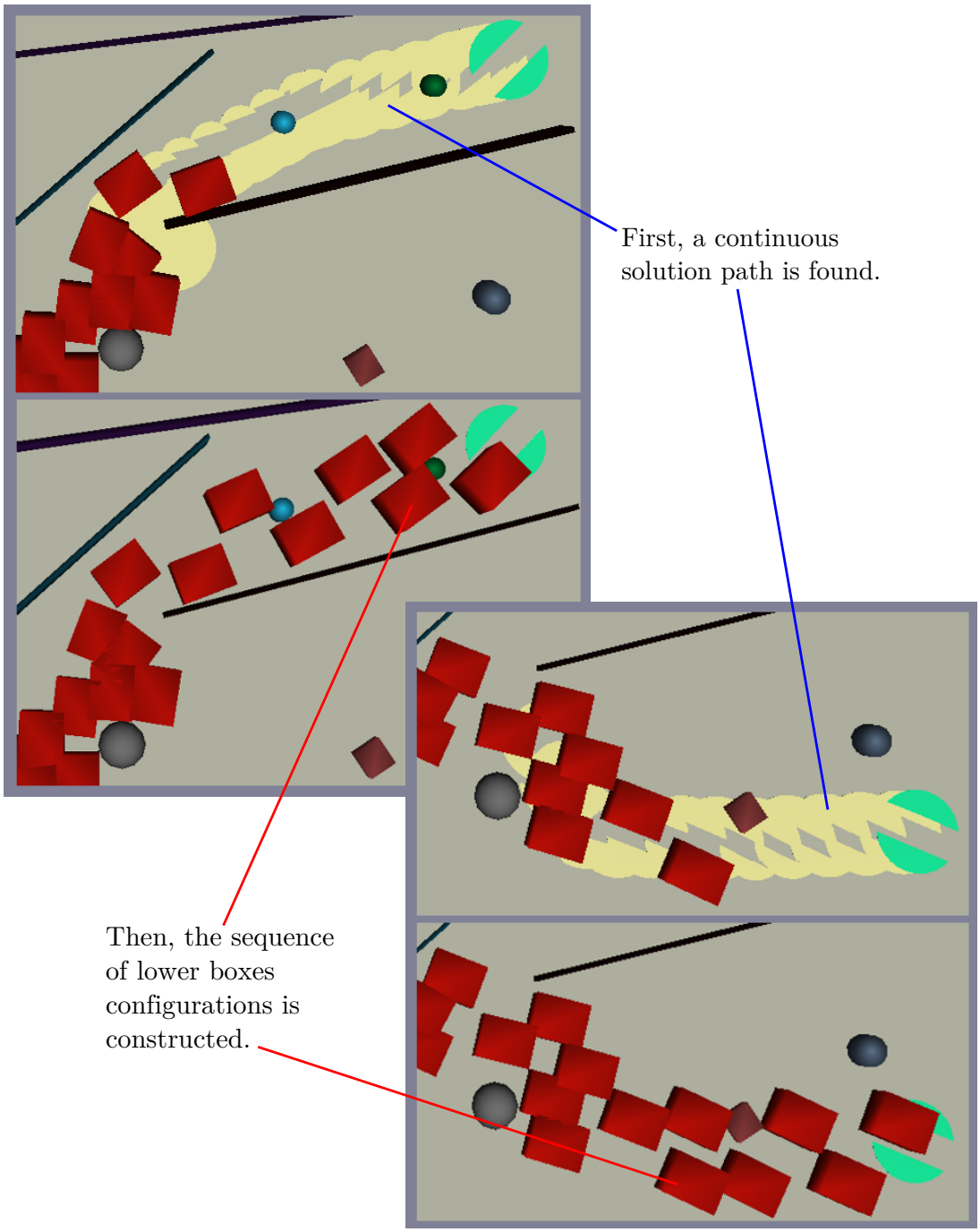


Fig. 6.16: Phases 1) and 2) of the algorithm: two examples. The initial walking trajectory (before smoothing) is such that below height h_m no point outside the lower boxes is touched by the robot (see Fig. 6.14).

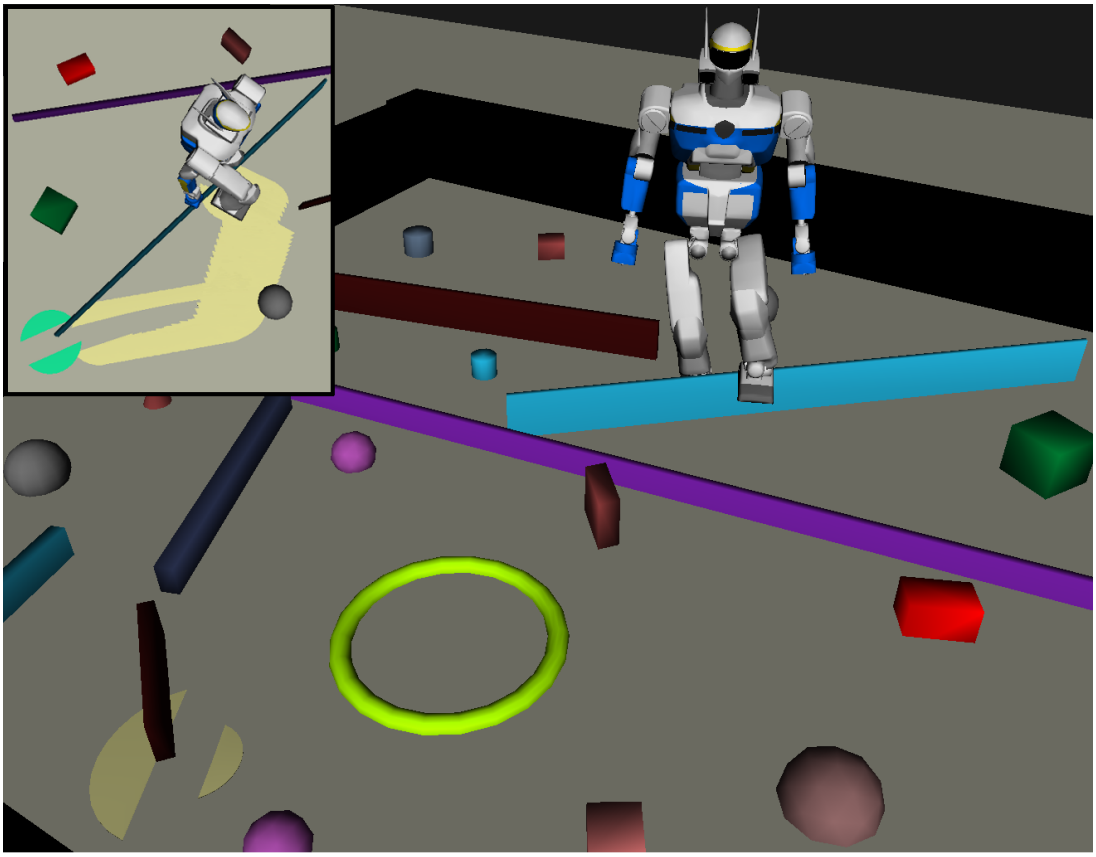


Fig. 6.17: A simulation of HRP-2 performing real-time footstep planning in an environment cluttered with 3D obstacles. On the bottom-left corner of the image a beige 2D shape made of two portions of disks can be seen. It is a representation of a weakly collision-free configuration $s(t)$ from the continuous path that the robot follows (one side is Φ_L while the other is Φ_R). Note that the configuration is weakly collision-free even though the shape intersects an obstacle. The area swept by this 2D shape along another continuous path is shown on the smaller image in the upper-left corner.

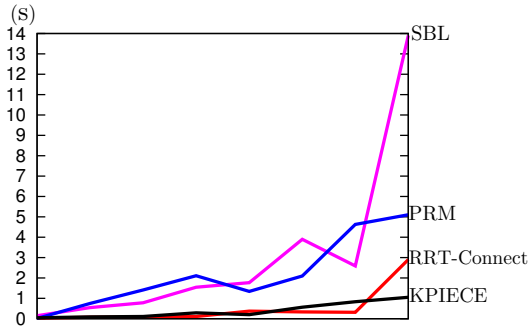


Fig. 6.18: Average durations of the planning phase for random goals sorted by increasing difficulty.

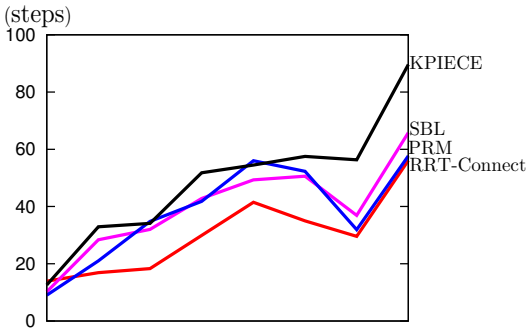


Fig. 6.19: Average number of steps of the solutions obtained for the same goals as in Fig. 6.18.

might be a waste of time to check new samples: indeed, maybe we already checked a collision-free sample that belongs to both Φ'_L and Φ_L (resp. Φ'_R and Φ_R). For this reason we keep track of the last few collision-free samples found and first test their membership to Φ_L (resp. Φ_R). This method speeds up our validity tests a bit. Note also that because many collision checks are grouped in each validity test, our algorithm is particularly well suited for parallel computations.

6.9 Conclusion and future work

In this chapter we have introduced a new bounding box method for humanoid robot footstep planning. The two-level bounding box, whose upper part moves continuously and lower part discretely, captures well the hybrid behaviour of humanoid robots (and could easily be extended to robots with more legs). Based on this bounding box our planning algorithm enables the robot to plan dynamic trajectories and accurately step over obstacles at a low computational cost. In fact, we have shown through simulations that even in a very cluttered environment, fast footstep planning can be achieved so that after receiving a new goal, the robot almost always starts to move without any noticeable delay.

Using an algorithm based on an original equivalence result, the motion of the whole hybrid bounding box is governed by continuous paths. As a result classical motion planning algorithms can be used effortlessly, and since the problem of footstep planning is very specific and much less studied than rigid body motion planning, this

generalization is clearly beneficial. In fact, our next objective is to reduce further the computation times by using more advanced motion planning libraries such as g-Planner ([Pan et al., 2010]) which takes advantage of GPU architectures to achieve extremely fast motion planning. Using such a tool for footstep planning has never been done before, and it should enable the construction of optimized path in constrained environments in the blink of an eye, and thus pave the way towards practical and robust real-time footstep planners. Adding new dimensions to the parameter space would also be interesting (for example to tackle the problems of walking on stairs or on uneven terrain) and more complex, but with the speed up brought by g-Planner the footstep planning could remain computationnally efficient. Finally, using variants of the notion of weak collision-freeness, we could try to adapt various motion planning strategies for mobile robots (see for example [Alami et al., 2007], [Pignon et al., 1993]), or improve other techniques of continuous footstep planning such as [Kanoun et al., 2009] or [Dalibard et al., 2011].

Chapter 7

Conclusion

7.1 General contributions

We believe that footstep planning is an interesting problem for 3 main reasons:

1. Of course, the first one is its usefulness: humanoid robots are built because we hope they will eventually be able to move like humans and reach their level of agility. The ability to plan steps with ease seems like a natural intermediate goal in this quest.

2. On a theoretical level, footsteps planning has both discrete and continuous aspects which makes it a very particular motion planning problem.

3. On a more practical level, it is too difficult a problem to be solved without introducing simplifying hypotheses: it thus becomes challenging to find the good simplifications that should at the same time lead to practical applications *and* let the robot exploit well its stepping capabilities. It is also better to avoid lists of heuristics because they are hard to reproduce and do not reveal general principles.

Throughout this thesis, we proposed original solutions while keeping those three points in mind.

In Chapter 2, we first considered the underlying "footprint planning" problem with the following initial remark: when we discretize this problem, we can do it so that to come down to a search problem on a 2D grid. We then wondered whether in that case there exist more efficient methods than A*-like search algorithms, for example by reasoning on the obstacles description more explicitely. That lead us to several theoretical results which in the end leave little hope for this ambition.

In Chapter 3, we started with the following remark: the complexity of the footstep planning problem is partly due to the obstacles which are *a priori* unknown, but it is also due to the intrinsic complexity of the biped robot itself, which has many restrictions and can self-collide in many ways. Since we know the model of the robot,

we can study it offline so that to precompute efficient data structures that could help us save a lot of time during online computations. Following this remark we built approximations of some feasibility tests, and used them in a few experiments with the robot HRP-2.

In Chapter 4, we introduced an original walking pattern generator with two special features that make it well suited for precomputation-based footstep planning: a) being based on half-steps, its input space has a reduced dimensionality, and b) the initially planned trajectories are improved thanks to a very fast smoothing homotopy. We also conducted a theoretical analysis of the sensitivity of this walking pattern generator, and its two main features were advantageously used in the next two chapters.

In Chapter 5, we obtained a dense discretization of the walking pattern generator input space with about 300 half-steps, and for each of them we precomputed an approximation of the volume swept by the legs of the robot. By combining these approximations with the features of the walking pattern generator we could realize fast footstep planning while allowing a large variety of steps, and notably a large variety of stepping-over motions. Using this approach we performed several challenging experiments with HRP-2.

In Chapter 6 we considered a continuous approach: starting with an equivalence result shown in the context of footprint planning, we obtained a novel algorithm that enables the transposition of footstep planning problems into more classical problems of continuous motion planning. Thanks to a hybrid bounding box for HRP-2, we used this algorithm to design an efficient method capable of quickly planning sequences of steps in very constrained environments where the robot has to step over many obstacles.

7.2 Limitations and perspectives

In this thesis we only considered footstep planning as a motion planning problem, when in fact it is obviously not limited to that. Even though we did use a stabilization algorithm in our implementations, we made the implicit assumption that trajectories can be followed with high accuracy. And with this assumption we could indeed solve some challenging problems with the real HRP-2 robot. However, in order to get a safer, more robust and more flexible behaviour, it would be better to take into account uncertainty and increase the robot compliance. To do this efficient methods for local trajectory correction and advanced control are needed. Nevertheless, fast motion planning will remain essential for local methods are all prone to local minima. Hence, for better planning and control of walking motions, the best combination is probably to first quickly generate a decent reference trajectory, and then convert it into sequences of control laws that will enable to roughly follow the reference while keeping the ability to cope with uncertainty and external perturbations, and possibly optimize important balance or energy-related criteria. The situation where the robot becomes unable to follow the reference trajectory must also be addressed, avoiding to fall being the number one priority.

In light of these facts, the three main axes of our future work will be uncertainty, compliance and control.

Appendix A

Proof of Theorem 2.2.2

Let (Q,T) be a 2-counter machine verifying the U-turn property, and, without loss of generality, let $(q_0,0,0)$ be the initial configuration.

We propose an algorithm of time complexity $O(|(Q,T)|_B)$ (we recall that $|(Q,T)|_B$ denotes the size of a classical binary encoding of (Q,T)) which decides whether all configurations are reachable or not. The algorithm works in four steps, and returns `true` if all the configurations are reachable, `false` otherwise. Here, we present it along with the proof of its correctness.

1. Let us consider (Q,T) as a graph, where the elements of Q are the vertices and the elements of T oriented edges. We first verify that all the vertices are reachable from q_0 (if not, return `false`). Then, we consider only the maximal strongly connected subgraph $(\tilde{Q},\tilde{T})$ containing q_0, which can be extracted in time $O(|(Q,T)|_B)$. The universal reachability property holds for $(\tilde{Q},\tilde{T})$ if and only if it holds for (Q,T), because since all the vertices are reachable from q_0, the universal reachability property is equivalent to $\forall (x,y)\in\mathbb{Z}^2,\ q_0 \xrightarrow[(x,y)]{*} q_0$, and sequences of transitions going from q_0 to q_0 necessarily contain only transitions of $(\tilde{Q},\tilde{T})$.

2. We consider now the 1-counter machine $(\tilde{Q},\tilde{T}_1)$ obtained from $(\tilde{Q},\tilde{T})$ by replacing each transition (q,x,y,q') by (q,x,q'). First, we check if there is any odd number $2k+1$ such that we have $q_0 \xrightarrow[(2k+1)]{*} q_0$. To do so, we replace temporarily each transition (q,x,q') by $(q,x \bmod 2,q')$, merging the identical results. We create then a new graph with twice as many vertices as $(\tilde{Q},\tilde{T}_1)$ and unweighted edges: for each $q\in\tilde{Q}$ we create $q^{(odd)}$ and $q^{(even)}$, for each transition $(q_1,0 \bmod 2,q_2)\in\tilde{T}_1$ we create the edges $(q_1^{(odd)},q_2^{(odd)})$ and $(q_1^{(even)},q_2^{(even)})$, and for each transition $(q_1,1 \bmod 2,q_2)\in\tilde{T}_1$ we create the edges $(q_1^{(odd)},q_2^{(even)})$ and $(q_1^{(even)},q_2^{(even)})$. An example of this construction is shown on Fig. A.1.

   A configuration of the form $(q_0,2k+1)$ is reachable in $(\tilde{Q},\tilde{T}_1)$ if and only if there is a path from $q_0^{(even)}$ to $q_0^{(odd)}$ in the newly created graph. We can verify that easily by using a breadth-first search algorithm (of complexity $O(|(Q,T)|_B)$).

   If no configuration $(q_0,2k+1)$ is reachable, it means that in (Q,T), configurations of the form $(q_0,2k+1,k')$ are not reachable, and therefore the algorithm

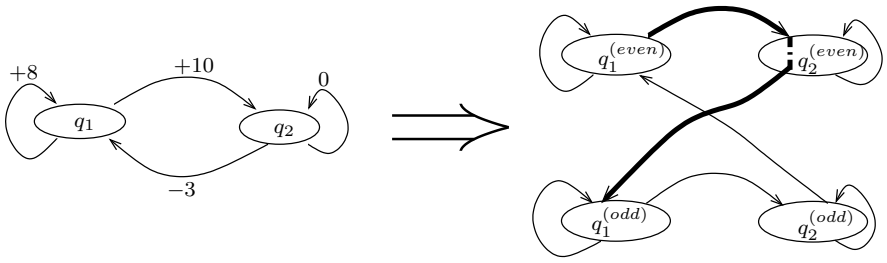


Fig. A.1: Checking whether an odd value is reachable for a 1-counter machine.

can return `false`. Otherwise, we proceed to next step.

3. We still consider $(\tilde{Q}, \tilde{T}_1)$. Let us pose $\{x_1, x_2, \ldots, x_K\} = \{x \in \mathbb{Z} | \exists (q, x, q') \in \tilde{T}_1\}$. We can calculate $gcd(x_1, \ldots, x_K)$ (in time linear in the encoding of $x_1, x_2, \ldots, x_K$) along with a linear combination $\sum_{i=1}^{K} a_i x_i = gcd(x_1, \ldots, x_K)$. If $gcd(x_1, \ldots, x_K)$ is not 1, then $(q_0, 1)$ cannot be reached, and we can return `false`. Thus we can assume $gcd(x_1, \ldots, x_K) = 1$. Let us show now that for each x_i, $(q_0, 2zx_i)$ is reachable for any $z \in \mathbb{Z}$.

   Let $q_0 \xrightarrow[(M)]{*} q \xrightarrow[(x_i)]{*} q' \xrightarrow[(N)]{*} q_0$ be a path from q_0 to itself. Thanks to the U-turn property, we know that the following path also exists:

$$q_0 \xrightarrow[(0)]{*} (q_0)^\top \xrightarrow[(-M)]{*} q^\top \xrightarrow[(0)]{*} q \xrightarrow[(x_i)]{*} q' \xrightarrow[(0)]{*} (q')^\top \xrightarrow[(-N)]{*} (q_0)^\top \xrightarrow[(0)]{*} q_0$$

   If we follow this path just after the previous one, we obtain a path leading to the configuration $(q_0, 2x_i)$, which we can repeat to reach any configuration $(q_0, 2zx_i)$ with $z \in \mathbb{N}$. Thanks to the U-turn property, any of these paths can be transformed into a path leading to configuration $(q_0, -2zx_i)$. Therefore, the configuration $(q_0, 2zx_i)$ is reachable for any $z \in \mathbb{Z}$.

   We deduce that we can combine several paths in order to reach the configuration $(q_0, 2gcd(x_1, \ldots, x_K))$, that is to say $(q_0, 2)$. The configuration $(q_0, -2)$ is also reachable.

   Since we know that there exists a reachable configuration of the form $(q_0, 2k+1)$, we can combine several paths in order to finally obtain a path reaching $(q_0, 1)$.

4. We now use the path $q_0 \xrightarrow[(1)]{*} q_0$ just obtained, but this time on the 2-counter machine $(\tilde{Q}, \tilde{T})$, and from the configuration $(q_0, 0, 0)$. It reaches a configuration $(q_0, 1, a)$. a can be computed efficiently if during the previous step, we keep updating the number of occurences of each transition in all the paths that were combined now to reach $(q_0, 1, a)$.

   We then change every transition $(q, x, y, q') \in \tilde{T}$ into $(q, x, y - ax, q')$. Let us call $(\tilde{Q}, \widehat{T})$ the 2-counter machine consequently obtained. $(\tilde{Q}, \widehat{T})$ keeps the U-turn

property, and every path $q \xrightarrow[(x,y)]{*} q'$ in $(\tilde{Q}, \tilde{T})$ corresponds to a path $q \xrightarrow[(x,y-ax)]{*} q'$ in $(\tilde{Q}, \widehat{T})$. The function $f : (x, y) \mapsto (x, y - ax)$ is a bijection of $\mathbb{Z}^2$, and therefore we deduce that the universal reachability property holds for $(\tilde{Q}, \tilde{T})$ if and only if it holds for $(\tilde{Q}, \widehat{T})$. In $(\tilde{Q}, \widehat{T})$, we have a path from the initial configuration $(q_0, 0, 0)$ to $(q_0, 1, a - a) = (q_0, 1, 0)$. Thanks to the U-turn property, we can transform it into a path from $(q_0, 0, 0)$ to $(q_0, -1, 0)$, and since we can repeat these paths at will, we have: $\forall z \in \mathbb{Z},\ q_0 \xrightarrow[(z,0)]{*} q_0$.

Let us now consider the 1-counter machine $(\tilde{Q}, \widehat{T}_2)$ obtained from $(\tilde{Q}, \widehat{T})$ by replacing each transition (q, x, y, q') by (q, y, q'). We prove that $(\tilde{Q}, \widehat{T})$ verifies the universal reachability property if and only if $(\tilde{Q}, \widehat{T}_2)$ verifies $\forall z \in \mathbb{Z},\ q_0 \xrightarrow[(z)]{*} q_0$.

The direct implication is trivial ($q_0 \xrightarrow[(w,z)]{*} q_0$ in $(\tilde{Q}, \widehat{T})$ implies $q_0 \xrightarrow[(z)]{*} q_0$ in $(\tilde{Q}, \widehat{T}_2)$). Conversely, if $(\tilde{Q}, \widehat{T}_2)$ verifies $\forall z \in \mathbb{Z},\ q_0 \xrightarrow[(z)]{*} q_0$, let (x, y) be a couple of integers. We have $q_0 \xrightarrow[(y)]{*} q_0$. This path can be directly applied to $(\tilde{Q}, \widehat{T})$ in which it leads, for some $b \in \mathbb{Z}$, to $q_0 \xrightarrow[(b,y)]{*} q_0$. On top of that, $(\tilde{Q}, \widehat{T})$ verifies $\forall z \in \mathbb{Z},\ q_0 \xrightarrow[(z,0)]{*} q_0$, so we have also: $q_0 \xrightarrow[(-b+x,0)]{*} q_0$. If we combine this path to $q_0 \xrightarrow[(b,y)]{*} q_0$, we obtain a path $q_0 \xrightarrow[(x,y)]{*} q_0$. It follows that $(\tilde{Q}, \widehat{T})$ verifies the universal reachability property.

So, we just have to check whether $(\tilde{Q}, \widehat{T}_2)$ verifies $\forall z \in \mathbb{Z},\ q_0 \xrightarrow[(z)]{*} q_0$. This is done with the same techniques as the ones used in steps 2 and 3 on $(\tilde{Q}, \tilde{T}_1)$ (using the greatest common divisor of the transition costs and the U-turn property). If the property is verified, the algorithm returns `true`, otherwise it returns `false`. □

Appendix B

Proof of Theorem 2.3.2

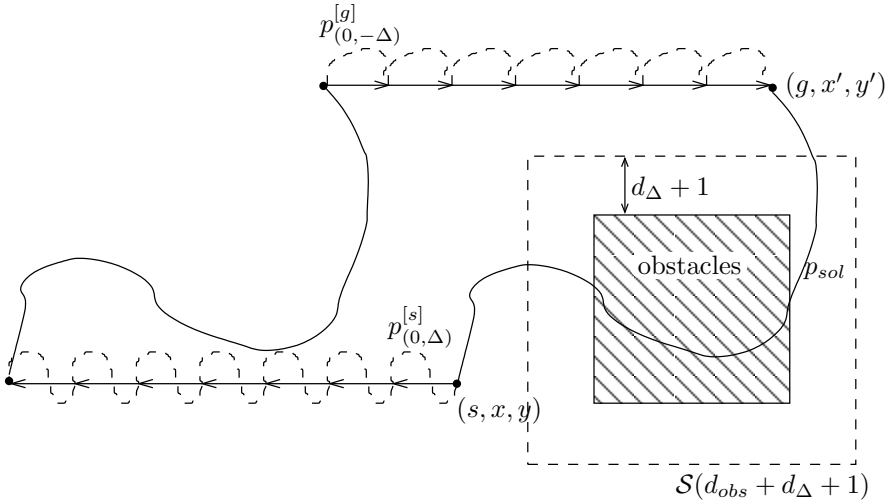


Fig. B.1: A construction to avoid the obstacles.

Let (Q, T) be a 2-counter machine with finite guards verifying the Four-Vectors property. Let $u = (u_1, u_2)$ and $v = (v_1, v_2)$ be two non-colinear vectors such that for each $q \in Q$, we have (when we ignore the guards): $q \xrightarrow[u]{*} q$, $q \xrightarrow[-u]{*} q$, $q \xrightarrow[v]{*} q$, and $q \xrightarrow[-v]{*} q$. Let us pose $\Delta = u_2 \times v_1 - u_1 \times v_2 \neq 0$. With simple combinations of the previous relations, we obtain: $q \xrightarrow[(0,\Delta)]{*} q$, $q \xrightarrow[(0,-\Delta)]{*} q$, $q \xrightarrow[(\Delta,0)]{*} q$, and $q \xrightarrow[(-\Delta,0)]{*} q$. For each state $q \in Q$, we choose in the graph (Q, T) four finite paths implementing these relations, and denote them by $p^{[q]}_{(0,\Delta)}$, $p^{[q]}_{(0,-\Delta)}$, $p^{[q]}_{(\Delta,0)}$, $p^{[q]}_{(-\Delta,0)}$.

For $d \in \mathbb{N}$, we denote by $\mathcal{S}(d)$ the set of positions $(v_1, v_2) \in \mathbb{Z}^2$ such that $|v_1| \leq d \wedge |v_2| \leq d$. Since the obstacles are finite, there exists an integer d_{obs} such that $\mathcal{S}(d_{obs})$ contains all the obstacles.

Besides, there exists $d_\Delta \in \mathbb{N}$ such that for all $q \in Q$, the paths $p^{[q]}_{(0,\Delta)}$, $p^{[q]}_{(0,-\Delta)}$, $p^{[q]}_{(\Delta,0)}$, and $p^{[q]}_{(-\Delta,0)}$, starting from configuration $(q, 0, 0)$, are all entirely contained in $\mathcal{S}(d_\Delta)$.

With these definitions, we are ready to prove the following lemma:

Lemma B.0.1. *Let (g, x', y') be a configuration to reach from a configuration (s, x, y), such that both (x', y') and (x, y) are outside $\mathcal{S}(d_{obs} + d_\Delta + 1)$. In these conditions,*

(g, x', y') is reachable from (s, x, y) in the environment with the obstacles if and only if it is also reachable from (s, x, y) in the free environment.

Of course, if (g, x', y') is reachable from (s, x, y) in the environment with the obstacles, it is also reachable from (s, x, y) in the free environment.

The demonstration of the other implication uses a construction illustrated on Fig. B.1: from (s, x, y), among $p^{[s]}_{(0,\Delta)}$, $p^{[s]}_{(0,-\Delta)}$, $p^{[s]}_{(\Delta,0)}$, and $p^{[s]}_{(-\Delta,0)}$, there is at most one path which, repeated, would lead to an obstacle. The same is true for (g, x', y') even if we apply the paths backwards. Thus, there exist two corresponding paths (say $p^{[s]}_{(0,\Delta)}$ and $p^{[g]}_{(0,-\Delta)}$, without loss of generality) which can be repeated endlessly from both configurations (backwards for $p^{[g]}_{(0,-\Delta)}$), without ever leading to a collision with the obstacles. If there exists a path p_{sol} from (s, x, y) to (g, x', y') in the free environment, then in the environment with obstacles, we can first "go far from the obstacles", repeating $p^{[s]}_{(0,\Delta)}$ from (s, x, y), then apply p_{sol} without any collision, and then come back with $p^{[g]}_{(0,-\Delta)}$, until ultimately reaching (g, x', y') without any collision along the whole path. This concludes the demonstration of Lemma B.0.1. □

Now we can use Lemma B.0.1 to finish the proof of Theorem 2.3.2. We demonstrate the decidability in the 4 following cases:

1. If (s, x, y) and (g, x', y') are both outside $\mathcal{S}(d_{obs} + d_\Delta + 1)$, then Lemma B.0.1 applies and we just have to check the reachability in the free environment, which has been shown decidable ($\in$ NP, Theorem 2.2.1).

2. If $(s, x, y) \in \mathcal{S}(d_{obs} + d_\Delta + 1)$ and $(g, x', y') \notin \mathcal{S}(d_{obs} + d_\Delta + 1)$, then if p_{sol} is a path going from (s, x, y) to (g, x', y'), there must be a first configuration (q, a, b) outside $\mathcal{S}(d_{obs} + d_\Delta + 1)$ in p_{sol}. Since there is a finite number of transitions, their "length" is bounded, and there is only a finite number of candidates for such a configuration (q, a, b). Besides, the path from (s, x, y) to (q, a, b) contains only one configuration outside $\mathcal{S}(d_{obs} + d_\Delta + 1)$. Since $\mathcal{S}(d_{obs} + d_\Delta + 1)$ contains $(2 \times (d_{obs} + d_\Delta + 1) + 1)^2$ positions, there are $(2 \times (d_{obs} + d_\Delta + 1) + 1)^2 \times |Q|$ possible configurations in it (where $|Q|$ is the number of states), and thus a path of size at least $(2 \times (d_{obs} + d_\Delta + 1) + 1)^2 \times |Q| + 1$ necessarily passes through the same configuration twice, and therefore its size can be reduced. It implies that we can search for paths from (s, x, y) to a candidate configurations (q, a, b) with an upper bound on the paths lengths: $(2 \times (d_{obs} + d_\Delta + 1) + 1)^2 \times |Q| + 2$. Hence, the total number of paths to consider is finite, and for each reachable candidate (q, a, b), we can check if there is a path leading to (g, x', y') without considering the obstacles, since both configurations are outside $\mathcal{S}(d_{obs} + d_\Delta + 1)$.

3. If $(s, x, y) \notin \mathcal{S}(d_{obs} + d_\Delta + 1)$ and $(g, x', y') \in \mathcal{S}(d_{obs} + d_\Delta + 1)$, we can reverse all transitions in (Q, T), and come down to case 2, considering the reachability of (s, x, y) from (g, x', y').

4. If $(s, x, y) \in \mathcal{S}(d_{obs} + d_\Delta + 1)$ and $(g, x', y') \in \mathcal{S}(d_{obs} + d_\Delta + 1)$, either the path between (s, x, y) and (g, x', y') stays inside $\mathcal{S}(d_{obs}+d_\Delta+1)$ and we need to consider only paths of length at most $(2 \times (d_{obs} + d_\Delta + 1) + 1)^2 \times |Q| + 1$, or there is a first configuration outside $\mathcal{S}(d_{obs} + d_\Delta + 1)$: (q_1, a_1, b_1), and a last configuration outside $\mathcal{S}(d_{obs} + d_\Delta + 1)$: (q_2, a_2, b_2). Using similar arguments as in cases 2 and 3, there are only a finite number of candidates for such configurations, and we

can check the reachability of (q_1, a_1, b_1) from (s, x, y) as well as the reachability of (g, x', y') from (q_2, a_2, b_2) with paths of bounded lengths. The reachability of (q_2, a_2, b_2) from (q_1, a_1, b_1) comes down to case 1. □

Appendix C

Proof of Theorem 2.3.3

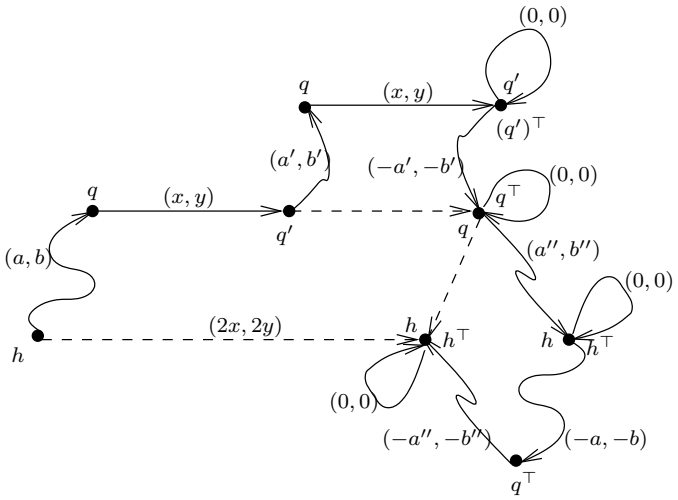


Fig. C.1: Combinations of paths to obtain $h \xrightarrow[(2x,2y)]{*} h$

Let (Q, T) be a strongly connected 2-counter machine with at least two non-colinear transitions, and verifying the U-turn property.
Let (q, x, y, q') and (p, z, t, p') be two non-colinear transitions, i.e. such that (x, y) and (z, t) are not colinear. For any $h \in Q$, since (Q, T) is strongly connected, we have for some $(a, b) \in \mathbb{Z}^2$, $h \xrightarrow[(a,b)]{*} q$, and therefore, $h \xrightarrow[(a+x,b+y)]{*} q'$. We have also for some $(a', b') \in \mathbb{Z}^2$, $q' \xrightarrow[(a',b')]{*} q$ (because (Q, T) is strongly connected). So we have also $q' \xrightarrow[(a'+x,b'+y)]{*} q'$. We can combine that to $q' \xrightarrow[(0,0)]{*} (q')^\top$ and $(q')^\top \xrightarrow[(-a',-b')]{*} (q)^\top$ in order to obtain: $q' \xrightarrow[(x,y)]{*} q$. Hence, $h \xrightarrow[(a+2x,b+2y)]{*} q$.

Now we show $q \xrightarrow[(-a,-b)]{*} h$. Since (Q, T) is strongly connected, we have again for some $(a'', b'') \in \mathbb{Z}^2$, $q \xrightarrow[(a'',b'')]{*} h$. Using $h^\top \xrightarrow[(-a,-b)]{*} q^\top$ after a U-turn from h, we get: $h \xrightarrow[(-a,-b)]{*} q^\top$. Since we have also $q^\top \xrightarrow[(-a'',-b'')]{*} h^\top$, we can combine the three paths and a U-turn to obtain $q \xrightarrow[(-a,-b)]{*} h$.

We combine $q \xrightarrow[(-a,-b)]{*} h$ to $h \xrightarrow[(a+2x,b+2y)]{*} q$, which finally leads to:

$$h \xrightarrow[(2x,2y)]{*} h$$

Fig. C.1 sums up the combinations used to obtain $h \xrightarrow[(2x,2y)]{*} h$.

Using the U-turn property, we easily obtain also:

$$h \xrightarrow[(-2x,-2y)]{*} h$$

With the same reasoning, but this time based on the transition (p, z, t, p'), we obtain:

$$h \xrightarrow[(2z,2t)]{*} h$$

and

$$h \xrightarrow[(-2z,-2t)]{*} h$$

Since $(2x, 2y)$ and $(2z, 2t)$ are not colinear, it concludes the demonstration. □

Appendix D

NP-hardness of a 2D discrete shortest path problem

Let us recall the input of our problem: an initial position $S \in \mathbb{Z}^2$, a goal $G \in Z^2$, and a list of disjoint polygons whose vertices are in $\mathbb{Z}^2$, the total number of vertices being n. The question is: considering the point robot described on Fig. 2.8 (it can jump in the 4 main directions, with jumps of length 1, 2, 3 or 4), what is the minimum number of jumps necessary to go from S to G while avoiding the obstacles?

To show that this problem is NP-hard, we reduce 3-SAT to it, and although we will also have to introduce radically different techniques, we mainly follow some of the key ideas of the proof in [Canny and Reif, 1987] where the 3D Euclidean shortest path problem is shown NP-hard.

Let us describe an instance of the 3-SAT problem: we have k variables $v_1, \ldots, v_k$, and denote all the literals (ie. variables or their negation) by $l_1, \ldots, l_{2k}$, with $l_i = v_i$ for $i \leq k$ and $l_{k+i} = \neg v_i$. There are K clauses, each clause being the disjunction of 3 literals. Thus, we can write the whole formula as follows:

$$\bigwedge_{i=1}^{K} (l_{p_i} \vee l_{q_i} \vee l_{r_i}),$$

and the question is: is there an assignment of the variables that evaluates this boolean formula to 1? We will now construct an instance of our problem which, if solved, gives an answer to this question.

Let us first introduce an operation of path splitting. In [Canny and Reif, 1987], the technique of path splitting is used to double the number of *shortest path classes* with only a constant number of obstacles. In our case, we do not reason on *shortest path classes* but on single shortest paths, and a similar path splitting tool cannot be obtained. Nevertheless we can create an exponential number of points at the same distance from the source with only a polynomial number of obstacles. As in the rest of the proof, a formal description of our gadgets would be very tedious, so we will highly rely on figures to convey the main ideas of the demonstration which is easier to "visualize" than to formally describe. We first illustrate our path splitting operation with Fig. D.1 which shows in broad outline how to easily obtain points that are at the same "distance" (in number of jumps) from the source, but with a large number of obstacles. The goal is to create somehow a tree of paths, and one important property is that for 2^k "output paths" there are only k strata of obstacles, and on each stratum, the obstacles are regularly spaced, i.e. their organization is periodic. We denote by

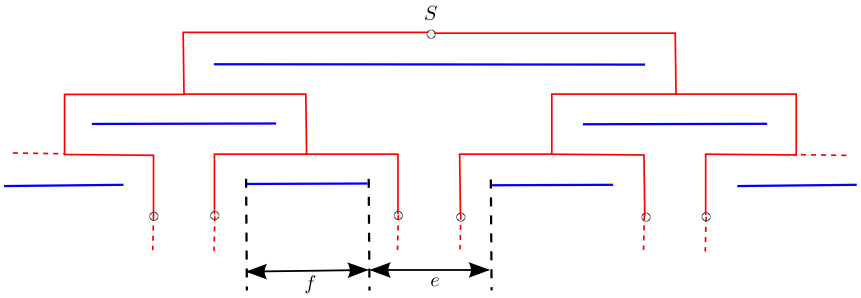


Fig. D.1: The global structure of a path splitter.

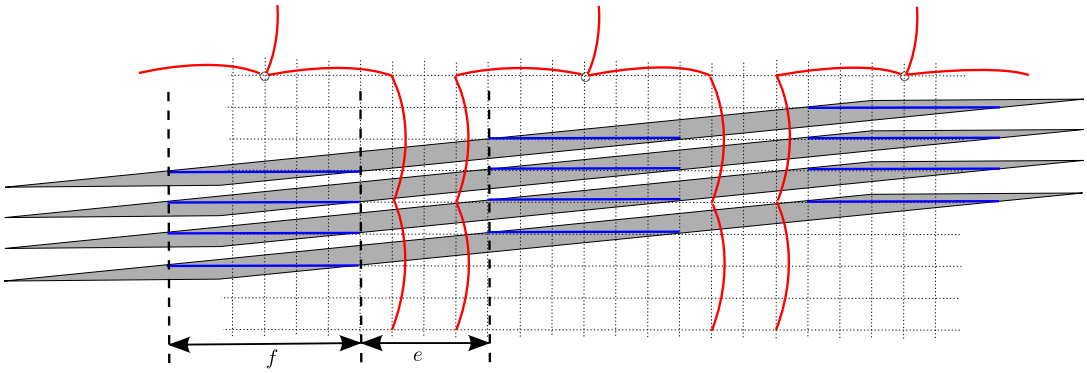


Fig. D.2: The obstacles for a stratum of path splitting.

(f, e) this periodicity: an obstacle of length f, then no obstacles for e columns, then an obstacle of length f, etc. On Fig. D.2, we show how one stratum can actually be obtained with 4 obstacles of 4 vertices each. Of course f and e can be changed at will, and using this construction for each stratum of the tree structure of Fig. D.1, we can obtain 2^k points at the same distance from the source with a number of obstacles polynomial in k. In fact, we modify the structure a bit in order to obtain 2^k groups of 3 points, as shown on Fig. D.3. These points correspond to shortest paths of same length, and for all the other points on the same horizontal line, at least one additional jump is required to reach them from S. Here we do not specify the distance between two consecutive groups or two consecutive points in the same group, but it should be quite large, and the distance between two consecutive groups must be a multiple of the distance between two consecutive points in the same group.

So, this operation of path splitting is represented by the symbol of Fig. D.3, and if we use the same construction upside down, we obtain a "path merger" that we put just above the goal G. This gives 3×2^k columns that will be of prime importance in the rest of the demontration.

In fact, to each group of 3 columns corresponds an assignment of the variables v_i. To get it we simply number the groups (from 0 to $2^k - 1$), and each assignment follows the binary encoding of its group: v_i takes the value 0 if the i-th bit is 0, and 1 otherwise. The 3-SAT formula has clauses with exactly 3 literals, and in each group the columns correspond to the first, second and third literal of a clause.

Here is an outline of the proof:

- We will consider the clauses one by one and for each clause, we will "filter" groups of columns, i.e. add some "delay" relatively to a "reference" (the meaning of these

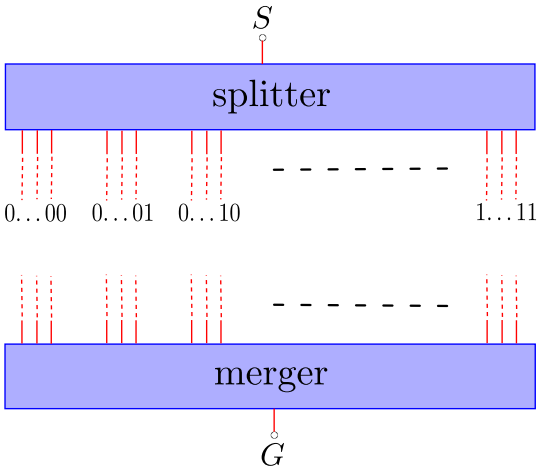


Fig. D.3: 3×2^k columns between S and G.

terms will become clearer as the demonstration goes on), with the following rule: columns are delayed if and only if they belong to a group whose assignment does *not* satisfy the clause.

- We will repeat this filtering sequentially, and in the end, there will exist a group of columns that has never been delayed if and only if the whole 3-SAT formula is satisfiable. For such columns the length ref of the shortest path (in number of jumps) from S to G can be computed, and what follows is that the satisfiability of the 3-SAT formula will be equivalent to the existence of a path of at most ref jumps going from S to G.

We now start explaining the proof in more details by illustrating the principles of "delaying" and "filtering". On Fig. D.4, there are five columns; on each column, the objective of the point robot is to go down as fast as it can, and thus if possible to perform jumps of length 4. What we call *reference* is somehow a maximum speed: it corresponds to the jumps performed by a point robot that came from S onto a virtual column which is unaffected by specific obstacles (i.e. this virtual column only "meets" the obstacles that affect all columns). This reference is also what we call "delay 0". On top of Fig. D.4, all the columns are assumed to be at delay 0. But because of the obstacles, the delay is increased on the second and fourth column. Indeed, when trying to follow the reference, just before the obstacle the point robot is obliged to make a smaller jump. On the example of Fig. D.4 a jump of length 3 has to be made, which sets the delay to +1. Then, a global obstacle that does not affect the reference further increases this delay, and it becomes +4 on columns 2 and 4.

A remark on delays: since the reference corresponds to jumps of length 4, delays cannot be deliberately reduced, but they can be deliberately increased by making smaller jumps. Thus, although we will keep the notations +1, +2, +3, etc., refering to minimum delays, a more accurate way to describe them would be to use the notations ≥ 1, ≥ 2, ≥ 3, etc.

The difficulty will be to delay a large number of columns with only a constant number of obstacles. To show how this is done, we start with the first clause: $l_{p_1} \vee l_{q_1} \vee l_{r_1}$. We will 3 "filters", one for each literal. With the first filter, our objective is to delay the columns whose assignment does not evaluate l_p to true (e.g. if l_p is $\neg v_i$, we

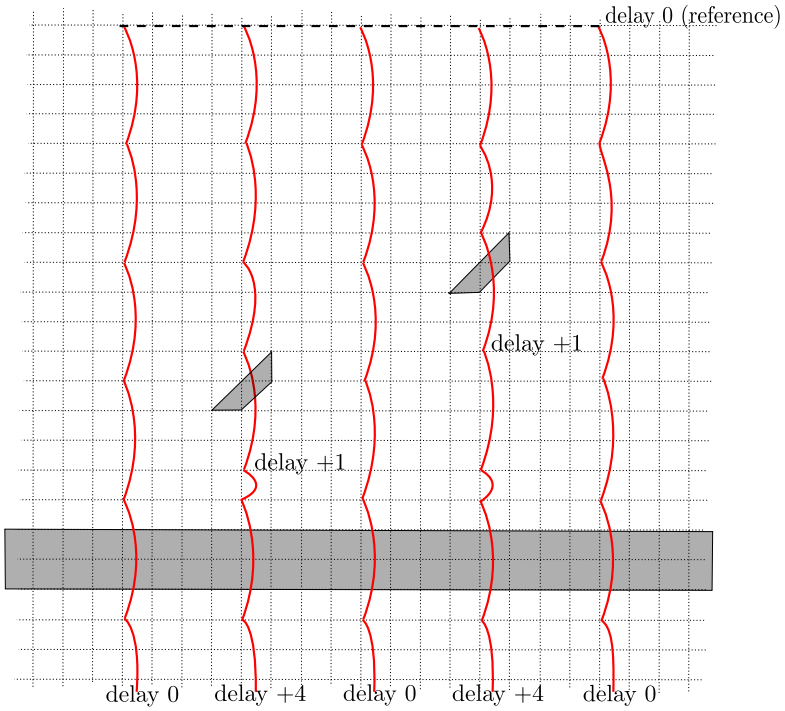


Fig. D.4: Adding delay with obstacles: an example.

want to delay the columns whose number has the i-th bit equal to 1). However, since we are dealing with the first literal of the clause we want to only affect the first column of each group. So, as a preliminary operation we somehow separate these columns from the other ones with the construction illustrated on Fig. D.5. It uses only one very large but thin obstacle which sets the delay on the first column of each group to +2. To put this obstacle at the right position, it is necessary to know where the jumps of the reference are, but this is something we can always keep track of. While Fig. D.5 shows the details, Fig. D.6 represents the same construction on a wider scale.

So, the first column of each group has now a delay of (at least) +2. Then we filter all the columns belonging to a group whose assignment does not evaluate l_{p_1} to true. The details of the construction are shown on Fig. D.7, and a more schematic view for two distinct examples is shown on Fig. D.8: by adapting the periodicity of the obstacles with values f and e, we can decide which groups are filtered. This corresponds to the "literal filtering" operation in [Canny and Reif, 1987]. As shown on Fig. D.7, the construction uses 4 obstacles of 4 vertices each, and has a very specific effect on delays. After the obstacle of Fig. D.6, there are two types of columns: the ones with delay 0, and the ones with delay +2. For the columns whose assignment evaluates l_{p_1} to true, the delay is unchanged. Otherwise, we can verify that the construction of Fig. D.7 increases the delay by either 1 or 2. The construction of Fig. D.7 might seem unnecessarily complicated, but it is actually one of the key gadgets of the proof, and a similar "literal filtering" could not be obtained if, for example, the point robot could only make jumps of length 3 or less. So, after this "literal filtering":

- for each group whose assignment evaluates l_{p_1} to true, the first column has delay +2,
- for each group whose assignment evaluates l_{p_1} to true, the second and third

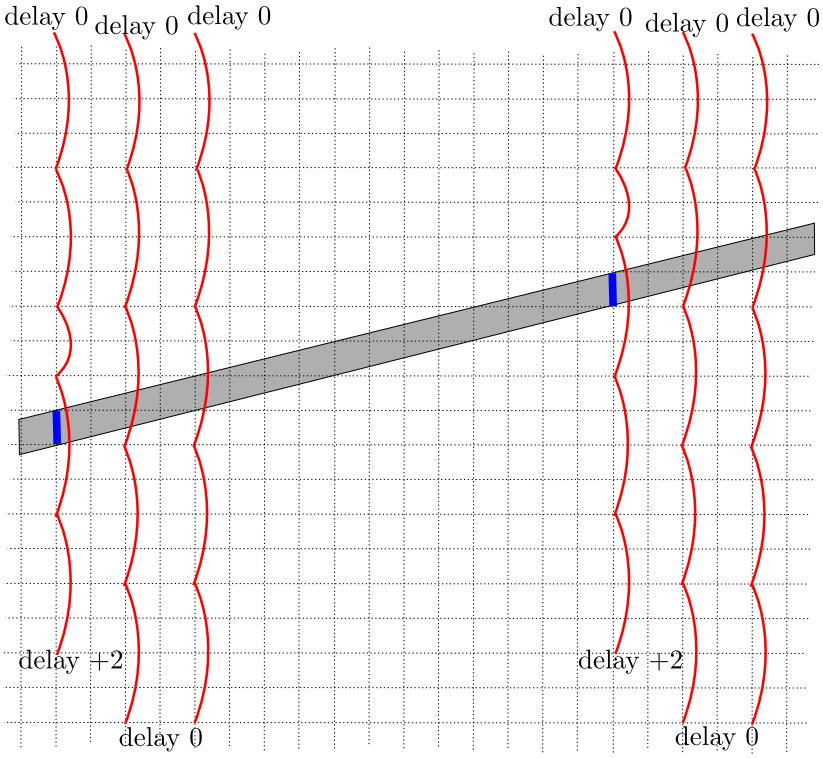


Fig. D.5: Delaying the first column of each group.

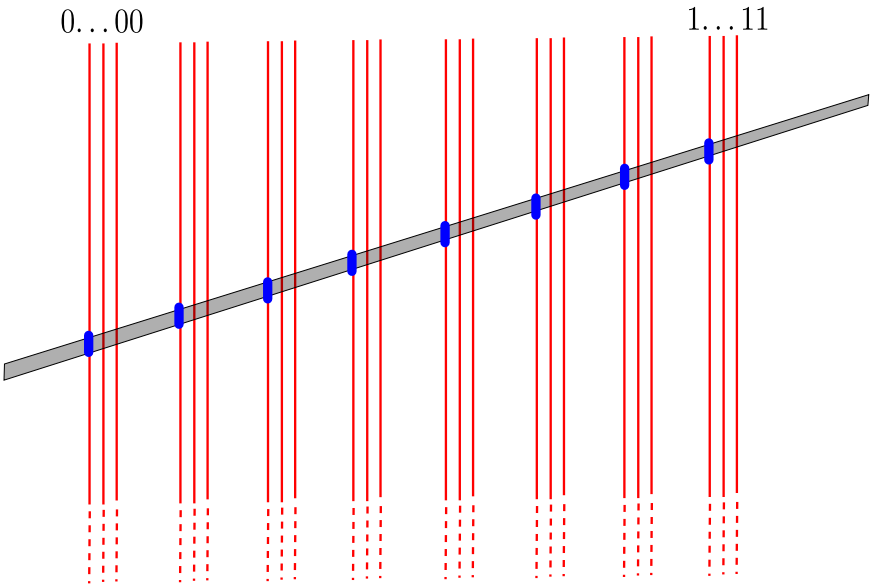


Fig. D.6: Delaying the first column of each group: representation on a wider scale.

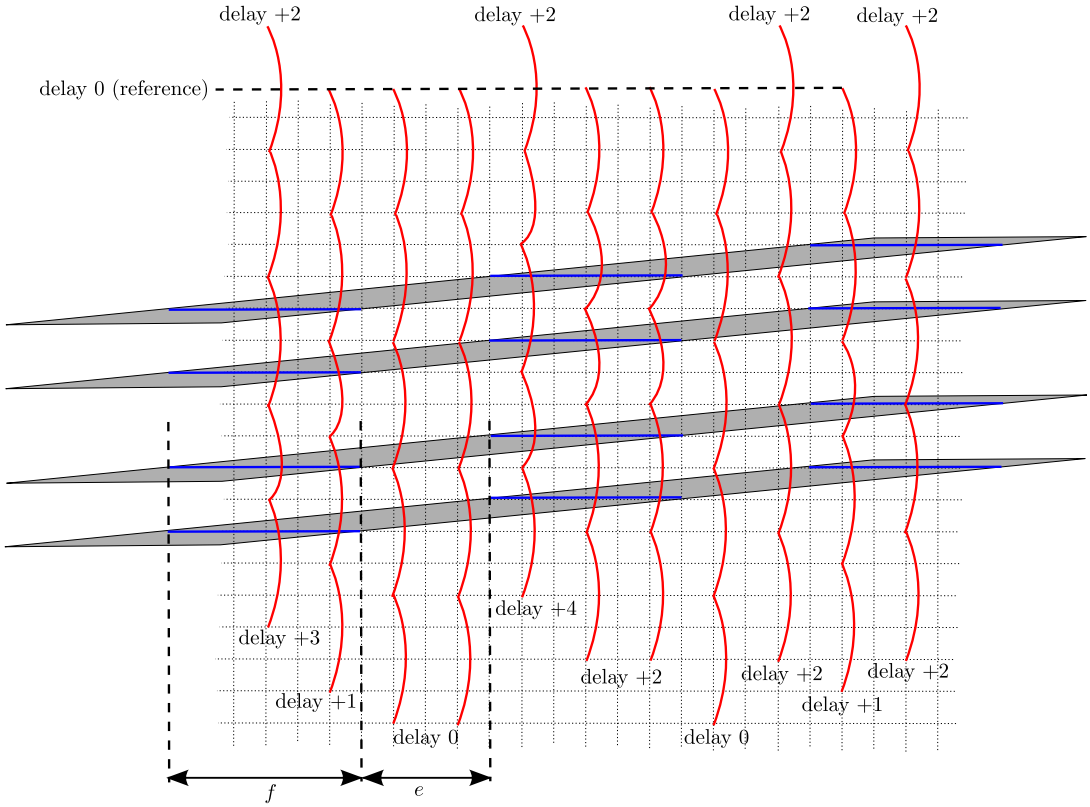


Fig. D.7: Detail of the obstacles used for literal filtering.

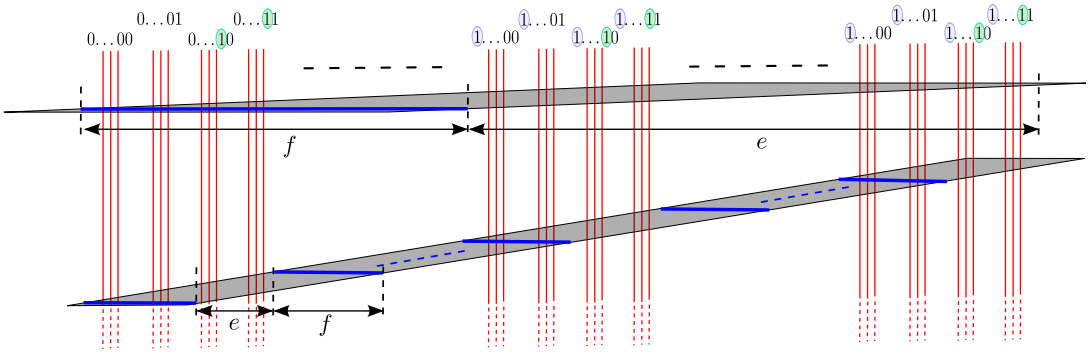


Fig. D.8: Adapting e and f for literal filtering: two examples.

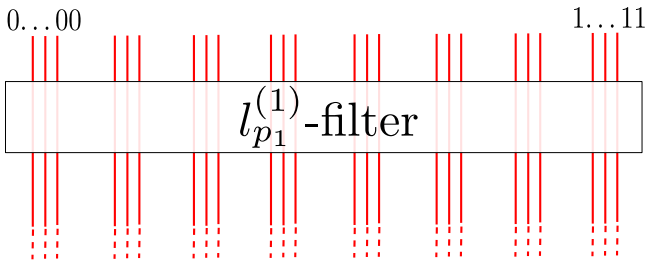


Fig. D.9: The symbol for literal filtering.

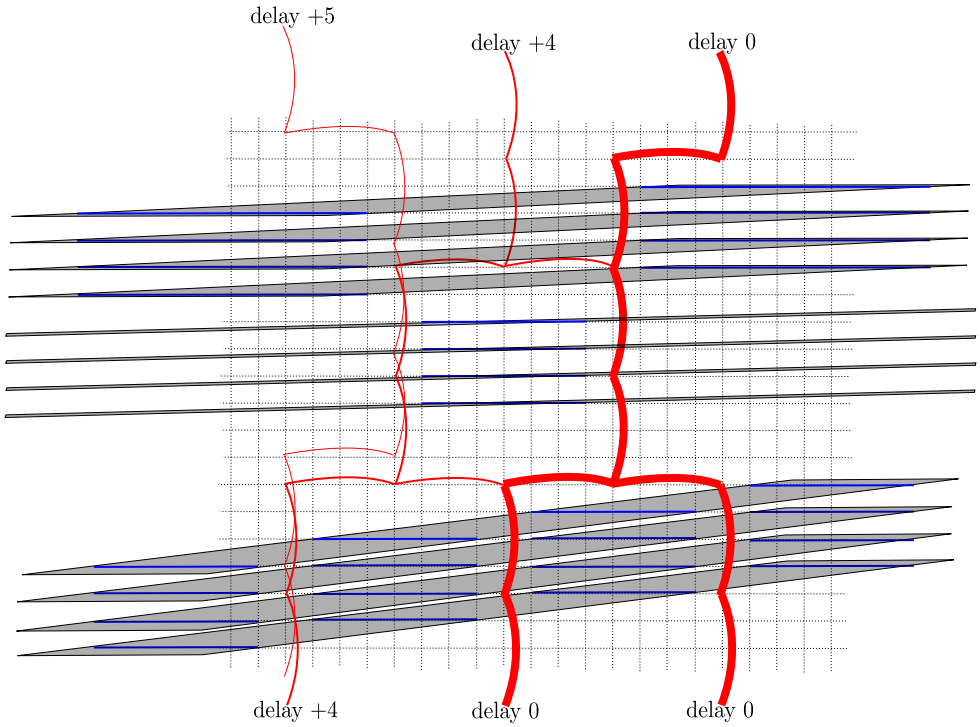


Fig. D.10: By repeating this construction twice, the lowest delay becomes the delay of all three columns.

column have delay 0,

- for each group whose assignment evaluates l_{p_1} to false, the second and third column have delay +1 or +2,
- for each group whose assignment evaluates l_{p_1} to false, the first column has delay +3 or +4,

We notice that only the first columns of the groups whose assignment evaluates l_{p_1} to false have a delay of at least +3. With an obstacle similar to the one at the bottom of Fig. D.4, it is easy to make the delays 0 and +1 become +2 (it will also make delays +4 become +6). After this obstacle, all the columns have a delay of at least +2, and this defines a new reference (so delay +2 becomes delay 0). Thus after this obstacle we have the following:

- for each group whose assignment evaluates l_{p_1} to false, the first column has a delay of at least +1,
- all the other columns have delay 0.

We managed to do what we aimed for: only the first columns of groups whose assignment evaluates l_{p_1} to false have been delayed. Using again an obstacle similar to the one at the bottom of Fig. D.4, we increase this delay to at least +4. We sum up all this literal filtering process with the symbol shown on Fig. D.9.

Once the filtering of l_{p_1} on the set of first columns is done, we use similar constructions to do the filtering of l_{q_1} on the set of second columns, and then l_{r_1} on the set of third columns. After this, in any group, there exists a column of delay 0 if and only if

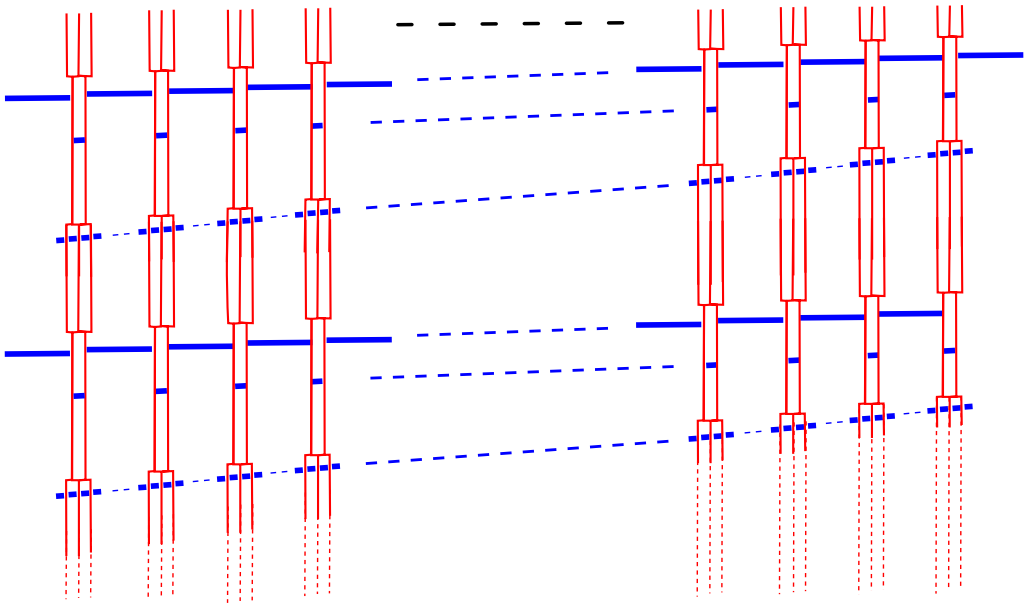


Fig. D.11: Levelling the delays of all groups of columns.

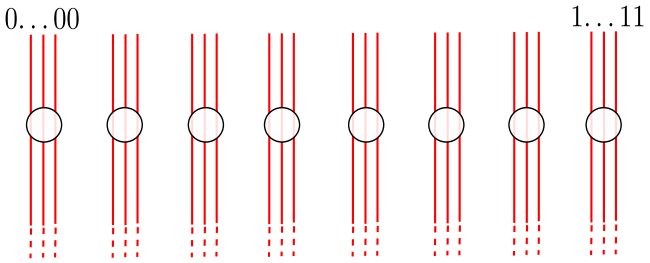


Fig. D.12: The symbol for the "delay levelling" operation.

the clause $l_{p_1} \vee l_{q_1} \vee l_{r_1}$ is verified by the group assignment. What we do next is that we use some merging and splitting so that in each group, the lowest delay becomes the delay of all three columns. We call this "levelling the delays", and it is done by repeating twice the construction shown on Fig.D.10.

Fig. D.11 illustrates this operation on a wider scale, and to simplify, we will just represent it with circles, as shown on Fig. D.12. What follows is that if we repeat the literal filtering and this process for all the clauses, in the end there exists a column of delay 0 if and only if the 3-SAT formula $\bigwedge_{i=1}^{K}(l_{p_i} \vee l_{q_i} \vee l_{r_i})$ is satisfiable. The whole construction is summed up on Fig. D.13, and this concludes the demonstration, because if we denote by ref the reference number of jumps that would be necessary to go from S to G with delay 0 (and ref can be effectively calculated), then the 3-SAT formula is satisfiable if and only if the solution to our shortest path problem is equal to ref (in number of jumps). Besides, it is easy to verify that firstly, the total number of vertices is polynomial in k and K, and secondly, the space needed for the whole construction is exponential in k and K and thus the encoding of all the input points (obstacle vertices, S and G) is polynomial in k and K. As a consequence, our 2D discrete shortest path problem is NP-hard. □

The NP-hardness of this problem raises a few natural questions. First, we could

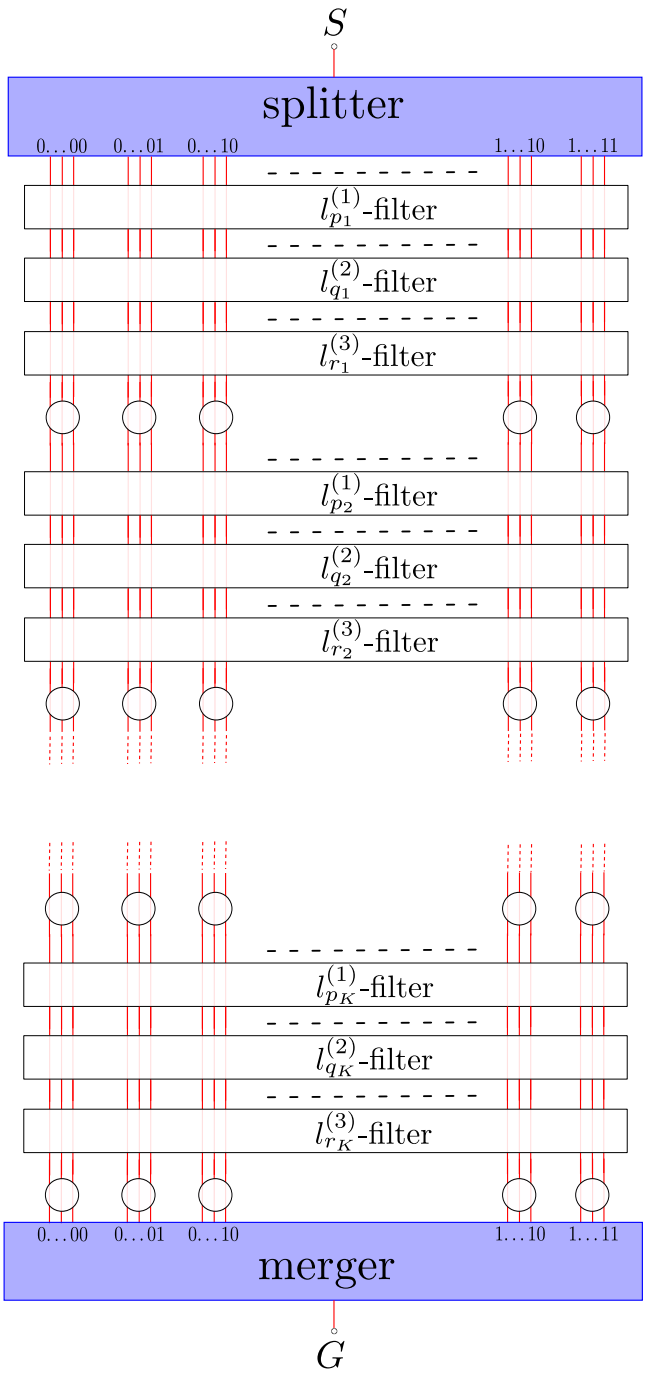


Fig. D.13: The whole construction that reduces 3-SAT to our 2D discrete shortest path problem.

wonder whether its decision version (i.e. we add a number b to the input, and the question becomes: can we go from S to G with less than b jumps?) is NP-complete. Of course, it is not the case because certificates (paths) can be exponentially large. However, if we denote the jumps by $\downarrow_1$, $\downarrow_2$, $\downarrow_3$, $\downarrow_4$, $\uparrow_1$, $\uparrow_2$, ..., $\rightarrow_4$ (the subscript is the length of the jump), then the sequence $\downarrow_4\leftarrow_1\downarrow_4\leftarrow_1\downarrow_4\leftarrow_1\downarrow_4\leftarrow_1\downarrow_4\downarrow_4\rightarrow_2$ could also be written $(\downarrow_4\leftarrow_1)^5(\downarrow_4)^2\rightarrow_2$. With this notation, some exponentially large sequences of jumps can be described with a polynomial encoding. Before trying to obtain some NP-completeness result, an interesting question would be: is there a polynomial P such that, if n is the total number of vertices of the environment, one can always find a shortest path (if one exists) described in the above fashion with at most $P(n)$ bits?

We let this question open, as well as the following question: if we restrict the jumping capabilities even further, allowing only unit jumps, is the discrete shortest path problem still NP-hard?

Appendix E

Bounds for α_1 and α_2: proof of inequalities (4.46) and (4.47)

Fig. E.1 illustrates the first phase of the leg motion, where the correct *hip joint - ankle joint* axis is set. It shows several equalities and inequalities. First, α being the angle between $\vec{L}$ and $\vec{L} + \vec{\Delta L}$ ($\alpha \in [0, \pi]$), κ, the displacement undergone by the ankle after the two first rotations, is such that $\kappa = 2l \sin(\lambda/2) \cdot 2\sin(\alpha/2)$. Then, since $2l \sin(\lambda/2) \cdot \sin(\alpha)$ is the distance between the initial position of the ankle joint and the axis defined by $\vec{L}+\vec{\Delta L}$, we have: $2l \sin(\lambda/2) \cdot \sin(\alpha) \leq \|\vec{\Delta L}\|$. Furthermore, if γ denotes the angle between the vertical axis ν and $\vec{\Delta L}$, then the distance between the intermediate axis and the final position of the ankle joint is equal to $2l \sin(\lambda/2) \cos\gamma \cdot \sin\alpha_2$ ($\alpha_2 \in [0, \pi]$), and it is necessarily less than than κ, thus:

$$2l \sin(\lambda/2) \cos\gamma \cdot \sin\alpha_2 \leq 2l \sin(\lambda/2) \cdot 2\sin(\alpha/2) \tag{E.1}$$

$$\cos\gamma \cdot \sin\alpha_2 \leq 2\sin(\alpha/2) \tag{E.2}$$

Assuming $\frac{2\sin(\alpha/2)}{\cos\gamma} \leq 1$ and $\alpha_2 \in [0, \frac{\pi}{2}]$ (since we have $\gamma < \pi/3$, these two assumptions hold when $\|\vec{\Delta L}\|$ is relatively small):

$$\alpha_2 \leq \arcsin\left(\frac{2\sin(\alpha/2)}{\cos\gamma}\right) \tag{E.3}$$

$$|2\sin(\alpha_2/2)| \leq 2\sin\left(\frac{1}{2}\arcsin\left(\frac{2\sin(\alpha/2)}{\cos\gamma}\right)\right) \tag{E.4}$$

As it can be seen on the view from the side (Fig. E.1), κ_1, the displacement undergone by the ankle joint through the first rotation of angle α_1 (with $\alpha_1 \in [0, \pi]$), is equal to $2l \sin(\lambda/2) \cdot 2\sin(\alpha_1/2)$, and it can be shown that it is also less than κ (this can be seen on the view from the front: the intermediate position of the ankle joint is, on the blue dashed circle, the closest point to the inital position of the ankle joint), so we have:

$$2l \sin(\lambda/2) \cdot 2\sin(\alpha_1/2) \leq 2l \sin(\lambda/2) \cdot 2\sin(\alpha/2) \tag{E.5}$$

$$|2\sin(\alpha_1/2)| \leq 2\sin(\alpha/2) \tag{E.6}$$

From the inequality $2l \sin(\lambda/2) \cdot \sin(\alpha) \leq \|\vec{\Delta L}\|$, and assuming $\alpha \in [0, \frac{\pi}{2}]$, we deduce:

$$\alpha \leq \arcsin\left(\frac{\|\vec{\Delta L}\|}{2l \sin(\frac{\lambda_{min}}{2})}\right) = \alpha_{max} \tag{E.7}$$

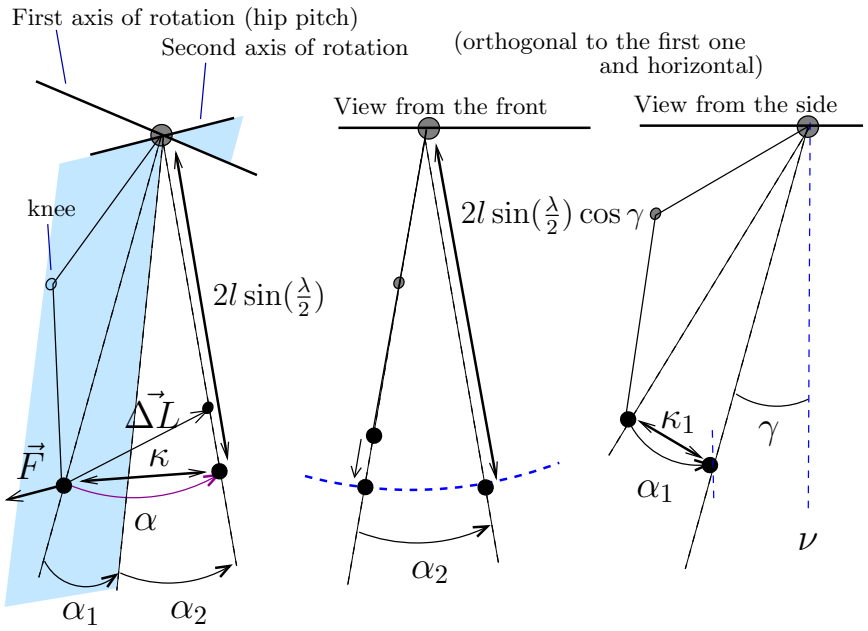


Fig. E.1: The two first rotations, to obtain the correct *hip joint - ankle joint* axis. Here "front" means that we watch the leg so that the hip, knee and ankle are aligned; the "view from the side" is orthogonal to the view from the front.

And from the equations (E.4) and (E.6) it follows (assuming $\frac{2\sin(\alpha_{max}/2)}{\cos\gamma_{max}} \leq 1$):

$$|2\sin(\alpha_2/2)| \leq 2\sin\left(\frac{1}{2}\arcsin\left(\frac{2\sin(\alpha_{max}/2)}{\cos\gamma_{max}}\right)\right) \tag{E.8}$$

$$|2\sin(\alpha_1/2)| \leq 2\sin(\alpha_{max}/2) \tag{E.9}$$

Appendix F

Bound for β: proof of inequality (4.48)

During the second phase of the motion of the leg, pitch rotations are used to put the ankle joint at the correct position (see Fig. F.1). Typically, this sliding motion is the one that produces the largest displacements, by potentially moving the knee much more than the ankle. The positive or negative angle β (variation of the hip pitch) is such that:

$$2l \sin\left(\frac{\lambda}{2} + \beta\right) = 2l \sin(\frac{\lambda}{2}) + \delta \tag{F.1}$$

$\frac{\lambda}{2} + \beta$ stays in the range $[\frac{\lambda_{min}}{2}, \frac{\lambda_{max}}{2}] \subset (0, \frac{\pi}{2})$, and therefore we have:

$$\frac{\lambda}{2} + \beta = \arcsin\left(\sin\left(\frac{\lambda}{2}\right) + \frac{\delta}{2l}\right) \tag{F.2}$$

It follows:

$$\beta = \int_{\sin(\lambda/2)}^{\sin(\lambda/2)+\delta/(2l)} \frac{dt}{\sqrt{1-t^2}} \tag{F.3}$$

We know: $\sin(\frac{\lambda}{2}) + \frac{\delta}{2l} \in [\frac{\lambda_{min}}{2}, \frac{\lambda_{max}}{2}]$, and $|\delta| \leq \|\vec{\Delta L}\|$, so since $x \mapsto \frac{1}{\sqrt{1-t^2}}$ is an increasing function, we can deduce:

$$|\beta| \leq \int_{\sin(\lambda_{max}/2)-\|\vec{\Delta L}\|/(2l)}^{\sin(\lambda_{max}/2)} \frac{dt}{\sqrt{1-t^2}} \tag{F.4}$$

Assuming $\sin\left(\frac{\lambda_{max}}{2}\right) \geq \frac{\|\vec{\Delta L}\|}{2l} - 1$ (true when $\|\vec{\Delta L}\|$ is relatively small), we obtain:

$$|\beta| \leq \underbrace{\frac{\lambda_{max}}{2} - \arcsin\left(\sin\left(\frac{\lambda_{max}}{2}\right) - \frac{\|\vec{\Delta L}\|}{2l}\right)}_{\leq \pi} = \beta_{max} \tag{F.5}$$

It implies: $|2\sin(|\beta|/2)| \leq 2\sin(\beta_{max}/2)$.

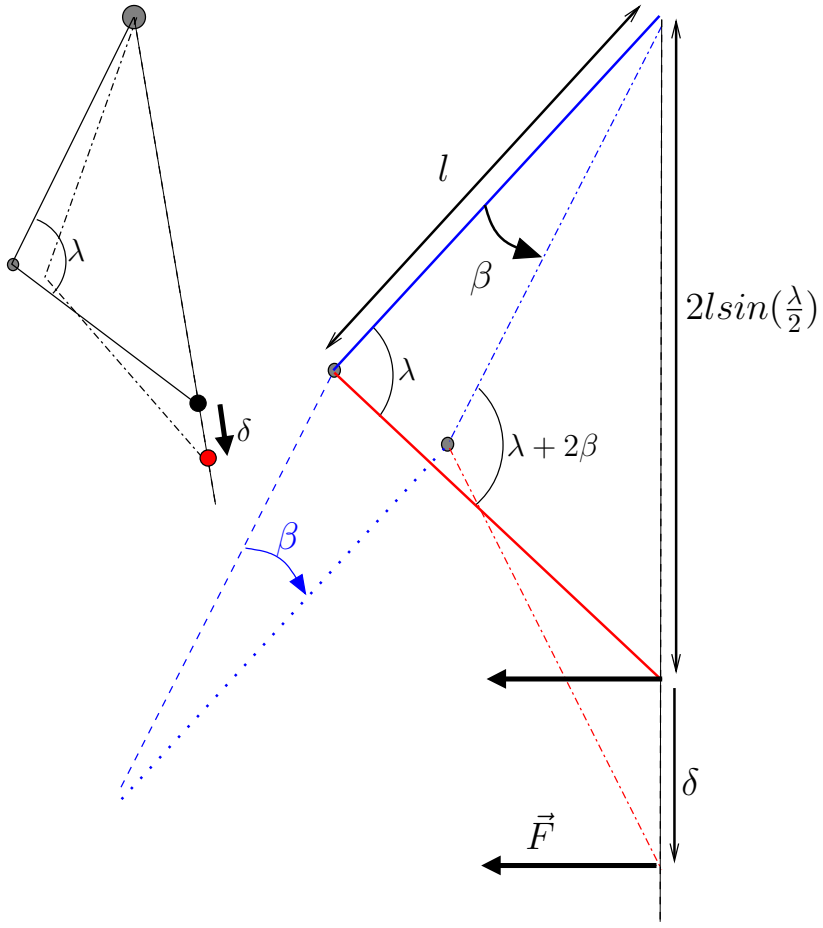


Fig. F.1: Sliding the ankle along the *hip joint* - *ankle joint* axis.

Appendix G

Bound for $\delta\theta$: proof of inequality (4.49)

After the second phase of the motion of the leg, the ankle is at the correct position and $\vec{F}$ is still unchanged, so what remains to be done is the modification of θ. To modify θ, we rotate the leg about the axis defined by $\vec{L}+\Delta\vec{L}$, and use the ankle pitch to keep the foot horizontal. Applying a rotation of angle $\delta\theta$ about this axis will not necessarily modify the foot yaw by $\delta\theta$. As shown on Fig. G.1, let us consider the horizontal frame $(\vec{n_1}, \vec{n_2})$ attached to the ankle joint, and such that $\vec{n_1}$ has the same yaw as $\vec{L}+\Delta\vec{L}$. We also call $\mathcal{P}$ the plane containing the ankle joint and orthogonal to $\vec{L}+\Delta\vec{L}$. During a rotation of angle $\delta\theta$ about the axis defined by $\vec{L}+\Delta\vec{L}$, the transformation applied to the foot can be decomposed into: 1) an orthogonal projection of $\vec{F}$ on the plane $\mathcal{P}$, then 2) a rotation of angle $\delta\theta$ about this axis, and finally 3) a projection along the direction defined by $\vec{L}+\Delta\vec{L}$ onto the horizontal frame $(\vec{n_1}, \vec{n_2})$, followed by a resizing. If we denote by $\vec{n_1'}$ the orthogonal projection of $\vec{n_1}$ on $\mathcal{P}$, and if $\vec{F} = x\cdot\vec{n_1} + y\cdot\vec{n_2}$, then its projection on $\mathcal{P}$ is $x\cdot\cos\gamma\cdot\vec{n_1'} + y\cdot\vec{n_2}$. Then, if we take $(\vec{n_1'}, \vec{n_2})$ as a basis of the plane $\mathcal{P}$, after the rotation of angle $\delta\theta$, the vector obtained has the following coordinates:

$$\begin{pmatrix} \cos\delta\theta & -\sin\delta\theta \\ \sin\delta\theta & \cos\delta\theta \end{pmatrix} \begin{pmatrix} \cos\gamma\cdot x \\ y \end{pmatrix} = \begin{pmatrix} \cos\gamma\cdot\cos\delta\theta & -\sin\delta\theta \\ \cos\gamma\cdot\sin\delta\theta & \cos\delta\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \tag{G.1}$$

And after the projection back onto the plane generated by $(\vec{n_1}, \vec{n_2})$, the vector $\vec{F'} = x'\cdot\vec{n_1} + y'\cdot\vec{n_2}$ will be obtained, with x' and y' such that:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \frac{1}{\cos\gamma} & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\gamma\cdot\cos\delta\theta & -\sin\delta\theta \\ \cos\gamma\cdot\sin\delta\theta & \cos\delta\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \tag{G.2}$$

$$\begin{aligned} \begin{pmatrix} x' \\ y' \end{pmatrix} &= \begin{pmatrix} \cos\delta\theta & \frac{-\sin\delta\theta}{\cos\gamma} \\ \cos\gamma\cdot\sin\delta\theta & \cos\delta\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \\ &= \begin{pmatrix} \cos\delta\theta & -\sin\delta\theta \\ \sin\delta\theta & \cos\delta\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \\ &\qquad \sin\delta\theta\cdot \begin{pmatrix} 0 & 1-\frac{1}{\cos\gamma} \\ \cos\gamma - 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \end{aligned} \tag{G.3}$$

After the resizing, the new value of $\vec{F}$ will be $\frac{\vec{F'}}{\|\vec{F'}\|}$. Let us call $\vec{F_{\delta\theta}}$ the result of the rotation of $\vec{F}$ by angle $\delta\theta$ in the frame $(\vec{n_1}, \vec{n_2})$. From the equation (G.3) we deduce

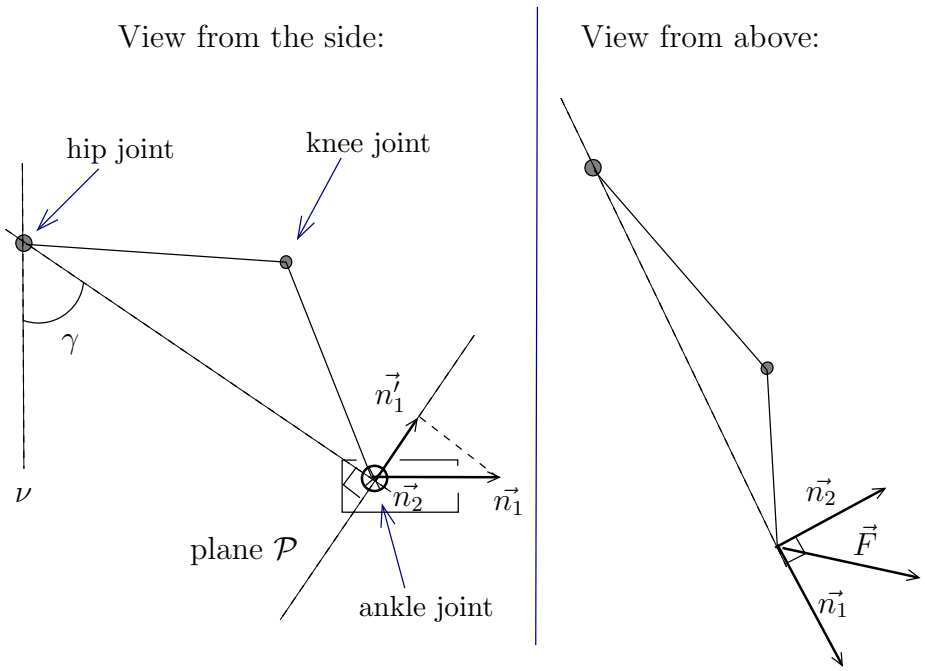


Fig. G.1: The frame $(\vec{n_1}, \vec{n_2})$.

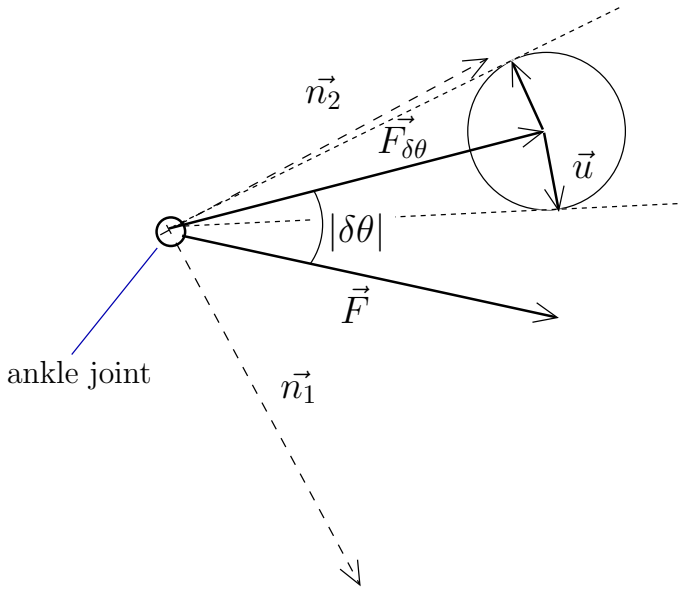


Fig. G.2: Finding an upper bound of the angle between $\vec{F}$ and $\vec{F_{\delta\theta}}$.

that $\vec{F'} = \vec{F_{\delta\theta}} + \vec{u}$, with:

$$\|\vec{u}\| \leq |\sin\delta\theta| \cdot \max\left(|1 - \frac{1}{\cos\gamma}|, |\cos\gamma - 1|\right) \tag{G.4}$$

$$\|\vec{u}\| \leq |\sin\delta\theta| \cdot \max\left(|1 - \frac{1}{\cos\gamma}|, |\cos\gamma| \cdot |1 - \frac{1}{\cos\gamma}|\right) \tag{G.5}$$

$$\|\vec{u}\| \leq |\sin\delta\theta| \cdot |1 - (\cos\gamma)^{-1}| \tag{G.6}$$

Since $\gamma < \frac{\pi}{3}$, $\cos\gamma > \frac{1}{2}$, and thus $0 \leq (\cos\gamma)^{-1} - 1 < 1$, so $\|\vec{u}\| < 1$. As a result (see Fig. G.2) the angle between $\vec{F_{\delta\theta}}$ and $\vec{F'}$ is at most:

$$\arcsin\left(|\sin\delta\theta| \cdot \left((\cos\gamma)^{-1} - 1\right)\right) \tag{G.7}$$

Since arcsin is convex on $[0, 1]$, we can also deduce that this value is bounded by $\left((\cos\gamma)^{-1} - 1\right) \cdot \arcsin(\sin|\delta\theta|)$, and therefore also by:

$$\left((\cos\gamma)^{-1} - 1\right) \cdot |\delta\theta| \tag{G.8}$$

It follows that, in order to obtain a rotation of angle $\Delta\theta$ in the frame $(\vec{n_1}, \vec{n_2})$, the rotation about the axis defined by $\vec{L} + \vec{\Delta L}$ must be of angle $\delta\theta$ such that:

$$|\delta\theta - \Delta\theta| \leq ((\cos\gamma_{max})^{-1} - 1) \cdot |\delta\theta| \tag{G.9}$$

And, as a result:

$$|\delta\theta| \leq \frac{\cos\gamma_{max}}{2\cos\gamma_{max} - 1} \cdot |\Delta\theta| \tag{G.10}$$

Assuming $\frac{\cos\gamma_{max}}{2\cos\gamma_{max}-1} \cdot |\Delta\theta| \leq \pi$ (true for $\Delta\theta$ relatively small):

$$|2\sin(|\delta\theta|/2)| \leq 2\sin\left(\frac{\cos\gamma_{max}}{4\cos\gamma_{max} - 2} \cdot |\Delta\theta|\right) \tag{G.11}$$

Bibliography

Alami, R., Krishna, K. M., and Siméon, T. (2007). Provably safe motion strategies for mobile robots in dynamic domains. In *Autonomous Navigation in Dynamic Environments*, volume 35 of *Springer Tracts in Advanced Robotics*, pages 85–106. Springer Berlin / Heidelberg.

Alami, R., Laumond, J.-P., and Siméon, T. (1994). Two manipulation planning algorithms. *1st Workshop on the Algorithmic Foundations of Robotics (WAFR'94)*.

Asano, T., Kirkpatrick, D., and Yap, C. K. (1996). d_1-optimal motion for a rod. In *12th Symp. on Computational Geometry*, SCG '96, pages 252–263. ACM.

Ayaz, Y., Munawar, K., Malik, M. B., Konno, A., and Uchiyama, M. (2006). Human-like approach to footstep planning among obstacles for humanoid robots. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'06)*.

Bardin, S., Finkel, A., Leroux, J., and Petrucci, L. (2003). Fast: Fast acceleration of symbolic transition systems. In *Int. Conf. on Computer Aided Verification (CAV'03)*, pages 118–121. LNCS 2725, Springer-Verlag.

Barraquand, J. and Latombe, J.-C. (1993). Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles. *Algorithmica*, 10(2):121–155.

Baudouin, L., Perrin, N., Moulard, T., Lamiraux, F., Stasse, O., and Yoshida, E. (2011). Real-time replanning using 3D environment for humanoid robot. In *IEEE/RAS Int. Conf. on Humanoid Robots (Humanoids'11)*.

Beatty, M. F. (1986). *Principles of Engineering Mechanics: Kinematics*. Plenum Press.

Benallegue, M., Escande, A., Miossec, S., and Kheddar, A. (2009). Fast c^1 proximity queries using support mapping of sphere-torus-patches bounding volumes. In *IEEE Int. Conf. on Robotics and Automation (ICRA'09)*, pages 483–488.

Boichut, Y., Héam, P.-C., and Kouchnarenko, O. (2009). How to tackle integer weighted automata positivity. In *3rd Int. Workshop on Reachability Problems (RP'09)*, pages 79–92.

Boissonnat, J.-D., Devillers, O., Donati, L., and Preparata, F. P. (1992). Motion planning for spider robots. In *IEEE Int. Conf. on Robotics and Automation (ICRA'92)*, pages 2321–2326.

Bourgeot, J.-M., Cislo, N., and Espiau, B. (2002). Path-planning and tracking in a 3d complex environment for an anthropomorphic biped robot. In *IEEE Int. Conf. on Intelligent Robots and Systems (IROS'02)*, volume 3, pages 2509–2514.

Byl, K., Shkolnik, A., Prentice, S., Roy, N., and Tedrake, R. (2009). Reliable dynamic motions for a stiff quadruped. In *Experimental Robotics*, volume 54 of *Springer Tracts in Advanced Robotics*, pages 319–328. Springer Berlin / Heidelberg.

Canny, J. and Reif, J. H. (1987). Lower bounds for shortest path and related problems. In *28th IEEE Symp. on Foundations of Computer Science (SFCS'87)*, pages 49–60.

Chestnutt, J. (2007). *Navigation Planning for Legged Robots*. PhD thesis, Carnegie Mellon University.

Chestnutt, J. and Kuffner, J. J. (2004). A tiered planning strategy for biped navigation. In *IEEE/RAS Int. Conf. on Humanoid Robots (Humanoids '04)*, pages 422–436.

Chestnutt, J., Kuffner, J. J., Nishiwaki, K., and Kagami, S. (2003). Planning biped navigation strategies in complex environments. In *IEEE/RAS Int. Conf. on Humanoid Robots (Humanoids'03)*.

Chestnutt, J., Lau, M., Cheung, G., Kuffner, J. J., Hodgins, J., and Kanade, T. (2005). Footstep planning for the honda asimo humanoid. In *IEEE Int. Conf. on Robotics and Automation (ICRA'05)*, pages 631–636.

Chestnutt, J., Michel, P., Nishiwaki, K., Kuffner, J. J., and Kagami, S. (2006). An intelligent joystick for biped control. In *IEEE Int. Conf. on Robotics and Automation (ICRA'06)*, pages 860–865.

Chestnutt, J., Nishiwaki, K., Kuffner, J. J., and Kagami, S. (2007). An adaptive action model for legged navigation planning. In *IEEE/RAS Int. Conf. on Humanoid Robots (Humanoids'07)*, pages 196–202.

Chestnutt, J., Takaoka, Y., Suga, K., Nishiwaki, K., Kuffner, J. J., and Kagami, S. (2009). Biped navigation in rough environments using on-board sensing. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'09)*.

Clarkson, K., Kapoor, S., and Vaidya, P. (1987). Rectilinear shortest paths through polygonal obstacles in o(n(logn)2) time. In *3rd Symp. on Computational Geometry (SCG'87)*, pages 251–257.

Comon, H. and Jurski, Y. (1998). Multiple counters automata, safety analysis and presburger arithmetic. In *Int. Conf. on Computer Aided Verification (CAV'98)*, pages 268–279. Springer.

Dalibard, S., El Khoury, A., Lamiraux, F., Taix, M., and Laumond, J.-P. (2011). Small-time controllability of a walking humanoid robot. In *IEEE/RAS Int. Conf. on Humanoid Robots (Humanoids'11)*.

de Berg, M., van Kreveld, M., Overmars, M. H., and Schwarzkopf, O. (2000). *Computational Geometry: Algorithms and Applications*. Springer-Verlag.

El Khoury, A., Taix, M., and Lamiraux, F. (2011). Path Optimization for Humanoid Walk Planning: an Efficient Approach. In *Int. Conf. on Informatics in Control, Automation and Robotics (ICINCO'11)*.

Elmogy, M., Habel, C., and Zhang, J. (2009). Online motion planning for hoap-2 humanoid robot navigation. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'09)*.

Ferré, E. and Laumond, J.-P. (2004). An iterative diffusion algorithm for part disassembly. In *IEEE Int. Conf. on Robotics and Automation (ICRA'04)*.

Finkel, A. and Sangnier, A. (2010). Mixing coverability and reachability to analyze vass with one zero-test. In *36th Conf. on Current Trends in Theory and Practice of Computer Science (SOFSEM'10)*, pages 394–406.

Fischer, M. J. and Rabin, M. O. (1974). Super-exponential complexity of presburger arithmetic. Technical report, Massachusetts Institute of Technology.

Frisken, S. F., Perry, R. N., Rockwood, A. P., and Jones, T. R. (2000). Adaptively sampled distance fields: a general representation of shape for computer graphics. In *Int. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH'00)*, pages 249–254.

Garey, M. R. and Johnson, D. S. (1990). *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co.

Geraerts, R. and Overmars, M. H. (2002). A comparative study of probabilistic roadmap planners. In *5th Workshop on the Algorithmic Foundations of Robotics (WAFR'02)*.

Gonzalez, J. P. and Likhachev, M. (2011). Search-based planning with provable suboptimality bounds for continuous state spaces. In *4th Symp. on Combinatorial Search (SOCS'11)*.

Goswami, A. (1999). Postural stability of biped robots and the foot-rotation indicator (FRI) point. *Int. Journal of Robotics Research*, 18:523–533.

Greibach, S. A. (1978). Remarks on blind and partially blind one-way multicounter machines. *Theor. Comput. Sci.*, 7(3):311–324.

Gutmann, J.-S., Fukuchi, M., and Fujita, M. (2005). Real-time path planning for humanoid robot navigation. In *Int. Joint Conf. on Artificial Intelligence (IJCAI05)*, pages 1232–1237.

Hachisuka, T., Jarosz, W., Weistroffer, R., Dale, K., Humphreys, G., Zwicker, M., and wann Jensen, H. (2008). Multidimensional adaptive sampling and reconstruction for ray tracing. In *Int. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH'08)*, pages 1–10, New York, NY, USA. ACM.

Halava, V. and Harju, T. (1998). Undecidability in integer weighted finite automata. *Fundamenta Informaticae*, 38(1-2):189–200.

Halava, V. and Harju, T. (1999). Languages accepted by integer weighted finite automata. In *Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa*, pages 123–134. Springer-Verlag.

Harada, K. (2010). Motion planning for a humanoid robot based on a biped walking pattern generator. In *Motion Planning for Humanoid Robots*, pages 192–197. Springer.

Harada, K., Kajita, S., Kaneko, K., and Hirukawa, H. (2006). An analytical method for real-time gait planning for humanoid robots. *Int. Journal of Humanoid Robotics*, 3(1):1–19.

Hardwick, P. and Stout, Q. F. (1998). Flexible algorithms for creating and analyzing adaptive sampling procedures. In *Developments and Applications in Experimental Design, IMS Lec. Notes – Monograph Series 34*, pages 91–105.

Hasegawa, T., Nakagawa, K., and Murakami, K. (2003). Collision-free path planning of a telerobotic manipulator based on swept volume of teleoperated manipulator. In *5th IEEE Int. Symp. on Assembly and Task Planning*.

Hauser, K. (2008). *Motion Planning for Legged Humanoid Robots*. PhD thesis, Stanford University.

Hayet, J.-B., Esteves, C., Arechavaleta, G., and Yoshida, E. (2009). Motion planning for a vigilant humanoid robot. In *Proceedings of 9th IEEE-RAS International Conf. on Humanoid Robots*, pages 196–201.

Herdt, A., Perrin, N., and Wieber, P.-B. (2010). Walking without thinking about it. In *IEEE Int. Conf. on Intelligent Robots and Systems (IROS'10)*, pages 190–195.

Himmelstein, J. C., Ferre, E., and Laumond, J.-P. (2009). Swept volume approximation of polygon soups. *IEEE Trans. on Automation Science and Engineering*, 7(1):177–183.

Hopcroft, J. E., Motwani, R., and Ullman, J. D. (2006). *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc.

Hopcroft, J. E. and Pansiot, J. (1976). On the reachability problem for 5-dimensional vector addition systems. Technical report, Cornell University.

Jaillet, L. and Siméon, T. (2006). Path deformation roadmaps. In *7th Workshop on the Algorithmic Foundations of Robotics (WAFR'06)*.

Kajita, S., Kanehiro, F., Kaneko, K., Fujiwara, K., Harada, K., and Yokoi, K. (2003). Biped walking pattern generation by using preview control of zero-moment point. In *IEEE Int. Conf. on Robotics and Automation (ICRA'03)*, pages 1620–1626.

Kajita, S., Kanehiro, F., Kaneko, K., Fujiwara, K., Yokoi, K., and Hirukawa, H. (2002). A realtime pattern generator for biped walking. In *IEEE Int. Conf. on Robotics and Automation (ICRA'02)*, pages 31–37, Washington, USA.

Kajita, S. and Tani, K. (1991). Study of dynamic biped locomotion on rugged terrain –derivation and application of the linear inverted pendulum mode–. In *IEEE Int. Conf. on Robotics and Automation (ICRA'91)*, volume 2, pages 1405–1411.

Kalakrishnan, M., Buchli, J., Pastor, P., Mistry, M., and Schaal, S. (2010). Fast, robust quadruped locomotion over challenging terrain. In *IEEE Int. Conf. on Robotics and Automation (ICRA'10)*, pages 2665–2670.

Kallmann, M. and Mataric, M. (2004). Motion planning using dynamic roadmaps. In *IEEE Int. Conf. on Robotics and Automation (ICRA'04)*, New Orleans, Louisiana.

Kambites, M. (2006). Word problems recognisable by deterministic blind monoid automata. *Theor. Comput. Sci.*, 362(1):232–237.

Kanoun, O., Laumond, J.-P., and Yoshida, E. (2011). Planning foot placements for a humanoid robot: A problem of inverse kinematics. *Int. Journal of Robotics Research*, 30(4):476–485.

Kanoun, O., Yoshida, E., and Laumond, J.-P. (2009). An optimization formulation for footsteps planning. In *IEEE/RAS Int. Conf. on Humanoid Robots (Humanoids'09)*.

Karaman, S. and Frazzoli, E. (2010). Incremental sampling-based algorithms for optimal motion planning. In *Robotics Science and Systems VI*.

Karp, R. M. and Miller, R. E. (1969). Parallel program schemata. *Journal of Computer and System Sciences*, 3(2):147–195.

Kavraki, L. E., Svestka, P., Latombe, J.-C., and Overmars, M. H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. on Robotics and Automation*, 12:566–580.

Kim, Y. J., Varadhan, G., Lin, M. C., and Manocha, D. (2003). Fast swept volume approximation of complex polyhedral models. In *8th ACM Symp. on Solid Modeling and Applications*, pages 11–22.

Kuffner, J. J. and Lavalle, S. (2000). RRT-Connect: An efficient approach to single-query path planning. In *IEEE Int. Conf. on Robotics and Automation (ICRA'00)*, pages 995–1001.

Kuffner, J. J., Nishiwaki, K., Kagami, S., Inaba, M., and Inoue, H. (2001). Footstep planning among obstacles for biped robots. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'01)*, pages 500–505.

Kuffner, J. J., Nishiwaki, K., Kagami, S., Kuniyoshi, Y., Inaba, M., and Inoue, H. (2002). Self-collision detection and prevention for humanoid robots. In *IEEE Int. Conf. on Robotics and Automation (ICRA'02)*, pages 2265–2270, Washington, USA.

Kuffner, J. J., Nishiwaki, K., Kagami, S., Kuniyoshi, Y., Inaba, M., and Inoue, H. (2003). Online footstep planning for humanoid robots. In *IEEE Int. Conf. on Robotics and Automation (ICRA'03)*.

Lamiraux, F., Bonnafous, D., and Lefebvre, O. (2004). Reactive path deformation for nonholonomic mobile robots. *IEEE Trans. on Robotics*, 20(6):967–977.

Laroussinie, F., Markey, N., and Schnoebelen, P. (2004). Model checking timed automata with one or two clocks. In *15th Int. Conf. on Concurrency Theory (CONCUR'04)*, pages 387–401.

Larsen, E., Gottschalk, S., Lin, M. C., and Manocha, D. (2000). Fast proximity queries with swept sphere volumes. In *IEEE Int. Conf. on Robotics and Automation (ICRA'00)*, pages 3719–3726.

Laumond, J.-P. (1986). Feasible trajectories for mobile robots with kinematic and environment constraints. In *Int. Conf. on Intelligent Autonomous Systems*, pages 346–354.

Lauterbach, C., Mo, Q., and Manocha, D. (2010). gProximity: Hierarchical GPU-based operations for collision and distance queries. *Comput. Graph. Forum*, pages 419–428.

LaValle, S. M. and Kuffner, J. J. (2000). Rapidly-exploring random trees: Progress and prospects. In *4th Workshop on the Algorithmic Foundations of Robotics (WAFR'00)*, pages 293–308.

Leroux, J. (2009). The general vector addition system reachability problem by presburger inductive invariants. In *IEEE Symp. on Logic in Computer Science (LICS'09)*, pages 4–13.

Leven, P. and Hutchinson, S. (2000). Toward real-time path planning in changing environments. *4th Workshop on the Algorithmic Foundations of Robotics (WAFR'00)*.

Mansard, N. and Chaumette, F. (2007). Task Sequencing for Sensor-Based Control. *IEEE Trans. on Robotics*, 23(1):60–72.

Mansard, N., Stasse, O., Evrard, P., and Kheddar, A. (2009). A versatile generalized inverted kinematics implementation for collaborative working humanoid robots: The stack of tasks. In *Int. Conf. on Advanced Robotics (ICAR'09).*, pages 1–6.

Mirtich, B. (1998). V-clip: fast and robust polyhedral collision detection. *ACM Trans. on Graphics (TOG)*, 17(3):177–208.

Mitrana, V. and Stiebe, R. (1997). The accepting power of finite automata over groups. In *New trends in formal languages*, pages 39–48. Springer.

Mitrana, V. and Stiebe, R. (2001). Extended finite automata over groups. *Discrete Appl. Math.*, 108(3):287–300.

Moenning, C. and Dodgson, N. A. (2003). Fast marching farthest point sampling for point clouds and implicit surfaces. Technical report.

Morisawa, M., Harada, K., Kajita, S., Nakaoka, S., Fujiwara, K., Kanehiro, F., Kaneko, K., and Hirukawa, H. (2007). Experimentation of humanoid walking allowing immediate modification of foot place based on analytical solution. In *IEEE Int. Conf. on Robotics and Automation (ICRA'07)*, pages 3989–3994.

Nakamura, Y. and Hanafusa, H. (1987). Optimal redundancy control of robot manipulators. *Int. Journal of Robotics Research*, 6(1):32–42.

Nakhaei, A. and Lamiraux, F. (2008). Motion planning for humanoid robots in environments modeled by vision. In *IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids'08)*, Daejeon, Korea.

Nishiwaki, K. and Kagami, S. (2010). Strategies for adjusting the zmp reference trajectory for maintaining balance in humanoid walking. In *IEEE Int. Conf. on Robotics and Automation (ICRA'10)*, pages 4230–4236.

Nishiwaki, K., Nagasaka, K., Inaba, M., and Inoue, H. (1999). Generation of reactive stepping motion for a humanoid by dynamically stable mixture of pre-designed motions. In *1999 IEEE International Conference on Systems, Man, and Cybernetics*, pages 902 – 907 vol.1.

Nishiwaki, K., Sugihara, T., Kagami, S., Inaba, M., and Inoue, H. (2001). Online mixture and connection of basic motions for humanoid walking control by footprint specification. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 4, pages 4110 – 4115 vol.4.

Nykänen, M. and Ukkonen, E. (1999). Finding paths with the right cost. In *16th Conf. on Theoretical Aspects of Computer Science (STACS'99)*, pages 345–355.

O'Kane, J. and LaValle, S. (2004). Sampling-based methods for discrete planning. In *Doctoral Consortium of the Int. Conf. on Automated Planning and Scheduling*.

OMPL (2010). The Open Motion Planning Library. `http://ompl.kavrakilab.org`.

Pan, J., Lauterbach, C., and Manocha, D. (2010). g-Planner: Real-time motion planning and global navigation using GPUs. In *24th AAAI Conf. on Artificial Intelligence*.

Papadimitriou, C. H. (1985). An algorithm for shortest-path motion in three dimensions. *Information Processing Letters*, 20(5):259–263.

Parikh, R. J. (1966). On context-free languages. *Journal of the ACM*, 13(4):570–581.

Perrin, N., Stasse, O., Baudouin, L., Lamiraux, F., and Yoshida, E. (2011a). Fast humanoid robot collision-free footstep planning using swept volume approximations. *IEEE Trans. on Robotics*, 27(5).

Perrin, N., Stasse, O., Lamiraux, F., Evrard, P., and Kheddar, A. (2009). On the problem of online footsteps correction for humanoid robots. In *27th Conf. of the Robotics Society of Japan*.

Perrin, N., Stasse, O., Lamiraux, F., and Yoshida, E. (2010a). Adaptive sampling-based approximation of the sign of multivariate real-valued functions. Technical report. Available at `http://hal.archives-ouvertes.fr/docs/00/54/48/91/PDF/approx.pdf`.

Perrin, N., Stasse, O., Lamiraux, F., and Yoshida, E. (2010b). Approximation of feasibility tests for reactive walk on HRP-2. In *IEEE Int. Conf. on Robotics and Automation (ICRA'10)*, pages 4243–4248.

Perrin, N., Stasse, O., Lamiraux, F., and Yoshida, E. (2011b). A biped walking pattern generator based on "half-steps" for dimensionality reduction. In *IEEE Int. Conf. on Robotics and Automation (ICRA'11)*, pages 1270–1275.

Perrin, N., Stasse, O., Lamiraux, F., and Yoshida, E. (2011c). Weakly collision-free paths for continuous humanoid footstep planning. In *IEEE Int. Conf. on Intelligent Robots and Systems (IROS'11)*, pages 4408–4413.

Pignon, P., Laumond, J.-P., and Hasegawa, T. (1993). Structuration de l'espace pour la planification hiérarchique des trajectoires d'un robot mobile. *Revue d'Intelligence Artificielle*, 7(4):431–449.

Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T. (1992). *Numerical Recipes in C: The Art of Scientific Computing*.

Reif, J. H. and Wang, H. (1998). The complexity of the two dimensional curvature-constrained shortest-path problem. In *3rd Int. Workshop on the Algorithmic Foundations of Robotics (WAFR'98)*, pages 49–57, Natick, MA, USA.

Reinhardt, K. (2008). Reachability in petri nets with inhibitor arcs. *Electronic Notes in Theor. Comput. Sci.*, 223:239–264.

Render, E. and Kambites, M. (2009). Rational subsets of polycyclic monoids and valence automata. *Information and Computation*, 207(11):1329–1339.

Rybina, T. and Voronkov, A. (2002). Brain: Backward reachability analysis with integers. In *Algebraic Methodology and Software Technology*, pages 489–494. Springer.

Sanchez, G. and Latombe, J.-C. (2001). A single-query bi-directional probabilistic roadmap planner with lazy collision checking. In *Int. Symp. on Robotics Research*, pages 403–417.

Schittkowski, K. (2005). Ql: A fortran code for convex quadratic programming - user's guide, version 2.11. Technical report, University of Bayreuth.

Schmidl, H., Walker, N., and Lin, M. C. (2004). CAB: Fast update of OBB trees for collision detection between articulated bodies. *Journal of Graphics Tools*, 9:1–9.

Schrijver, A. (1986). *Theory of linear and integer programming.* John Wiley & Sons, Inc., New York, NY, USA.

Schürer, R. (2002). Adaptive quasi-monte carlo integration based on miser and vegas. In *5th Int. Conf. on Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing.*

Schwartz, J. T., Sharir, M., and Hopcroft, J. E., editors (1986). *Planning, geometry, and complexity of robot motion.* Ablex Publishing Corp.

Schwarzer, F., Saha, M., and Latombe, J.-C. (2002). Exact collision checking of robot paths. In *5th Workshop on the Algorithmic Foundations of Robotics (WAFR'02).*

Seidl, H., Schwentick, T., Muscholl, A., and Habermehl, P. (2004). Counting in trees for free. In *Automata, Languages and Programming, LNCS 3142.* Springer.

Sellen, J., Choi, J., and Yap, C. K. (1995). Precision-sensitive euclidean shortest path in 3-space. In *11th ACM Symp. on Computational Geometry*, pages 350–359.

Sharir, M. and Schorr, A. (1986). On shortest paths in polyhedral spaces. *SIAM Journal of Computing*, 15(1):193–215.

Siciliano, B. and Khatib, O., editors (2008). *Springer Handbook of Robotics.*

Smola, A. J. and Schölkopf, B. (2004). A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222.

Stasse, O., Evrard, P., Perrin, N., Mansard, N., and Kheddar, A. (2009a). Fast foot prints re-planning and motion generation during walking in physical human-humanoid interaction. In *IEEE/RAS Int. Conf. on Humanoid Robots (Humanoids'09)*, pages 284–289.

Stasse, O., Perrin, N., Wieber, P.-B., Mansard, N., and Lamiraux, F. (2009b). Very fast decision making for whole body motion generation with humanoid robots. In *IEEE RO-MAN Workshop on Human-Synergy*.

Stentz, T. (1994). Optimal and efficient path planning for partially-known environments. In *IEEE Int. Conf. on Robotics and Automation (ICRA'94)*, pages 3310–3317.

Stilman, M., Nishiwaki, K., Kagami, S., and Kuffner, J. J. (2006). Planning and executing navigation among movable obstacles. In *IEEE/RSJ Int. Conf. On Intelligent Robots and Systems (IROS'06)*, pages 820–826.

Storer, J. A., Unilwslty, B., and Reif, J. H. (1994). Shortest paths in the plane with polygonal obstacles. *J. ACM*, 41:982–1012.

Sucan, I. A. and Kavraki, L. E. (2008). Kinodynamic motion planning by interior-exterior cell exploration. In *8th Workshop on the Algorithmic Foundations of Robotics (WAFR'08)*.

Sugiyama, M., Hachiya, H., Towell, C., and Vijayakumar, S. (2007). Value function approximation on nonlinear manifolds for robot motor control. In *IEEE Int. Conf. on Robotics and Automation (ICRA'07*.

Takubo, T., Tanaka, T., Inoue, K., and Arai, T. (2007). Emergent walking stop using 3-d zmp modification criteria map for humanoid robot. In *IEEE Int. Conf. on Robotics and Automation (ICRA'07)*, pages 2676–2681.

Tang, M., Kim, Y. J., and Manocha, D. (2010). CCQ: Efficient local planning using connection collision query. In *9th Workshop on the Algorithmic Foundations of Robotics (WAFR'10)*, pages 229–247.

Tate, S. R. (1991). *Arithmetic Circuit Complexity and Motion Planning*. PhD thesis, Duke University.

Ting, J.-A., D'Souza, A., Vijayakumar, S., and Schaal, S. (2008). A bayesian approach to empirical local linearization for robotics. In *IEEE/RAS Int. Conf. on Robotics and Automation (ICRA'08)*, pages 2860–2865.

Vardi, M. Y. and Wolper, P. (1986). An automata-theoretic approach to automatic program verification (preliminary report). In *IEEE Symp. on Logic in Computer Science (LICS'86)*, pages 332–344.

Verma, K. N. and Goubault-Larrecq, J. (2004). Karp-miller trees for a branching extension of vass. Technical report, Laboratoire Spécification et Vérification.

Vukobratovic, M. and Borovac, B. (2004). Zero-moment point – thirty five years of its life. *Int. Journal of Humanoid Robotics*, 1(1):157–173.

Vukobratovic, M. and Juricic, D. (1969). Contribution to the synthesis of biped gait. *IEEE Trans. on Biomedical Engineering*, BME-16(1):1 –6.

Xia, Z., Chen, G., Xiong, J., Zhao, Q., and Chen, K. (2009). A random sampling-based approach to goal-directed footstep planning for humanoid robots. In *IEEE/ASME Int. Conf. on Advanced Intelligent Mechatronics (AIM'09)*, pages 168–173.

Yoshida, E., Belousov, I., Esteves, C., and Laumond, J.-P. (2005). Humanoid motion planning for dynamic tasks. In *IEEE/RAS Int. Conf. on Humanoid Robots (Humanoids'05)*, pages 1–6.

Yoshida, E., Esteves, C., Belousov, I., Laumond, J.-P., Sakaguchi, T., and Yokoi, K. (2008). Planning 3D collision-free dynamic robotic motion through iterative reshaping. *IEEE Trans. on Robotics*, 24(5):1186–1198.

Zhang, X., Redon, S., Lee, M., and Kim, Y. J. (2007). Continuous collision detection for articulated models using taylor models and temporal culling. In *Int. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH'07)*.