Planning contact points for humanoid robots

Adrien Escande[a], Abderrahmane Kheddar[b], Sylvain Miossec[c]

[a] *CNRS-AIST Joint Robotics Laboratory, UMI3218/CRT, Tsukuba, Japan*
[b] *CNRS-AIST Joint Robotics Laboratory, UMI3218/CRT, Tsukuba, Japan*
CNRS-UM2 LIRMM, Montpellier, France
[c] *PRISME, Université d'Orléans, IUT de Bourges, France*

Abstract

We present a planner for underactuated hyper-redundant robots, such as humanoid robots, for which the movement can only be initiated by taking contacts with the environment. We synthesize our previous work on the subject and go further into details to give an in-depth description of the contact planning problem and the mechanisms of our contact planner. We highlight the structure of the problem with a simple example, present the contact space and the heuristics we use for planning and explain thoroughly the implementation of the different parts of the planner. Finally we give examples of planning results for complex scenarios with the humanoid robot HRP-2.

Keywords:
multi-contact planning, humanoid robot, posture optimization

1. Introduction

Underactuated systems such as humanoid robots can only move by contacts (and sometimes grasps) with the environment. In an obstacle-free, flat, horizontal or lightly-sloped terrains a humanoid robot moves by alternating contacts between the feet and the ground. Each contact creates a constraint on a foot and the reconfiguration of the posture by the actuators induces a movement of the remaining links. For such environments, several approaches proved to be efficient to plan motion for humanoid walking [1][2][3][4] or even running [5][6][7]. The seminal work in [8] illustrates motion generation from contact prints for general biped computer graphics figures. More recently impressive walking behaviors are generated automatically [9][10] even for

highly uneven virtual terrains [11][12][13] (see also [14] in robotics). In cluttered environments, specific motion planners are devised. Situations where the humanoid is asked to cross under or over obstacles are tackled respectively by [15] and [16]. In another work, a dedicated approach deals with situations where crossing under the table or move through a narrow tunnel are asked [17]. In the latter case, crawling using knees and elbows is the most appropriate strategy for humanoids. In other studies [18], planning and control of dynamic transition between walking, swinging with hands, and crawling for Gibbon (monkey-like) robot illustrates the need to use several links for locomotion.

There is however the need for more general planners. One could gather the above different approaches into contact postures and contact transitions primitives, whose combination could be used to plan motion for various situations. But it is difficult to unify these approaches because they use different control strategies and they require tremendous efforts and tricks for tuning. Even if such primitives could be devised, each of them would be valid for predefined links and for a predefined contact spot for each link. Ideally, in cluttered environments, a humanoid robot must be capable of planning motion without restricting the choices of possible contacts that can be made, i.e. considering contacts between any part of its body and any parts of the environment (and even its own body). This is what we are tackling here. Working with contacts is not straightforward: contacts are taken on the surfaces of objects that are also obstacles we do not want the robot to collide with. They also occur for subspaces of the configuration space with zero measure. For these reasons, classical planning algorithms cannot be used readily. Our work is a solution to that problem. It proposes a planning algorithm that achieves general non-gaited multi-contact planning. It combines an adapted best first planning algorithm (BFP), to span a tree of possible contacts and a posture generator that checks the feasibility of a contact configuration. The BFP algorithm is entirely revisited to plan in the space of contacts and the posture generator is formulated as a non-linear optimization problem. To our best knowledge, our planner is to this day the only one that has been successfully applied to humanoid robots (or systems of equivalent complexity) in complex, cluttered environments, such as presented in the two last scenarios of this paper.

Our planner belongs to the class of contact-before-motion planners in that the obtained whole-body trajectory is the consequence of a sequence of contacts (whereas in the motion-before-contact planners, the contacts are

chosen after a trajectory has been planned). Earlier works in this class include the planner of Bretl [19] for climbing robots that has been later adapted to humanoid robots by Hauser [20]. The contacts happen at fixed predefined locations in the environment, sometimes sampled at random on surfaces in the latter. Due to the use of RRT, the output plan is jerky and needs some smoothing in post-processing, that cannot eliminate all unnecessary movements, hence a lack of naturalness. This lack is due both to the use of RRT and the fixed locations of the contacts that might not be suitable. In [21], motion primitives are used to get smoother paths and chose more natural contacts. The solution is however limited by the library of primitives, losing some generalness. By taking another algorithmic approach, our planner does not suffer these drawbacks. The contacts can be taken anywhere on predefined surfaces and both the postures and the choices of contacts are obtained by minimizing a cost function. This enforces the naturalness while keeping the generalness. Moreover our approach let the user have some control on the overall look of the solution, as will be detailed later. The contribution of this paper is twofold: (i) we illustrate the difficulties of contact planning by an example with a simplified robot, (ii) we propose an algorithm for the general case and give a detailed and comprehensive description of all its components along with the justification of the heuristics used and the implementation details, going far beyond the synthesis of our previously published work [22][23][24][25].

The paper is organized as follow: we illustrate the contact planning structure (section 2) then give a detailed description of our contact planner with a presentation of both the concepts (section 4) and the implementation (section 5 for a first basic yet complete presentation of the different parts, section 6 for the posture generation, and section 7 for advanced planning modules). We then exemplify its usage with scenarios involving the humanoid robot HRP-2 [26].

2. Problem overview

In this section, we illustrate the structure of the contact points planning problem. We consider a simplified 2-dimensional tripod robot evolving on a flat floor (at the elevation $y = 0$). It is reduced to a central point and three legs, each of which has a hip and a knee joints (see Fig. 1, left). The feet are considered to be single points. The configuration space of this robot $\mathcal{C}$ can thus be described by the 2D position of the mass, and two angles per

leg (for which we do not consider any joint limits). $\mathcal{C}$ is thus a 8-dimensional manifold: $\mathcal{C} = \mathbb{R}^2 \times \mathbb{S}^6$ (with $\mathbb{S}$ the unit circle). To model an underactuated robot, we consider that this robot can only move when at least one of its feet is in contact with the floor. Furthermore, the robot's movement is limited to lay in $\mathcal{C}_{\text{free}}$ defined as the set of configurations for which all points P of the robot except the feet in contact must have an elevation $y_P > 0$. Finally, we consider non-sliding contacts. This is a purely geometric approach since we do not account for the gravity field so that there is no equilibrium issue.

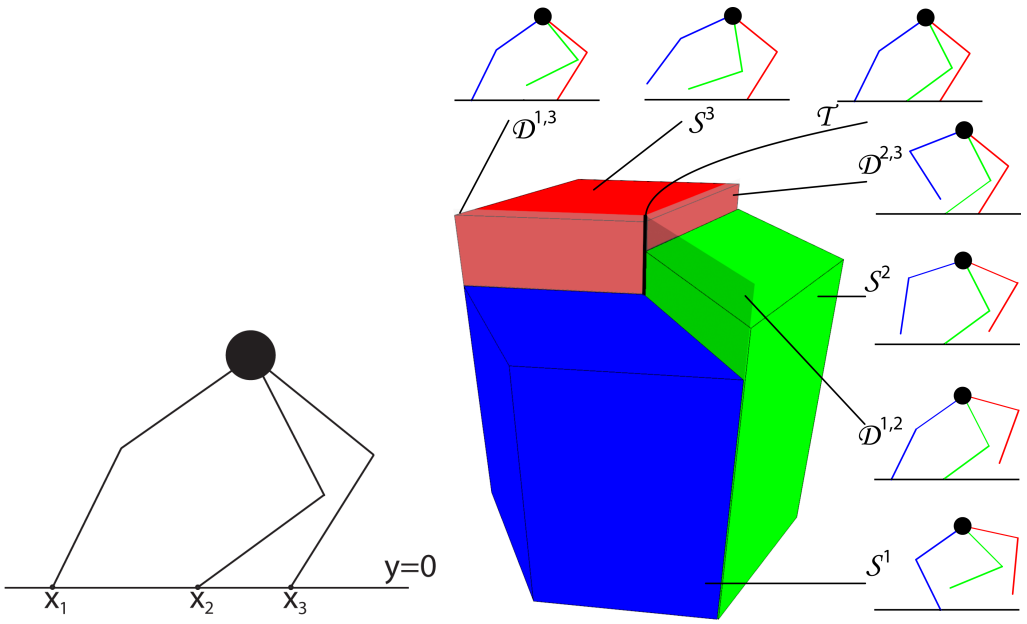

Figure 1: Left: a 2d tripod robot with 2-dofs legs and single point contacts. Right: a representation of the 8-dimensional configuration manifold of the robot, with the 7 submanifolds corresponding to the different contact combinations. Note that each submanifold's representation corresponds to a different projection so as to depict all of them in a 3d drawing which captures the connectivity of these spaces.

2.1. Contact submanifolds

A foot contact is geometrically specified by a 2-dimensional constraint: the position $(x_i, 0)$ of the foot on the floor. When a single foot F_i (i=1, 2 or 3) is in contact at $(x_i, 0)$, the robot evolves on a 6-dimensional submanifold $\mathcal{S}^i_{x_i}$ of $\mathcal{C}_{\text{free}}$ which is the null-space of the constraint. We also consider the 7-dimensional submanifold $\mathcal{S}^i = \bigcup_{x_i \in \mathbb{R}} \mathcal{S}^i_{x_i}$, which is the set of all configurations

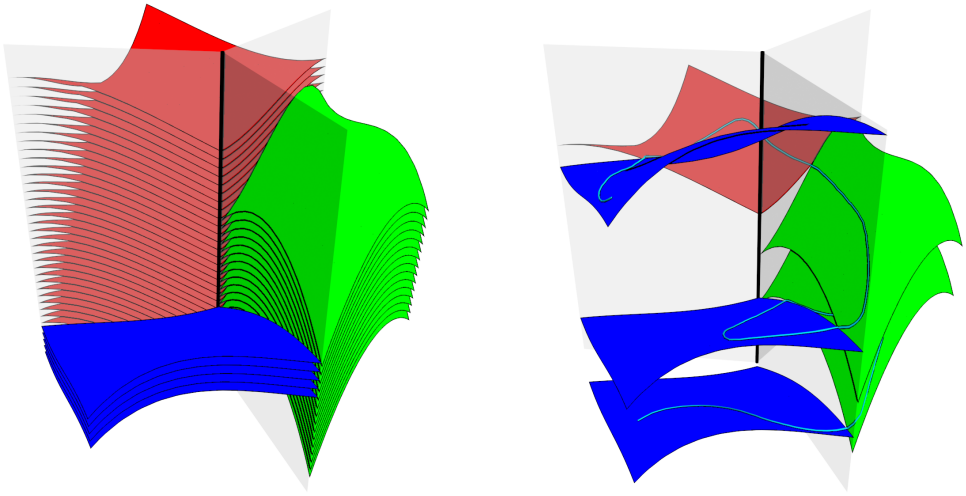

Figure 2: Left: the foliations of the single support submanifolds. Right: an example of path (from bottom leaf to top one) with the following contact sequence: $\{F_1\}$, $\{F_1, F_2\}$, $\{F_2\}$, $\{F_1, F_2\}$, $\{F_1\}$, $\{F_1, F_2\}$, $\{F_2\}$, $\{F_2, F_3\}$, $\{F_3\}$, $\{F_1, F_3\}$, $\{F_1\}$. The movements in double support phases cannot be represented due to the lack of dimensions of the 3d representation.

for which F_i is in contact, regardless of contact position.
Likewise, we define the 4-dimensional submanifolds of $\mathcal{C}$ for two feet F_i and F_j $(i \neq j)$ respectively in contact at x_i and x_j (we drop the always-zero elevation of the contact for conciseness) as $\mathcal{D}^{i,j}_{x_i,x_j}$. These sets are bounded by the constraints on x_i and x_j induced by the robot geometry: the feet cannot be further apart than the legs lengths allow it. Thus $\mathcal{D}^{i,j}_{x_i,x_j} \subset \partial\mathcal{S}^i_{x_i} \cap \partial\mathcal{S}^j_{x_j}$, where ∂X denotes the boundary of X.
From there we obtain the 5-dimensional manifolds $\mathcal{D}^{i,j}_{x_j} = \bigcup_{x_i \in \mathbb{R}} \mathcal{D}^{i,j}_{x_i,x_j} \subset \partial\mathcal{S}^i \cap \partial\mathcal{S}^j_{x_j}$ and $\mathcal{D}^{i,j}_{x_i} = \bigcup_{x_j \in \mathbb{R}} \mathcal{D}^{i,j}_{x_i,x_j} \subset \partial\mathcal{S}^i_{x_i} \cap \partial\mathcal{S}^j$, as well as the 6-dimensional submanifolds $\mathcal{D}^{i,j} = \bigcup_{(x_i,x_j) \in \mathbb{R}^2} \mathcal{D}^{i,j}_{x_i,x_j} \subset \partial\mathcal{S}^i \cap \partial\mathcal{S}^j$. The first one is the set of configurations for which the foot F_j is fixed as a specified position x_j while F_i has the correct elevation, but its position on the floor is unspecified (yet within the geometrical limits). The second one is the symmetric case where F_i is fixed and F_j is not. $\mathcal{D}^{i,j}$ is the set of configurations where both F_i and

F_j are contacting with the floor without being fixed to it.
Lastly, we can define a 2-dimensional submanifold for triple contacts $\mathcal{T}_{x_1,x_2,x_3}$, with F_1, F_2 and F_3 in contact at x_1, x_2 and x_3, and all the 3,4,5-dimensional manifold obtained by relaxing the constraint on the position of one or several feet, among which is $\mathcal{T} = \bigcup_{(x_1,x_2,x_3)\in\mathbb{R}^3} \mathcal{T}_{x_1,x_2,x_3}$. The submanifolds $\mathcal{S}^i$, $\mathcal{D}^{i,j}$ and $\mathcal{T}$ represent the 7 possible contact combinations of the robot in our example, with respectively one, two or all three of the feet F_i in contact (see Fig. 1).

2.2. Movement limitations

For a given i, $\left\{\mathcal{S}^i_{x_i}\right\}_{x_i\in\mathbb{R}}$ is a family of disjoint manifold. Mathematically it forms a *foliation* of $\mathcal{S}^i$, and a submanifold $\mathcal{S}^i_{x_i}$ is called a *leaf*. Such structures have already been exhibited in manipulation planning [27][28]. In term of movements for the robot, two distinct leaves correspond to two distinct placement of F_i, and thus it is not possible to go directly from one leaf to another: that would mean a contact sliding. The movements in $\mathcal{S}^i$ are thus restricted to happen in the leaf the robot is currently in. To move from one leaf to the other, the robot will need to go out of $\mathcal{S}^i$, which means taking (at least) another contact.
Likewise, $\left\{\mathcal{D}^{i,j}_{x_i,x_j}\right\}_{(x_i,x_j)\in\mathbb{R}^2}$ (resp. $\left\{\mathcal{D}^{i,j}_{x_i,x_j}\right\}_{x_i\in\mathbb{R}}$, $\left\{\mathcal{D}^{i,j}_{x_i,x_j}\right\}_{x_j\in\mathbb{R}}$, $\left\{\mathcal{D}^{i,j}_{x_i}\right\}_{x_i\in\mathbb{R}}$ and $\left\{\mathcal{D}^{i,j}_{x_j}\right\}_{x_j\in\mathbb{R}}$) is a foliation of $\mathcal{D}^{i,j}$ (resp. $\mathcal{D}^{i,j}_{x_j}$, $\mathcal{D}^{i,j}_{x_i}$, $\mathcal{D}^{i,j}$ and $\mathcal{D}^{i,j}$) for which it is not possible to go from one leaf to the other. In each case, it means that in order to go to another leaf of the same foliation, the robot must travel out of this foliation, which correspond to adding or removing a contact. Similar structures and conclusion arise in the triple contact case, see Fig. 2.

2.3. The complexity of contact planning

Due to the structure of the configuration space and the restriction on the robot movements, planning a path with contacts is much more complex than the classical piano mover's planning problem where we search a path in a continuous space. The contact planning problem, on the contrary, is an hybrid problem which mix three kind of choices. Firstly, when the robot is in a given contact combinations, there is a discrete choice to be made concerning the next combinations: if it is in $\mathcal{S}^1$ for example, the robot can attempt to add a contact and go either to $\mathcal{D}^{1,2}$, $\mathcal{D}^{1,3}$ or simultaneously two contacts and go to $\mathcal{T}$. From $\mathcal{D}^{1,2}$, it could go to $\mathcal{S}^1$, $\mathcal{S}^2$ or $\mathcal{T}$, by removing

a contact in the two first cases, or adding one in the last case. Secondly, when adding a contact, there is a continuous choice to be made as to where the contact should occur, i.e. what will be the leaf of the next submanifolds. Lastly, within a leaf, we compute the path between the point where the robot enter the leaf and the one planed for its exit.

Several difficulties occur in more complex situations: the presence of obstacles and the need for configuration taking into account the stability of the robot reduce the feasible space, so that it might not always be possible to find a path on a leaf between the entering and exiting configuration. There can also be several contact surfaces, increasing the number of combinations for single, double and triple contact submanifolds, since not only the choice of the feet in contact must be made, but also the pairing of each contact foot with a surface.

To summarize, contact planning features three kinds of choices:

- the discrete choice of the sequence of contact combinations, i.e. a sequence of foliations,
- the continuous choice of the positions of the contacts, i.e. the choice of a leaf within a foliation, and
- the continuous choice of a path on a leaf between two contact combinations.

This triple level of choices is particularly complex because of the huge size of the set of discrete choices and because each of the choices is highly dependent on all the other choices previously made. In particular, the continuous choice of a leaf at a time will impact the further possibilities of discrete choices. The addition of physics considerations such as the equilibrium of the robot adds complexity to the problem and prevents from using straightforwardly the reduction property used in [27].

3. Best First Planning

The contact planner we are presenting in this paper is derived from a potential-field-based planner called Best First Planning (BFP) [29], briefly describe hereafter.

3.1. Algorithm

The main principle of BFP is to grow a tree of configurations based on a discretization of the configuration space $\mathcal{C}$ by expanding at each iteration

Data: $\mathbf{q}^{\text{init}}$, $\mathbf{q}^{\text{end}}$,$U(\mathbf{q})$
Result: a sequence $\left[\mathbf{q}^0, \mathbf{q}^1, \cdots, \mathbf{q}^l\right]$, with $\mathbf{q}^0 = \mathbf{q}^{\text{init}}$, $\mathbf{q}^l = \mathbf{q}^{\text{end}}$ and $\mathbf{q}^{i+1}$ son of $\mathbf{q}^i$ for $0 \leq i < l$

```
1  - L is a sorted list, initially empty
2  - T is a tree, initially empty
3  begin
4      insert(q^init,L)
5      add(q^init,T)
6      while no isEmpty(L) do
7          q ← first(L)
8          for each neighbor q' of q do
9              if U(q') < M and no isAlreadyIn(q',T) then
10                 setFather(q', q)
11                 add(q',T)
12                 insert(q',L)
13                 if q' = q^final then
14                     return backtrackPath(q')
15     return failure
```

Algorithm 1: Best First Planning Algorithm

the best leaf of the actual tree, where *best* is defined by a potential function over $\mathcal{C}$. The root node is the configuration $\mathbf{q}^{\text{init}}$, and the expansion of the tree ends when one of the new leaves is (very close to) the target configuration $\mathbf{q}^{\text{end}}$.

The expansion of a leaf relies on the notion of m-neighbor in a discretized space: if δ is the discretization step, a m-neighbor of $\mathbf{q}$ is obtained by (separately) incrementing or decrementing at most m coordinates of $\mathbf{q}$ by δ. For instance, if $\mathcal{C} = \mathbb{R}^2$ and $\delta = 0.1$, the 2-neighbors of $(1, 0)$ are $(0.9, 0)$, $(0.9, -0.1)$, $(1, -0.1)$, $(1.1, -0.1)$, $(1.1, 0)$, $(1.1, 0.1)$, $(1, 0.1)$ and $(0.9, 0.1)$.

The algorithm is described in Algo. 1. T is a tree whose nodes values are configurations, and L is the list of its leaves, i.e. the nodes that have not been visited for expansion yet. L is sorted in increasing order, according to the potential function U. Two operations are defined on T:

- *add*($\mathbf{q}$,T) adds a node with value $\mathbf{q}$ to T,
- *isAlreadyIn*($\mathbf{q}$,T) checks if T has a node with value $\mathbf{q}$.

For the list structure, three operations are available:

- *insert*($\mathbf{q}$,L) inserts a leaf with value $\mathbf{q}$ in L according to its potential value $U(\mathbf{q})$,
- *first* returns the first element of L, i.e., the leaf with smallest potential $U(\mathbf{q})$, and deletes it from the list,
- *isEmpty*(L) returns *true* when the list L is empty, *false* otherwise.

Finally, *setFather*(l,n) assigns n as the father node of the leaf l, and *backtrackPath*($\mathbf{q}$) goes back from father to father from the node with value $\mathbf{q}$ to the root of T and returns the path from the root to $\mathbf{q}$.
By always expanding the best non-visited node of T, BFP acts as a steepest descent algorithm with predefined possible descent direction (the choice of m for the m-neighbors), and fixed step length, until it reaches a local minimum of U. Then it consider every discretized configurations in the basin of attraction of this minimum by increasing order of potential value, until it finds a way out and resumes its steepest descent-like search. A physical image of this process would be water following a slope, arriving in a basin and filling it up until it goes over the edge and flows away from it.

3.2. Features

It is worth pointing out several features of the plain BFP algorithm, to see how they will translate when tackling the contact planning:

1. a *potential field function* U: it gives a guide for the search. This guide is however not perfectly followed (i.e. the algorithm does not descend following exactly the gradient) because of the way the sons of a node are chosen. In order to have the algorithm return a correct path, U should be an attractive field toward the target configuration, and be repulsive when nearing the obstacles.
2. a *discretization* of the search space, on which relies the notion of neighbors and hence the way sons are generated. The discretization step is a crucial parameter of the algorithm. It must be small enough to ensure passing through corridor of the configuration space, yet big enough so as to not increase the computation time unnecessarily.
3. *free path between neighbors*: it is a strong hypothesis of BFP, that the path between two m-neighbors lies in the free space if both neighbors does. This hypothesis is most likely valid when the discretization is small enough. Otherwise line 9 of the algorithm should include an additional checking function *freePath*($\mathbf{q}$,$\mathbf{q}$').

4. *limitation of the search space*: if U globally increases when going away from the target configuration, the choice of M defines a closed set of the configuration space. This is necessary for termination.
5. *loop avoidance*: the test *isAlreadyIn*($\mathbf{q}$', T) prevents the algorithm to visit twice the same configuration. With 4, this ensures termination.
6. *new leaves*: the way to generate the sons of a node implies the direction the algorithm can take to perform the search. In BFP, the sons are the m-neighbors, but other choices could be made.
7. *ending condition*: the way to define the target was reached. Here it is when a leaf is equal (or very near to) the target configuration.

Some of these features are trivial. They are nonetheless crucial and will get more complex in contact planning.

4. Contact planner: specificities and concepts

4.1. Contacts and planning space

From here, we consider a robot with a free-flying root and n unit joints. The particularity of contact planning is to look for a path in submanifolds of the configuration space corresponding to the surfaces of the $\mathcal{C}$-obstacles: denoting $r_i(\mathbf{q})$ the volume of the workspace occupied by the i-th body r_i of the robot at configuration $\mathbf{q}$, $\mathcal{R}(\mathbf{q})$ the volume occupied by the whole robot, and O_j the volume of the j-th object of the environment, the planning occurs in

$$\mathcal{C}_c = \left\{ \mathbf{q} \in \mathcal{C} | \exists i, \mathcal{R}(\mathbf{q}) \cap O_i \neq \emptyset, \mathcal{R}(\mathbf{q}) \cap \overset{\circ}{O}_i = \emptyset \text{ and } \forall j \neq k, r_j(\mathbf{q}) \cap r_k(\mathbf{q}) = \emptyset \right\}$$

where $\overset{\circ}{X}$ denotes the interior of X.
Not all points of $\mathcal{C}_c$ are interesting however, because some surfaces of the environment or the robot may not be appropriate for contact: a window may not bear the robot's weight, some parts of the robot may be too fragile, etc. To account for this and to simplify the problem, we consider contacts between predefined spots of the robot and predefined surfaces of the environment. We also restrain to fixed contacts (e.g. non-sliding or non-rolling).

A *contact spot* on the robot is defined by a triple $\left(r_i, H_j^i, h_j^i\right)$ where r_i is a body of the robot, H_j^i is an element of SE(3) expressing the position of a contact frame with respect to the body's frame, such that the origin of this

frame is a point of the body's surface, and the z-axis of this frame is an inner normal to this surface. h^i_j is a set of points forming the convex hull of the contact surface. Note that for a body r_i we can define several contact spots. The set of all contact spots is noted $\mathcal{S}_{\text{contacts}}$.

For the environment, we select smooth surfaces that can support a contact and denote their set $\mathcal{S}_{\text{surfaces}}$. For each surface, it is possible to define a continuous mapping giving the normal vector and a tangent one for each point (x, y). For the sake of simplicity, we will consider only planar surfaces, but our algorithms extends easily to the surfaces of general convex objects by taking correctly into account the contacting area between the body and the surface, and less trivially to non-convex cases by carefully considering the collisions.

A robot-environment contact c is fully determined by the choice of a couple of $\mathcal{S}_{\text{contacts}} \times \mathcal{S}_{\text{surfaces}}$ and the triple (x, y, θ), where (x, y) denotes a point of the environment surface, and θ is the angle around the normal vector at (x, y) made by the tangent vector at (x, y) and the x-axis of the contact frame of the robot. The choice of the couple being discrete, the set of all contacts c is a subset of the 4-dimensional space $\mathbb{N} \times \mathbb{R}^3$. For a given k, we consider the k-tuples of contacts such that no body appears more than once, and we denote $\mathcal{S}_k$ the corresponding set. k must be at least one, because the robot needs at least one contact to be stable, and cannot be more than $n+1$ since a robot with n unidimensional joints and a free flying root has $n+1$ bodies. Consequently, we define the set

$$\mathcal{S} = \bigcup_{k=1}^{n+1} \mathcal{S}_k \tag{1}$$

This set of sets of contacts appears to be the natural planning space for our problem: moving a robot from one place to another amounts to searching for a sequence of sets of contacts achievable by the robot. In this space, a m-neighbors of an element is another set of contacts obtained by at most m contact changes (addition or removal, but not displacement: going from (c_1, c_2) to (c_1, c'_2), where c_2 and c'_2 involve the same body, is done in two steps by first removing c_2 then adding c'_2).

4.2. Feasible sequence of contacts sets

We define here what is an *achievable sequence* for the robot.
$\mathcal{S}$ is more meaningful for our purpose than $\mathcal{C}$ and is smaller since effectively

the number of contacts is less than 4 or 5. The size of $\mathcal{S}$ is thus less than 20 while the size of $\mathcal{C}$ ranges usually from 30 to more than 100. Yet, we cannot forget completely about $\mathcal{C}$ because we need the configuration of the robot to test some physical and geometrical properties. We say that *a set of contacts* $s \in \mathcal{S}$ *is feasible* if there exists (at least) one posture for the robot, such that the contacts are effective and the geometrical constraints (joint limits, collision avoidance...) and physical constraints (equilibrium, possible torque limits...) are satisfied. Such a posture is a *witness posture*. We denote $\mathcal{Q}_s$ the set of all witness postures for a contacts set s. Whether it is empty or not can be decided by a feasibility problem (FP), that we describe in section 6.

A sequence of feasible contacts sets $(s_1, s_2, \cdots, s_l)$ is *feasible* itself, if there is a continuous path satisfying successively each s_i:

$$\begin{aligned} \exists \varphi : [1,l] \to \mathcal{C}, \quad & (1) \quad \varphi \text{ is } C^0, \\ & (2) \quad \forall i \in [|1, l-1|]\,,\ \forall u \in [i, i+1\,[\ , \varphi(u) \in \mathcal{Q}_{s_i} \\ & (3) \quad \varphi(l) \in \mathcal{Q}_{s_l} \end{aligned}$$

Finding such a path might be done by using classical planning methods on constrained submanifolds [30][31]. These approaches however are too costly to be used as checking routines within our planner; therefore we use a heuristic we now describe.

For φ to be continuous, we need two conditions: (i) $\mathcal{Q}_{s_i} \cap \mathcal{Q}_{s_{i+1}} \neq \oslash$ and (ii) $Q_{s_{i-1}} \cap Q_{s_i}$ and $Q_{s_i} \cap Q_{s_{i+1}}$ both belong to the same connected component of $\mathcal{Q}_{s_i}$.
The first condition means that one can find a path to change the contacts from s_i to s_{i+1}. We first remark that $\mathcal{Q}_{s_i}$ can be written as the intersection of two subsets of $\mathcal{C}$, one defined by the stability (i.e. equilibrium) constraints derived from s_i that we denote $\mathcal{Q}_s^{\text{stab}}$ and the other defined by all the remaining (and mostly geometric) constraints, $\mathcal{Q}_s^{\text{geom}}$.
(i) implies that $\mathcal{Q}_{s_i}^{\text{geom}} \cap \mathcal{Q}_{s_{i+1}}^{\text{geom}} \neq \oslash$. A sufficient way to ensure this is to plan only with 1-neighbors, i.e. one passes from s_i to s_{i+1} only by adding or removing exactly one contact. Let us assume that, without loss of generality, $s_i \subset s_{i+1}$, which implies $\mathcal{Q}_{s_i}^{\text{geom}} \supset \mathcal{Q}_{s_{i+1}}^{\text{geom}}$ and $\mathcal{Q}_{s_i}^{\text{stab}} \subseteq \mathcal{Q}_{s_{i+1}}^{\text{stab}}$. We then have:

$$\begin{aligned} \mathcal{Q}_{s_i} \cap \mathcal{Q}_{s_{i+1}} \neq \oslash \quad & \Leftrightarrow \quad \mathcal{Q}_{s_i}^{\text{geom}} \cap \mathcal{Q}_{s_i}^{\text{stab}} \cap \mathcal{Q}_{s_{i+1}}^{\text{geom}} \cap \mathcal{Q}_{s_{i+1}}^{\text{stab}} \neq \oslash & (2) \\ & \Leftrightarrow \quad \left(\mathcal{Q}_{s_i}^{\text{geom}} \cap \mathcal{Q}_{s_{i+1}}^{\text{geom}}\right) \cap \left(\mathcal{Q}_{s_i}^{\text{stab}} \cap \mathcal{Q}_{s_{i+1}}^{\text{stab}}\right) \neq \oslash & (3) \\ & \Leftrightarrow \quad \mathcal{Q}_{s_{i+1}}^{\text{geom}} \cap \mathcal{Q}_{s_i}^{\text{stab}} \neq \oslash & (4) \end{aligned}$$

This means that there is a path between two 1-neighbors s_i and s_{i+1} if one can find a posture satisfying the geometric constraints of the biggest (for the inclusion) of the two sets and the stability constraints of the smallest one.

Condition (ii) means we can find a path within $\mathcal{Q}_{s_i}$ to perform the contact changes from s_{i-1} to s_{i+1}. In some specific cases (for example with a thin obstacle) this path may not exist, but these cases being rare, we suppose there is always one.

We thus construct our feasible sequence by adding or removing contacts, one at a time, and finding at each step a witness posture taking into account the changing contact geometrically but not physically (i.e. for the stability). This can be seen as checking the feasibility just before a new contact is made or just after an existing one was released.

5. Contact planner: implementation

This section discusses the implementation of our contact planner by describing its different components. We give a first practical way to implement each of them with the exception of the Posture Generator, in charge of the feasibility checking, which is described on its own in the next section. This first implementation gave good results in not-too-complicated scenarios [22]. Section 7 goes into the details of more sophisticated implementations for some of these components, aiming at tackling more complex environments and achieving better speed.

5.1. Contacts BFP

Algorithm 2 presents the main routine of our planning, called *Contacts Best First Planning*. While very similar to algorithm 1, it is however more complex to implement.

Since we work in $\mathcal{S}$ but need to prove the feasibility of a contact set by a witness posture, the input and output of the algorithm are based on couples from $\mathcal{S} \times \mathcal{C}$. A node can then be seen as a triple $(f, s, \mathbf{u})$ with f the father node, $s \in \mathcal{S}$ and $\mathbf{q} \in \mathcal{C}$. The goal, determining the successful termination of the algorithm is given as a subset $\mathcal{Q}_{\text{goal}} \subset \mathcal{C}$, possibly of lower dimension, described as a set of equalities and inequalities on $\mathbf{q}$. It is reached for the set of contacts s_e if a posture $\mathbf{q}^e$ exists in $\mathcal{Q}_{s_e} \cap \mathcal{Q}_{\text{goal}}$. To guide the planning toward the goal we also provide the algorithm with a potential field U defined over $\mathcal{S} \times \mathcal{C}$ (although one might want to use only $\mathcal{S}$ or $\mathcal{C}$). Unlike the classical approach in potential field planning, U cannot incorporate readily a

Algorithm: Contacts BFP

Data: $(\mathbf{s}_{\text{init}}, \mathbf{q}^{\text{init}})$, $\mathcal{Q}_{\text{goal}}$,$U(\mathbf{s}, \mathbf{q})$

Result: a sequence $[(\mathbf{s}_0, \mathbf{q}^0), (\mathbf{s}_1, \mathbf{q}^1), \cdots, (\mathbf{s}_e, \mathbf{q}^e)]$, with $(\mathbf{s}_0, \mathbf{q}^0) = (\mathbf{s}_{\text{init}}, \mathbf{q}^{\text{init}})$, $\mathbf{q}^e \in \mathcal{Q}_{\text{goal}}$ and $\mathbf{s}_{i+1}$ 1-neighbor of $\mathbf{s}_i$

```
- L is a sorted list, initially empty
- l is a list
- n and n' are nodes
begin
    n ← (⊘, s_init, q^init)
    insert(n,L)
    while no isEmpty(L) do
        n ← first(L)
        if allowsToReachGoal(n) then
            return backtrackPath(n)
        l ← generateSons(n)
        for each node n' in l do
            if U(n') < M and trajectoryExistsTo(n') then
                insert(n',L)
    return failure
```

Algorithm 2: Contacts BFP

repulsive component based on the distance between the robot and the environment since at any time, parts of the robot need to be in contact with the environment. The robot however needs to be guided around (or under/over) the obstacles. We introduce in section 7.2 a way to build a potential field that copes with this obstacle/support duality. For the sake of simplicity, we will write $U(n)$ for $U(s, \mathbf{q})$, where $n = (f, s, \mathbf{q})$. Likewise we write $\mathcal{Q}_n$ for $\mathcal{Q}_s$.

The algorithm still grows a tree by expanding the best leaf according to U, that can be found in the sorted list L. The function *allowsToReachGoal*(n) checks whether $\mathcal{Q}_n \cap \mathcal{Q}_{\text{goal}}$ is empty or not by trying to solve the corresponding feasibility problem, and returns *true* upon success, producing a witness posture $\mathbf{q}^e$. We suppose that the path between $n.\mathbf{q}$ and $\mathbf{q}^e$ exists. *generateSons*(n) generates the sons of n and is detailed in the next section. It includes what is equivalent to the loop avoidance, discretization and leaves generation in the classical BFP. Lastly, we mention the function *trajectoryExistsTo*(n) which returns *true* if a path exists between n and its father.

In our implementation, this function is trivial and always returns *true*, since we follow the heuristic mentioned above.

5.2. Sons generation

Algorithm: generateSons
Data: n, a node
Result: l, the list of leaf expanded from n, initially empty
1 - s, a list of contacts, initially empty
2 - N, the set of the nodes already generated during the whole planning
3 **begin**
4 $(f, s, \mathbf{q}) \leftarrow n$
`// Suppressing a contact`
5 **if** length$(s) \geq 2$ **then**
6 **for** *each* c *in* s **do**
7 $s' \leftarrow s \setminus \{c\}$
8 $n' \leftarrow (n, s', \mathbf{q})$
9 **if** add(n', N) **then**
10 **if** generate(n', c) **then**
11 $l \leftarrow l \cup \{n'\}$
`// Adding a contact`
12 **for** $i \leftarrow 1$ **to** $size(\mathcal{E}_{contacts})$ **do**
13 **if** *no* isAlreadyInContact(s, i) **then**
14 **for** $j \leftarrow 1$ **to** $size(\mathcal{E}_{surfaces})$ **do**
15 $l \leftarrow l \cup generateNewContacts(n,i,j)$
16 $return(l)$

Algorithm 3: Sons generation

For a node $n = (f, s, \mathbf{q})$, the generation of sons is based on the 1-neighbors of s, i.e. the sets of contacts obtained from s by either adding or removing a contact. The function *generateSons* presented in algorithm 3 is therefore divided in two parts. The first part explores the possibility to remove a contact from s. It trivially tries every contact, one by one. A new node candidate n' is create at line 8, using the witness posture of its father for the initialization. The block on lines 9-11, which also appears in algorithm 4, checks the validity of this candidate with respect to two criteria. The first

Algorithm: generateNewContacts

Data: n, a node, i and j, indices of the contact spot and the surface
Result: l, a list of leaves whose father is n, initially empty

```
1  - N is the set of the nodes already generated
2  begin
3      if forbiddenContact(i,j) then
4          return(⊘)
5      (f, s, q) ← n
6      contacts ← getContactCandidates(i,j)
7      for each c in contacts do
8          s' ← s ∪ {c}
9          n' ← (n, s', q)
10         if add(n',N) then
11             if generate(n',c) then
12                 l ← l ∪ {n'}
13     return(l)
```

Algorithm 4: New contacts for a given couple (contact spot, surface)

test is the transposition of the loop avoidance: $add(n, N)$ checks if n appears in N. If not, it adds it to N and returns *true*, otherwise it does nothing and returns *false*. It is implemented by defining a strict order relation $\prec$ on $\mathcal{S}$ (see 5.3). This way N is represented as a red-black binary tree and the existence test/insertion can be performed in $\mathcal{O}(\ln(\text{size}(N)))$.
The second test is the feasibility test. *generate*(n,c) with $n = (f, s, \mathbf{q})$ attempts at finding a witness posture $\mathbf{q}_w$ for the non-emptiness of the set $\mathcal{Q}_{s\cup\{c\}}$ with c not participating to the stability. This is the transcription of the sufficient conditions of (i) in 4.2. Note that in algorithm 4, c is already in s, yet it will not participate to the stability. *generate* is implemented as a feasibility (or optimization) problem as described in section 6. The process is initialized with the actual configuration $\mathbf{q}$ attached to n and replaces it by $\mathbf{q}_w$ upon completion. This is why in both algorithms 3 and 4 we construct the new nodes with the configuration of their fathers: the sets $\mathcal{Q}$ attached to the father and the sons being largely similar, the father's configuration already satisfies many constraints of the sons' feasibility problems.

The second part is more complex because there is usually an infinite number of possible new contacts. This complexity is handled by the function

generateNewContacts presented in algorithm 4 and its sub-function *getContactCandidates*. This second part iterates over the possible new pairs in $\mathcal{S}_{\text{contacts}} \times \mathcal{S}_{\text{surfaces}}$. The sub-function *isAlreadyInContact*(s, i) returns *true* if the body corresponding to the i-th contact spot in $\mathcal{S}_{\text{contacts}}$ is already participating to a contact in s. Remember that the sets of contacts in $\mathcal{S}$ cannot involve twice the same body of the robot.

The working of algorithm 4 is fairly straightforward. Two sub-functions need to be explained: *forbiddenContact*(i,j) relies on a list of invalid pairs of $\mathcal{S}_{\text{contacts}} \times \mathcal{S}_{\text{surfaces}}$ given by the user and returns *true* if (i,j) denotes a pair in this list. The use of this list can be motivated by security issues, the control of the overall aspect of the result or the introduction of some a priori knowledge. For example, one might want to avoid the robot putting its feet on a table. *getContactCandidates*(i,j) returns a list of potential contacts between the contact spot i and the surface j, that have then to be validated by the block we described above. How to build this list is discussed in section 5.4.

5.3. Ordering $\mathcal{S}$

A contact can be described by the tuple (i, x, y, θ) with i the index identifying a couple in $\mathcal{S}_{\text{contacts}} \times \mathcal{S}_{\text{surfaces}}$. For $s \in \mathcal{S}$, we define $ind(s)$=$[i_1, .., i_k]$ the *sorted* list of the indices corresponding to the contacts in s. We also construct $p_s = (x_1, y_1, \theta_1, \cdots, x_k, y_k, \theta_k)$, the vector of these contacts parameters, in the same order as the indices in $ind(s)$.
If s and s' are such that $ind(s)$=$ind(s')$, then we say that s and s' are equal *iff* $\|D\,(p_s - p_{s'})\| \leq k \cdot \epsilon$, with D a diagonal matrix to account for the non-homogeneity between angles and distances, and ϵ a positive constant. We write $p_s \neq p_{s'}$ if the previous inequality is not verified. One must note that this equality relation is not transitive.

Using the lexicographic order on the vectors, we can then define an order relation $\prec$ on $\mathcal{S}$: $s \prec s'$ *iff*

- $\text{card}(s) < \text{card}(s')$
- or $\text{card}(s) = \text{card}(s')$ and $ind(s) < ind(s')$
- or $ind(s)$=$ind(s')$ and $p_s \neq p_{s'}$ and $p_s < p_{s'}$

Through the use of the set N with the equality presented here, we obtain an implicit discretization of $\mathcal{S}$ whose parameters are D and ϵ: when a node n is inserted in N, it defines a region of $\mathcal{S}$ in which we cannot chose a new set of contacts (because the test $add(n', N)$ would fail). Provided the environment

is bounded, we thus have a finite (yet huge) number of contacts sets. How these sets are distributed in $\mathcal{S}$ is determined by the way of generating contact candidates.

5.4. Generating contacts candidates

A pair of $\mathcal{S}_{\text{contacts}} \times \mathcal{S}_{\text{surfaces}}$ being given, generating candidates amounts to choosing triples (x, y, θ).
Ideally, the generation of candidates should verify the three following points:

- (i) ensure a good coverage of the surface, to avoid ignoring possible paths for the robot,
- (ii) give a small number of candidates to avoid combinatorial explosion,
- (iii) give feasible contacts as much as possible. Indeed, the feasibility checking is the most expansive part of the planner, and takes more time when failing to find a witness posture than when it manages to produce one.

Determining a strategy to take into account these three points is a crucial step of our contact planner. Such a strategy must be cheap enough so that it does not cost more than its benefit, especially regarding point (iii). Reducing the contact surface we consider is a necessity and can be roughly done by intersecting the surface with a sphere centered at the root of the limb containing the contact spot and a conservative radius.

Our first implementation of the candidates generation is based on this sphere-surface intersection and a random generation of (x, y, θ) with (x, y) on the intersection. The compromise between (i) and (ii) is done by choosing the number of generated triples. It appears experimentally that taking a number proportional to the square root of the surface's area gives good results: this gives us enough samples for small surfaces yet not too much for larger ones.

5.5. Potential field

Constructing a potential field efficiently guiding the planning is the second crucial point in our work. We discuss here some basic solutions and will present more complex one in section 7.2. The idea is above all to have a guide in the workspace: we do not aim at a precise posture, but usually at a place in space or more precisely, a subset of $\mathcal{C}$. Thus, a first idea is to construct a simple attractive potential $\|x - x^0\|^2$ based on the position (and possibly the orientation) of the robot's root or CoM and a target position. For a given

node, this potential will be evaluated at its witness posture.Another idea is to base this potential on the position of the contacts or their barycenter, and a target barycenter. Being defined only on $\mathcal{S}$, this potential field has the advantage of not relying on the witness posture that is dependent of the way to produce it. These types of potential may be combined. We can also add to them bonus or penalty terms in order to favor or penalize some class of contacts sets, for motivation ranging once again from security to aesthetic. For example, when we wanted to run the output of the planner on a real robot, we greatly penalized the sets with the hands only, to avoid the robot walking on its hands, while permitting it to use them in conjunction with other parts, for instance to pass under a table.

An important drawback of the potential fields presented here is that they do not take obstacle into account. This will be tackled in section 7.2.

6. Posture Generation

This section describes a tool that is extensively used throughout the planning process, since it provides the witness postures attesting for the feasibility of a set of contacts. It relies on the notion of *task*, which denotes in this work a set of equalities and inequalities over $\mathcal{C}$, namely $g(\mathbf{q}) = 0$ and $h(\mathbf{q}) \leq 0$. A contact is a special case of task. The naming *task* was already used for equalities in a planning context [31], and is closely related to the notion of task function in control [32].

6.1. Feasibility problem

Given a set of tasks $\{\mathcal{T}_i\}$, the subset $\mathcal{Q} \in \mathcal{C}$ of the geometrically and physically correct configurations verifying these tasks is written (see Fig. 3):

$$\mathcal{Q} = \begin{cases} q_i^- \leq q_i \leq q_i^+, \quad \forall i \in [|1, n|], & (5) \\ \epsilon_{ij} \leq d(r_i(\mathbf{q}), r_j(\mathbf{q})), \quad \forall (i,j) \in \mathcal{I}_{\text{auto}}, & (6) \\ \epsilon_{ik} \leq d(r_i(\mathbf{q}), O_k), \quad \forall (i,k) \in \mathcal{I}_{\text{coll}}, & (7) \\ s(\mathbf{q}) \leq 0, & (8) \\ g_i(\mathbf{q}) = 0, \quad \forall\ \mathcal{T}_i, & (9) \\ h_i(\mathbf{q}) \leq 0, \quad \forall\ \mathcal{T}_i. & (10) \end{cases}$$

where inequalities (5) describe the joint limits. We consider 1d joints. Bounds for more complex joints could be written in a more general form $b(\mathbf{q}) \leq 0$.

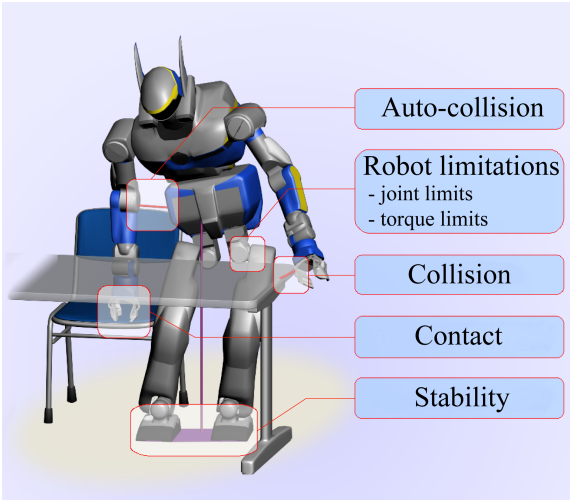

Figure 3: The different constraints defining $\mathcal{Q}$

Equations(6) and (7) express the auto-collision and collision avoidance constraints, $d(O_1, O_2)$ being the (signed) distance between objects A and B. As in section 4, $r_i(\mathbf{q})$ (resp. O_k) denotes the volume of the workspace occupied by the i-th body r_i of the robot at configuration $\mathbf{q}$ (resp. k-th obstacle). ϵ_{ij} and ϵ_{ik} are security distances, thus usually positive. $\mathcal{I}_{\text{auto}}$ is a subset of $[|1, n+1|]^2$: some pairs of bodies can never collide because of joint limits or of other collisions that will always occur before, and it is not always possible to write collision avoidance constraint as a positive distance requirement. Denoting N_O the number of objects in the environment, $\mathcal{I}_{\text{coll}}$ is a subset of $[|1, n+1|] \times [|1, N_O|]$: pairs (r_i, O_k) involved in a contact should not be part of it, for example.

s is a stability criterion written only on the configuration $\mathbf{q}$ and will be described in section 6.4.

One might want to take into account the limitation on the actuators of the robot. This can be done by introducing additional variables, namely the torques $\boldsymbol{\tau}$ and the contact forces $\mathbf{f}$. They are linked to $\mathbf{q}$ by the dynamic equation of a multibody system in the static case:

$$\begin{pmatrix} \mathbf{g}_1(\mathbf{q}) \\ \mathbf{g}_2(\mathbf{q}) \end{pmatrix} = \begin{pmatrix} \boldsymbol{\tau} \\ \mathbf{0} \end{pmatrix} + \begin{pmatrix} J_{c_1}^T(\mathbf{q}) \\ J_{c_2}^T(\mathbf{q}) \end{pmatrix} \mathbf{f} \tag{11}$$

which should then be added as a constraint (J_c being the jacobian matrix of the contact points corresponding to $\mathbf{f}$ and $\mathbf{g}$ the gravity term). One can

then impose bounds on $\boldsymbol{\tau}$: $\tau_i^- \leq \tau_i \leq \tau_i^+, \quad \forall i \in [|1, n|]$ and the stability constraint will be rewritten as $s(\mathbf{f}) \leq 0$ and accounts for the contact forces being inside friction cones. This solution however increases the number of variables, which induces a greater cost to solve the feasibility problem. We do not consider bounds on $\boldsymbol{\tau}$ in this work.

6.2. Positioning constraints

Geometrically imposing the contact between a body r_i of the robot and a surface of the environment is a special case of positioning a body at a given location in the workspace.

We denote H_i^j the element of SE(3) describing the position of a frame i w.r.t a frame j. For a vector v (resp. a point p), we write $\mathbf{v}^i$ (resp. $\mathbf{p}^i$) its coordinates in the frame i and $\mathbf{v}^j$ (resp. $\mathbf{p}^j$) in the frame j. We have the non-linear relation $\mathbf{v}^j = R_i^j \mathbf{v}^i$ (resp. $\mathbf{p}^j = H_i^j \mathbf{p}^i$), where R_i^j is the rotation part of H_i^j. If k is a third frame, we have $H_i^k = H_j^k H_i^j$ (see [33], chap. 2). We denote the world frame by 0.

Positioning the body r_i in the workspace is done by superimposing a frame attached to this body (we called it *contact frame* in section 4.1) with a frame in the environment. Let i denote the local frame of r_i, j the attached frame and e the environment frame. The superimposition constraint can be written $H_j^0(\mathbf{q}) \left(= H_i^0(\mathbf{q}) H_j^i\right) = H_e^0$. It is conveniently expressed in $\mathbb{R}^6$ by

$$\mathbf{t}_j^0(\mathbf{q}) - \mathbf{t}_e^0 = 0 \tag{12}$$

$$\log\left(\left(R_e^0\right)^{-1} R_j^0(\mathbf{q})\right) = 0 \tag{13}$$

where $\mathbf{t}$ is the translation part of H. For R a rotation of axis $\mathbf{u}$ and angle θ, we have $\log(R) = \theta\mathbf{u}$.

For a contact task, the frame j is chosen as described in 4.1 and e is a frame such that its origin is on an environment surface, and its z-axis correspond to the outer normal of the surface at this point.

For more modularity, we implemented the rotation part 13 by expressing the perpendicularity between the axes of frames j and e:

$$\mathbf{i}^0(\mathbf{q}).\mathbf{b}^0 = 0 \tag{14}$$

$$\mathbf{i}^0(\mathbf{q}).\mathbf{n}^0 = 0 \tag{15}$$

$$\mathbf{j}^0(\mathbf{q}).\mathbf{n}^0 = 0 \tag{16}$$

$$-\mathbf{j}^0(\mathbf{q}).\mathbf{b}^0 \leq 0 \tag{17}$$

$$-\mathbf{k}^0(\mathbf{q}).\mathbf{n}^0 \leq 0 \tag{18}$$

with $\mathbf{i}^j = \mathbf{d}^e = (1,0,0)^T$, $\mathbf{j}^j = \mathbf{b}^e = (0,1,0)^T$, $\mathbf{k}^j = \mathbf{n}^e = (0,0,1)$.
The two inequalities are necessary to discriminate between the four possible cases satisfying the equalities.

The advantage of such a set of constraints over eq. (13) is the possibility to take a subset of it. We use later the following weaker positioning constraints:

$$\begin{aligned} \left(\mathbf{t}_j^0(\mathbf{q}) - \mathbf{t}_e^0\right).\mathbf{n}^0 &= 0 && (19)\\ \mathbf{i}^0(\mathbf{q}).\mathbf{n}^0 &= 0 && (20)\\ \mathbf{j}^0(\mathbf{q}).\mathbf{n}^0 &= 0 && (21)\\ -\mathbf{k}^0(\mathbf{q}).\mathbf{n}^0 &\leq 0 && (22) \end{aligned}$$

These constraints force the origin of frame j to lie in the plan x-y of frame e but not at a particular position, and the z-axis of j is aligned with the z-axis of e, with the rotation around this axis let free. For a planar surface described by e, it means the contact spot j (of body r_i) is in contact with this surface, but the parameters (x, y, θ) of this contact are not fixed.

Other geometric tasks can be described this way, see for example [24].

6.3. Collision constraints

To express that two objects O_1 and O_2 (bodies of the robot or parts of the environment) are not in collision, we use a signed pseudo-distance, i.e. a function d such that

$$\begin{cases} d(\mathbf{q}) > 0 & \text{if } O_1(\mathbf{q}) \cap O_2(\mathbf{q}) = \emptyset & (23)\\ d(\mathbf{q}) = 0 & \text{if } O_1(\mathbf{q}) \cap O_2(\mathbf{q}) = \partial O_1(\mathbf{q}) \cap \partial O_2(\mathbf{q}) & (24)\\ d(\mathbf{q}) < 0 & \text{otherwise} & (25) \end{cases}$$

Additionally, the gradient of this function needs to be continuous for the optimization solver[1].
The Euclidean distance between the objects, extended to the penetration case by quantifying the depth [34], is a suitable candidate for eqs. (23) to (25), and its gradient is continuous when both objects are convex and one of them is strictly convex. We presented in [35] a strictly convex bounding volume to ensure this gradient's continuity.

[1] non-smooth optimization routines for problems with generic objective and constraint functions are difficult to find off-the-shelf and usually not as fast and perfected as state-of-the-art algorithms for smooth optimization.

To compute the gradient of d, we extend the reasoning found in [36] to the case of two moving objects: if P_1 and P_2 are the *fixed* points of O_1 and O_2 that coincide with the witness points of d (i.e. the points realizing the distance) at configuration $\mathbf{q}$, denoting $\mathbf{p}_1^0(\mathbf{q})$ and $\mathbf{p}_2^0(\mathbf{q})$ their coordinates in the world frame, we have

$$\frac{\partial d}{\partial \mathbf{q}} = \frac{1}{d(\mathbf{q})} \left(\mathbf{p}_1^0(\mathbf{q}) - \mathbf{p}_2^0(\mathbf{q})\right)^T \left(\frac{\partial \mathbf{p}_1^0}{\partial \mathbf{q}} - \frac{\partial \mathbf{p}_2^0}{\partial \mathbf{q}}\right) \tag{26}$$

6.4. Stability constraints

We give here two expressions for the stability constraint $s(\mathbf{q})$. The simplest one is based on the classical projection of the CoM in the sustentation polygon [37]: for a contact, the contact spot gives us a set of points $h = \left\{\mathbf{p}_1^j, \cdots, \mathbf{p}_k^j\right\}$ expressed in its contact frame j. When the contact is realized, we have $H_j^0(\mathbf{q}) = H_e^0$, so that the contact points are at the positions $H_e^0 h = \left\{H_e^0 \mathbf{p}_1^j, \cdots, H_e^0 \mathbf{p}_k^j\right\}$. Now, if we have all l contacts, the sustentation polygon is the (2d) convex hull of all the projected contact points: $\text{conv}\left(\pi\left(H_{e_1}^0 h_1\right) \cup \cdots \cup \pi\left(H_{e_l}^0 h_l\right)\right)$, where $\pi : \mathbb{R}^3 \longrightarrow \mathbb{R}^2$ denotes the projection onto the x-y plane. We denote $\boldsymbol{\pi}_i$ the m points of this convex hull indexed in the trigonometric order, and $\mathbf{n}_i = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} (\boldsymbol{\pi}_{i+1} - \boldsymbol{\pi}_i)$ (with the convention $\boldsymbol{\pi}_{m+1} \equiv \boldsymbol{\pi}_0$), the outer normal to segment $\boldsymbol{\pi}_i\boldsymbol{\pi}_{i+1}$. Then, $\mathbf{G}(\mathbf{q})$ being the CoM expressed in the world frame, we have:

$$s(\mathbf{q}) = \begin{pmatrix} \left[\boldsymbol{\pi}_1 - \pi(\mathbf{G}(\mathbf{q}))\right]^T \mathbf{n}_1 \\ \vdots \\ \left[\boldsymbol{\pi}_m - \pi(\mathbf{G}(\mathbf{q}))\right]^T \mathbf{n}_m \end{pmatrix} \tag{27}$$

This criterion is valid if all contacts are on the same horizontal plane. It is conservative if the contacts are on different horizontal planes (see [37]). For both situations, this is the criterion we adopt because it is fast. In all other cases we use a second criterion, which is a static version of the one proposed in [38, 39] and similar to [40]: if we consider that $\boldsymbol{\tau}$ is not bounded, then the first n lines of eq. (11) can always be satisfied, and the robot is only constrained by the last 6 lines which are equivalent to the Euler-Newton equations: $\mathbf{g}_2(\mathbf{q}) = J_{c_2}(\mathbf{q})^T \mathbf{f}$. Note that if the equation is expressed in the world frame, what we consider now, J_{c_2} is independent of the configuration. Furthermore, to avoid sliding, each contact force must be inside a Coulomb

friction cone and so $\mathbf{f}$ is inside a product of cones: $\mathbf{f} \in \mathcal{F} = \times \mathcal{C}_j$. If we consider discretized friction cones, then $\mathcal{F}$ is also a discretized (or polyhedral) cone (in $\dim(\mathbf{f})$ dimensions) so that $\mathbf{f}$ can be written as a positive linear combination of rays: $\mathbf{f} = \sum_{\alpha_i \geq 0} \alpha_i \mathbf{f}_i$ where the $\mathbf{f}_i$ are deduced from the rays of the discretized friction cones. It follows that $\mathbf{g}_2(\mathbf{q}) = \sum_{\alpha_i \geq 0} \alpha_i J_{c_2}^T \mathbf{f}_i$, hence $\mathbf{g}_2(\mathbf{q})$ belongs to a (possibly open) polytope that is the 6d convex hull of the $J_{c_2}^T \mathbf{f}_i$. Using a double description algorithm such as `cddlib` [41], we can get a representation of this polytope as a set of linear inequalities and so write $s(\mathbf{q}) = A\mathbf{g}_2(\mathbf{q}) + \mathbf{b}$. With further operations, this criterion can even be expressed in 3 dimensions [42]. Computing the linear inequalities takes up to a few ms for several contacts, each with 4 contact points and 4-sided discretized cones.

6.5. Adding a criterion

If $\mathcal{Q}$ is not empty, it is generally a non-countable set. It might then be useful to specify a criterion o to be minimized, so as to select a particular posture. The feasibility problem becomes then a minimization problem

$$\min_{\mathbf{q} \in \mathcal{Q}} o(\mathbf{q}), \tag{28}$$

An advantage to using a criterion is that the obtained witness posture is less dependent on the solver mechanisms and the initial guess (the optimization problem may however have some local minima, even if o is convex, because of the constraints). This is important when the potential field of the planning is based on $\mathbf{q}$ so that nodes are "fairly" evaluated. To this end using the planner's potential field as a posture criterion is a possibility we use later.

Otherwise, the criterion can have several forms, from maximizing the stability of the robot (by rewriting each line of the stability constraint as $s_i(\mathbf{q}) \leq w$, and minimizing w) to notions of comfort posture (see e.g. [43]). We first used the distance to a reference posture: $\|\mathbf{q} - \mathbf{q}^*\|^2$ which gives good results for simple scenarios [22]. In the next section, we propose a variant of a criterion based on the distance to a reference path $s \in [0,1] \rightarrow \mathbf{q}^*(s) \in \mathcal{C}$:

$$\min_{s, \mathbf{q}} \|\mathbf{q} - \mathbf{q}^*(s)\|^2 \tag{29}$$

6.6. Implementation details

To solve the above feasibility/optimization problem we use FSQP [44].Our posture generator is build over two parts as advocated in [45] (Chapter 1): the

algorithm part which consists in an object encapsulation of FSQP and knows nothing about the problem to be solved, and the simulation part, which we called *Posture Calculator*, that performs all posture-related computation at the request of the algorithm, and knows nothing about optimization (see Fig. 4).

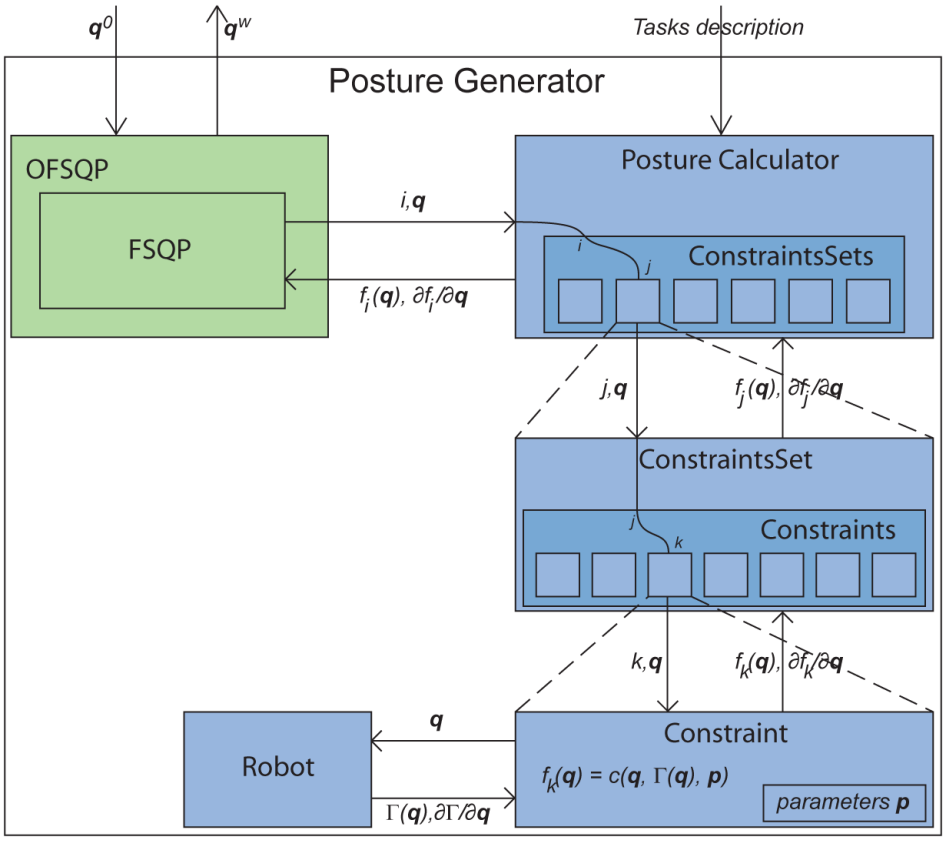

Figure 4: The architecture of the posture generator, as a dialogue between an optimization algorithm (in green) which asks for the values or derivatives at a point of the problem functions and a simulator (in blue) which computes them.

The *Robot* class performs all the computations related to the kinematics of the robot, for instance the position of a point of the robot in the world frame, or its gradient. The *Constraint* class represents a function from $\mathbb{R}^m$ into R (with m the number of variables) that can be considered either as constraint or objective function by the solver. It implements the different kinds of parametric function that we need to describe the problem.

For example the collision avoidance constraint between the objects O_i and O_j computes $a \cdot d(i, j, \mathbf{q}) + b$ where a, b, i and j are the parameters and $d(i, j, \mathbf{q}) = d(O_i(\mathbf{q}), O_j(\mathbf{q}))$.
ConstraintsSet implements a group of constraints describing a task, for instance a contact or the stability criterion. It is meant to help the user by providing a higher level interface, so that he just has to give the task parameters, and the corresponding constraints are automatically generated.
Lastly, the *Posture Calculator* handles two functionalities: it ensures the sorting of the constraints according to FSQP requirement (non-linear inequalities first, then non-linear equalities...) and it keeps up-to-date the stability and collision constraints according to the contacts added or removed.

The computations are made using lazy evaluation so as to avoid evaluating twice the same data.

7. Improving Planning

7.1. Advanced contact generation

We saw that for a pair of $\mathcal{S}_{\text{contacts}} \times \mathcal{S}_{\text{surfaces}}$ a contact is specified by a triple (x, y, θ). For this pair, the sets of all triples is a subset V of $\mathbb{R}^3$ which can be seen as the product of the surface (or the domain of its parameters) and the segment $[0, 2\pi]$ (see Fig. 5).

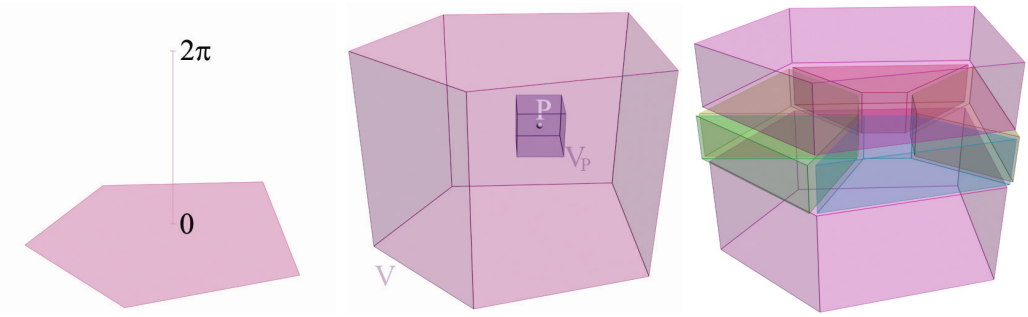

Figure 5: The search volume V is build as the product of the surface's parameters domain and the segment $[0, 2\pi]$ (left). When a new contact with parameters P is found, a volume V_P is constructed around (middle) and $V \backslash V_P$ is divided into convex sub-volumes (right).

Searching for contact candidates amounts to finding points in this volume, and the three points of section 5.4 can be rewritten as:

- (i) ensure a good coverage of V,

- (ii) give points not too close from each other,
- (iii) give points corresponding to feasible contacts as much as possible.

(iii) remains the difficult point: if s is the set of contacts we try to add a contact to, the projection of $\mathcal{Q}_s$ onto V is far from being trivial so that biasing the search toward "good" points in V does not seem to be practical.

The solution we propose is to let (x, y, θ) be chosen by the Posture Generator, rather than proposing candidates which may not be feasible: using eqs. (19) to (22) instead of eqs. (14) to (18) for a new contact involving r_i constrains only 3 dofs of r_i, and lets (x, y, θ) free. Upon successful completion of the posture generation with this weaker form of contact, we have the parameters of a feasible contact. If the generation fails, then we assume no contact can be found for the current pair (body, surface). This assumption may not hold, because the Posture Generator does not guarantee to find a solution if there is one. However, it happens to be true in practice for the vast majority of cases since the optimization is started with the configuration of the father, hence not far from the solution.

Once a first point $P = (x, y, \theta)$ has been found in V, we have to find another one, not too close from it (condition (ii)). To do so, the idea is to generate a new point in $V \backslash V_P$ where V_P is a small volume centered around P, then, for this new point, exclude another volume from V and generate a new one, repeating these steps until the Posture Generator cannot find a solution. In practice, we want to work with convex search volumes to avoid introducing (more) local minima in the optimization scheme, and we furthermore restrain these volumes to have boundaries in θ of the form $\theta =$ constant. Such a volume can be described by the following inequalities:

$$A\left(\mathbf{t}_j^0(\mathbf{q}) - \mathbf{t}_e^0\right) + b \leq 0 \tag{30}$$

$$\theta^- \leq \theta(\mathbf{q}) \leq \theta^+ \tag{31}$$

where A and b describe the limitation of the surface (that we assume to be convex or divisible in convex pieces) and $\theta(\mathbf{q})$ is the angle between the projection of $\left(\mathbf{t}_j^0(\mathbf{q}) - \mathbf{t}_e^0\right)$ on the surface and the reference direction. The volume V_P is chosen to be a parallelepiped of dimensions $\delta l \times \delta l \times \delta\theta$, with δl and $\delta\theta$ two parameters that control the density of points. $V \backslash V_P$ is then divided in convex volumes of the same shape (see Fig. 5). In each of these volumes a generation will be attempted before further subdivision, until no solution exists. This leads to algorithm 5.

The *construcSearchVolume*(j) method divides the surface with index j into convex pieces and builds the corresponding search volumes by computing their product with $[0, 2\pi]$. *generate*(n, i, V) performs a posture generation with the contacts of n and the weaker form of contact involving r_i within the search volume V. *retrieveContact*(n, i, j) identifies the values (x, y, θ) for the contact between the pair identified by (i, j) at configuration $n.\mathbf{q}$ and returns the contact. Lastly, *divideSearchVolume*(V, c) subdivides $V \backslash V_P$ into convex search volumes, where P is the point corresponding to c, and returns these new volumes, if any ($V \backslash V_P$ eventually gets empty). The algorithm terminates when there is no more search volumes in which to find contacts.

Algorithm: generateNewContacts2
Data: n, a node, i and j, indices of the contact spot and the surface
Result: l, a list of leaves whose father is n, initially empty
- N is the set of the nodes already generated
begin
  if forbiddenContact*(i,j)* **then**
    return($\oslash$)
  $n' \leftarrow n$
  $\mathcal{V} \leftarrow$ *construcSearchVolume*(j)
  while *no* isEmpty*($\mathcal{V}$)* **do**
    $V \leftarrow$ *first*($\mathcal{V}$)
    if generate*(n', i, V)* **then**
      $c \leftarrow$ *retrieveContact*(n',i,j)
      $n' \leftarrow (n,\ n'.s \cup \{c\},\ n'.\mathbf{q})$
      $\mathcal{V} \leftarrow \mathcal{V} \cup$ *divideSearchVolume*(V, c)
      if add*(n',N)* **then**
        $l \leftarrow l \cup \{n'\}$
  return(l)

Algorithm 5: Contact generation with search volumes

7.2. Guide path and potential field

Using solely an attracting potential field to guide the planner toward the goal appears to be limited because the presence of obstacles will create local minima in $\mathcal{C}_{\text{free}}$ whose size will be correlated to the size of the obstacles, and the number of contact spots. We thus need a field that guides the planning

around (or above/under) the obstacles. We achieve that by constructing a descending valley-like shape potential field following a rough path in $\mathcal{C}$ (see Fig 6). This rough path gives an idea of the overall trajectory without the detailed movements due to the contacts. We define it as a linear interpolation between a small numbers of key postures. It does not matter if the path thus obtained does not lay fully in $\mathcal{C}_{\text{free}}$. These key postures can be computed automatically [46] or given manually. In the latter case, it gives the user a way to encode high level specification about the motion like going around or under the table, entering the house by the front or rear door...

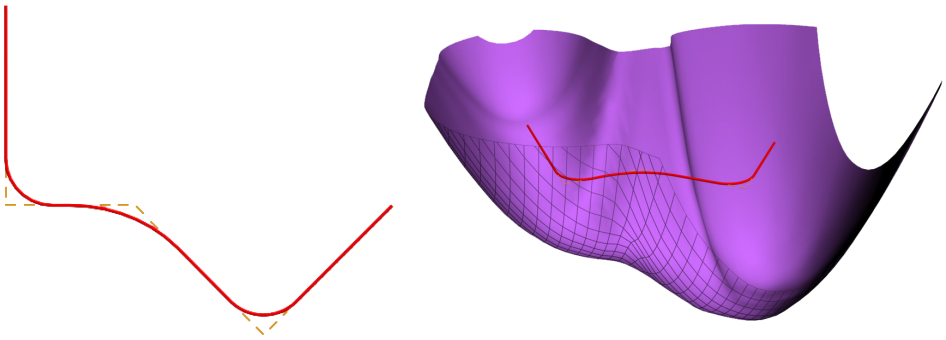

Figure 6: Starting from a piecewise linear path (dotted line, on the left) we construct a smoother path (plain line), upon which we base a descending valley-like shaped potential field (right).

Let $\mathfrak{p} : [0,1] \longrightarrow \mathcal{C}$ be a continuous path, and $\pi : \mathcal{C} \longrightarrow \mathcal{C}_{\text{root}}$ the projection onto the configuration space of the robot's root. We denote $\mathbf{q}'$ and $\mathfrak{p}'$ the images of a configuration $\mathbf{q}$ and the path $\mathfrak{p}$ by π. We can safely assume that π is bijective from $\mathfrak{p}$ to $\mathfrak{p}'$: it would not be the case if $\mathfrak{p}'$ was self-intersecting, which would imply the robot is coming back to a place he was before (all the movements in between were unnecessary) or if the posture was changing while the root is kept fixed. This second case can be easily avoided: it does not hurt to change a bit the root's position, the path being only approximate. Then, denoting $\mathbf{q}'_{\mathfrak{p}}$ the closest point of $\mathfrak{p}'$ from $\mathbf{q}'$, we define $\mathbf{q}_{\mathfrak{p}} = \pi^{-1}(\mathbf{q}'_{\mathfrak{p}})$, and write the potential field as

$$U_{\mathfrak{p}}(\mathbf{q}) = \|\mathbf{q} - \mathbf{q}_{\mathfrak{p}}\|^2 - \alpha \cdot l' \tag{32}$$

where l' is the curvilinear coordinate of $\mathbf{q}'_{\mathfrak{p}}$ along $\mathfrak{p}'$. The first term of this function is a pseudo-distance to the path and gives the valley-like shape; the

second accounts for the distance traveled along the path, and gives the slope toward the goal. The α factor weights the contribution of the two terms.

If $\mathfrak{p}$ is piecewise linear, this potential field however is not even continuous because the closest point $\mathbf{q}'_{\mathfrak{p}}$ jumps when $\mathbf{q}'$ passes from the voronoi region of a segment to the other, implying a discontinuity of l'. This can be partially corrected by "rounding" the angles of $\mathfrak{p}$: we replace each angle by an arc of circle (in dimension $n+6$) for which we chose the maximal radius such as the distance to the original path is less than a given constant. As long as the distance to the modified path is less than the radii, $U_{\mathfrak{p}}$ is C^1.

Two terms can be added to this potential field. First, from $\mathfrak{p}$, it is easy to derive guide paths $\mathfrak{p}_i$ for the bodies r_i, and build potential fields $U_{\mathfrak{p}_i}(\mathbf{q})$ based on the distance from the bodies to their guide paths. Adding such terms to $U_{\mathfrak{p}}$ for selected bodies (typically the feet and hands, sometimes the knees) enables the planner to anticipate the movements of these bodies which is useful when the robot needs to radically change the posture (for example to pass from walking to crawling) (see [25]).
Second, we can add a term U_S over $\mathcal{S}$, as devised in 5.5, to account for the characteristics of the set of contacts.

7.3. Very Best First Contact Planning

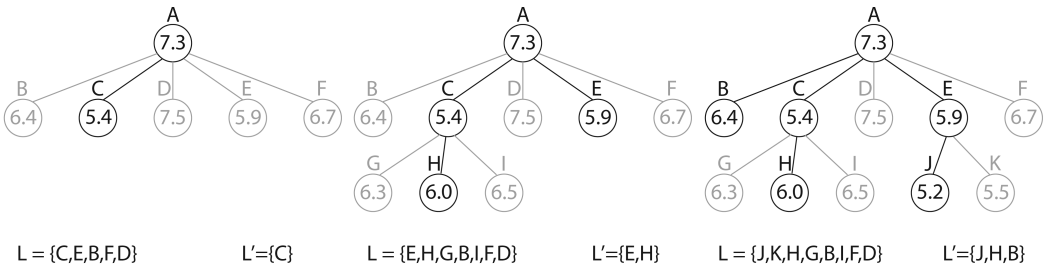

Figure 7: VBFP (black) versus BFP (black and gray). L is the list of leaves in BFP, L' in VBFP. Starting from node A, only the best son (C) is generated at first with VBFP whereas BFP would generate all five sons (left). Then (center), C is chosen to be expanded in which case its best sibling (E) replaces it in L', and the best son of C is generated. Finally (right), E being the best leaf in L' (or L), it is expanded by only its best sons J, and replaced by its best sibling (B) in L'. The first element of L' is always the same as the first one of L, ensuring the correctness of VBFP.

Unless the planner gets stuck in a huge local minima, what should be avoided by the use of the previously proposed potential field if a solution exists, most of the nodes generated during the planning will never be visited

for further expansion, and thus are generated to no avail. We present here a method to drastically reduce the number of generated nodes.

Suppose at first that we have an oracle capable to tell us at no cost for a node f which of its sons n will have the best rating according to the potential field. Then, when f is expanded, it suffices to generate n and to add it in the list of leaves L. Indeed, the siblings of n being worse than n they will never be selected for expansion as long as n has not been selected itself. When, at an ulterior iteration, n is selected, its best sibling n' has to be put in L because it might be better than all the other elements already in L. So, each time a node is selected for expansion, only its best son and its best sibling are generated and inserted in L (see fig. 7), which greatly reduce the number of unused nodes generated. We call this principle *Very Best First Planning*.

Unfortunately, in the case of contact planning, we do not have such an oracle, but we have a semi-oracle in the sense that we are able to predict that some nodes will be better than some others without generating them. This is achieved by using the potential field of the planner as the objective function for the posture generation. In this case, the new contact found for a given search volume will yield a better node than the contacts found for the subsequent sub-volumes. This leads to a modification of Alg. 2 we named *Contacts VBFP*, where the line 11: $l \leftarrow$ *generateSons*(n) is replaced by

$$l \leftarrow \textit{generateBestSons}(n) \cup \textit{generateNextSiblings}(n)$$

The method *generateBestSons*(n) tries to generates all the nodes obtained by removing one contact from n, and for each couple of $\mathcal{S}_{\text{contacts}} \times \mathcal{S}_{\text{surfaces}}$, only the first node that would be obtained by algorithm 5. We cannot indeed predict the ranking order between theses nodes. *generateNextSiblings*(n) returns an empty list when n has been obtained by removing a contact. If n has been created by adding a contact c, then the method tries to generate the nodes for the same couple of $\mathcal{S}_{\text{contacts}} \times \mathcal{S}_{\text{surfaces}}$ and the sub-volumes issued from the volume in which c was generated.

We observed that this lazy-generation scheme leads to creating 3 to 5 times less nodes than in the CBFP algorithm, with a similar gain in computation time and memory.

8. Results

In this section, we present the output of our planner applied to a HRP-2 robot in three scenarios of increasing complexity (see Fig. 8): climbing

a ladder, crawling through a narrow tunnel, and sitting at a table while carrying a glass. The computations are performed on a single thread on a Pentium 4 at 3.2 GHz, with 2 Go of RAM. 117 pairs of bodies (r_i, r_j) are considered for auto-collision, all pairs are considered for robot-environment collision, and additional analytical constraints on the joint angles are written for collision avoidance at the hips and the neck (they have been identified empirically).

8.1. Ladder scenario

We begin by taking a situation similar to one in [20], in order to get a comparison: the robot has to climb a ladder, about 2m tall. To get as close as possible to the conditions of [20], we restrict the robot to use only its feet and one face of each hand, and fix the orientation of the contacts. We however let the contact occur at any place on the lower and upper grounds and on the rungs of the ladder. For this experiment, while all the support surfaces are horizontal, we use the general stability criterion presented in section 6.4, the CoM projection criterion being too conservative for this case. The robot starts in front of the ladder and the goal is specified as a region (at the top of the ladder) in which the waist needs to be.
Our planner finds a path with 35 nodes in about 8.5 minutes which is only slightly slower than [20] where the possible contact placements are given beforehand. A total of 497 nodes were generated.

8.2. Tunnel scenario

In this scenario, we demonstrate the ability of our planner to cope with cluttered environments and particularly to enter in and evolve through a narrow space, which is usually recognized as difficult for planners: the robot, which is 1.54m tall and 63cm wide has to go through a tunnel 73cm wide, 54cm high and 90cm long. The entrance is in front of its initial position, 35cm above the ground. Ten contact spots are defined on the robot: the feet, the knees, the elbows, the side of each hand, and the top the pinches. Four key postures are given to define the guide path: standing up in front of the tunnel, on its hands and knees at the entrance and at the exit of the tunnel and standing up after. The goal is specified as a region in wich the waist needs to be.
The solution is found in 56.2 minutes and consists in 101 nodes, for a total of 3656 nodes generated. The planning time is divided in three roughly equal parts: entering the tunnel, going through it and getting up.

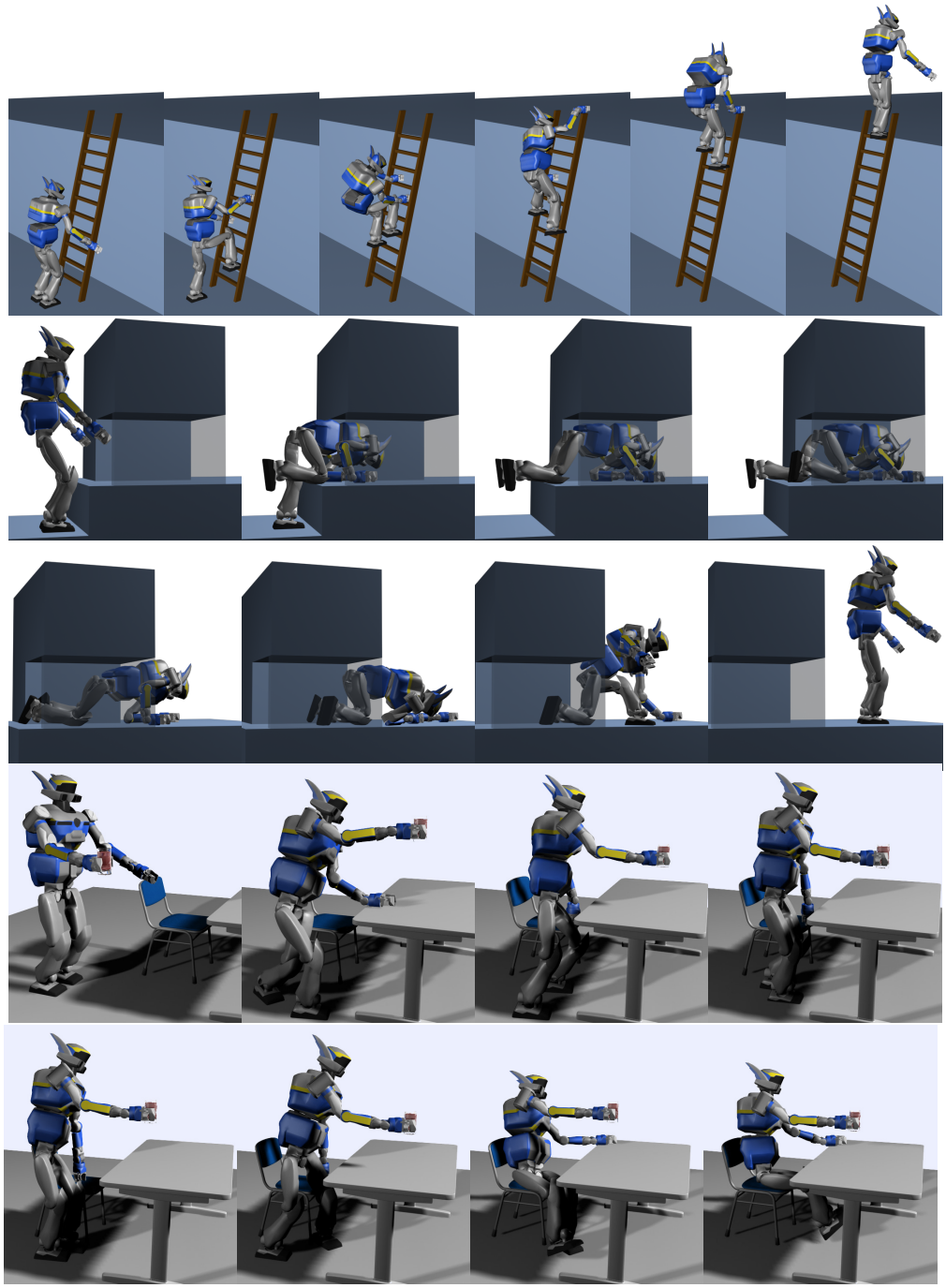

Figure 8: Initial, final and some inbetween steps for the three scenarios.

8.3. Table scenario

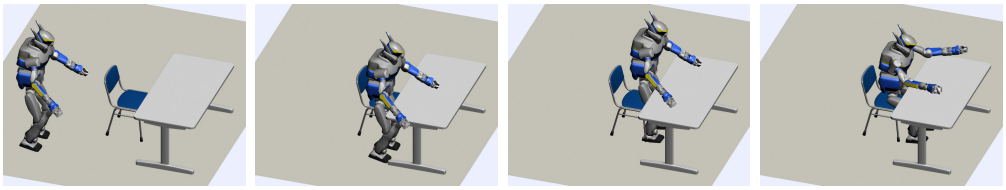

Figure 9: Key postures to define the rough path $\mathfrak{p}$ for the table scenario.

In this third scenario, the robot has to go and sit at a table while carrying a glass full of liquid in its right hand. The planning problem is defined as follow: there are 3 objects in the environment: the floor, the chair and the table. Twelve contact spots are defined: the feet, the knees, the thighs, the side of each hand, and the top the pinches and three environment surfaces are considered for contact: the floor, the seat of the chair, and the top surface of the table. We forbid the (hand, floor), (foot, chair) and (foot, table) contact pairs. An additional task $\mathcal{T}_{\text{glass}}$ is added to every posture generation to keep the glass vertical. Consequently, all contacts with the right hand are forbidden. The main difficulty of the planning is that the robot has to enter the narrow space between the table and the chair, space in which it cannot be in full upright position. In particular, while the space seems more open than in the previous scenario, it is more difficult to find stable posture when evolving around the chair. The task $\mathcal{T}_{\text{glass}}$ adds complexity to the posture generation and prevents the robot to use its right hand for locomotion. We give manually the rough path with 4 key postures as depicted in Fig 9.

The result, depicted in Fig. 8 is a sequence of 69 nodes, obtained in about 3 hours, after 3800 nodes have been generated. The robot quickly gets to the side of the chair, before taking some time to put its feet in front of the chair (after about 50 nodes). Once it has entered this narrow passage, it reaches the goal in about 20 nodes. When near to the chair, the robot takes support on the chair and the table with its left hand. Experiments on a real HRP-2 robot of this scenario are reported thoroughly in [23][24].

9. Conclusion

Planning a general multipurpose and multi-contact motion for humanoid robots is possible by relaxing the contacts to occur with any part of the robot and any part of the environment. Our approach consists in building a tree

that explores possible contact configurations thanks to a generalized inverse kinematics that is formulated as a non-linear optimization problem in order to account for robot limitations in terms of joints limits, static equilibrium, non-desired collision avoidance, etc. Several complex scenarios were successfully planned using our planner. Further extensions of our method are already under implementation and consist in generalizing it to more generic agents as well as unifying multi-contact locomotion and manipulation in a single planning framework for all the components (the posture generator in [47], the planner in [48]). We also considered deformable environments [49]. This demonstrates that our algorithm foundations are viable enough to allow such extensions. As future work, we will need to reconsider the issue of scalability with respect to the number of agents and the complexity of the situations. The bottleneck for the planning is the posture generation. Several improvements can be made on this point to speed up the computations. First, the sons generation is highly parallelizable: in the Contact VBFP presented here, all the posture generation problems are independent when expanding a node. We will soon make the necessary code changes to benefit from the multi-core feature of modern computers. It is also necessary to reconsider the solver with respect to our problem. By taking into account the particularities of the problem we solve, it is possible to write a dedicated solver and tune it carefully to converge rapidly toward an existing solution or detect quickly when there is no solution. We expect a huge speed gain from both the parallelization and the solver specialization.

Our planner generates contact sets; the motion transition between these sets is computed in a second step which is not considered here, but we started investigating two possible approaches: the first one uses semi-infinite optimization techniques [50][51], the other one is a closed-loop control approach using multi-objective optimization [52][53].

References

[1] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi, H. Hirukawa, Biped walking pattern generation by using preview control of zero-moment point, in: IEEE International Conference on Robotics and Automation, 1620–1626, 2003.

[2] T. Takenaka, T. Matsumoto, T. Yoshiike, Real Time Motion Generation and Control for Biped Robot –1st Report: Walking Gait Pattern

Generation–, in: IEEE/RSJ International Conference on Robots and Intelligent Systems, St. Louis, USA, 1084–1091, 2009.

[3] A. Herdt, H. Diedam, P.-B. Wieber, D. Dimitrov, K. Mombaur, M. Diehl, Online Walking Motion Generation with Automatic Foot Step Placement, Advanced Robotics 24 (5-6) (2010) 719–737.

[4] A. Herdt, N. Perrin, P.-B. Wieber, Walking without thinking about it, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, 190–195, 2010.

[5] T. Takenaka, T. Matsumoto, T. Yoshiike, S. Shirokura, Real Time Motion Generation and Control for Biped Robot –2st Report: Running Gait Pattern Generation–, in: IEEE/RSJ International Conference on Robots and Intelligent Systems, St. Louis, USA, 1092–1099, 2009.

[6] T. Takenaka, T. Matsumoto, T. Yoshiike, Real Time Motion Generation and Control for Biped Robot –3rd Report: Dynamics Error Compensation–, in: IEEE/RSJ International Conference on Robots and Intelligent Systems, St. Louis, USA, 1594–1600, 2009.

[7] T. Takenaka, T. Matsumoto, T. Yoshiike, T. Hasegawa, S. Shirokura, H. Kaneko, A. Orita, Real Time Motion Generation and Control for Biped Robot –4th Report: Integrated Balance Control–, in: IEEE/RSJ International Conference on Robots and Intelligent Systems, St. Louis, USA, 1601–1608, 2009.

[8] M. Kalisiak, M. van de Panne, A Grasp-based Motion Planning Algorithm for Character Animation, The Journal of Visualization and Computer Animation 12 (3) (2001) 117–129.

[9] K. Yin, K. Loken, M. van de Panne, SIMBICON: simple biped locomotion control, ACM Transactions on Graphics (SIGGRAPH) 26, ISSN 0730-0301.

[10] U. Muico, Y. Lee, J. Popović, Z. Popović, Contact-aware Nonlinear Control of Dynamic Characters, ACM Transactions on Graphics (SIGGRAPH) 28 (3).

[11] I. Mordatch, M. de Lasa, A. Hertzmann, Robust Physics-Based Locomotion Using Low-Dimensional Planning, ACM Transactions on Graphics 29 (3).

[12] S. Coros, P. Beaudoin, M. van de Panne, Generalized Biped Walking Control, ACM Transctions on Graphics 29 (4) (2010) Article 130.

[13] J. chi Wu, Z. Popović, Terrain-adaptive bipedal locomotion control, ACM Transactions on Graphics (SIGGRAPH) 29 (4) (2010) 72:1–72:10.

[14] L. Sentis, M. Slovich, Motion Planning of Extreme Locomotion Maneuvers Using Multi-Contact Dynamics and Numerical Integration, in: IEEE/RAS International Conference on Humanoid Robotics, 2011.

[15] K. Yokoi, E. Yoshida, H. Sanada, Unified motion planning of passing under obstacles with humanoid robots, in: IEEE International Conference on Robotics and Automation, 2009.

[16] B. Verrelst, O. Stasse, K. Yokoi, B. Vanderborght, Dynamically Stepping Over Obstacles by the Humanoid Robot HRP-2, in: IEEE RAS/RSJ Conference on Humanoid Robots, Genova, Italy, 117–123, oral Presentation, 2006.

[17] F. Kanehiro, T. Yoshimi, S. Kajita, M. Morisawa, K. Fujiwara, K. Harada, K. Kaneko, H. Hirukawa, F. Tomita, Whole Body Locomotion Planning of Humanoid Robots based on a 3D Grid Map, in: IEEE International Conference on Robotics and Automation, 1072–1078, 2005.

[18] Z. Lu, T. Aoyama, K. Sekiyama, Y. Hasegawa, T. Fukuda, Walk-to-brachiate Transfer of Multi-Locomotion Robot with Error Recovery, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, Taipei, Taiwan, 166–171, 2010.

[19] T. Bretl, Multi-Step Motion Planning: Application to Free-Climbing Robots, Ph.D. thesis, Stanford University, 2005.

[20] K. Hauser, T. Bretl, J.-C. Latombe, Non-gaited humanoid locomotion planning, in: IEEE/RSJ International Conference on Humanoid Robots, 7–12, 2005.

[21] K. Hauser, T. Bretl, K. Harada, J.-C. Latombe, Using Motion Primitives in Probabilistic Sample-Based Planning for Humanoid Robots, in: Workshop on the Algorithmic Foundations of Robotics, 2006.

[22] A. Escande, A. Kheddar, S. Miossec, Planning support contact-points for humanoid robots and experiments on HRP-2, in: IEEE/RSJ International Conference on Robots and Intelligent Systems, Beijing, China, 2974–2979, 2006.

[23] A. Escande, A. Kheddar, S. Miossec, S. Garsault, Planning support contact-points for acyclic motions and experiments on HRP-2, in: International Symposium on Experimental Robotics, Athens, Greece, 2008.

[24] A. Escande, A. Kheddar, Contact planning for acyclic motion with tasks constraints, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, St. Louis, USA, 435–440, 2009.

[25] A. Escande, A. Kheddar, Planning contact supports for acyclic motion with task constraints and experiment on HRP-2, in: IFAC 9th International Symposium on Robot Control (SYROCO'09), Gifu, Japan, 259–264, 2009.

[26] K. Kaneko, F. Kanehiro, S. Kajita, H. Hirukawa, T. Kawasaki, M. Hirata, K. Akachi, T. Isozumi, Humanoid robot HRP-2, in: IEEE International Conference on Robotics and Automation, New Orleans, LA, 1083–1090, 2004.

[27] T. Siméon, J. Cortès, J.-P. Laumond, A. Sahbani, Manipulation planning with probabilistic roadmaps, The International Journal of Robotics Research 23 (7-8) (2004) 729–746.

[28] K. Hauser, V. Ng-Thow-Hing, H. Gonzalez-Banos, Multi-Modal Motion Planning for a Humanoid Robot Manipulation Task, in: International Symposium on Robotics Research, 2007.

[29] J.-C. Latombe, Robot motion planning, Kluwer Academic Publishers, Boston-Dordrecht-London, ISBN 0-7923-9129-2, 1991.

[30] J. Cortés, Motion Planning Algorithms for General Closed-Chain Mechanisms, Ph.D. thesis, 2003.

[31] M. Stilman, Task Constrained Motion Planning in Robot Joint Space, in: IEEE/RSJ International Conference on Robots and Intelligent Systems, 2007.

[32] C. Samson, M. Le Borgne, B. Espiau, Robot Control: the Task Function Approach, Clarendon Press, Oxford, United Kingdom, 1991.

[33] V. Duindam, S. Stramigioli, Modeling and Control for Efficient Bipedal Walking Robots - A Port-Based Approach, vol. 53 of *Springer Tracts in Advanced Robotics*, Springer, ISBN 978-3-540-89917-4, 2009.

[34] G. van den Bergen, Collision detection in interactive 3D environments, The Morgan Kaufmann Series in Interactive 3D Technology, Morgan Kaufmann Publishers, 2004.

[35] A. Escande, S. Miossec, A. Kheddar, Continuous gradient proximity distance for humanoids free-collision optimized-postures, in: IEEE-RAS Conference on Humanoid Robots, Pittsburg, Pennsylvania, 2007.

[36] O. Lefebvre, F. Lamiraux, D. Bonnafous, Fast Computation of Robot-Obstacle Interactions in Nonholonomic Trajectory Deformation, in: IEEE International Conference on Robotics and Automation, Barcelona, Spain, 2005.

[37] P.-B. Wieber, On the stability of walking systems, in: The third IARP International Workshop on Humanoid and Human Friendly Robotics, AIST, Tsukuba, Japan, 53–59, 2002.

[38] S. Garsault, Non-gaited dynamic motion with muti-contact transition, Tech. Rep., Ecole Centrale de Paris, URL `http://sylvain.garsault.free.fr/pub/finalReportSGarsault.pdf`, 2008.

[39] S. Barthélemy, P. Bidaud, Stability Measure of Postural Dynamic Equilibrium based on Residual Radius, in: RoManSy'08 : 17th CISM-IFToMM Symposium on Robot Design, Dynamics and Control, 2008.

[40] T. Bretl, S. Lall, Testing Static Equilibrium for Legged Robots, IEEE Transactions on Robotics 24 (2008) 794–807.

[41] K. Fukuda, A. Prodon, Double Description method revisited, in: Lecture Notes in Computer Science, vol. 1120, 91–111, 1995.

[42] Z. Qiu, A. Escande, A. Micaelli, T. Robert, Human motions analysis and simulation based on a general criterion of stability, in: International Symposium on Digital Human Modeling, Lyon, France, 2011.

[43] J. Yang, T. Marler, H. Kim, J. Arora, K. Abdel-Malek, Multi-Objective Optimization for Upper Body Posture Predicition, in: 10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, Albany, New York, 2004.

[44] C. Lawrence, J. L. Zhou, A. L. Tits, User's Guide for CFSQP Version 2.5: A C Code for Solving (Large Scale) Constrained Nonlinear (Minimax) Optimization Problems, Generating Iterates Satisfying All Inequality Constraints, 1997.

[45] F. Bonnans, C. Gilbert, C. Lemaréchal, C. A. Sagastizábal, Numerical optimization- Theoretical and Practical Aspects, Springer, 2002.

[46] K. Bouyarmane, A. Escande, F. Lamiraux, A. Kheddar, Collision-free Contacts Guide Planning Prior to Non-Gaited Motion Planning for Humanoid Robots, in: IEEE International Conference on Robotics and Automation, 2009.

[47] K. Bouyarmane, A. Kheddar, Static multi-contact inverse problem for multiple humanoid robots and manipulated objects, in: IEEE-RAS International Conference on Humanoid Robots, Nashville, TN, 8–13, 2010.

[48] K. Bouyarmane, A. Kheddar, Multi-Contact Stances Planning for Multiple Agents, in: IEEE International Conference on Robotics and Automation, Shanghai, China, 2011.

[49] K. Bouyarmane, A. Kheddar, FEM-based Static Posture Planning for a Humanoid Robot on Deformable Contact Support, in: IEEE-RAS International Conference on Humanoid Robots, Bled, Slovenia, 2011.

[50] S. Miossec, K. Yokoi, A. Kheddar, Development of a software for motion optimization of robots - Application to the kick motion of the HRP-2 robot, 2006.

[51] S. Lengagne, P. Mathieu, A. Kheddar, E. Yoshida, Generation of dynamic motions under continuous constraints: Efficient computation using B-Splines and Taylor polynomials, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, Taipei, Taiwan, 698–703, 2010.

[52] M.-X. Liu, A. Micaelli, P. Evrard, A. Escande, C. Andriot, Interactive dynamics and balance of a virtual characters during manipulation tasks, in: IEEE International Conference on Robotics and Automation, 2011.

[53] K. Bouyarmane, A. Kheddar, Using a Multi-Objective Controller to Synthesize Simulated Humanoid Robot Motion with Changing Contact Configurations, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, San Fransico, CA, 2011.