SAPIENZA
UNIVERSITÀ DI ROMA

Master Thesis in
Artificial Intelligence and Robotics

Academic Year 2012/13

Whole-Body Motion Planning for Robotic Manipulation of Articulated Objects

---

Author: Felix Burget

Submitted on Dezember 14, 2012 in Partial Fulfillment of the Requirements for the Degree of Master of Science

Supervisors (Università di Roma - La Sapienza):
Prof. Giuseppe Oriolo
Prof.ssa Marilena Vendittelli

Supervisors (Albert-Ludwigs-Universität Freiburg):
Prof. Maren Bennewitz
Dipl.-Inf. Armin Hornung

In memory of my beloved grandfather Alois Burget,
a great engineer, artist and person

Declaration

I hereby declare, that I am the sole author and composer of my Thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work. I hereby also declare, that my Thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Place, date

Signature

Abstract

In this thesis, we present a sampling-based approach to generate whole-body motions for a humanoid robot. In particular, the planner developed is capable to generate valid whole-body motions for body repositioning as well as for grasping and manipulation of articulated objects, such as doors or drawers. Besides the generic constraints involved in motion planning, such as joint limits and collision avoidance, motions of a humanoid robot are required to satisfy stability and closure constraints. Additionally manipulation constraints arise, once the robot has grasped the handle of an object to be manipulated. In the work at hand, our planner, which is an extended variant of the RRT-CONNECT algorithm, will be presented. In the experiments we will show that the whole-body motion planner allows to perform tasks that are not feasible by a decomposition approach that plans motions for the lower and upper body consecutively. Furthermore, examples of manipulation planning will be presented and the planner is thoroughly evaluated. The experiments have been carried out successfully on both, the simulated and real humanoid robot platform.

Acknowledgements

At this point I would like to thank everyone who helped and supported me throughout the development of this thesis. I would particularly like to thank Prof. Maren Bennewitz and Prof. Giuseppe Oriolo, who supervised this thesis and helped a lot with their experience and advice. Special thanks also goes to Dipl.-Inf. Armin Hornung, Dipl.-Inf. Daniel Maier and Prof. Marilena Vendittelli, who contributed to the success of this thesis by stimulating discussions and versatile and helpful remarks. Furthermore, I would like to express my thanks to Dr. Ioan Sucan from Willow Garage Inc. for his continuous assistance with the MoveIt! framework in ROS and for always being available for questions concerning implementation issues. Another thank you goes to all of my friends in Rome, especially to Pouya, Gio, Enrico, Simone, Daniele, Martina and Taigo; the 2 years in Italy have been a great experience that I don't want to miss. A big thank you of course is dedicated to my family, friends in Germany and especially to Britta, who always believed in me and have been an essential support during my studies and adventures abroad.

Contents

List of Figures

1 Introduction

In recent years robots have made significant progress in hardware such that they are nowadays expected to be able to perform complicated tasks in complex cluttered environments. Today various kinds of robots exist, among most of them are particularly designed to meet the requirements of a specific workspace in mind. One of the most interesting and challenging field of robotics, subject to ongoing research, is the development of cognitive as well as physical skills of humanoid robots (see Fig. 1.1). As opposed to other kinds of robots, humanoids are expected to operate in environments originally designed for humans. With their body composition being based on the human skeleton, humanoid robots are required to imitate human motions such as walking, climbing stairs, grasping and manipulating objects as well as interacting with other individuals no matter whether humans or other robots.

Figure 1.1: Humanoid Robot Platforms. From left to right: Asimo (Honda Motor Co. [1]), HRP-2 (Kawada Industries Inc. [2]), Nao (Aldebaran Robotics [3]), Hubo (Korea Advanced Institute of Science and Technology [4]).

From a roboticist point of view a humanoid robot represents a highly redundant and complex system with usually a number of degrees of freedom n in the order of $n > 20$. Generating motions for these systems is a very difficult task for several reasons. First, we have to find a way to deal with redundancy. Unlike humans which naturally exploit redundancy to select motions according to ergonomic considerations or simply to sustain their health, humanoid robots must often choose a motion from an infinite set of solutions achieving the task based on some energy-related measures. Second, the sought motion must satisfy many fundamental constraints: the physical capabilities of the robot such as joint limits and actuator differential properties must be respected, collisions with obstacles in the environment must be avoided, and particulary the robot must keep

the projection of its center of mass inside the support polygon, i.e., keeping balance. Additional constraints are imposed when the robot is asked to grasp or to manipulate objects. Grasping objects, for instance, often comes with a restriction on the possible hand orientations or carrying a glass of water requires to keep the hands orientation fixed throughout the motion.

Due to the high-dimensionality of the motion planning problem for humanoid robots, many of the current approaches generate motions in advance by a functional decomposition of the degrees of freedom into a set of lower body and upper body joints. With this approach locomotion and manipulation can be considered as two decoupled problems. Manipulation has been subject to extensive research with the emergence of industrial robot arms and reliable methods has been developed to realize bipedal locomotion (see Fig. 1.2).

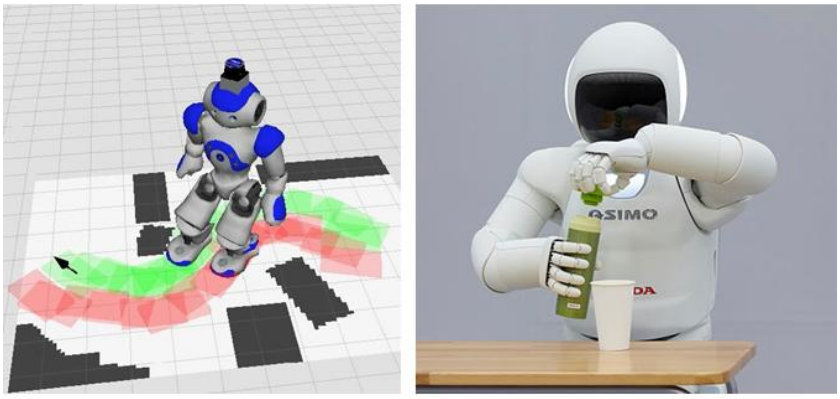

Figure 1.2: Locomotion and Manipulation for Humanoid Robots. Left: The Nao robot (Aldebaran Robotics) is walking from a start to a goal location [5]. Right: The Asimo robot (Honda Motor Co.) opens a bottle of water [6].

The major drawback of this decoupling approach is that a manipulation task cannot affect the locomotion and vice versa. Thus, only tasks that allow a sequencing of walking and manipulation, i.e., actuating the lower body and upper body limbs consecutively, can be performed. Many task in the real world however, such as picking up an object from the floor or opening a door or drawer, require to plan a coordinated motion of the lower and upper body parts.

This requirement leads us to the motivation of this work. Interacting with the environment can be generally seen as performing three tasks: *reach*, *grasp* and *manipulate*. As humans naturally do, we want humanoid robots to exploit their kinematic redundancy to accomplish these task in an efficient way. For doing so, all of the humanoids physical capabilities need to be taken into account when planning motions for the robot structure. Considering all degrees of freedom will permit to execute tasks through a whole-body motion, which before have been impossible to realize by a locomotion-manipulation sequence.

1.1 Contribution

In this work, the generic RRT-CONNECT planner, introduced in [7], has been modified to obtain a planner capable of planning statically stable collision-free whole-body motions for a humanoid robot. In order to speed up the search for valid configurations during planning, our RRT-CONNECT variant relies on a pre-computed set of stable whole-body configurations within the tree expansion process. To find a valid path, i.e., a sequence of configurations and links between them, the RRT-CONNECT planner generally performs a bidirectional search in the configuration space by growing two trees rooted at a start and goal configuration, respectively. Whereas the start configuration usually corresponds to the current state of the robot, the goal configuration is typically not known in advance. In this thesis, we generate a set of goal configurations that have the robot's hand at a desired pose and select the best among them for planning considering the manipulability of the kinematic structure. In a second step, the planner has been extended towards manipulation of articulated objects. With the robot's hand being attached to an object's handle, the planning algorithm is required to ensure that the hand remains on the handle trajectory throughout the whole-body motion. In the experiments, the planner developed has shown to reliably plan whole-body motions for collision-free body repositioning as well as for grasping and manipulating objects. For the planning queries of different complexity, the performance of the planner has been thoroughly evaluated. Furthermore, the whole-body motion trajectories generated from the geometric path, obtained from the planner, have proven to be safely executable on both, the simulated and the real robot platform.

1.2 Structure of the Thesis

The reminder of this thesis is structured as follows:

In Chapter 2 we will first discuss the related work. Here, we consider several approaches from the literature that deal with the problem of generating whole-body motions for complex high-DOF robots. Furthermore, techniques addressing the problem of object manipulation are introduced.

In Chapter 3 we cover the theoretical background for the thesis. After describing the general concept of motion planning we will elaborate on probabilistic motion planning, which is the method of our choice. Then, two sampling-based algorithms are described before introducing the constraints typically involved in motion planning for humanoid robot platforms. At the end of the chapter, we treat the topic of manipulability of kinematic structures, which is an important component of our planner, used to evaluate the quality of configurations.

Chapter 4 presents the planner developed in this thesis for whole-body motion and manipulation planning. For simplicity, we will first explain the functionality of the planner in the absence of manipulation constraints, i.e., for planning statically stable collision-free body repositioning motions. Then, the extensions added to the planner in order to allow manipulation of articulated objects are presented.

Chapter 5 presents the experimental setup and results. In this context, the performance of the planner has been evaluated considering different planning scenarios. Furthermore, the feasibility of motion plans generated by our planner has been examined by executing them on the real robot platform.

Finally, Chapter 6 will give an overview of the issues covered in this thesis. In the summary we discuss the positive and negative aspects of the choices made in this work. Additionally an outlook to possible future works and improvements on our planning framework will be given.

2 Related Work

The problem of generating whole-body motions for complex, high-DOF robots can be considered as a relatively new research area in the young field of robotics. Approaches addressing this problem, gathered in the literature so far, can generally be classified into one of the two categories: Jacobian-based methods or randomized, sampling-based motion planning techniques. Although approaches belonging to the former category rather deal with online motion generation, many of their concepts can be inherited for planning motions offline. In the following we will present some of the approaches for each category.

2.1 Jacobian-based approaches

Jacobian-based approaches follow the idea of defining multiple task, eventually of different priority, whose associated task function depends on the current robot configuration. Using generalized inverse kinematics with task specific jacobians, joint velocities that minimize these task functions are iteratively computed. In the following a couple of works belonging to this category are presented.

The work of Kanoun *et al.* in [8] is based on the prioritized kinematic control scheme to plan local motions for a humanoid robot. For any kind of articulated structure a motion of the structures joints is calculated to achieve a certain goal task. Mostly this task is related to a desired position and/or orientation, defined in the workspace, of a body in the articulated structure. Considering a task function and an associated desired goal value, a task jacobian can be calculated in order to compute joint velocities that cause the function to tend towards the desired value. Since humanoid robots are highly redundant systems, the authors define multiple task, represented by linear equality and inequality systems. The novelty of their approach is that these tasks/systems can be organized in an arbitrary order of priority, usually going from the most to least critical. To solve the systems in the specified order of priority, a resolution algorithm is provided. This algorithm iteratively searches solutions for lower priority systems within the set of higher priority systems solutions. The approach has been applied successfully on the humanoid robot platform HRP-2, for the task of reaching a ball underneath an object. Although the authors stated that their algorithm is generally capable of dealing with tasks requiring locomotion, no such applications has been presented at this stage.

In 2010, Kanoun *et al.* [9] presented an extension to their approach from [8]. In this work also locomotion is considered by building a virtual kinematic structure composed

of the robot kinematic model and a number of articulated footsteps. Here, the footsteps are modeled as a chain of rotational and prismatic joints, which allows the application of their numerical inverse kinematics framework. First, they defined a fixed number of footsteps for the virtual chain. In a reaching task those footsteps, being initially folded, are iteratively unfold until the task is completed. The actual motion for the legs, required to follow the sequence of footsteps, is obtained using the dynamic walking pattern generator from [10]. In a second stage they introduced an algorithm and a specific criterion to adapt the number of footsteps progressively. This approach however bears the risk of the virtual chain being trapped in a local minima, an issue generally encountered with numerical optimization methods.

Another contribution to the work of Kanoun is presented by Dang *et al.* in [11]. As before robot motion is resolved in the form of an optimization problem. This time however motion planning is modified to run online. Using visual sensor feedback in the control loop the proposed approach allows to adapt tasks and hence to perform real time replanning.

In [12], Yoshida *et al.* present an approach that couples generalized inverse kinematics with Kajita's dynamic walking pattern generator. The foundation is again a set of prioritized tasks, as described in [8]. Using the associated task jacobians, the generalized inverse kinematics algorithm generates whole-body motions that gradually perform these tasks. Meanwhile during the motion, several criteria such as manipulability and joint limits are monitored. As soon as the monitor detects that the task is unfeasible due to unsatisfied criteria, a support polygon planner is activated. This planner computes a new location for one of the feet in order to extend the robot's reachable space. The stepping motion itself is created by the walking pattern generator, mentioned earlier. A remarkable feature of this work is that the task execution further proceeds taking into account the simultaneously performed stepping motion.

In [13], Mansard *et al.* propose to implement visual servoing to generate full-body motions for a humanoid robot. In their work they present the idea of splitting the control into several sensor-based control tasks that are executed simultaneously by a structure termed *stack of tasks*. This structure can be used for task level control and in combination with the *task sequencing* framework, already introduced in the literature. When the control law is computed, only the tasks currently being in the stack are considered. Tasks are sorted in increasing order of priority and can be removed or added to the stack during execution, while the latter is only admissible if some DOFs remain free after applying the active tasks. With control at the task level the humanoid robot HRP-2 was able to grasp an object while walking. Due to the robustness of visual servoing even grasping a slowly moving object was successfully performed by the robot. Obstacle and self-collision avoidance however has not been taken into account.

2.2 Sampling-based approaches

Motion planning algorithms commonly represent the robot as a point in a n-dimensional *configuration space* $\mathcal{C}$, where n corresponds to the number of degrees of freedom of the robot. The intend of these methods is to search in the configuration space for a collision free sequence of motions, also referred to as a *path*, connecting a given start and goal configuration. Several algorithms, such as the *Retraction method*, *Cell Decomposition* or *Artificial Potential Fields* has been developed, which however require the availability of an explicit representation of the configuration space [14]. In high-dimensional configuration spaces, with robot structures having $n > 6$ DOF's , such a representation does not exist and therefore methods based on the idea of a probabilistic exploration of the search space are adopted. In recent years a number of such randomly exploring methods have been successfully proposed in the literature [15, 16, 17, 7, 18]. Although these methods have proven to be efficient for planning collision free paths in high-dimensional configuration spaces, their application to humanoid robots is still a challenging task. The main issue that prohibits the direct application of classical probabilistic motion planning techniques lies in the fact that humanoid robots are subject to a high number of constraints in addition to collision avoidance.

In 2002, Kuffner *et al.* [19] were the first to apply the idea of probabilistic motion planning to generate whole-body motions for humanoid robots. In their work, they presented an extended variant of the RRT-CONNECT planner, that mainly differs from the classical version in the way in which the configuration space is sampled and how new samples are connected to the search trees by the local planner. Since it is very unlikely that samples, randomly generated from the configuration space, are valid their planner resorts to a pre-computed set of statically stable configurations in the tree expansion process. In a second phase the collision free statically stable path returned by the planner is smoothed and transformed into a dynamically stable trajectory. The planner proposed has shown to produce nice results for both simulated and real humanoid robots. A limitation of their planner is that the location of the support foot (or feet in the case of double-support) is not allowed to change during planning. In the thesis at hand, this work is extended towards manipulation of articulated objects. In manipulation planning additional constraints on the robot's end-effector motion, imposed by the object trajectory, need to be considered.

Another approach presented by Stilman in [20] deals with motion planning under task constraint. These task constraints are defined as a restriction on the freedom of motion of a robot end-effector. As in [19], an extension of the RRT algorithm is used to find a valid motion sequence. In particular, the local planner used in the tree expansion process is enhanced by a method projecting the sampled configurations on the sub-manifold of $\mathcal{C}$ that solves the task. The approach has shown to work well for a mobile manipulator opening a door and drawer. The drawback of this work is that it only considers constraints regarding the relative motion of the robot end-effector with respect to some fixed world frame, which is not sufficient for planning motions for systems

involving a static stability constraint. Another disadvantage is that the proposed method is not capable to deal with multiple constraints.

Berenson *et al.* [21] follow an approach similar to the work of Stilman. In their work, they introduce the Constrained Bidirectional RRT planner, which considers constraints as Task Space Regions (TSR), a concept previously used for goal-specification. To describe more complex constraints, these TSR's can also be linked together thus building TSR Chains. Results of motion planning for the HRP3 humanoid robot, considering stability, manipulation and closure constraints, are presented.

In [22], Vahrenkamp *et al.* follow a multi-level motion planning approach to move a robot to a target object. To bypass the complexity of motion planning for high-dimensional configuration spaces they divide the robot into subsystems like arms or hands and adaptively control the set of joints considered for planning, depending on the current situation. This approach leads to a sequence of motion planning problems of lower dimensionality. The decision about which joints to consider for the respective planning phase is delivered by measuring the distance of the robot to the target object's location. Thus, only the lower body joint are involved in motion planning until the target object enters the robots upper body workspace. In their work results using both, the unidirectional and the bi-directional RRT planner, are presented. With a humanoids upper body mounted on a mobile base, also here equilibrium constraints are not taken into account for motion planning.

In [23], Oriolo and Vendittelli propose a control-based approach to task-constrained motion planning. As opposed to previous approaches their RRT planner makes use of a motion generation scheme that allows continuous constraint satisfaction when moving towards randomly sampled configurations. Given a sequence of task coordinates along a predefined cartesian path the planner produces forward, backwards and self-motions motions traversing the associated task-constrained configuration space leaves $\mathcal{C}_{task}$. Since back and forth motions of the end-effector are not desired, arcs representing backwards motions are reversed before storing them in the tree structure. The tree expansion process proceeds iteratively until either a forwards motion to a configuration in the last, or a backwards motion to a configuration in the first, task-constrained configuration space leave has been found. In this case the algorithm checks whether a optimal path from the start to the goal configuration exists. In the negative case the search continues, otherwise the path is returned. Results are presented for a fixed-base manipulator and a robot with free-flying base. Other constraints such as closure constraints and stability constraints, required in motion planning for humanoids, are not considered at this stage. Generally however, it is possible to consider such constraints by extending the task jacobian used in the motion generation scheme.

In [24], Dalibard *et al.* propose a way of using local jacobian-based methods within randomized motion planning. The core of their planner is based on the RRT-CONNECT algorithm. In a first stage they define a number of tasks with different priorities, where

task are considered equivalent to constraints in this context. Using these tasks a local planner which employs a prioritized pseudo-inverse technique, already introduced in [12], is adopted. This local planner is used for two purposes, namely to generate a number of goal configurations prior planning and to project randomly generated samples onto the constraint manifold in the tree expansion process. Generating more than one goal configuration has the major benefit that multiple goal trees can be grown for the path search. When a solution path, connecting the start tree with one of the goal trees, has been found, an additional optimization of the reached goal posture is performed in order to obtain a more natural whole-body motion. As in [19], this work does not permit the support state (single-leg or double support) to change during motion planning.

3 Background

In this chapter, we will describe the theoretical background used throughout the remainder of the thesis. First, the fundamental concepts and principles of motion planning will be covered. Then, we will elaborate on probabilistic motion planning, the method of our choice, and present some of the most widespread sampling-based algorithms. Afterwards we will discuss various kinds of constraints involved in motion planning for humanoid platforms. Finally, we will conclude by highlighting the topic of manipulability of kinematic structures, an important component of our approach.

3.1 The Concept of Motion Planning

We will first give a definition of the basic motion planning problem. Furthermore, we will introduce the notion of *configuration space* and describe how distances in that space can be measured. At the end, we will outline the correlation between the workspace and the configuration space in terms of obstacles.

3.1.1 Problem Definition

In the last decades several types of robotic platforms, such as mobile robots, fixed and mobile base manipulators, as well as humanoid robots have been developed. While the workspace of these platform may differ, all of them are expected to perform a variety of tasks. In general, tasks can be of different nature. Robots, for example, consisting of a single rigid body, also referred to as mobile robots, are typically required to drive from one location to another. Fixed base manipulators, on the other hand, are usually expected to interact with objects in their accessible workspace. In the case of mobile base manipulators or humanoid platforms, tasks often involve both locomotion and simultaneous manipulation. Here, the fundamental problem is to find a sequence of motions that allow the robot to successfully execute such tasks. A first solution to this problem is provided by trajectory planning methods. These methods have proven to be efficient, but are only applicable under the simplifying assumption that the workspace is empty, i.e., when it does not contain any obstacles. In most real world applications however, the workspace is populated by obstacles and another technique, referred to as *Motion Planning*, is adopted to solve the problem. As opposed to trajectory planning methods, motion planning techniques enable a robot to successfully execute tasks while avoiding collisions with obstacles. Another important issue in motion planning is whether the obstacles are considered to be static or moving. In the former case, a geometric representation of the obstacles can be made available in advance and planning can be

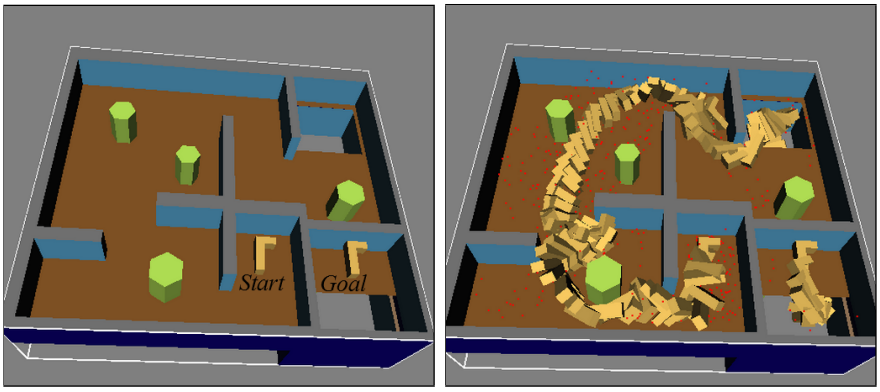

Figure 3.1: The Piano Mover's Problem in $\mathbb{R}^3$. A start and a goal posture in $\mathcal{W}$ (left) and a solution to the motion planning problem (right), [26].

performed *off-line*. In the latter case the robot is required to adjust its motion according to environment changes perceived via its on-board sensors. In this case planning is said to be performed *on-line*. According to the literature [14] the problem of motion planning is then defined as follows.

Let us consider a free-flying robot $\mathcal{B}$ moving in an Euclidean space $\mathcal{W} = \mathbb{R}^N$, with $N = 2$ or 3, also called the *workspace*. The workspace is assumed to be populated by a number of i fixed obstacles, referred to as $\mathcal{O}_1, \ldots, \mathcal{O}_i$. Furthermore, both the geometry of $\mathcal{B}$ and $\mathcal{O}_1, \ldots, \mathcal{O}_i$ as well as the pose of $\mathcal{O}_1, \ldots, \mathcal{O}_i$ in $\mathcal{W}$ are supposed to be known in advance. Given these assumption and a start and goal posture p_s, p_g for $\mathcal{B}$ in $\mathcal{W}$, the problem of motion planning is then formulated as follows: Find a sequence of postures in $\mathcal{W}$, also called a *path*, connecting the two postures p_s and p_g without hitting any obstacles. If a path exists, return the solution found, otherwise report failure. A classical version of this motion planning problem is sometimes referred to as the *Piano Movers Problem* [25], in which movers are asked to transfer a piano in $\mathbb{R}^2$ from an initial to a final posture. In the generalized version of the piano movers problem, as shown in Fig. 3.1, the same task needs to be solved by moving the piano in $\mathbb{R}^3$.

Obviously, many of the assumption made in the classical formulation of the motion planning problem are not valid in practice. First, many robotic platform are not free-flying and are subject to nonholonomic constraints, i.e., the robot is not capable of instantly moving in any direction. Secondly, the geometry and pose of obstacles are not always known in advance and may probably change while the robot is moving, thus making on-line motion planning inevitable. Moreover, manipulation and assembly problems, that can be seen as a controlled collision between the robot and an object, clearly do not conform with the assumptions described above. Although the classical formulation does not capture these problems, it still represents a foundation to solve a variety of more advanced motion planning problems.

While tasks assigned to the robot are usually defined in the workspace either in $\mathbb{R}^2$ or $\mathbb{R}^3$, the problem of motion planning operates in another space, called the *configuration space*, which will be the subject of the next section.

3.1.2 The Configuration Space

In motion planning, a complete description of the geometry of a robot $\mathcal{B}$ and a workspace $\mathcal{W}$ is provided. The workspace is a static environment populated with obstacles. In order to find a collision-free path for $\mathcal{B}$ to move from an initial to a final location, a complete specification of every point on the robot geometry is required. To achieve this, motion planning approaches commonly use a convenient abstraction of the problem, called the configuration space $\mathcal{C}$. The advantage of this representation is that all possible poses of the robot in $\mathcal{W}$ are reduced to single points in $\mathcal{C}$, called *configurations* q. Thus, the problem of motion planning for a robot becomes equivalent to motion planning for a point in $\mathcal{C}$ [27]. When planning is performed in the configuration space it is of course also mandatory to properly map the obstacles $\mathcal{O}$ contained in $\mathcal{W}$ to $\mathcal{C}$. The resulting portion of the configuration space occupied by these obstacles, also called *obstacle region* $\mathcal{C}_{obs}$, is then defined as

$$\mathcal{C}_{obs} = \{q \in \mathcal{C} \mid \mathcal{B}(q) \cap \mathcal{O} \neq \emptyset\}. \tag{3.1}$$

where $\mathcal{B}(q) \subset \mathcal{W}$ denotes the set of points occupied by the robot in the workspace, when being in the configuration q. The remaining set of collision-free configurations is then finally given by $\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obs}$, where $\setminus$ is the subtraction operator used for sets. For a detailed explanation of the relation between workspace obstacles and the configuration space, we want to refer the reader to Sec. 3.1.4. While the workspace is either of dimension 2 or 3, the dimension of the configuration space n is determined by the minimum number of parameters needed to specify the configuration of the robot. In practice, this number usually corresponds to the number of degrees of freedom of the robot system. The simplest example one can think of is presented by a circular mobile robot moving in a two-dimensional Euclidean space, whose configuration can be specified by the location of its center (x, y), expressed w.r.t. some fixed coordinate system (see Fig. 3.2). Then, given the radius of the robot platform r, one can easily deduce the set of points occupied by the robot. Note that the workspace and the configuration space are not identical in this case, although they have the same dimension.

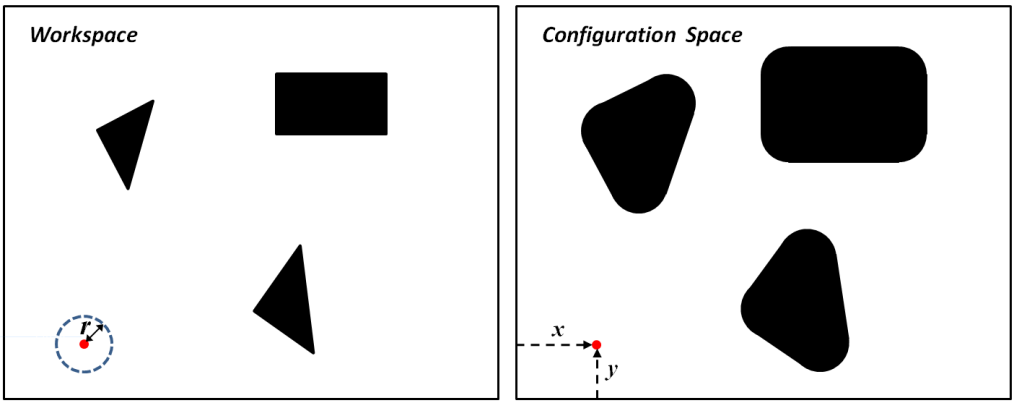

Figure 3.2: Workspace and Configuration Space of a circular mobile robot. Obstacles are mapped into $\mathcal{C}$ following the procedure explained in Sec. 3.1.4.

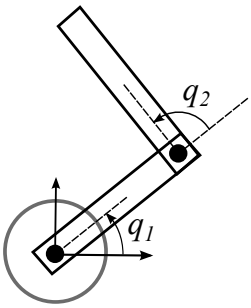

Figure 3.3: Fixed base planar manipulator with two revolute joints (2R).

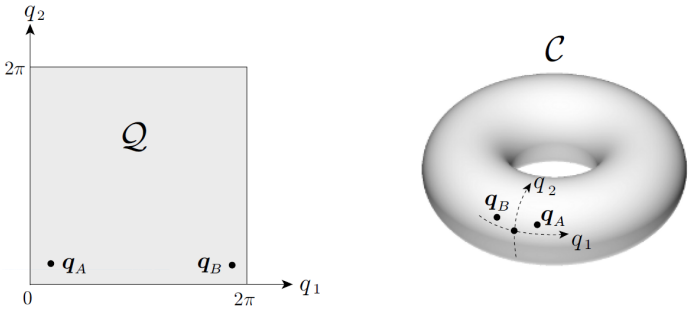

Figure 3.4: Configuration space of a 2R planar manipulator. A locally valid representation as a Euclidean space (left) and the correct representation as a torus shaped space (right), [14].

This fact becomes particularly clear when we consider more complex robotic systems. The configuration vector of a polygonal mobile robot, for example, contains a rotational component θ in addition to a translation, thus making the $\mathcal{C}$-space three-dimensional. Even more complex is the problem of motion planning for fixed and mobile base manipulators, where the configuration space often has a dimension of $n > 5$. The significant difference between the configuration space of a circular mobile robot and mechanical structures, whose configuration vector contain angular coordinates, is that the topology of $\mathcal{C}$ does no longer correspond to an Euclidean space. To illustrate this issue, let us consider a fixed-base planar manipulator with two revolute joints q_1, q_2, as shown in Fig. 3.3, and two configurations q_A and q_B.

When representing the configurations q_A and q_B in an Euclidean space $\mathcal{Q}$ (see Fig. 3.4) it obviously seems as the two configurations are far away from each other. However, examining the corresponding postures of the robot in the workspace $\mathcal{W}$, it turns out that q_A and q_B are actually very similar. This implies, that the representation as an Euclidean space is only valid considering small fractions of $\mathcal{C}$. A topology of the configuration space that correctly reflects the properties of the workspace, is obtained by first bending $\mathcal{Q}$ such that the two horizontal lines at $q_2 = 2\pi$ and $q_2 = 0$ touch each other. The resulting 'tube' then needs to be bent a second time such that the two ends are connected. At the end we obtain a topology in the shape of a torus.

3.1.3 Distance Metric

An important notion, used throughout all motion planning methods, is the *configuration space distance*. The choice of the distance metric adopted, generally depends on the topology of the configuration space $\mathcal{C}$. For simplicity, let us again consider the circular mobile robot, shown in Fig. 3.2, whose configuration is defined by the vector $q = (x, y)$. An approach to describe the distance between two configurations q_A, q_B, is presented by considering the corresponding space in $\mathcal{W}$, occupied by the set of points $\mathcal{B}(q)$ on the robot $\mathcal{B}$. With $p(q_A)$ and $p(q_B)$, being the location in $\mathcal{W}$ of a point p on the robot in configuration q_A and q_B respectively, the distance can be defined as follows

$$d_1(q_A, q_B) = \max_{p \in \mathcal{B}} \|p(q_A) - p(q_B)\|, \tag{3.2}$$

where $\|\cdot\|$ denotes the Euclidean distance, for our example in $\mathbb{R}^2$. Concisely said, Eq. (3.2) defines the distance w.r.t. two points on the robot geometry that maximize the displacement in the workspace.

In practical motion planning algorithms however, the computational expense to find these points is not acceptable and the simple Euclidean norm, defined as

$$d_2(q_A, q_B) = \|q_A - q_B\|, \tag{3.3}$$

is used instead. Recalling the discussion of the configuration space in the previous section it reveals that the distance measures d_1 and d_2 are only appropriate as long as $\mathcal{C}$ is euclidian or rather small fractions of $\mathcal{C}$ are considered.

For the 2R planar manipulator, such as the one shown in Fig. 3.3, the distance measures do not correctly reflect the distance between two configurations on the torus. When the configurations q_A and q_B are represented by a vector (q_1, q_2) of angular coordinates, the following additional computations need to be performed

$$min\_diff\,[i] = \min(|q_A[i] - q_B[i]|, 2\pi - |q_A[i] - q_B[i]|), \quad \text{for } i = 1, 2 \tag{3.4}$$

$$d_3(q_A, q_B) = \sqrt{(min\_diff\,[1])^2 + (min\_diff\,[2])^2}. \tag{3.5}$$

where $q_A[i]$ and $q_B[i]$ denote the value of the i-th angular component in configuration q_A and q_B, respectively. Note that, besides being a correct measure for the distance between two points on the torus, d_3 only involves simple math operations. This issue becomes particularly important when considering motion planning algorithms regarding their runtime.

3.1.4 C-Space Obstacles

So far, we have only discussed the representation of the robot $\mathcal{B}$ in the configuration space. When searching in $\mathcal{C}$ for a collision-free path, the obstacles of the workspace $\mathcal{O}$ obviously need to be represented in the same space as well. Here, the mapping of obstacles again strongly depends on the configuration space topology. For the sake of convenience, let us first consider $\mathcal{C}$ to be an Euclidean space, either $\mathbb{R}^2$ or $\mathbb{R}^3$. In this case a representation of the obstacles in $\mathcal{C}$ can be obtained by circling the obstacles in $\mathcal{W}$ with the point on the robot geometry, previously selected to represent the robot in the configuration space. In Fig. 3.5, two examples of this mapping are illustrated.

As one can easily see, the shape of workspace obstacles simply grow when they are mapped into $\mathcal{C}$. When considering non-Euclidian spaces, as encountered with the 2R planar manipulator, the same mapping becomes slightly more difficult. In this case, a graphical representation of $\mathcal{C}$-space obstacles can be generated by sampling the configuration space with a proper resolution and applying collision checking (see Sec. 3.4.2) for each of these configurations. In comparison to the mapping of obstacles in Euclidean space this operation obviously comes along with a much higher computational effort. Fig. 3.6 shows the result of the $\mathcal{C}$-space obstacles building procedure for the 2R planar manipulator in two different cases. Note that the configuration spaces shown are only illustrated as squares for the purpose of a better understanding.

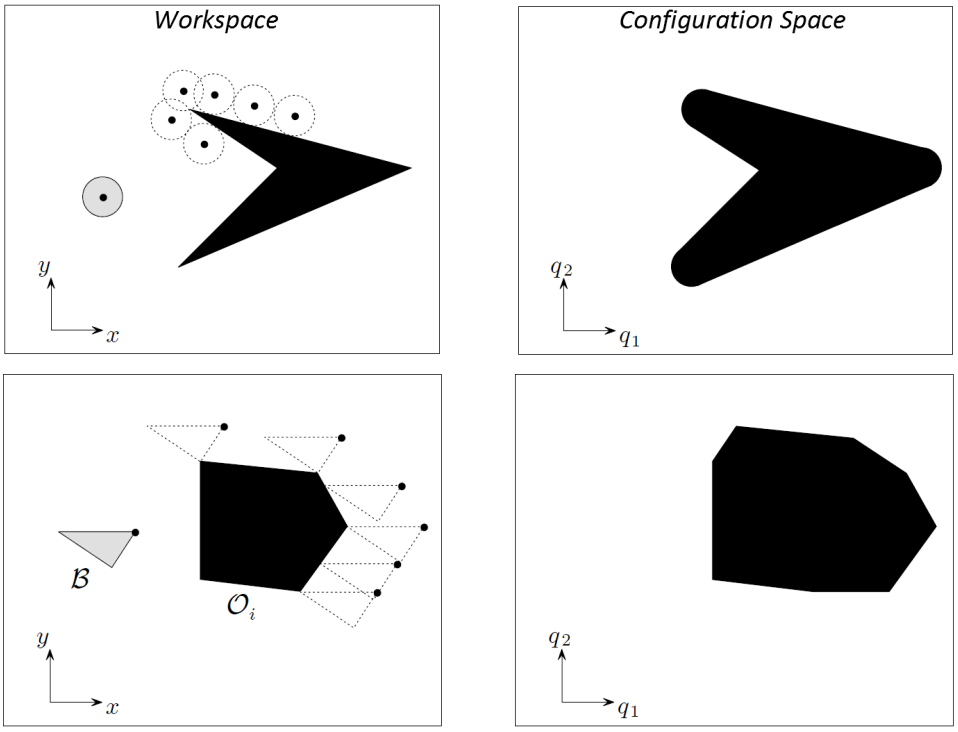

Figure 3.5: Examples of $\mathcal{C}$-space obstacles. Mapping of a workspace obstacle (left column) into the configuration space (right column) for a circular robot (top) and a polygonal robot moving translational in $\mathcal{W}$ (bottom), [14].

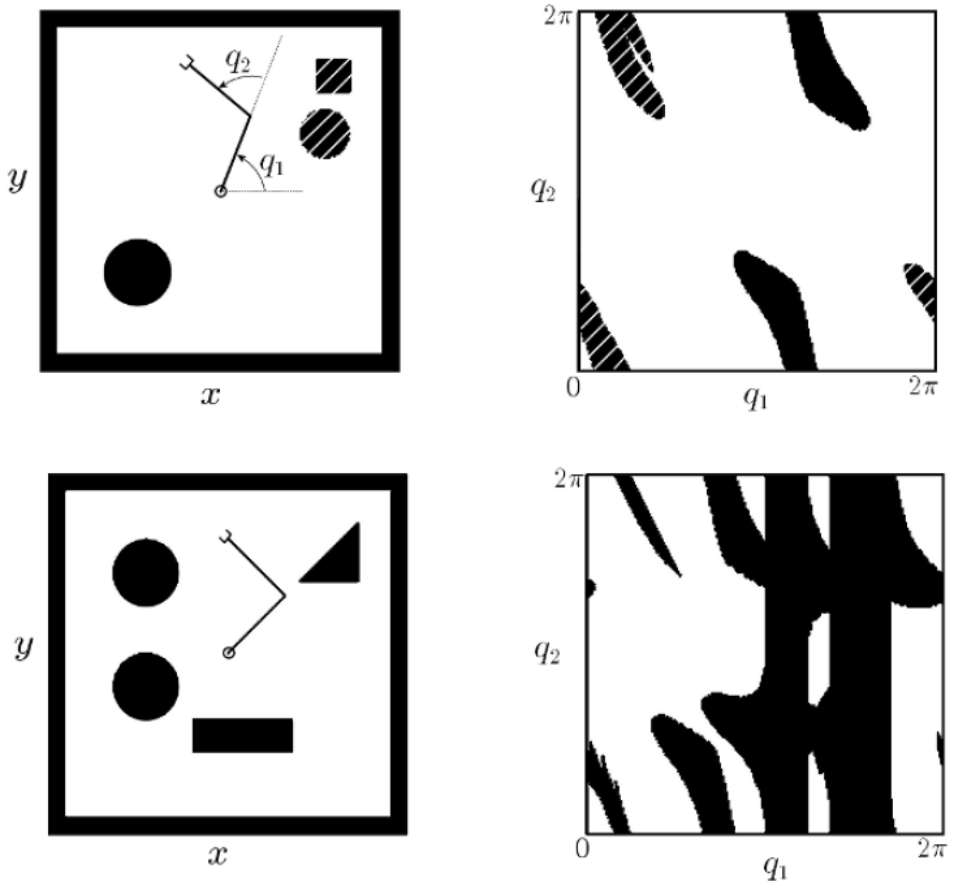

Figure 3.6: Examples of $\mathcal{C}$-space obstacles for a 2R planar manipulator. The robot and the obstacles in the workspace (left column) and the corresponding $\mathcal{C}$-space obstacles in the configuration space build by generating samples and applying collision checks (right column), [14].

The correct visualization of the $\mathcal{C}$-space as a two-dimensional torus would require to apply the folding procedure described in Sec. 3.1.2. Examining Fig. 3.6, one can also see that the assignment of configuration space obstacles to the respective workspace obstacles becomes everything else than straightforward.

Many motion planning methods from the literature, such as the *Cell Decomposition* [16] and *Planning via Retraction* [28] method, rely on the availability of an explicit representation of the $\mathcal{C}$-space obstacles. In high-dimensional configuration spaces however, the computational expense required for the building procedure typically becomes intractable and other techniques such as *Probabilistic Motion Planning* are employed. As opposed to the previously mentioned methods, probabilistic motion planning methods are capable of finding collision-free paths without prior computation of such a representation.

3.2 Probabilistic Motion Planning

Probabilistic approaches to motion planning follow the idea of randomly exploring the configuration space in order to find a collision-free path. In probabilistic algorithms, configurations are sampled from $\mathcal{C}$ using a specific sampling scheme and connected by a local planner. In the following section, a brief overview of the basic components of a probabilistic planner will be given. Moreover, we will present the available practices to perform path search and path smoothing. At the end, the section concludes by introducing the notion of completeness.

3.2.1 Sampling Strategies

Over the years, several strategies for sampling the configuration space have been developed. Depending on the complexity of the motion planning problem, the choice of the sampling strategy can have a significant impact on the performance of a planner.

The simplest strategy, suitable for many problems, is to sample configurations randomly from a uniform distribution. Although this strategy is easy to implement and works well for high-dimensional configuration spaces, it has the disadvantage that, in difficult planning problems, the running time of a planner might vary across different runs [15]. Another drawback of the strategy reveals when considering planning problems in which the robot is required to traverse narrow passages in the configuration spaces in order to reach a desired final destination. In this scenario, the probability of sampling collision-free configurations within the narrow passage is very small and thus the planner is forced to invest a lot of work, leading to a poor overall performance.

This problem motivates another kind of sampling strategy, referred to as *obstacle-based sampling* [29]. Here, the idea is to generate samples in the vicinity of obstacle boundaries, thus allowing to discover collision-free paths through narrow passages. Initially the strategy proceeds by sampling a number of random configurations using a uniform distribution. Then each of these configurations is checked for collision. Instead of simply discarding those configurations q_c found to be in collision, the sampler generates stepwise new configurations from q_c in an arbitrary direction until a collision-free configuration q_{free} has been found. Given q_c and q_{free}, it finally performs a binary search to find the closest configuration q to the contour of the obstacle contained in $\mathcal{C}_{free}$. At the end of this procedure, q is added to the data structure maintained by the planner, while q_c and q_{free} are discarded.

Another approach to address the narrow passage problem is to sample the configuration space by a Gaussian distribution that is biased towards the contour of obstacles [30]. The procedure applied by the so called Gaussian sampler is relatively simple. First, a configuration q_1 is randomly sampled from a Gaussian distribution. Then a second configuration q_2 at a distance d from q_1 is generated. Here, the distance d is chosen randomly from a uniform distribution. If both configurations are detected to be in collision

or collision-free, both of them are discarded. Otherwise, if one sample is collision-free and the other one is in collision, only the collision-free sample is added to the data structure maintained by the planner.

An alternative sampling strategy that tries to keep the number of samples a planner needs to maintain as low as possible is presented by the *visibility-based sampling* method [31]. In this context the visibility of a configuration q is defined with respect to the number of previously sampled configurations that the local planner is capable to connect to it. In order to be added to the list of configurations a planner maintains, a configuration needs to satisfy one of the following criteria. Either none of the existing configurations in the list can be connected to q, in which case q is considered a new component, or q allows to connect at least two elements from the list. By following these rules, visibility-based sampling provides a way for exploring the configuration space while keeping the storage requirements low.

Manipulability-based sampling is a further sampling strategy that considers the manipulability measure associated with a manipulator for generating samples in $\mathcal{C}$. Generally, the manipulability indicates the freedom of motion of a manipulator in a given configuration. In configurations, for example, with a high manipulability measure the kinematic structure exhibits a high dexterity. Low values for the manipulability, on the other hand, imply to be close to a singular configuration and thus allow only a limited range of admissible motions for the manipulator. Considering this aspect in sampling, it is naturally to sample those regions of $\mathcal{C}$ with a low manipulability more densely than areas where manipulability is high. For doing so, the manipulability-based sampler builds a cumulative density function (CDF) for the manipulability measure. Then, samples are generated from the configuration space and rejected with a probability proportional to the associated CDF value of their manipulability value [15].

3.2.2 Local Planner

The *Local Planner* of a probabilistic algorithm is the component in charge of connecting points in the configuration space by a collision-free path. To do so it refers to the list of configurations already sampled from $\mathcal{C}$ and maintained by the planning algorithm. Then, given a randomly sampled configuration q_{rand}, it first searches for the closest configurations to q_{rand} in that list, adopting one of the distance metrics introduced in Sec. 3.1.3. In the following, this configuration is referred to as the *nearest neighbor* q_{near}. The simplest strategy for connecting the two configurations is based on generating a discrete set of intermediate states, equally spaced along the straight line segment joining q_{near} and q_{rand} in $\mathcal{C}$. Note that while probabilistic roadmap methods typically consider the whole segment, other methods only generate intermediate states towards q_{rand} until a certain distance ε from q_{near} is reached. Once the intermediate states have been created, the local planner checks whether they are collision free (see Sec. 3.4.2). If yes, both the last configuration along the line and the edge connecting it to q_{near} are added to the data structure maintained by the algorithm. In Fig. 3.7 an example for the simple connection

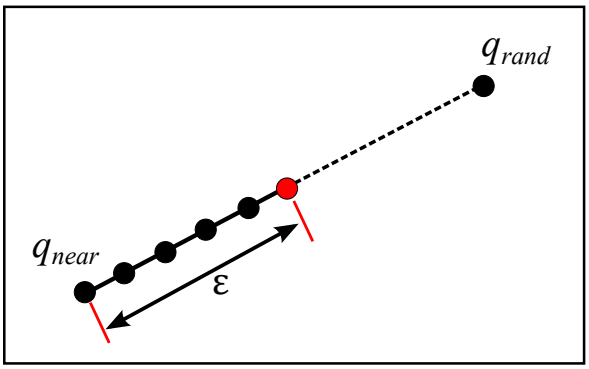

Figure 3.7: Local Planner. Connecting configurations by a discrete set of intermediate states along the straight line segment joining q_{near} and q_{rand}.

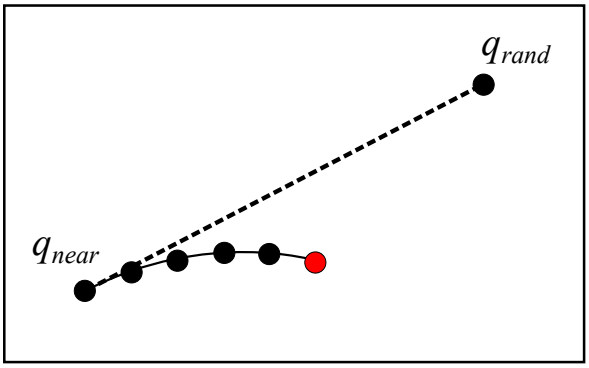

Figure 3.8: Local Planner. Connecting configurations by incrementally generating control inputs that move the robot from q_{near} towards q_{rand}.

strategy is illustrated. Although the simple connection strategy represents an easy and efficient way for connecting configurations, it often does not comply with constraints imposed by robotic systems or tasks assigned. When dealing with systems subject to nonholonomic or task constraints, another more advanced connection strategy needs to be adopted. In the literature, kinematic or task constraints are typically formulated in the context of control laws. Then, given a configuration q_{near}, these control laws produce control inputs for the robotic system that generate an incremental motion from q_{near} towards q_{rand} while keeping the respective constraints satisfied. As before, collision checks are performed as the point moves in $\mathcal{C}$. Finally, the integration stops when either q_{rand} has been reached, an invalid state has been encountered or a certain preset time has expired. The resulting motion in $\mathcal{C}$, shown in Fig. 3.8, usually deviates from the straight line segment between q_{near} and q_{rand}. As with the simple connection strategy, the final configuration reached and the edge from q_{near} is stored in the data structure maintained by the planner.

3.2.3 Path Search

In probabilistic motion planning we differentiate between single-query and multi-query path search algorithms. Multi-query algorithms, like the *Probabilistic Roadmap Planner* [18], construct a graph by first finding a certain number of collision-free configurations in $\mathcal{C}$. After that, each of them is connected to the configurations in its neighborhood. For all subsequent queries, the start and goal configuration simply need to be connected to the nearest configuration of the graph. Then, a solution is found by traveling from the start to the goal configuration along the edges of the roadmap while considering a specific heuristic to decide which road to follow. This heuristic assigns a cost to each configuration of the graph that usually corresponds to the distance to the goal configuration.

Single-query algorithms, like the *Rapidly Exploring Random Trees (RRT) Planner*, search for a collision-free path by growing trees in the configuration space. Here, the tree growing procedure is repeated for each new query, for which reason these methods are referred to as single-query methods. Approaches belonging to this kind of methods can be additionally divided into unidirectional and bidirectional path search algorithm. In unidirectional search, a single tree is grown from the start configuration until one of the tree branches can be connected to the goal configuration. In bidirectional search, two trees, rooted at the start and the goal configuration respectively, are maintained. When performing a search both trees are grown in the configuration space until two branches of trees are capable of joining each other. In order to accelerate the search, trees are often instructed to grow towards each other rather than exploring the configuration space completely at random.

3.2.4 Path Smoothing

Paths generated by probabilistic motion planning algorithms can be quite ugly and unnecessarily long [32]. Therefore, smoothing techniques are often applied to the raw solution path as a post-processing step. Given a path P, i.e., the output of a probabilistic planner, an improved result can be obtained by the following procedure.

Perform a certain number of iterations. At each iteration, pick two random configurations q_1 and q_2 from P and try to connect them by using the local planner. If the new path segment between q_1 and q_2 is valid, i.e., it does not violate any constraints and is shorter than the original path segment in P, the original path segment is replaced by the new one. By repeatedly performing this operation the raw solution path becomes smoother and smoother. The smoothness of the final solution path obtained obviously depends on the maximum number of iterations specified. Generally, path optimization can be also carried out during the planning phase. In this way, the planner directly returns a nice path. Doing so however, leads to an increase of the planning time. Fig. 3.9 shows an example of the smoothed path P_{new} obtained from the raw solution path P after performing a certain number of iterations.

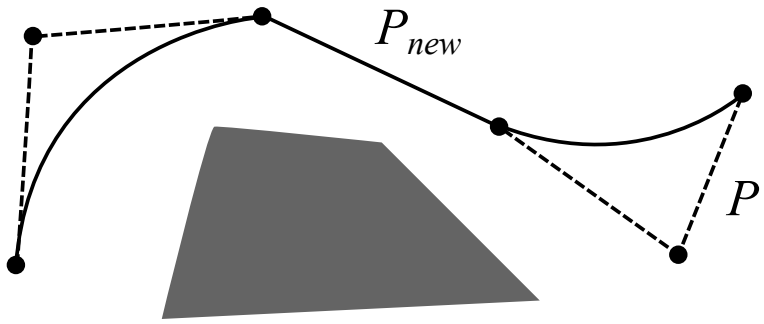

Figure 3.9: Example of the smoothing operation applied to the raw solution path (dashed line) obtained from the probabilistic planner.

Here, the original path P is illustrated as a dashed line. The course of the smoothed path P_{new} after a certain number of iterations is shown as a solid line. The grey surface represents a configuration space obstacle.

3.2.5 Notion of Completeness

In motion planning, a planner is said to be *complete* if it is capable of producing a solution within finite time or otherwise correctly reports that there is none. Often the notion of completeness is defined with respect to some specific properties of the motion planning algorithm. For grid-based search algorithms, like the *Approximate Cell Decomposition* for example, the weaker notion of *resolution completeness* is used. This notion indicates that the algorithm will find a solution to the motion planning problem, given that the resolution of the underlying grid is chosen fine enough. In probabilistic motion planning an even weaker notion, called *probabilistic completeness* is adopted. By definition, this notion says that, given a problem that is solvable in the open free configuration space, the probability that the planner solves the problem approaches 1 as the running time of the algorithm goes to infinity [33]. According to these definitions. one should keep in mind that it is not possible to conclude that there is no solution to the problem, when probabilistic or resolution complete algorithms fail to find a path within a preset time or resolution. On the other hand, complete algorithms are usually considerably more expensive than probabilistic or resolution-complete methods, from a computational point of view.

3.3 Sampling-based Algorithms

Over the years, several probabilistic motion planning algorithms have been proposed in the literature. For each of these algorithms, a variety of variants exist, that mainly differ in the sampling and connection strategy adopted. Comparing the pros and cons of these algorithms would certainly go beyond the scope of this work. For this reason we will focus in the following on two algorithms, the *Probabilistic Roadmap Planner* and the *Rapidly Exploring Random Trees (RRT) Planner*, as representatives of multi-query and single-query planning algorithms.

3.3.1 Probabilistic Roadmap Planner

Creating a path in the configuration space using the Probabilistic Roadmap Planner is considered as a two phase process: *planning*, and *query* [34]. In the planning phase the planner keeps on sampling the configuration space until a certain number of collision-free configurations has been generated. Each sampled configuration, or simply said *point* in $\mathcal{C}$, is connected to the neighbors lying within an area of predefined size. Here, the connection between two points is established by a straight line path, that is only maintained by the planner if it is found to be collision-free. The resulting network, called a *roadmap*, stored by the planner is then used to solve all subsequent motion planning queries. A summary of the steps required for the roadmap construction is presented by Alg. 1 in pseudo-code, where S and R denote the set of configurations sampled from $\mathcal{C}$ and the roadmap constructed, respectively [18].

Note that this operation needs to be performed only once, given that the planning environment is static. An example of a roadmap build in $\mathcal{C}$ is illustrated in Fig. 3.10.

Algorithm 1: Roadmap Construction

```
1  S ← ∅ ;
2  R ← ∅ ;
3  repeat
4      q ← random configuration from C ;
5      if q is collision-free then
6          begin
7              add q to S ;
8              choose subset S_q of S with candidate neighbors for q ;
9              forall q' in S_q do
10                 begin
11                     if the local planner can connect q with q' then
12                         add connection (q, q') to R ;
13                     end
14                 end
15             end
16         end
17     end
18 until maximum number of samples in S is reached;
```

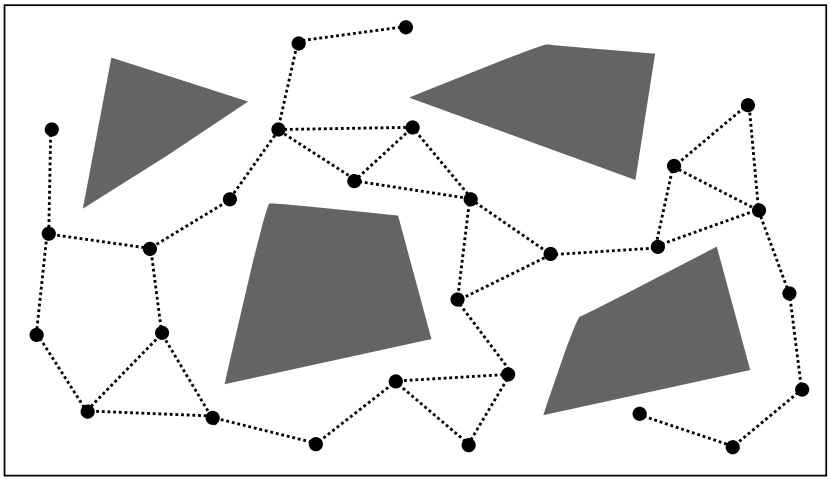

Figure 3.10: Probabilistic Roadmap Planner. Roadmap construction.

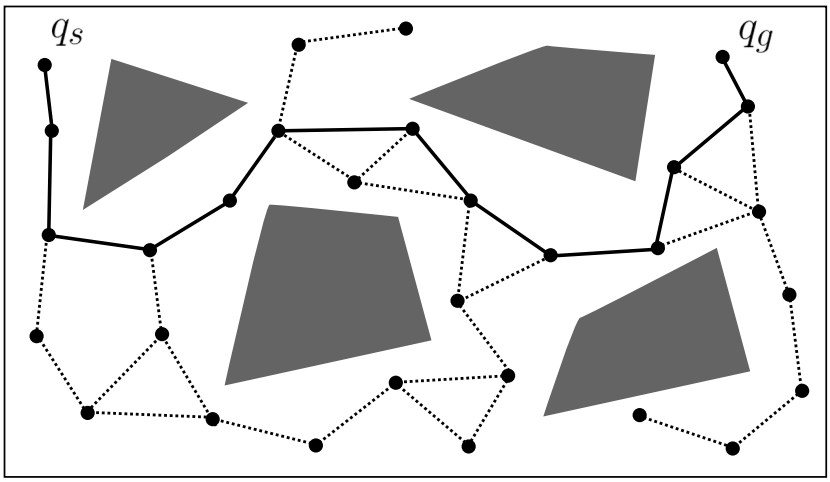

Figure 3.11: Probabilistic Roadmap Planner. Solution to a motion planning query (solid path).

A motion planning query is specified by a start and goal configuration, q_s and q_g. In order to solve a query, both configurations are connected in a first step to the respective closest configuration of the roadmap. Then, starting from q_s a solution path is found by moving iteratively to the neighboring node providing the minimum distance to the goal configuration. The distance metric used, usually corresponds to the Euclidean norm. The solution for a given start and goal configuration is shown in Fig. 3.11 as a thick, solid path.

Here, the roadmap shown in Fig. 3.11 consists of a single connected component. This means, that one can travel from any point to any other point by following the roads of the graph. Sometimes however, and especially when $\mathcal{C}$ contains narrow passages, the

roadmap has disconnected components. Sampling the configuration space extensively might solve this problem but would lead to a much higher computational expense. In particular, this effort is exaggerated when only rather simple motion planning problems are to be considered in the query phase. Moreover, in non-static planning environments, the roadmap needs to be rebuilt every time the environment changes. In this case it is often more efficient to refer to single-query motion planning methods.

3.3.2 Rapidly Exploring Random Trees Planner

A RRT Planner is a single-query probabilistic motion planning algorithm that searches for a path by incrementally expanding trees $\mathcal{T}$ in the configuration space. From an algorithmic point of view, trees are represented by a data structure, also referred to as a *Rapidly Exploring Random Tree*. As opposed to the PRM planner, this kind of algorithm only explores a subset of the configuration space, relevant for solving the query in mind [14]. Furthermore, the tree growing procedure is repeated for each new query, for which reason this approach is referred to as a single-query method. The routine to be repeatedly applied for growing the trees in $\mathcal{C}$ proceeds as follows.

At the beginning of each iteration a random configuration q_{rand} is sampled from $\mathcal{C}$ according to a uniform probability distribution. Then, the closest configuration to q_{rand} in $\mathcal{T}$, also called the nearest neighbor q_{near}, is found and the local planner generates a new configuration q_{new} at a distance ε from q_{near} along the segment connecting q_{near} and q_{rand}. Afterwards, both q_{new} and the segment joining it to q_{near} are checked for collisions. If they are found to be collision-free, the search tree $\mathcal{T}$ is expanded by the new configuration and that segment. Note that no collision check needs to be performed for q_{rand}, since it is only used to indicate a direction for the tree expansion and is rejected by the planner at the end of each iteration anyway.

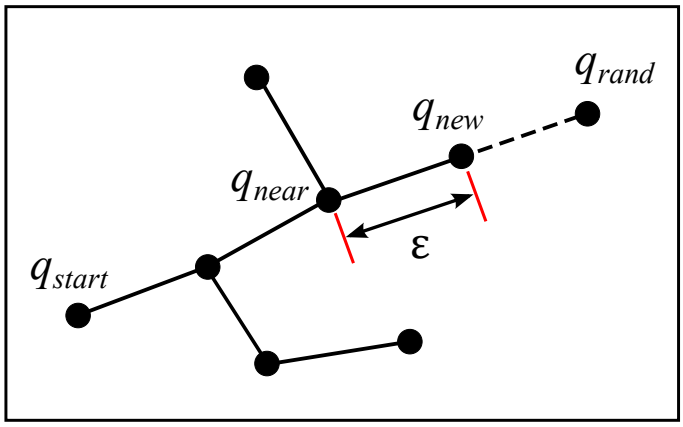

Figure 3.12: Example of a tree expansion step.

The simplest form of a RRT Planner grows a single tree rooted at the start configuration q_{start}. Then, during the search the tree tries occasionally to connect to the goal configuration by using it as q_{rand} in the tree expansion procedure. Often however, a faster and

more efficient variant, called the *bidirectional RRT*, is used instead. With this variant, two trees, rooted at the start and goal configuration respectively, are grown. Here, the latest configuration added to a tree $\mathcal{T}_a$ is used as q_{rand} for the expansion of the other tree $\mathcal{T}_b$ in order to minimize the effort required for the search.

A further, greedier version of the RRT Planner, is presented by the *RRT-CONNECT* algorithm [7]. Note that this planner will serve as a foundation for the whole-body motion planner developed in this thesis. Furthermore, the planner will be extended towards manipulation of articulated objects. The pseudo-code of the generic RRT-CONNECT algorithm is described in Alg. 2.

After initialization (Line 1), two search trees $\mathcal{T}_a$ and $\mathcal{T}_b$ are grown from q_{start} and q_{goal}. At each iteration i the function RAND_CONFIG returns a random configuration sampled from $\mathcal{C}$. Afterwards, the tree denoted by $\mathcal{T}_a$ is expanded by a new configuration q_{new} and path segment, as depicted in Fig. 3.12. Here, the outcome of the EXTEND function (Line 4) can be of three different kinds: *REACHED*, *ADVANCED* or *TRAPPED*. *TRAPPED* is returned, when either q_{new} is found to be in collision or the local planner failed to generate a valid path segment between q_{near} and q_{new}. *ADVANCED* is the return value obtained when q_{new} and the path segment are both valid. For obtaining *REACHED*, the same conditions as with *ADVANCED* need to be satisfied. Additionally *REACHED* indicates that q_{rand} has been chosen as q_{new}, which is the case when the distance between q_{near} and q_{rand} is below ε. The greedy element of the algorithm is introduced by the CONNECT function (Line 5 of Alg. 2), described in Alg. 3. As opposed to the classical bidirectional RRT, the second tree $\mathcal{T}_b$ is expanded multiple times in the direction of the latest configuration added to tree $\mathcal{T}_a$. As indicated by Line 3 of Alg. 2, the expansion of $\mathcal{T}_b$ proceeds until the configuration of $\mathcal{T}_a$ is reached or an invalid configuration has been encountered. If CONNECT returns *REACHED* a solution has been found and the resulting path is returned by the planner. Otherwise, the two trees are swaped (Line 12 of Alg. 2), i.e the tree denoted by $\mathcal{T}_a$ becomes $\mathcal{T}_b$ and vice versa. These steps are iteratively repeated until a solution has been found or a predefined maximum number of iterations I has been reached, in which case the planner reports failure.

Algorithm 2: Generic RRT-Connect(q_{start}, q_{goal})

```
1  T_a.init(q_start); T_b.init(q_goal) ;
2  for i = 1 to I do
3  |   q_rand ← RAND_CONFIG() ;
4  |   if not (EXTEND(T_a, q_rand) = TRAPPED) then
5  |   |   if (CONNECT(T_b, q_new) = REACHED) then
6  |   |   |   return PATH(T_a, T_b);
7  |   |   end
8  |   end
9  |   SWAP(T_a, T_b) ;
10 end
11 return FAILURE
```

Algorithm 3: CONNECT($\mathcal{T}$, q)

```
repeat
    S ← EXTEND(T, q) ;
until not (S = ADVANCED) ;
return S ;
```

3.4 Motion Constraints

The goal of motion planning is to find a path, i.e., a sequence of configurations, that drives a robot's end-effector from a start to a goal pose in the workspace. So far, a solution has been considered feasible if the waypoints and local paths between them are collision-free. In practice however, several other constraints need to be taken into account for planning. In the following, the basic constraints involved in motion planning for humanoid robot platforms will be presented.

3.4.1 Actuator Limitations

Robots are mechanical systems consisting of serial and/or parallel kinematic chains, whose links are actuated by a sequence of motors, also called joints. Motors or joints are generally subject to geometric and differential constraints, typically specified in the robot data sheet or a robot description file made available by the robotic platform provider. Geometric constraints disclose a range of admissible values a joint can take. Accounting for these kinds of constraints in motion planning is a relatively simple issue. Given a list of joint limit pairs, probabilistic algorithms often randomly sample joint values within the bounds according to a uniform distribution. Differential constraints, on the other hand, specify the maximum velocities and accelerations the robot's actuators are capable to execute. While motion planning can be performed considering only geometric constraints, also referred to as *geometric planning*, differential constraints are essential when it comes to motion execution. Geometric motion planners proceed by first searching the configuration space for a collision-free path. Afterwards the geometric path is transformed into a valid joint trajectory, taking into account the differential constraints. Another class of planners use control laws to generate a sequence of collision-free configurations in $\mathcal{C}$. In this case differential constraints are already involved in the planning phase.

3.4.2 Collision Avoidance

Collision detection is a fundamental element of the sampling and validate scheme applied by probabilistic motion planning algorithms. From a theoretical point of view, the problem of collision detection can be considered as a separate field of research. Due to its complexity, the choice of the collision detection algorithm applied in motion planning significantly influences the overall performance of a planner. A first approach to figure out whether a certain configuration is in collision, is presented by performing an interference check between the robot and obstacle geometry. In practice, this is done by finding the minimum distance between the geometries. If the distance is negative, the current configuration is found to be in collision. Often however, robots are constituted by complex geometries and finding the minimum distance becomes a computational expensive operation. Therefore, the robot geometry is usually approximated by simple geometric shapes, such as cylinders and spheres, in order to speed up the computation. Additionally these shapes are enlarged to obtain a so called safety margin. In Fig. 3.13, the approximation of the HRP-2 robot model used for collision checking is illustrated.

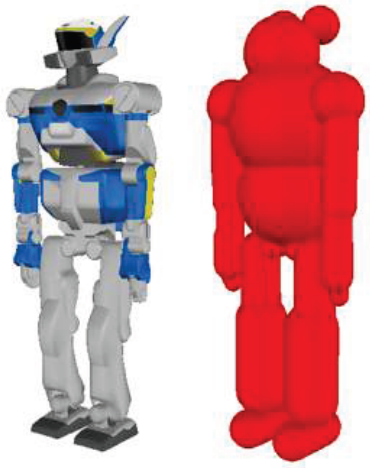

Figure 3.13: Original model of the HRP-2 robot (left) and approximation model for collision detection and distance computation (right), [35].

Another approach for collision detection is to consider the area or volume swept by the robot by moving through a sequence of configurations. Here, as opposed to the previous approach, an entire path can be checked for collisions at once. As before, an interference check is performed, this time considering the robot and obstacle volumes. On the other hand, the computation of these volumes can be very expensive. Note that with some robotic platforms, especially those containing parallel kinematic chains, one needs to check additionally for self-collisions, i.e., when different parts of the robot come into contact. An example for the swept volume technique, applied to the DLR's humanoid robot *Justin*, is shown in Fig. 3.14.

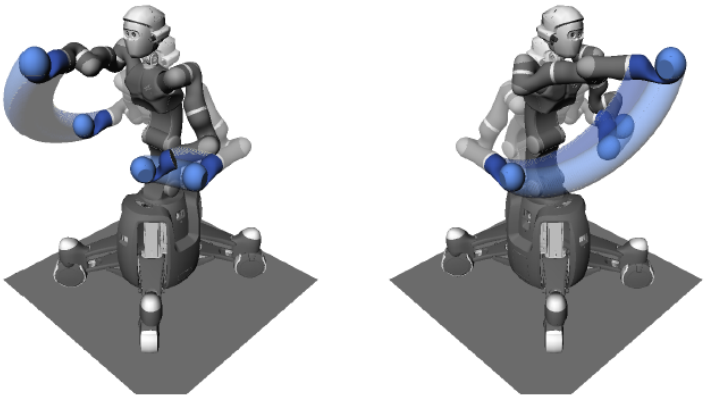

Figure 3.14: Right and left side view of swept volumes generated by a motion from one to another configuration, [36].

In practical motion planning applications, the collision detection algorithm used is often considered a black box. In the literature, several collision detection libraries, such as *FCL*, *PQP* and *ODE* [37, 38, 39] are presented and provided to the motion planner developer.

3.4.3 Closure Constraint

In the sampling schemes presented in Sec. 3.2.1, joint values were assumed to be independent from each other. This assumption however is no longer valid when considering kinematic structures whose links are arranged to form loops. In this case, some of the joints angles must be chosen such that the loop remains closed. In the case of a humanoid robot, for example, closed kinematic chains occur when an object is manipulated using both hands or the legs of the robot are assumed to remain fixed on the ground during a motion. A general representation of a closed kinematic chain is shown in Fig. 3.15, where two fixed base manipulators with a common joint (red disc) form a closed loop together with the ground plane.

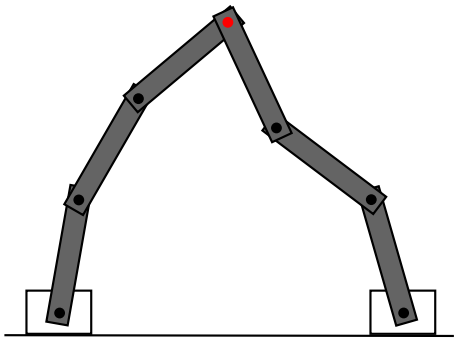

Figure 3.15: Two fixed base manipulators with a common joint (red disc) build a closed kinematic chain.

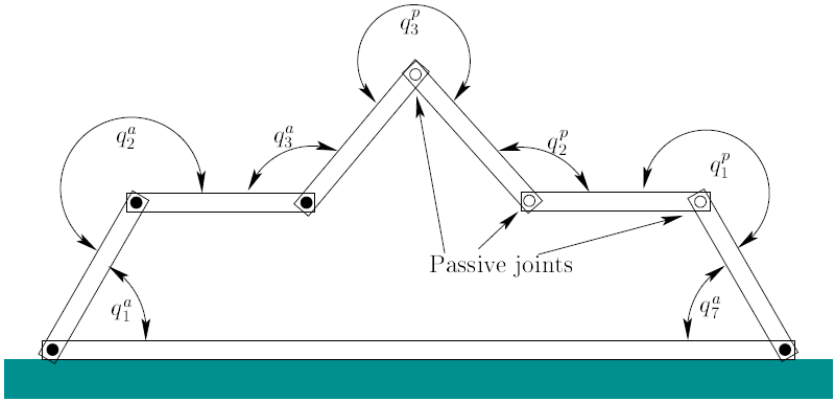

Figure 3.16: Example of a link decomposition into active and passive vectors of joint variables, [17].

In order to account for the closure constraint, the sampling procedure of motion planning algorithms need to be adapted. A first approach is presented by the *Active-Passive Link Decomposition* method [40]. With this approach, the configuration vector q is split into a vector of *active* joint variables q^a and a vector of so called *passive* joint variables q^p. A possible decomposition for a robot constituted by seven revolute joints is shown in Fig. 3.16. With this choice, the kinematic structure has four degrees of freedom. Then, the method proceeds by randomly sampling values for the active joint variables q^a. Afterwards the remaining passive joint variable q^p are determined by solving the inverse kinematics (IK) problem in order to satisfy the closure constraint. Note that while for some choices of q^a multiple IK solutions for q^p exist, most of the sampled configurations q^a will not allow to find a solution to the IK problem.

Another method, called the *Random Loop Generator* [41], improves the constraint satisfaction success rate by iteratively choosing values for the variables in q^a that allow an inverse kinematic solution for the passive variables in q^p. A prerequisite for this approach is that the chosen active joints appear sequentially along the kinematic chain, i.e., they are not interrupted by a passive joint. In a first step an interval I_1 of admissible values for the first active joint q_1^a is computed. According to some geometrical analysis, it is known that for all values q_1^a outside this interval, no IK solution for the passive chain exists. In practice, this analysis can be performed by checking whether the passive chain can reach the accessible workspace spanned by the active chain, given a value for q_1^a. Once a value for q_1^a has been sampled from I_1, an interval I_2 of allowable joint values for q_2^a is computed. As before, a value of q_2^a is sampled from I_2 and an interval for the subsequent active joint is determined. This procedure is repeated until a value has been assigned to each active joint or the generator has detected that no admissible assignment is left for one of the joints. If the set of admissible values I_i for an active joint q_i^a is found to be empty, all previous assignments are discarded and the whole procedure is repeated. Otherwise, the vector q^a is used to find values for the passive variables in q^p through inverse kinematics.

3.4.4 Stability Constraint

When planning is performed for humanoid robots or mobile manipulators, it is important to ensure that the motions generated do not cause the robot to fall over. From a geometrical point of view, the stability constraint implies that the projection of the Center of Mass (CoM) of the robotic structure is required to stay inside the support polygon throughout the motion. This formulation of the constraint is also referred to as *static stability*. An extension of this concept is presented by considering *dynamic stability*, where the dynamics of a system are taken into account to maintain equilibrium [42]. Motion planning algorithms however, often aim at generating a statically stable path in the configuration space. Afterwards, this path can be transformed into a dynamically stable path in a post-processing step. Recalling the sampling procedures from Sec. 3.3, we know that each new configuration needs to be checked for validity before adding it to the data structure maintained by the planner. In order to determine validity of a configuration concerning the stability constraint, the first task is to find the location of the CoM. For simplicity, let us consider the serial chain with four links joined by revolute joints, as depicted in Fig. 3.17.

For each link of the kinematic chain, the length l_i and the mass m_i are assumed to be given. Furthermore, it is supposed that the robot description provides the location of the center of mass $c_i = [c_{ix}, c_{iy}, c_{iz}]^T$ for each link, expressed in the respective reference frame. From the geometric description of the chain bodies, we can derive the homogeneous transformation matrices T_i, defined as

$$T_i = \begin{bmatrix} R_i & d_i \\ 0 & 1 \end{bmatrix}, \tag{3.6}$$

where R_i and d_i denotes the 3×3 rotation matrix and the translation vector of the i-th link frame with respect to the preceding link frame, respectively. The total mass of the system is simply obtained by summing up the individual masses of the links $M = m_1 + m_2 + m_3 + ... + m_i$.

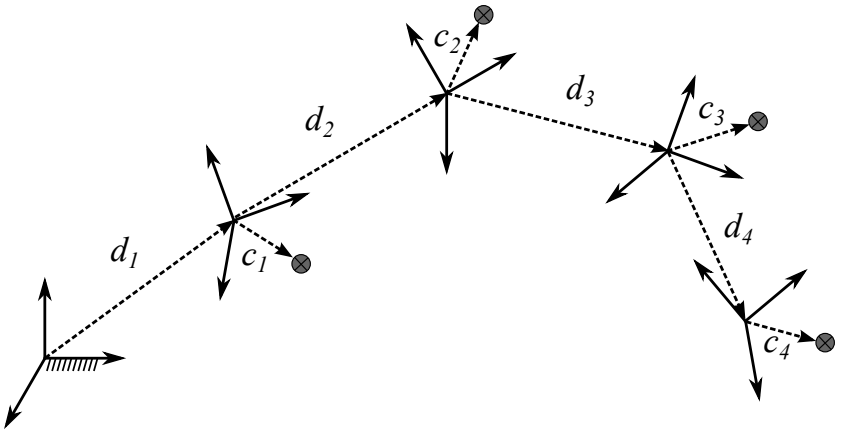

Figure 3.17: Parameters of a 4 link serial chain manipulator.

The CoM location for the entire system in Fig. 3.17 is obtained by computing the sum of each link's CoM divided by the total mass M [43]:

$$CoM = \frac{m_1 T_1 \begin{Bmatrix} c_1 \\ 1 \end{Bmatrix}}{M} + \frac{m_2 T_1 T_2 \begin{Bmatrix} c_2 \\ 1 \end{Bmatrix}}{M} + \frac{m_3 T_1 T_2 T_3 \begin{Bmatrix} c_3 \\ 1 \end{Bmatrix}}{M} + \frac{m_4 T_1 T_2 T_3 T_4 \begin{Bmatrix} c_4 \\ 1 \end{Bmatrix}}{M}. \quad (3.7)$$

A humanoid robot is composed of several such kinematic chains. In this case, the overall CoM location can be obtained by computing the weighted average of the individual chain CoMs, determined according to Eq. (3.7).

Once the CoM for the anthropomorphic structure is known, we need to check whether its projection onto the support plane is within the boundaries of the so called support polygon. The support polygon is defined as the convex hull established by the pressure point of the structure on the ground. Therefore, the shape and size of this polygon depends on whether the robot is currently in single leg (one foot on the ground) or double (both feet on the ground) support mode. An example of the support polygon (solid black line) for a double support configuration and the location of the projected CoM (green dot) is shown in Fig. 3.18.

If the projection of the CoM is inside the support polygon, as depicted, the configuration is said to be statically stable. Otherwise, if the point is found to be outside the boundaries of the polygon the robot would fall over when adopting that pose. In motion planning algorithms, this check is performed iteratively during the roadmap construction or tree expansion process. Based on the outcome of this operation, configurations are rejected or further processed by the respective planner.

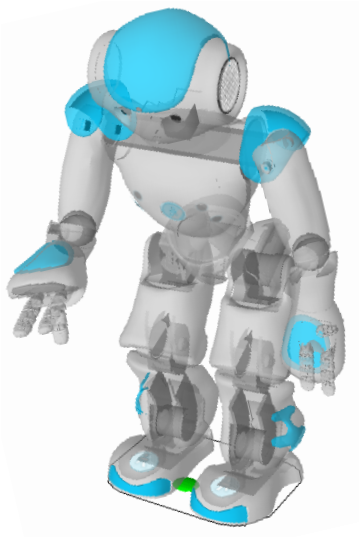

Figure 3.18: Support polygon (black line) and projected CoM (green dot).

3.5 Manipulability of Kinematic Structures

Manipulability is defined as the ease of arbitrarily changing the position and orientation of the end-effector located at the tip of a manipulator. This freedom of motion plays an important role when an interaction between the robot and the environment is intended. Since the ability to manipulate strongly depends on the configuration of the robot, it is necessary to recall some general mathematical relationships. A task r is usually defined by a vector of m variables and the relation between a configuration q, i.e a vector of n joint values θ, and r is expressed as

$$r = f(q), \tag{3.8}$$

also known as the forward kinematics equation. Then, the relation between the joint velocities $\dot{q}$ and the task velocity $\dot{r}$ is given by the derivative of Eq. (3.8) as follows

$$\dot{r} = J(q)\dot{q}, \tag{3.9}$$

where $J(q)$ denotes the Jacobian matrix of the manipulator in configuration q. Analyzing the rank of $J(q)$ reveals some important properties of the kinematic structure. When the following condition is satisfied

$$\max_{q} \operatorname{rank} J(q) = m, \tag{3.10}$$

the manipulator is said to have a degree of redundancy of $(n - m)$. Otherwise, if the manipulator jacobian looses rank in a certain configuration q^*, i.e.,

$$\operatorname{rank} J(q^*) < m \tag{3.11}$$

the robot is said to be in a singular configuration. In this case, the task vector r specifying the end-effector pose cannot move in a certain direction and thus the manipulability is reduced. An explicit measure of the manipulability, represented by a scalar value w, has been introduced by Yoshikawa in [44]:

$$w = \sqrt{\det J(q)J^T(q)}. \tag{3.12}$$

When considering non-redundant manipulators, i.e., when $m = n$, Eq. (3.12) can be reduced to

$$w = |\det J(q)|. \tag{3.13}$$

There also exists a graphical representation of manipulability. When considering the set of joint velocities of unit norm, defined according to

$$\dot{q}^T\dot{q} = 1, \tag{3.14}$$

and substituting in the above equation $\dot{q}$ with the expression obtained by rearranging Eq. (3.9), one gets

$$\dot{r}^T(J(q)J^T(q))^{-1}\dot{r} = 1, \tag{3.15}$$

which is the equation of points on the surface of an ellipsoid in the end-effector velocity space [14]. Here, the superscript T denotes the transpose of the matrix. The direction and dimension of the principal axes of the ellipsoid, centered at the end-effector frame, then follow by computing the *singular value decomposition* (SVD) of the matrix JJ^T. Note that in the following, the argument q of the Jacobian is simply omitted for convenience. The directions of the principal axes are given by the eigenvectors of the matrix JJ^T while the dimensions are determined by the singular values σ_i of J, computed as

$$\sigma_i = \sqrt{\lambda_i(JJ^T)}, \tag{3.16}$$

where λ_i denotes the i-th eigenvalue of JJ^T. Examples of the velocity ellipsoid for an 2R planar manipulator in different configurations is shown in Fig. 3.19.

The shape of the ellipsoid indicates that large velocities can be applied in the direction of the major principal axis while only small velocities are possible in the direction of the minor axis. A similar representation for the manipulability is also available considering forces instead of velocities. The principal axes of the force ellipsoid have the same direction as the axes of the velocity ellipsoid. The dimensions of the axes however, are interchanged with respect to the velocity ellipsoid. This fact indicates, that only small forces can be applied in directions allowing large velocities and vice versa. For further reading about velocity manipulability and a detailed explanation of the force manipulability the reader is referred to the literature [44, 45, 14, 27].

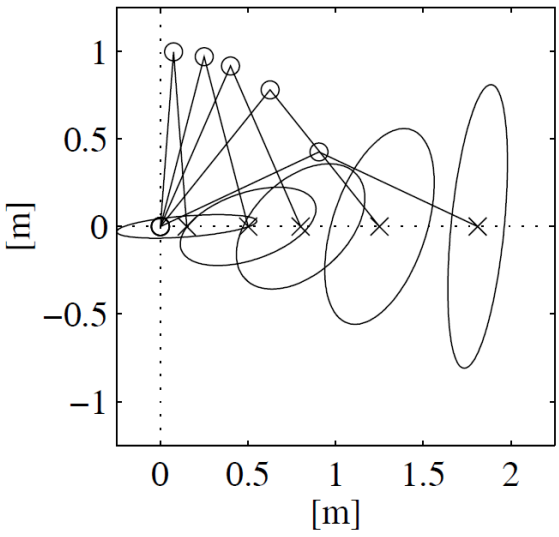

Figure 3.19: Manipulability ellipsoid for a 2R planar manipulator in different configurations, [14].

4 Motion Planning for Humanoids

Planning whole-body motions for a humanoid robot is a challenging task due to the high dimensionality of the configuration space and the constraints involved. In this work, a probabilistic approach has been chosen in order to realize whole-body motion and manipulation planning, since the method is known to be of remarkable efficiency for these kind of problems. In particular, when whole-body motions for manipulation actions have to be planned, several constraints need to be considered during the search, e.g., stability of the humanoid needs to be ensured, collisions need to be avoided and manipulation constraints must be taken into account such that the robot's hand remains attached to an articulated object to be manipulated. To ensure contact with the articulated object, our planner forces hand poses of generated configurations to follow the trajectory of an object handle. Here, we consider the entire task of object manipulation as two motion planning problems: *reaching* and *manipulation*. The sampling-based motion planner, developed in this thesis, is capable to solve both of these problems efficiently. In the following, we will first present the assumptions considered for planning whole-body motions. Then, we will focus on the task of planning statically-stable collision-free whole-body motions in the absence of manipulation constraints. Afterwards, the planner is extended towards manipulation of articulated objects. At the end of the chapter, additional information on the software and tools used for the implementation will be given.

4.1 Planning Assumptions

Currently, we consider the environment of the robot to be known, i.e., the robot does not need to execute any sensor actions during planning. Thus, the joints of the head, which contains sensors for perceiving the environment, remain unarticulated. Additionally the hands are only required when grasping. Therefore, planning is performed considering only a subset of the total DOF of the robot. Note that although both, the head and hands are not actuated during planning, their contribution to the robot's overall CoM location is considered for checking static stability. Furthermore, both of them are taken into account in the collision checking procedure.

Moreover, it is assumed that the robot remains in double support, i.e., with both feet fixed on the ground, during the search for a whole-body motion. As we will see later, this condition is essential to account for the stability constraint during the exploration of the configuration space.

Another important issue in motion planning is collision avoidance. As mentioned earlier, probabilistic motion planning algorithms do not require to compute an explicit represen-

tation of the free configuration space in advance. Instead, collision checks are iteratively applied to the randomly sampled configurations using external algorithms. Detecting collisions requires the knowledge of the geometry and location of the obstacles in the workspace. In this work, we assume the information to be given. Furthermore, we suppose that the shape and the location of obstacles do not change during planning, i.e., the environment is considered to be static.

Manipulating articulated objects requires to solve two motion planning problems. First, starting from an initial statically-stable double support configuration, a whole-body motion needs to be planned for reaching the object handle with the robot's hand. For the scope of this work, we assume that the handle is already within the reachable workspace of the robot. Once the handle has been grasped, a second motion plan for manipulating the object needs to be found, taking into account the manipulation constraints. For the former motion planning problem, we assume that the pose of the object's handle is specified with respect to some fixed coordinate frame in advance. For the latter, it is supposed that a model of the articulated object is provided. Within the scope of this work, we will consider a door and a drawer as objects to be manipulated (see Sec. 4.3.1). In this case, manipulation requires to ensure that the robot's hand remains on a circular arc or a straight line during the motion. Generally however, it is important to note that the planner developed is also capable of dealing with any other type of articulated objects.

4.2 Whole-Body Motion Planning

The planner developed in this thesis builds upon the RRT-CONNECT algorithm [7], which has already demonstrated the ability to efficiently find solutions to planning problems in high-dimensional domains. The basic idea of RRT-CONNECT is to grow two search trees, one from the start and one from the goal configuration. The search trees are iteratively connected by randomly sampling configurations and extending the trees. In the following, the variant of RRT-CONNECT for whole-body motion planning under stability and collision avoidance constraints will be described.

4.2.1 RRT-CONNECT Planner for Humanoids

As described in Sec. 3.3.2, the generic version of the RRT-CONNECT planner explores the configuration space by expanding two trees towards randomly sampled configurations. With a humanoid robot however, most of these samples correspond to unstable whole-body poses and moving towards these configurations would likely cause the robot to lose its balance. Hence, our planner follows the idea of pre-computing a set of statically stable configurations [19] that are subsequently used to guide the search in the tree expansion process (see Sec. 4.2.2). In the following, we will explain the basic functionality of the RRT-CONNECT planner for a humanoid robot in the absence of manipulation constraints.

Algorithm 4: Humanoid RRT-CONNECT (q_{start}, q_{goal})

```
1  n_s.q ← q_start
2  n_g.q ← q_goal
3  T_a.init(n_s); T_b.init(n_g);
4  for i = 1 to max_iter do
5  |   q_rand ← RAND_DS_CONFIG(DS_DATABASE)
6  |   if not (EXTEND(T_a, q_rand) = TRAPPED) then
7  |   |   if (CONNECT(T_b, T_a.last.q) = REACHED) then
8  |   |   |   PATH(T_a, T_b) ← PATH_SHORTCUTTER(T_a, T_b)
9  |   |   |   return PATH(T_a, T_b)
10 |   |   end
11 |   end
12 |   SWAP(T_a, T_b)
13 end
14 return FAILURE
```

Algorithm 5: EXTEND ($\mathcal{T}$, q_{ref})

```
1  n_near ← FIND_NEAREST_NEIGHBOR(T, q_ref)
2  n_new.q ← NEW_CONFIG(q_ref, n_near.q)
3  if IS_CONFIG_VALID(n_new.q) then
4  |   T.add_node(n_new)
5  |   T.add_link(n_near, n_new)
6  |   if n_new.q = q_ref then return REACHED
7  |   else return ADVANCED
8  end
9  return TRAPPED
```

Algorithm 6: CONNECT($\mathcal{T}$, q_{ref})

```
1  repeat
2  |   S ← EXTEND(T, q_ref) ;
3  until not (S = ADVANCED) ;
4  return S ;
```

In Alg. 4 the individual steps performed by our planning algorithm is shown in pseudo-code. Each element of the search trees contains a whole-body configuration q and additional information, that will become particularly important when dealing with planning under manipulation constraints. Therefore, we will refer to the elements of a tree as nodes n rather than only configurations. As input, the planner takes two collision-free statically stable double support configurations q_{start} and q_{goal}. While the start configuration usually corresponds to the current state of the robot, the goal configuration is generally not known in advance but has to be generated from the task constraints as described in Sec. 4.2.3.

After initialization (Line 3 of Alg. 4), two search trees $\mathcal{T}_a$ and $\mathcal{T}_b$ are grown from q_{start} and q_{goal} until a solution has been found or a maximum number of iterations max_iter has

been reached. At each iteration i the function RAND_DS_CONFIG returns a random statically stable whole-body pose q_{rand} from the precomputed database *DS_DATABASE*. Then, the tree $\mathcal{T}_a$ is expanded by a single step (see Fig. 3.12) calling the EXTEND procedure, as described in Alg. 5. Here, the function FIND_NEAREST_NEIGHBOR (Line 1 of Alg. 5) finds the node n_{near} containing the configuration q_{near}, i.e., the nearest configuration to q_{rand} in $\mathcal{T}_a$, by evaluating the Euclidian norm of the configuration space distances as follows

$$q_{near} = \arg\min_{q_i \in \mathcal{T}_a} \|q_{rand} - q_i\|. \tag{4.1}$$

Once q_{near} has been found, the function NEW_CONFIG (Line 2 of Alg. 5) first computes the unit direction vector q_{dir} for the tree expansion, defined as

$$q_{dir} = \frac{q_{rand} - q_{near}}{\|q_{rand} - q_{near}\|}, \tag{4.2}$$

and tries to extend $\mathcal{T}_a$ by a new configuration q_{new} generated at a distance ε from q_{near} in the direction q_{dir}:

$$q_{new} = q_{near} + \varepsilon \cdot q_{dir}, \tag{4.3}$$

with $\varepsilon \in [0, 1]$ being a global step parameter, specified in the planner setup. The choice of the ε parameter plays an important role regarding the performance of the planner. While large values for ε are advantageous to obtain a fast tree growth in largely free configuration spaces, small values are more appropriate to safely explore highly cluttered environments. Note, that if the distance between q_{near} and q_{rand} is found to be below ε, q_{rand} is chosen as the new configuration q_{new} instead.

With the IS_CONFIG_VALID function (Line 3 of Alg. 5) the algorithm then checks whether q_{new} and the path joining it to q_{near} satisfy all constraints involved. Checking for validity in the absence of manipulation constraints means to determine whether q_{new} and the sequence of configurations joining it to q_{near} are statically-stable and collision-free. Since collision checking is far more computationally expensive than determining static stability, it is natural to perform the stability check first. While for geometric planning it might be sufficient to check whether the projection of the CoM is inside the support polygon for a given configuration, the system dynamics may cause the CoM to leave the support polygon when executing the whole-body motion trajectory. Therefore, the stability check is performed using a scaled version of the original support polygon and thus the planning algorithm is capable to approximate the system dynamics to a certain extent. As the ε parameter, the scaling factor is specified in the planner setup and should be chosen according to the desired path execution speed. Using a scaled double support polygon for the stability check, as shown in Fig. 4.1, has shown to produce safe whole-body motions while maintaining a sufficient freedom of motion for the robot.

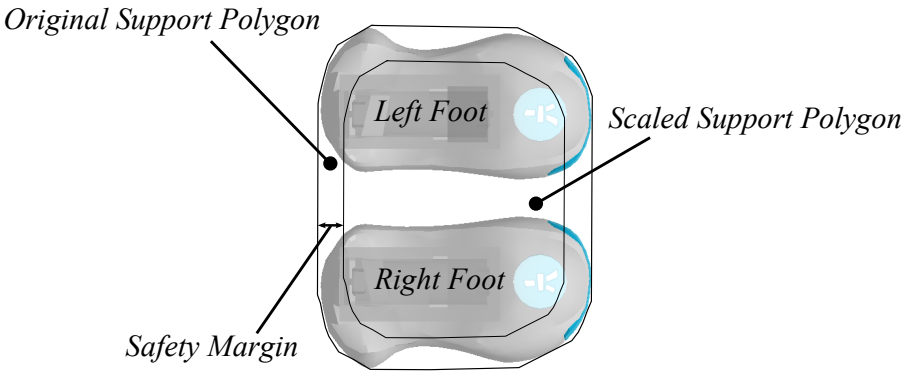

Figure 4.1: Original and scaled support polygon used for checking static stability.

If the new configuration q_{new} and the path segment have passed the stability check, the IS_CONFIG_VALID function proceeds with the collision check. As described in Sec. 3.4.2, the computational effort for this operation can be reduced by using an approximation of the original robot model geometries. In this work, an approximation has been obtained by generating a low-vertex version of the original robot mesh model. Since a humanoid robot is composed of parallel kinematic chains one should note that configurations need to be checked for self-collisions in addition to collisions with obstacles.

If q_{new} or the path segment has been found to be unstable or in collision, both of them are rejected and EXTEND returns *TRAPPED*. In this case, the two trees are swapped (Line 12 of Alg. 4), i.e., the tree denoted by $\mathcal{T}_a$ becomes $\mathcal{T}_b$ and vice versa and the algorithm proceeds with a new iteration $i + 1$. Otherwise, if the new configurations q_{new} and the path segment joining it to q_{near} have been determined to be valid, a new node n_{new} containing the new configurations and the path segment are added to the tree $\mathcal{T}_a$. Then, the EXTEND function returns either *REACHED* or *ADVANCED*, depending on whether q_{new} corresponds to the random configuration q_{rand} or not.

Provided that the return value is not *TRAPPED*, the algorithm proceeds with the extension of the tree $\mathcal{T}_b$ towards the configuration q_{new} (which is now denoted as $\mathcal{T}_a.last.q$ in Alg. 4) just added to $\mathcal{T}_a$. Here, the CONNECT function (Line 7 of Alg. 4), described in Alg. 6, extends the tree $\mathcal{T}_b$ multiple times by iteratively calling the EXTEND function. The search for the nearest neighbor to $\mathcal{T}_a.last.q$ in $\mathcal{T}_b$, as defined by Eq. (4.1), actually needs to be performed only at the first EXTEND iteration. For all subsequent EXTEND operations, the nearest configuration simply corresponds to the configuration of the last node added to $\mathcal{T}_b$. As indicated in Line 3 of Alg. 6, this extension continues until either $\mathcal{T}_a.last.q$ has been reached by $\mathcal{T}_b$ or an invalid configuration has been encountered. An example of the extension procedure applied by the CONNECT function is shown in Fig. 4.2.

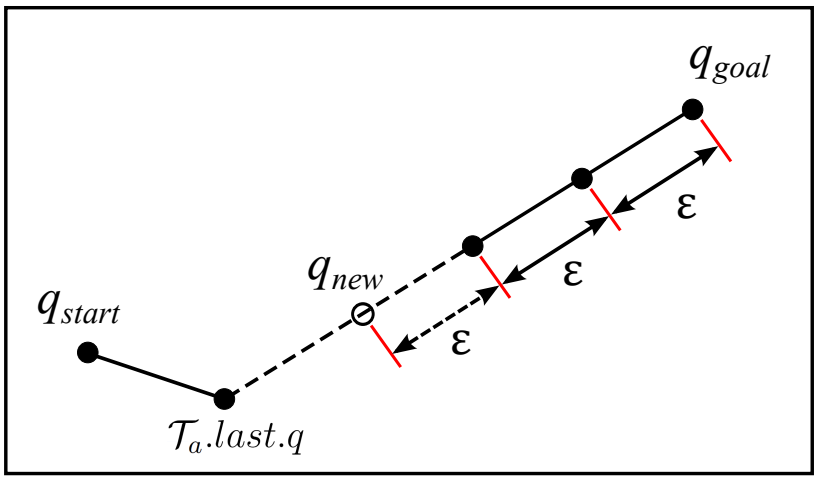

Figure 4.2: Example of the tree expansion with CONNECT.

When an invalid configuration has been encountered during the expansion of $\mathcal{T}_b$, the CONNECT function returns *TRAPPED*, the two trees are swapped and $\mathcal{T}_a$ is expanded towards another randomly chosen configuration from the database *DS_DATABASE* in the next iteration. Otherwise, if $\mathcal{T}_a.last.q$ is reached from $\mathcal{T}_b$, the two trees are connected and thus a valid path has been found. Often, and especially when planning is performed in cluttered environments, the resulting path contains a high number of extraneous nodes. Therefore, the planner finally calls the PATH_SHORTCUTTER (Line 8 of Alg. 4), that tries to remove unnecessary nodes from the solution path while maintaining the validity of the solution (see Sec. 4.2.4). Note that creating shortcuts between configuration of the path is only admissible for motion plans regarding posture changes. The application of this procedure to a manipulation motion plan could cause a violation of the manipulation constraints defined by the articulated object in Cartesian space. After planning, the geometric path obtained is transformed into a smooth whole-body motion trajectory using an approach similar to the one described in [46] (see Sec. 4.2.5).

4.2.2 Precomputing Stable Configurations

Inspired by the idea of Kuffner *et al.* [19], a pre-computed set of stable whole-body configurations is used for tree expansion to speed up the search for valid states. This set is built by iteratively sampling whole-body configurations respecting the joint limits. In the sampled configurations, the leg joint angles are adjusted so that the robot is in double support mode. To do so, we follow the *Active-Passive Link Decomposition* method introduced in Sec. 3.4.3. In this work, we have chosen the frame of the right foot $\mathcal{F}_{rfoot}$ as the root of the kinematic model (see Fig. 4.3) and adapt the left leg configuration to reach the desired pose of the left foot. Generally, one should note that the choice of the active chain is arbitrary in double support mode.

For each sample generated, the 6D rigid body transform of the hip frame T_{hip}^{rfoot}, expressed in $\mathcal{F}_{rfoot}$, is obtained from the forward kinematics of the right leg chain (i.e., the *active*

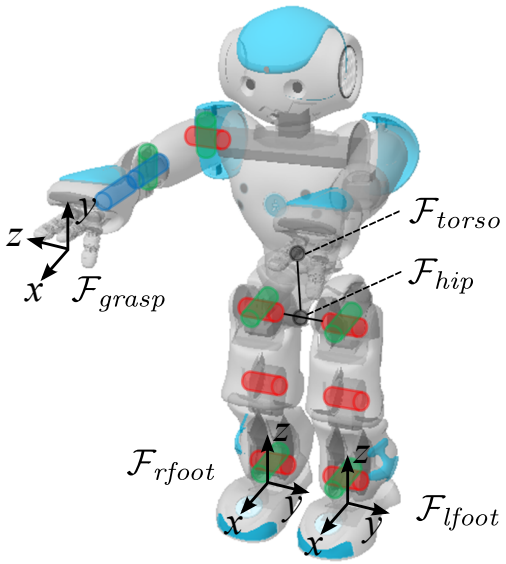

Figure 4.3: Frames of the kinematic model.

chain). From the fixed transformation T_{lfoot}^{rfoot}, which denotes the desired pose of the left foot relative to the right, T_{lfoot}^{hip} expressed in the hip frame $\mathcal{F}_{hip}$ can be computed as follows

$$\begin{aligned} T_{lfoot}^{hip} &= (T_{hip}^{rfoot})^{-1} \cdot T_{lfoot}^{rfoot} \\ &= T_{rfoot}^{hip} \cdot T_{lfoot}^{rfoot}, \end{aligned} \tag{4.4}$$

where $(\cdot)^{-1}$ denotes the inverse of a transformation matrix T. Considering the hip frame as the root of the left leg chain (i.e., the *passive* chain), the inverse kinematics (IK) problem for the desired left foot pose is solved numerically using the recursive Newton-Raphson algorithm [47]. Here, the right leg configuration of the sample is used as an initial guess for the joint values of the left leg chain. Then, velocities for the left leg joints are computed in order to incrementally reduce the error between the current and the desired left foot pose. Finally, the IK recursion stops when the error between the current and desired pose goes below a preset threshold. If an IK solution exists, the modified whole-body configuration is added to the database *DS_DATABASE* if it is free of self-collisions and statically stable. Moreover, it is important to note that these configuration only serve to specify a direction for the tree expansion process and thus stability can be checked using the original support polygon, as shown in Fig. 4.1. Generally, building this database of statically stable configurations needs to be performed only once since it is independent of the planning scenario or environment. Sets for single support mode (left or right) can be constructed in a similar fashion, without the second leg adjustment step. Examples of statically stable double support configurations from the generated database are shown in Fig. 4.4.

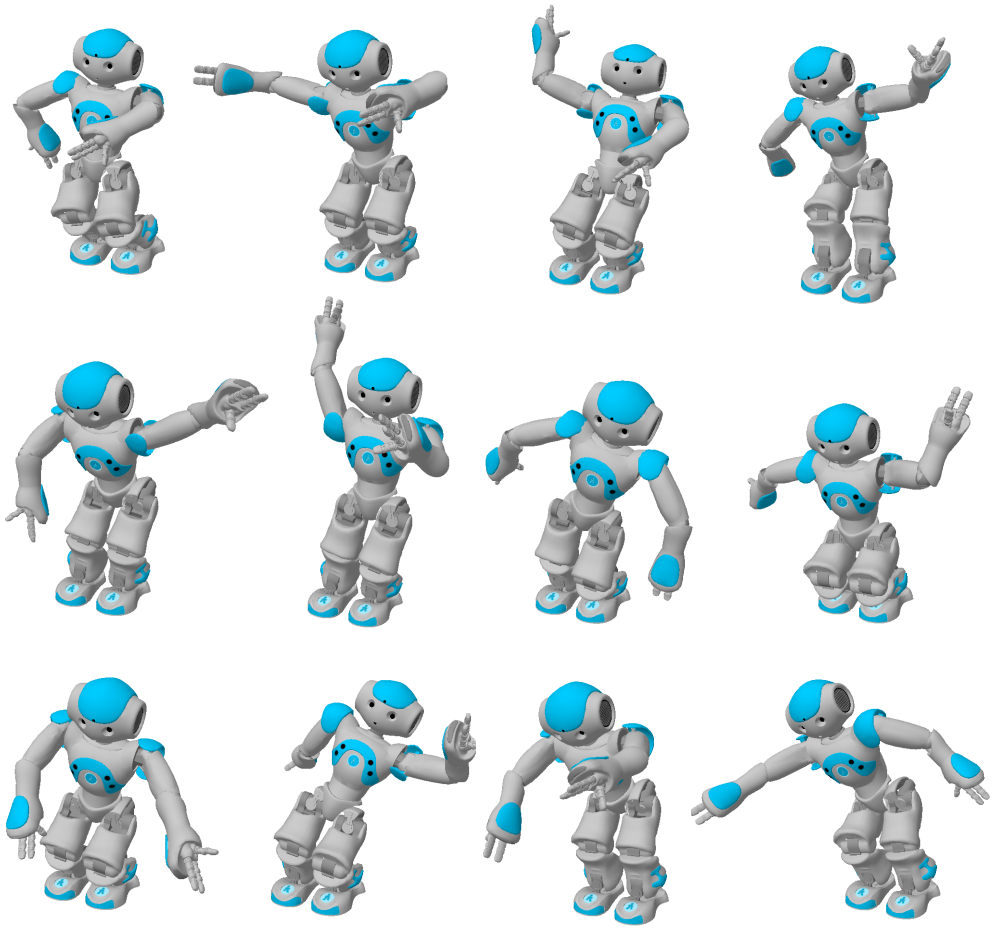

Figure 4.4: Examples of statically stable double support configurations from the database *DS_DATABASE* for the NAO robot.

4.2.3 Goal Pose Generation

When planning whole-body motions for manipulation, the goal configuration has to fulfill several requirements. For manipulation actions, the hand pose of the goal configuration used within RRT-Connect depends on the object to be manipulated. In order to obtain a valid goal configuration with a specific hand pose, our approach first generates double support configurations according to the method presented in Sec. 4.2.2 and then adapts the arm configuration as explained in the following.

In this work, we assume that the 5D grasping goal as desired 3D world coordinate and z-axis direction of the grasp frame $\mathcal{F}_{grasp}$ (see Fig. 4.3), located at the tip of the hand, is given according to the object to be manipulated. Accordingly the roll and pitch angles (i.e. x, y-axis rotations) of the grasp frame are fixed. From this data, the desired hand pose can be expressed w.r.t. the chosen root frame $\mathcal{F}_{rfoot}$ by a homogeneous transformation matrix T^{rfoot}_{grasp}. Then, for each double support configuration generated, the transformation T^{rfoot}_{torso} is obtained by computing the forward kinematics for the serial kinematic chain rooted at the right foot and considering $\mathcal{F}_{torso}$ as the end-effector frame.

Afterwards, the grasp frame $\mathcal{F}_{grasp}$ can be expressed w.r.t the torso frame $\mathcal{F}_{torso}$ by means of basic kinematics computations:

$$\begin{aligned} T_{grasp}^{torso} &= (T_{torso}^{rfoot})^{-1} \cdot T_{grasp}^{rfoot} \\ &= T_{rfoot}^{torso} \cdot T_{grasp}^{rfoot}. \end{aligned} \tag{4.5}$$

Now, considering the serial chain composed by the joints between $\mathcal{F}_{torso}$ and $\mathcal{F}_{grasp}$, the goal configuration generator tries to solve the 5D inverse kinematics problem for the desired hand pose in closed form. In general, it is also possible to solve the IK for a full 6D hand pose. In this application however, we want to leave the the yaw angle (z-axis rotation) of $\mathcal{F}_{grasp}$ unconstrained to allow for a larger set of possible grasp poses. If no solution has been found to the inverse kinematics problem, the whole-body configuration is rejected and a new double support configuration is generated. Otherwise, if the desired hand pose is within the reachable workspace of the robot in the current configuration, often multiple IK solutions for the arm chain exist. In this case, the IK solutions found are evaluated considering the manipulability measure introduced in equation Eq. (3.13). Recalling that in this work, object manipulation is considered as the solution to two motion planning problems, one for reaching and another one for manipulation, two goal configurations need to be generated. A common desired property of these goal configurations is that they allow a high freedom of motion for the arm chain. High flexibility of the final grasp configuration is important for the subsequent manipulation task. On the other hand, the final configuration of the manipulation plan should allow the robot to easily release the object's handle and to move to an arbitrary other whole-body configuration without colliding with the object. The overall quality of an IK solution is then evaluated according to

$$\text{eval}(q^{arm}) = q^{arm}[0] \cdot |\det(J(q^{arm}))|. \tag{4.6}$$

Here, $\det(J(q^{arm}))$ is the determinant of the Jacobian associated with the right arm configuration q^{arm} and denotes a measure of manipulability [44]. Additionally, this measure is combined with the shoulder pitch angle $q^{arm}[0]$ to prefer elbow-down configurations which are considered to be more natural than elbow-up configurations. The best IK solution maximizing Eq. (4.6) is assigned to the arm joints of the initially sampled double support configuration. Then, the resulting whole-body configuration is stored together with the corresponding value obtained from Eq. (4.6) in a goal configuration database. Note that not all double support configurations providing an IK solution for the arm chain are actually what one would consider a natural whole-body pose for an human-like structure. Therefore, the entire procedure is repeated until a predefined number of goal configurations has been generated to increase the probability to get a more natural whole-body pose.

Fig. 4.5 shows a set of example goal configurations generated for a desired grasp frame position and z-axis direction. Currently, the planner developed uses only the overall best whole-body configuration, i.e., the one with the highest value for Eq. (4.6) in the

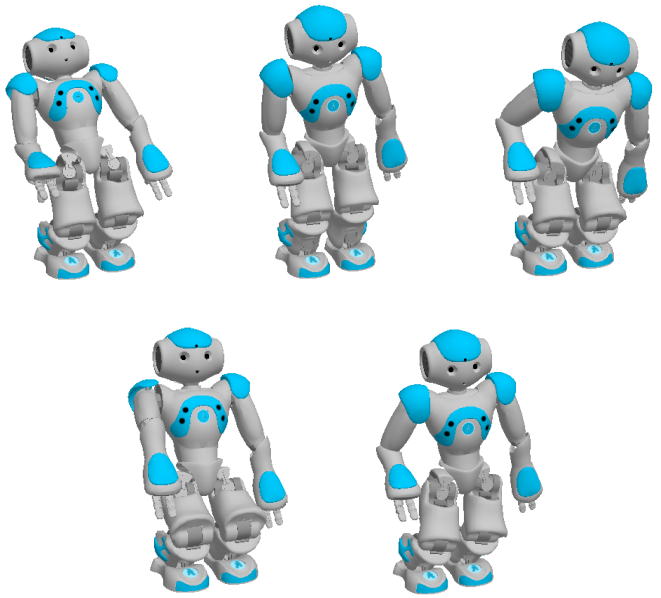

Figure 4.5: Examples of valid goal configurations for a given pose of the robot's right hand. The feet of the robot remain fixed, and the 5D goal pose for the hand is identical in all configurations. In this case, the left arm was set to a safe configuration adjacent to the robot's body.

generated database, as the goal pose. Generally, one can also think of rooting multiple trees at the generated goal configurations and initializing different RRT searches from which the best solution is chosen afterwards.

4.2.4 Path Shortcutter

From the literature, it is well known that probabilistic motion planning algorithms typically produce configuration space paths of unnecessary length. As opposed to roadmap techniques or the basic RRT algorithm using single step tree expansions, the greedy component of the RRT-CONNECT algorithm already tries to keep the number of nodes generated as low as possible. Though, if motion planning is performed in environments with many obstacles, also paths generated by RRT-CONNECT may still contain a high number of extraneous nodes and thus extensive whole-body motions occur. Therefore, the raw solution path P generated by the planner is iteratively shortened by additionally calling the PATH_SHORTCUTTER function (Line 8 of Alg. 4), which will be explained in detail in the following.

As input this function takes the original solution path, which is composed of a sequence of nodes from $\mathcal{T}_a$ and $\mathcal{T}_b$ (see black dashed line in Fig. 4.6). Initially, the procedure starts by creating a shortcut between the start and goal configuration through a straight line path. Then, the shortcut is examined for validity by performing stability and collision checks along the new path segment. If the shortcut is determined to be valid, the entire

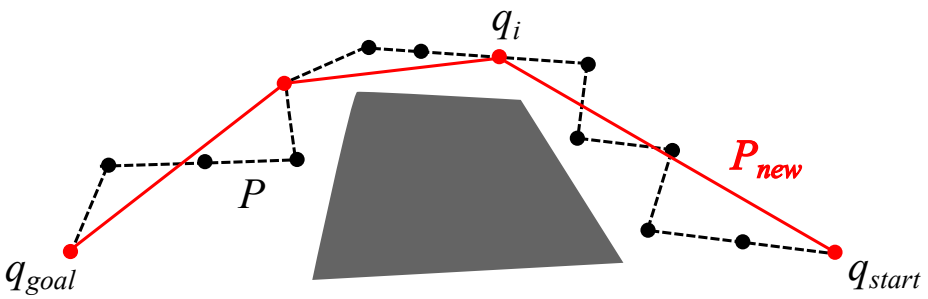

Figure 4.6: Original (black dashed line) and shortened solution path (red solid line).

original path between the two configurations is replaced by the new path P_{new} (see red solid line in Fig. 4.6). Here, the attempt of directly connecting the start and the goal configuration is very optimistic and usually fails in the presence of obstacles. In this case, the function proceeds by iteratively selecting nodes from the original path, going backwards from the goal configuration, and tries to connect them to q_{start} by a shortcut. Once a valid shortcut to a configuration q_i has been found, the original path segment between q_i and q_{start} is replaced by the new path segment. Then, in the next iteration the function further tries to shorten the original path by creating a straight line connection between q_i and q_{goal} and repeats the procedure describe above in the case of failure. Finally, the entire procedure stops when a path waypoint q_i has been successfully connected to q_{goal} by a shortcut. For completeness we want to remind the reader at this point, that path shortcuts are only admissible for motion plans regarding posture changes. For manipulation plans, all configurations along the original path need to be considered because shortcuts would cause a violation of the manipulation constraints.

4.2.5 Motion Trajectory

So far, we focused on the problem of planning a geometric path through the configuration space, which neither collides with obstacles nor violates any other constraints. Thus, the path returned by our planner consists of a sequence of configurations and straight line segments between them. In order to execute the planned motion afterwards, this path needs to be converted into a time-parameterized trajectory that follows the path within the capabilities of the robot. These capabilities usually refer to a limitation of the dynamic properties of the motors (or joints), actuating the links of the mechanical structure. In this work we are considering the maximum joint velocities and accelerations, as they are specified in the robot datasheet. Alternatively, one could also use the maximum torques applicable by the joints, if made available by the robot platform provider. In our application a trajectory is obtained using the *Iterative Parabolic Smoother* method, already implemented in the *MoveIt!* planning framework (see Sec. 4.4). Although there is no detailed description of the method provided, the authors stated that their approach is very similar to the work in [46].

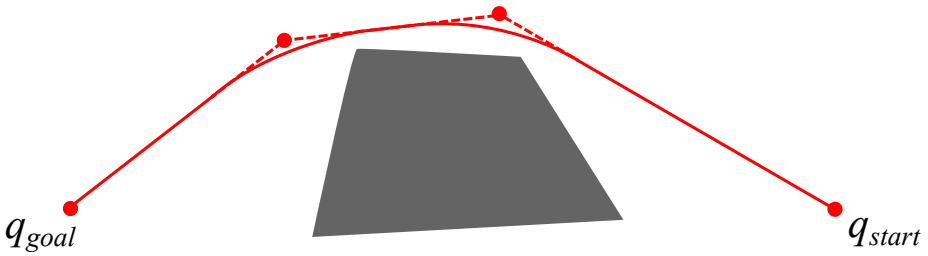

Figure 4.7: Geometric path returned by the planner (dashed line) and time-parameterized motion trajectory (solid line).

With this method a time-optimal trajectory that follows a given differentiable joint space path within given bounds on joint velocities and accelerations can be generated. Since the geometric path returned by the planner is clearly non-differentiable, it is made differentiable in a pre-processing step by adding circular blends between waypoints (see Fig. 4.7). Here, the circles positioned tangential to two linear path segments respectively are defined by the position of their center and radius. Within the pre-processing procedure, these parameters are chosen such that the new circular segments do not replace more than half of the adjacent straight line segments of the original path. Moreover, the procedure takes care that the overall deviation from the original path remains below a certain threshold. Since the circles are quite large, they are not shown in Fig. 4.7 for convenience. In a second step, the new path is parameterized by time taking into account the bounds on joint velocities and accelerations specified before. A detailed description of this procedure would certainly go beyond the scope of this work. Therefore, we want to refer the reader at this point to the paper in [46].

4.3 Whole-Body Manipulation Planning

After planning and executing a collision-free reaching motion, the palm of the robot's hand encloses the object's handle. Then, once the hand has been closed, another motion needs to be planned for manipulating the articulated object. In addition to closure, stability and collision avoidance also the constraints imposed by the object to be manipulated now need to be taken into account during the tree expansion process. In practice, the manipulation constraint implies that the robot's hand must follow a trajectory, defined by the articulated object in Cartesian space. In the following, we will first give a definition of articulated objects. Afterwards, we describe how to consider motion constraints for manipulating articulated objects once the handle has been grasped. In particular, we will focus on the extensions for manipulation planning, made to the Humanoid RRT-CONNECT algorithm, described in Alg. 4 of the previous section.

4.3.1 Articulated Objects

Articulated objects are represented by a volumetric model, a joint with the corresponding position, and a handle position. From this, a generative model can predict the trajectory of the articulated parts and the handle while the object is being manipulated. Articulation models can be learned from previous experience while carefully manipulating the environment [48]. For the scope of this work, we assume the model to be given. Here, we are particularly interested in prismatic (e.g., drawers) and revolute models (e.g., doors), as illustrated in Fig. 4.8. A prismatic model has one degree of freedom as a translation along a fixed axis. The handle trajectory is constrained on a 3D vector denoting the opening direction. A revolute model has one degree of freedom around a specified 3D axis of rotation. The handle trajectory is constrained on the circular arc described by the axis of rotation and the radius of the handle position. When planning for manipulation of articulated objects, the respective handle trajectory and the robot's grasp frame need to be expressed w.r.t the same reference coordinate system.

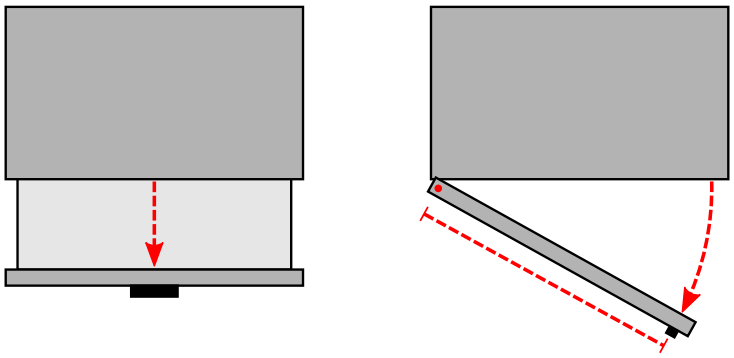

Figure 4.8: Two examples of articulated objects: A drawer and a door.

4.3.2 Extended RRT-CONNECT Planner

In order to realize whole-body manipulation planning the algorithm Alg. 4, described in Sec. 4.2.1, has been extended by several functions. As before, the planner requires a start and a goal configuration to be given in advance. Here, the start configuration used for manipulation planning simply corresponds to the final whole-body pose of the robot after performing the grasping motion. Then, given the initial hand pose of the start configuration and a model of the articulated object, one can compute the desired final hand pose along the trajectory of the object handle. For the final 5D hand pose obtained, a whole-body goal configuration for planning is generated following the procedure described in Sec. 4.2.3. An example of a start and goal configuration used for manipulating a drawer and a door are shown in Fig. 4.9.

The overall RRT-CONNECT algorithm used for planning both, reaching and manipulation motions is described as pseudo-code in Alg. 7. With the robot having grasped the

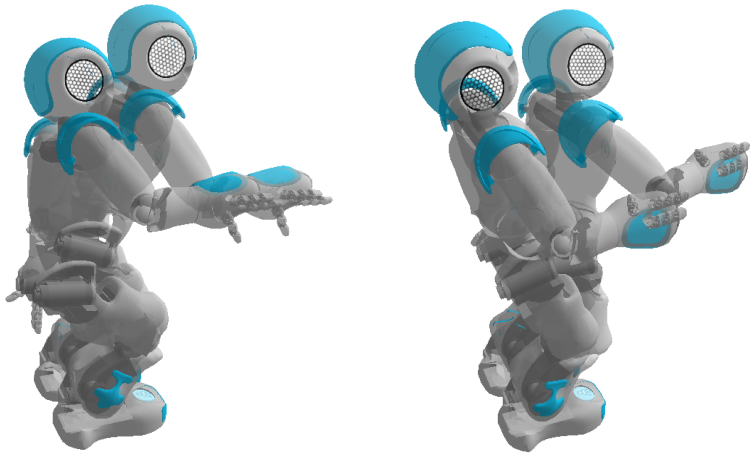

Figure 4.9: Start and goal configuration for manipulating a drawer (left) and a door (right).

object's handle (see Line 3), the extended RRT-CONNECT planner now takes additionally to a start and goal configuration a list of object parameters *object_params* as input. Before entering the main loop of the algorithm (see Line 9), the root nodes of the two trees are initialized based on the list of object parameters. Furthermore, the EXTEND function, described in Alg. 8, now performs additional operations to ensure that the robot's hand remains on the trajectory of the articulated object handle throughout the whole-body motion. In the following sections, the tree initialization and the expansion of the trees under manipulation constraints will be described in detail.

Algorithm 7: Extended Humanoid RRT-CONNECT (q_{start}, q_{goal}, *object_params*)

```
1  n_s.q ← q_start
2  n_g.q ← q_goal
3  if object_grasped = true then
4  |   ⟨h_0, ..., h_k⟩ ← GEN_HAND_TRAJ(q_start, object_params)
5  |   n_s.h ← 0
6  |   n_g.h ← k
7  end
8  T_a.init(n_s); T_b.init(n_g);
9  for i = 1 to max_iter do
10 |   q_rand ← RAND_DS_CONFIG(DS_DATABASE)
11 |   if not (EXTEND(T_a, q_rand) = TRAPPED) then
12 |   |   if (CONNECT(T_b, T_a.last.q) = REACHED) then
13 |   |   |   if object_grasped = false then
14 |   |   |   |   PATH(T_a, T_b) ← PATH_SHORTCUTTER(T_a, T_b)
15 |   |   |   end
16 |   |   |   return PATH(T_a, T_b)
17 |   |   end
18 |   end
19 |   SWAP(T_a, T_b)
20 end
21 return FAILURE
```

Algorithm 8: EXTEND ($\mathcal{T}$, q_{ref})

```
1  n_near ← FIND_NEAREST_NEIGHBOR(T, q_ref)
2  n_new.q ← NEW_CONFIG(q_ref, n_near.q)
3  if object_grasped = true then
4  |   n_new ← ENFORCE_MANIP_CONSTRAINT(n_new, n_near)
5  end
6  if IS_CONFIG_VALID(n_new.q) then
7  |   T.add_node(n_new)
8  |   T.add_link(n_near, n_new)
9  |   if n_new.q = q_ref then return REACHED
10 |   else return ADVANCED
11 end
12 return TRAPPED
```

4.3.3 Tree Initialization Considering Manipulation Constraints

Given a start configuration q_{start} with a hand attached to the object handle and the object parameters *object_params*, our approach computes a sequence of hand poses $\langle h_0, \ldots, h_k \rangle$ for T^{rfoot}_{grasp} along the handle trajectory (GEN_HAND_TRAJ in Line 4, Alg. 7). With the objects considered in this work, all these hand poses have the same direction for the z-axis of $\mathcal{F}_{grasp}$, pointing through the closed palm of the robot's hand. For the initialization of the search trees, the nodes n_s and n_g that contain q_{start} and q_{goal} are assigned the hand pose indices 0 and k, respectively (Lines 5 and 6 in Alg. 7). In the subsequent planning loop, the trees rooted at n_s and n_g are intended to be expanded iteratively by new configurations with hand pose indices higher than 0 or lower than k. Fig. 4.10 illustrates examples for linear and circular object trajectories with the generated hand poses (red dots).

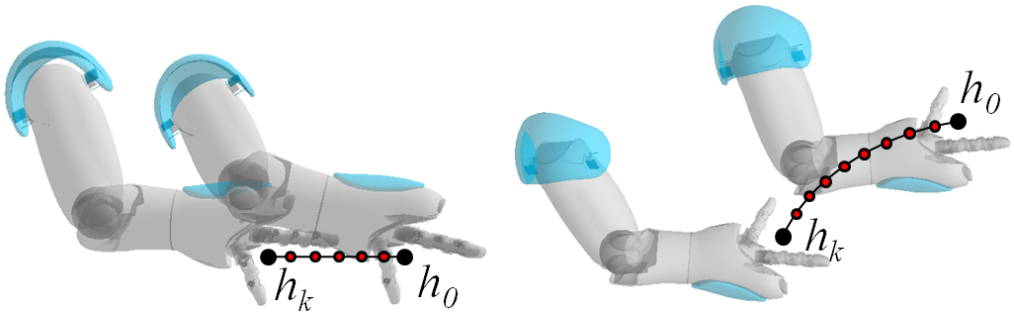

Figure 4.10: Example end-effector trajectories, i.e., desired hand poses for opening a drawer (left, side view) and a door (right, top view).

4.3.4 Tree expansion under Manipulation Constraints

During planning it is required that the robot's hand remains on the object trajectory for each new configuration added to one of the trees by the EXTEND function. Given a configuration q_{new} generated by NEW_CONFIG by stepping from q_{near} towards q_{rand}, the manipulating hand needs to move along the handle trajectory. This requirement is satisfied by ENFORCE_MANIP_CONSTRAINT (Line 4 in Alg. 8). In practice, this means that the hand needs to move from the hand pose with index $n_{near}.h$ to the hand pose with index $r = n_{near}.h+1$ or $r = n_{near}.h-1$, depending on the tree to be expanded. Thus, the first expansion of the start tree should result in a new configuration with the hand at the pose with index 1. Similarly, the first configuration added to the goal tree will have its hand at the pose with index $k-1$. Following this rule, it is guaranteed that the branches of the start tree correspond to continuous forward motions of the hand along the object's handle trajectory. On the other hand, branches of the goal tree correspond to backward motions of the hand, which are reversed once the two trees are connected. To obtain the desired hand pose in the tree expansion step, our approach again solves the IK for the arm chain in closed form, as already described in Sec. 4.2.3. If no solution has been found, q_{new} is rejected by the subsequent IS_CONFIG_VALID function (Line 6 in Alg. 8) and the algorithm proceeds with the expansion of the other tree. Otherwise, the planner continues with an evaluation of the solutions found. From the set *IKsol* of IK solutions for the desired hand pose, the planner selects q_c^{arm} , i.e., the solution that minimizes the configuration space distance to the arm configuration of q_{near}

$$q_c^{arm} = \underset{q_i^{arm} \in IKsol}{\arg\min} \|q_i^{arm} - q_{near}^{arm}\|, \tag{4.7}$$

and sets q_{new}^{arm} to q_c^{arm} and $n_{new}.h = r$ before performing the validity check. If the new configuration is also found to be valid a new node is added to the corresponding tree, otherwise the configuration is rejected.

At this point, it remains to describe how a connection between the two trees is detected. At each iteration, the function ENFORCE_MANIP_CONSTRAINT first computes the forward kinematics (FK) for the kinematic chain from $\mathcal{F}_{rfoot}$ to $\mathcal{F}_{grasp}$ in configuration q_{new}, previously generated by NEW_CONFIG. From the transformation T_{grasp}^{rfoot} thus obtained, the planner checks whether or not the hand is already at the desired pose with index $n_{near}.h+1$ or $n_{near}.h-1$. If this is not the case, the arm configuration is adapted by applying the procedure as described above. Otherwise, when the FK determines the hand to be already in the correct pose, a path has been found. Observing the tree expansion process, depicted in Fig. 4.11, it reveals that this case occurs exactly once, namely when $\mathcal{T}_b$ is expanded by the CONNECT function (Line 12 in Alg. 7), $\mathcal{T}_a.last.q$ has been reached from the last node added to $\mathcal{T}_b$, i.e., $\mathcal{T}_a.last.q = q_{new}$, and the respective hand pose indices of q_{near} and q_{new} are already adjacent to each other. When a path has been found, the value of the variable *object_grasped* is queried once again (Line 13 in Alg. 7), in order to prevent the planner from building shortcuts on manipulation paths and thereby violating the manipulation constraints.

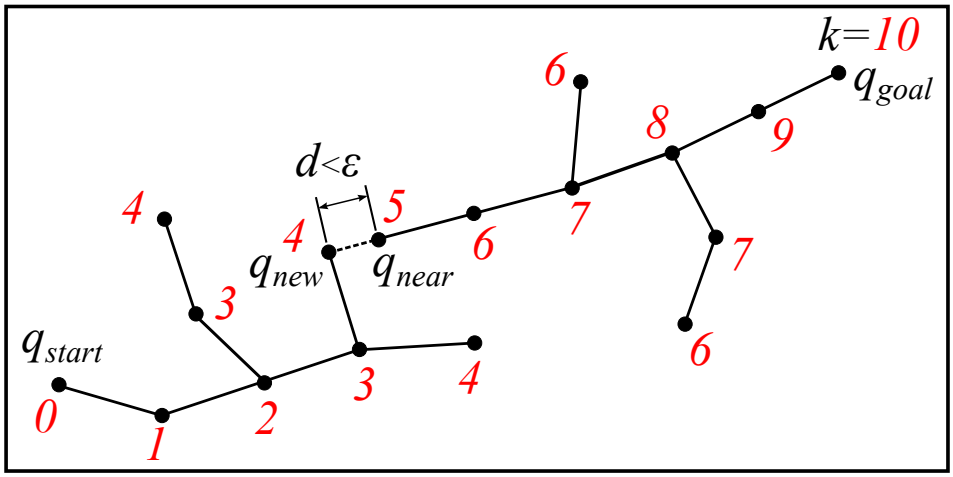

Figure 4.11: Nodes of the start tree growing from the left and the goal tree growing from the right, with their respective hand pose indices (red numbers) when manipulating an articulated object. A connection between the two trees can be established when two nodes with adjacent hand pose indices can be connected.

4.4 Implementation Details

The planner developed in this thesis is implemented in the *MoveIt!* framework, which is a part of the Robotic Operating System (ROS) [49]. This framework provides several tools for developing planning algorithms. Given a description file of a robot, for example, one can define arbitrary groups, such as legs or arms, for planning. Moreover, *MoveIt!* contains the *FCL* library [37] used for collision checks in this work. When the planner checks a configuration for validity, the collision mesh model of each robot link is tested for self-collisions and collisions with the environment. For the stability constraint, the Zero Moment Point generally indicates a humanoid's dynamic stability [42]. In this work, we use the simplification of static stability which is a valid approximation for slow motions. To analytically solve IK, we use *IKfast* [50]. In our case, this results in 5D goal poses with a 5 DOF arm. Within the procedure of generating statically-stable double support configurations, the leg closure is achieved by solving the IK for the left leg chain numerically using the iterative Newton-Raphson algorithm, implemented in *KDL* [47].

5 Experiments

In this chapter the experimental results for planning and executing whole-body motions are presented. After introducing the robot platform used, the trajectory execution method applied and the planning setup chosen, the performance of our planner is evaluated on several motion planning problems, such as collision-free body repositioning, reaching as well as manipulation of different objects.

5.1 Humanoid Robot Platform

For the experimental evaluation of our approach, we use a V4 Nao humanoid by Aldebaran Robotics (see Fig. 5.1). The robot is 58 cm tall and has 25 DOF: 2 in the neck, 6 in each arm (including one to open and close the hand), and 5 in each leg. In addition, the legs share a common (linked) hip joint that cannot be controlled independently. To perceive its own state and the environment, the robot is equipped with a large network of proprioceptive and exteroceptive sensors, including 2 cameras located in the head, 4 microphones, a sonar rangefinder, 2 infrared transmitters and receivers, an inertial board, 9 tactile and 8 pressure sensors. The head of the robot contains an Intel ATOM 1,6GHz CPU, that runs a Linux kernel and supports Aldebaran's *NAOqi* software, used to communicate with the robot and controlling it.

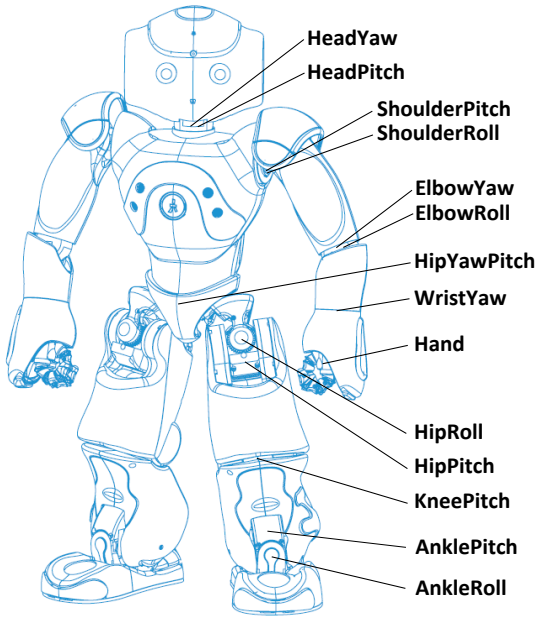

Figure 5.1: Kinematic model of the NAO robot (version H25), [3] .

Since planning is performed off-line, the planner developed in this work does not rely on sensor feedback. Therefore, the head of the robot containing the cameras remains unarticulated. Furthermore, the hands are only required when grasping. Hence, our planner operates on a total number of 20 DOFs. In the context of motion planning a configuration is then defined by the following vector of joint angles:

$$q = [q_{RLeg}, q_{LLeg}, q_{RArm}, q_{LArm}] \tag{5.1}$$

with

$$q_{R/LLeg} = [q_{AnkleRoll}, q_{AnklePitch}, q_{KneePitch}, q_{HipPitch}, q_{HipRoll}] \tag{5.2}$$

$$q_{R/LArm} = [q_{ShoulderPitch}, q_{ShoulderRoll}, q_{ElbowYaw}, q_{ElbowRoll}, q_{WristYaw}] \tag{5.3}$$

Actuating the *HipYawPitch* joint (see Fig. 5.1) would require to lift off one foot or to allow the feet to slip on the ground while remaining in double support. In this work however, we consider slippage as invalid and therefore this joint value is rigidly set to zero. To correct for backlash of the gears while executing joint angle trajectories, Nao can measure each angle with Hall effect magnetic rotary encoders at a precision of 0.1°. Inertia, mass, and CoM of each link are known from CAD models. For efficient collision checks, we created a low-vertex collision mesh model for each of the robot's links from the CAD models (see Fig. 5.2). Very subtle geometries of the original robot mesh, like the hands, have been entirely replaced by simple geometric primitives.

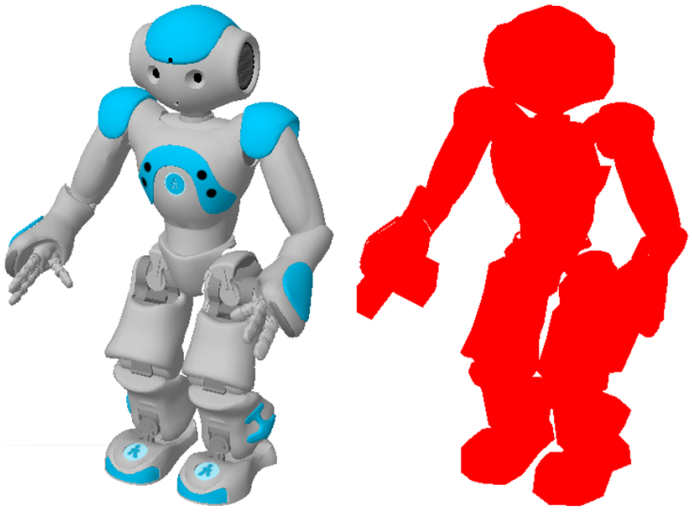

Figure 5.2: Original model of the NAO robot (left) and approximation used for collision checking (right).

5.2 Trajectory Execution for the NAO Robot

The whole-body motion trajectory finally obtained from the planner can be executed in various ways. A first simple possibility to visualize the trajectory generated is to use the tools provided by the *MoveIt!* framework of ROS (see Sec. 4.4). In this case, the sequence of configurations of the robot along the trajectory are shown in the *RViz* visualizer of ROS. Although the configurations of the trajectory are not interpolated with this tool and thus the path is not shown as a fluent transition between the configuration waypoints, this option allows a quick evaluation of the planning results. Alternatively, the trajectory can be directly send to the *angleInterpolation* function of *NAOqi*, which is the main software running on the robot and controlling it [3]. As input the function takes the names of the joints to be moved, a list of configurations and a list of associated timestamps. For security reasons, the *NAOqi* software can be also run locally on a computer in order to observe the motion on a simulated robot model. Then, when the whole-body motion has been determined to be safe, the same trajectory can be executed on the real robot platform by sending the *angleInterpolation* command to *NAOqi* running on the on-board computer of the robot.

5.3 Planning Setup

For planning whole-body motions we used a database of 463 statically stable double support configurations, generated within 10 000 iterations. The success rate of only 4.63% demonstrates the low probability of generating valid configurations, when the configurations space is sampled completely at random during the search. For generating goal poses, we allow a maximum number of 3 000 iterations. For efficiency, we stop searching when more than 5 goal poses have been found and choose the best one according to Eq. (4.6). The maximum number of iterations max_iter in Alg. 7 was set to 3 000. The ε parameter used in the EXTEND function to generate new configurations from q_{near} along the path segment joining it to q_{rand} (see Eq. (4.3)), is set to 0.1. The stability check for configurations generated during planning is performed on a by factor 0.8 scaled original support polygon. With this choice, the planned motions have shown to be safely executable while maintaining a sufficient freedom of motion for the robot within the planning phase.

Planning was performed off-board on a single core of a standard desktop CPU (Intel Core2 Duo, 3 GHz).

5.4 Evaluation of Whole-Body Motion Planning

In the first experiment, we evaluate the performance of the planner with goal pose generation. Thus, the robot had to perform stable whole-body motions to reach a 5D manipulation goal with the right arm (similar as in Fig. 4.5). The z-axis of the grasp frame $\mathcal{F}_{grasp}$ is horizontal and perpendicular to the robot's orientation, e.g., to grasp the handle of a drawer, at a height of $z = 0.2\,\text{m}$. The x and y coordinates of the goal

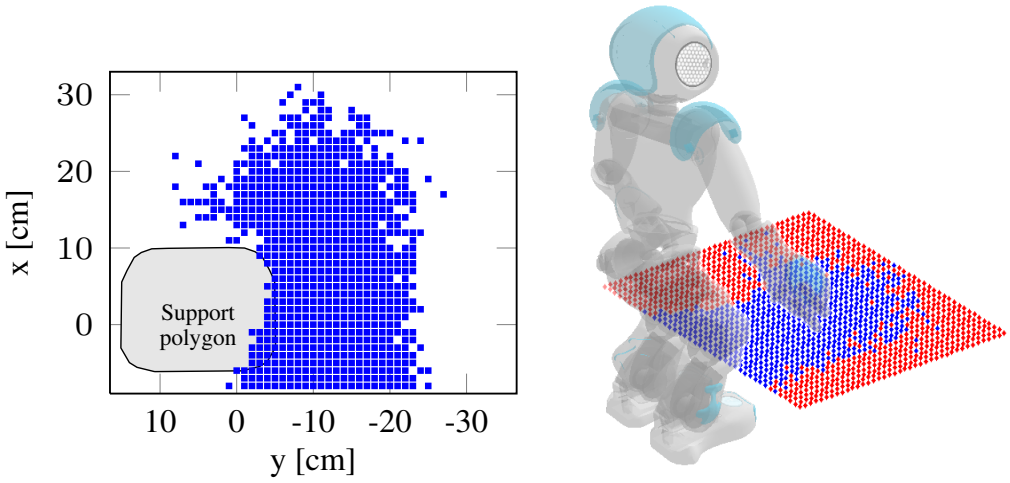

Figure 5.3: Reachability map of the right hand as 2D projection relative to the right stance leg (left) and shown relative to the robot model (right). Blue squares denote successful planning results to the 5D end-effector pose at height z=0.2 m.

are varied in intervals of 1 cm in a 40 cm × 40 cm area. Fig. 5.3 shows the resulting reachability map, in which squares denote successful results. In the other locations, the goal pose generation failed, i.e., they are not reachable. It is obvious that whole-body motion planning extends the manipulation range around the robot. When using solely its 5 DOF arm for planning, Nao reaches only a subset of these poses.

For the reachable locations, it takes 6.62 s ± 6.33 s to generate the first goal pose, and 0.18 s ± 0.19 s for planning with RRT-CONNECT expanding 49.79 ± 29.15 nodes on average.

5.5 Planning Collision-Free Motions

In a second experiment, we evaluate the performance of the planner in the presence of obstacles seriously constraining the possible motions. Here, two motion planning problems have been solved subsequently. In a first step, a motion for the robot has been planned to reach a desired 5D goal pose for its hand, located inside the upper shelf of a cabinet. In a second motion plan the robot then had to escape from the shelf and move its hand inside another shelf without colliding.

The entire whole-body motion plan to solve these tasks was successfully generated within 12.64 s, and executed collision-free and statically stable (see 5.4). A goal configurations for the upper and lower shelf were generated within 19.43 s. Note that this scenario is usually problematic for Jacobian-based optimization techniques due to the local minima of the cavities.

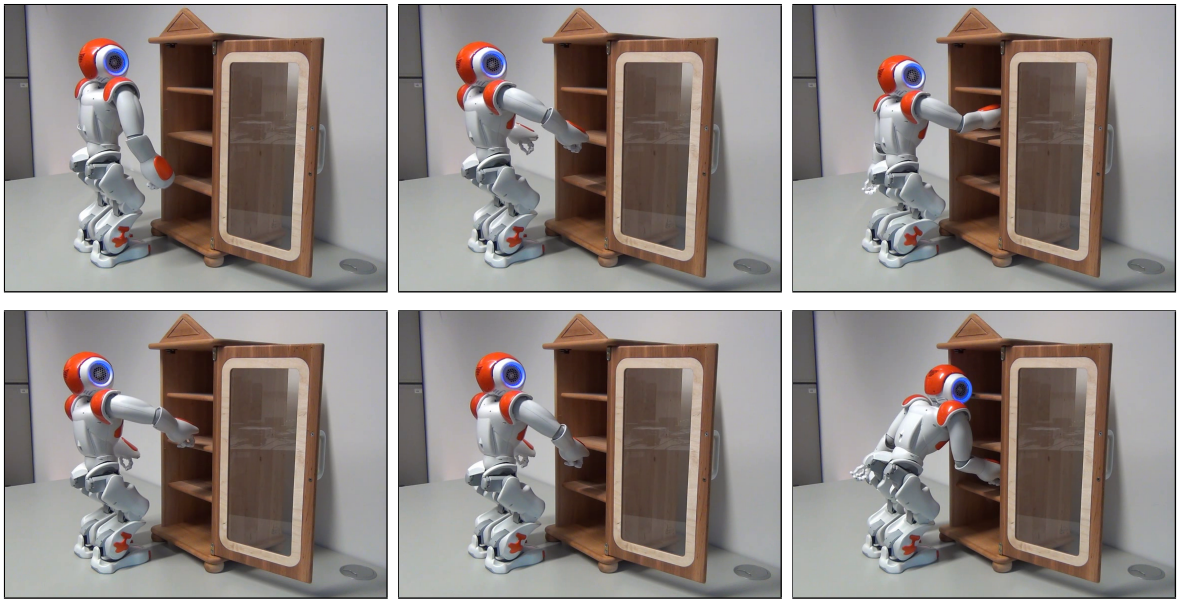

Figure 5.4: Execution of a whole-body plan to reach into different shelfs of a cabinet. The sequence of pictures shown are snapshots taken from a video (left to right, top to bottom).

5.6 Manipulating Articulated Objects

We now evaluate the manipulation of articulated objects with a Nao humanoid. From its initial configuration, the robot first has to plan for reaching the known handle location, grasp it, and then plan an end-effector trajectory given by the parametrization of the articulated object. Fig. 5.5 and 5.6 show snapshots from videos in which the robot executes the whole-body motion plans. It successfully grasped the handle and opened the drawer and door while keeping its balance. Fig. 5.7 shows the trajectories of the robot's right hand and CoM, as measured by the joint angle sensors during execution. The hand trajectory closely follows the given model, while the CoM remains safely within the support polygon.

To quantitatively evaluate the reliability of our randomized planning approach both scenarios, reaching and opening a drawer and a door, were planned 100 times. To plan for reaching the drawer's and door's handle it took 0.08 s ± 0.04 s and 0.09 s ± 0.05 s, expanding 32.81 ± 8.41 and 32.16 ± 9.43 nodes on average, respectively. A goal pose for the object's handle location was generated within 5.42 s ± 5.17 s and 8.59 s ± 7.54 s. For manipulation of the object, planning took 0.11 s ± 0.07 s and 0.15 s ± 0.08 s, expanding 30.36 ± 12.69 and 35.17 ± 14.51 nodes on average, respectively. A goal pose for the final drawer and door handle location was generated within 4.14 s ± 3.94 s and 2.63 s ± 2.47 s.

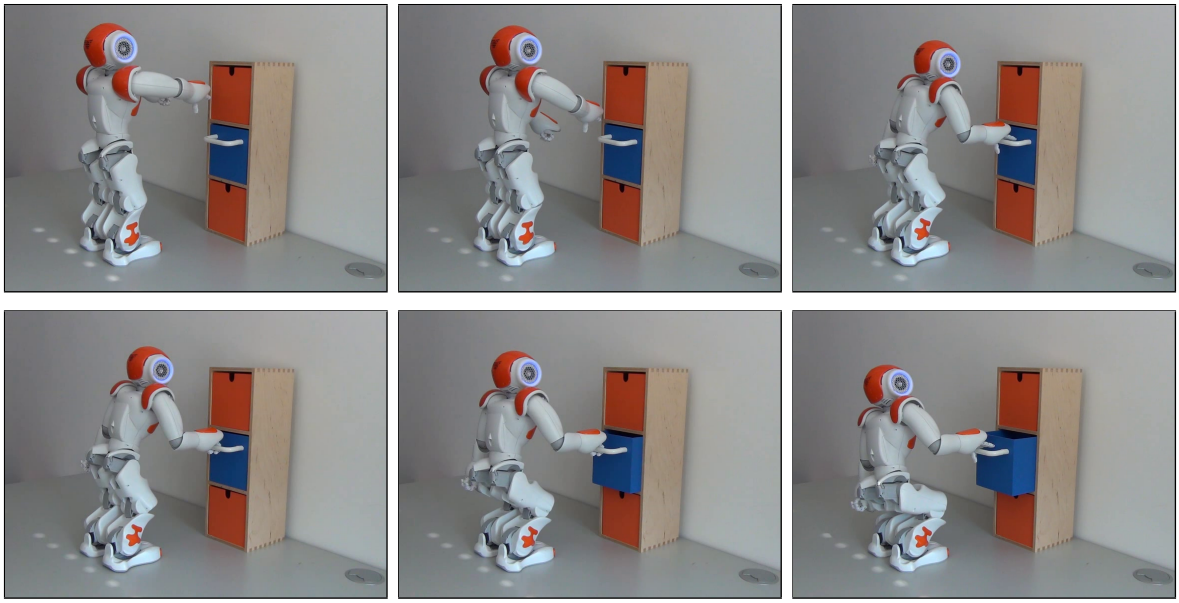

Figure 5.5: Execution of a whole-body manipulation plan for a drawer. First, a motion for reaching the drawer handle is planned and executed. Then, planning is performed for manipulation of the object once the handle has been grasped. Pictures are snapshots taken from a video (left to right, top to bottom).

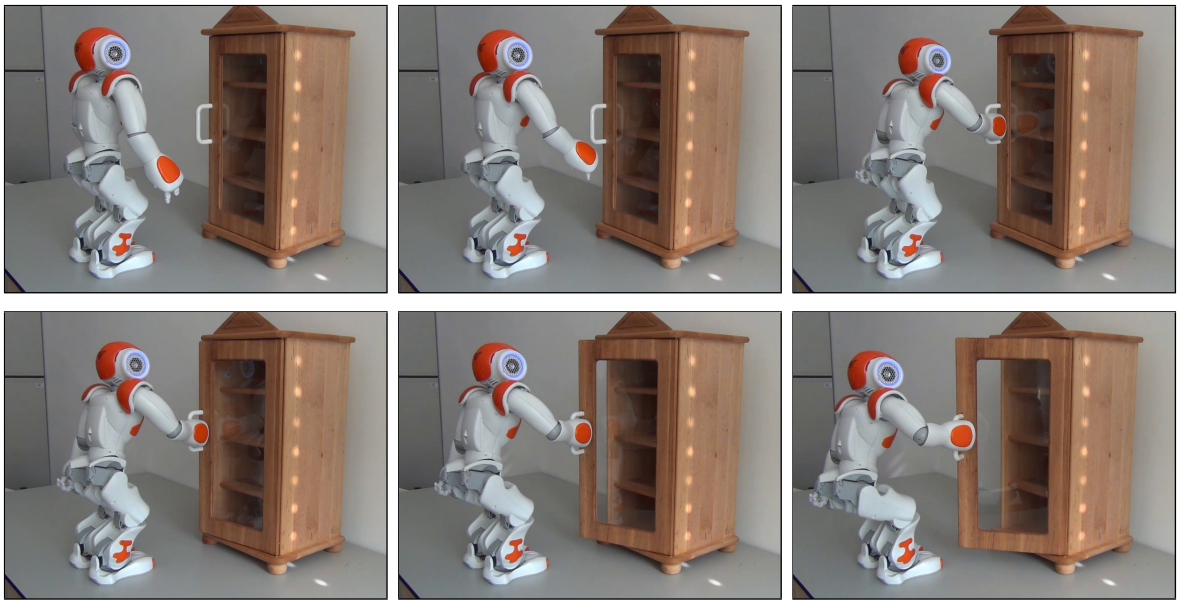

Figure 5.6: Execution of a whole-body manipulation plan for a door. First, a motion for reaching the door handle is planned and executed. Then, planning is performed for manipulation of the object once the handle has been grasped. Pictures are snapshots taken from a video (left to right, top to bottom).

With the chosen parameters of our algorithm, the task of reaching and manipulating a drawer succeeded in 89% of all attempts. For the door, planning succeeded in 78% percent of the runs. These rates certainly increase as more planning time is allowed. Note that the execution of planned motions on the real robot platform succeeded to 100% without losing balance.

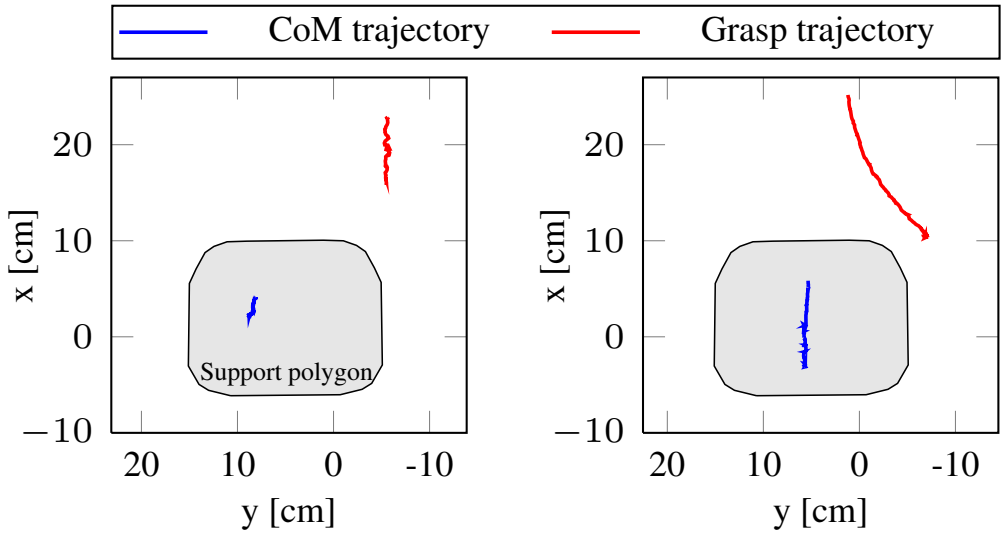

Figure 5.7: Trajectory for the right hand and center of mass over the support polygon while opening a drawer (left) and a door (right)

5.7 Collision-Free Object Manipulation

In this experiment we evaluate the capability of our algorithm to plan a reaching and manipulation motion for the robot in the presence of an additional obstacle. Starting from an initial configuration the robot first needs to perform a motion to reach the drawer's handle without colliding with the obstacle lying between the initial and desired final right hand location (see Fig. 5.8). Then, once the handle has been grasped by the robot, a second planned motion is executed on the Nao robot to open the drawer without hitting the obstacle with any part of the body.

Also here, we evaluated the reliability of the planner by repeating the experiment 50 times. To plan for reaching the drawer's handle it took 7.71 s ± 3.0 s, expanding 1067.93 ± 333.54 nodes on average. A goal pose for the drawer's handle location was generated within 16.24 s ± 12.64 s. For manipulation of the object, planning took 0.2 s ± 0.19 s, expanding 32.16 ± 21.09 nodes on average. A goal pose for the final drawer handle location was generated within 8.53 s ± 7.21 s. The entire task of planning the reaching and manipulation motion for the robot succeeded in 86% of the runs. All of the planned motions were executed on the real robot platform collision-free and statically stable.

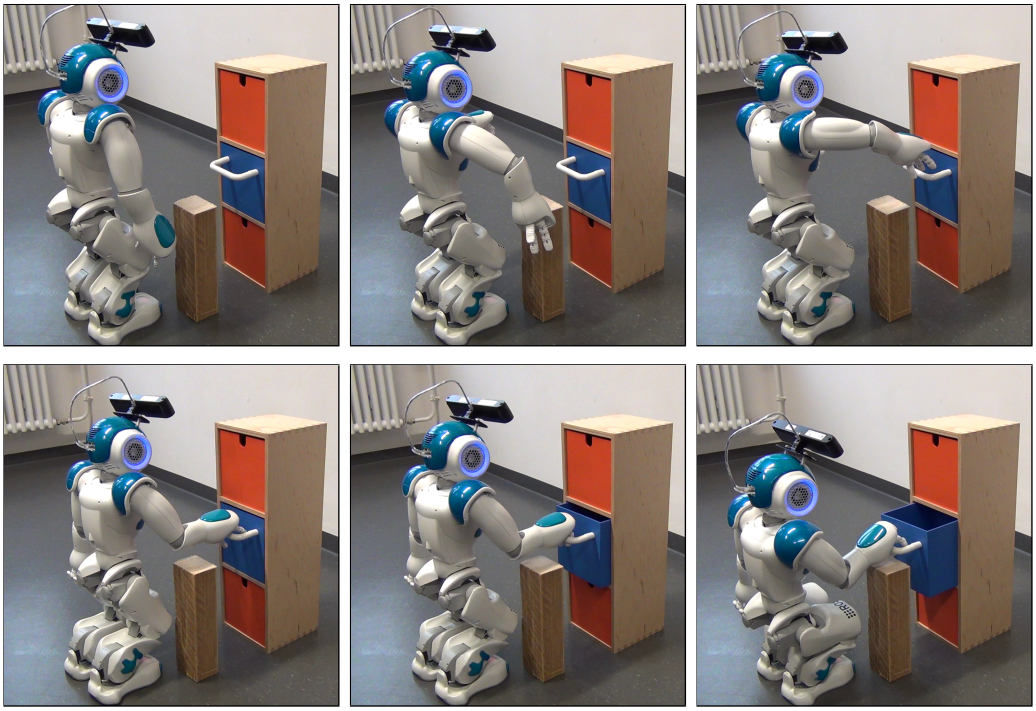

Figure 5.8: Execution of a whole-body manipulation plan for a drawer with collision avoidance. First, a collision-free motion for reaching the drawer handle is planned and executed. Then, planning is performed for manipulation of the object once the handle has been grasped. Pictures are snapshots taken from a video (left to right, top to bottom).

5.8 Pick and Place an Object

In the last experiment the robot is asked to pick up an object with its hand and to move it to another location. As before, we consider the entire task as two consecutive motion planning problems. First, planning is performed to find a valid whole-body motion for reaching the object with the hand. Once the object has been grasped, a second motion for manipulation is generated. As an example object for the pick and place task, we have chosen a small box located on top of a bigger box (see Fig. 5.9). After grasping the box the robot is expected to transfer it to an adjacent basket without colliding with the other objects contained in the environment.

The major difference with respect to the previous experiments presented so far is that for the second motion planning problem collisions additionally need to be checked between the carried object and the static obstacles. In this work, this is achieved by increasing the collision mesh model of the right hand such that the object becomes part of the robot geometry. Another critical issue for this kind of tasks is the abrupt shift of the CoM location, when lifting the object. In this experiment however, the shift of the CoM caused by the object with a weight of 25g can be considered rather small and easily compensable by the support polygon's safety margin (see Fig. 4.1). A problem often encountered in the execution of the planned motion on the real robot platform is that

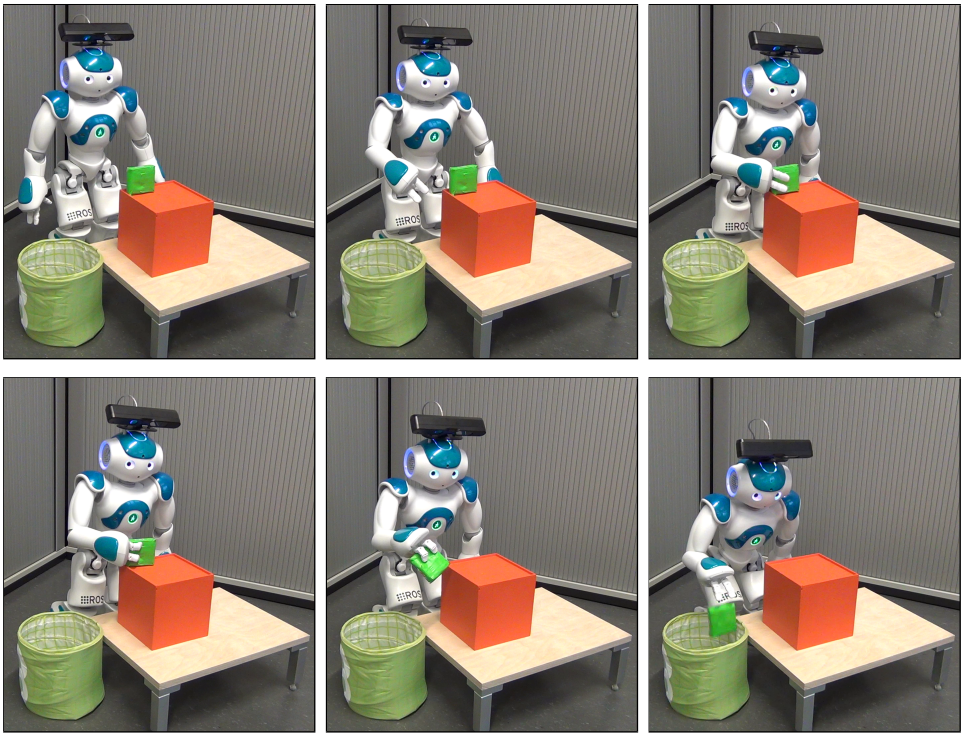

Figure 5.9: Execution of a whole-body manipulation plan for a small box. First, a collision-free motion for reaching the box is planned and executed. Once it has been grasped, planning is performed for manipulation of the box. Pictures are snapshots taken from a video (left to right, top to bottom).

the robot does not grasp the object tight enough to compensate for the initial sliding motion of the object on the lower box in the liftoff phase. Therefore, we decided to perform a predefined liftoff motion after the grasping motion to safely detach the object from the lower box before planning the manipulation task.

For planning the reaching motion our algorithm required 0.37 s, expanding 62 nodes. The generation of a goal pose took 3.28 s for the first motion planning problem. For the manipulation task planning took 0.18 s, expanding 38 nodes. A goal pose for the second motion planning problem was generated within 9.16 s.

5.9 Discussion of the Results

In the experiments we have evaluated the performance of our planner concerning the reachability of different hand poses, the capability of generating collision-free statically stable whole-body motions for body repositioning, manipulation of articulated objects as well as pick and place tasks. For all these scenarios whole-body motions were successfully planned and safely executed on the real robot platform. In the absence of obstacles seriously constraining the robot's motion, planning took less than 1 second to solve the respective query. As expected, the time required and number of nodes generated by our

algorithm significantly increased when planning is performed in environments cluttered by obstacles. Considering the complexity of these tasks however, a planning time around 7-12 seconds can still be considered satisfactory.

The sampling-based goal configuration generation takes up the major part of the overall time required to answer a motion planning query. Furthermore, the procedure is currently repeated for each new call to the planner. In order to avoid the future efforts for this operation one could think of storing the goal configurations already generated for previous motion plans in a database and reusing them as goal configurations for subsequent motion planning queries with the same desired final hand pose. Alternatively, one could speed up the goal configurations generation by using the generalized inverse kinematics for the whole-body kinematic structure. Another weakness of the current approach is that the probability of getting whole-body goal configurations for hand poses, reachable only from a specific double support configuration, is very low. This is especially the case for hand poses close to the floor.

Currently, planning is performed off-line in a virtual representation of the real world. Afterwards, the planned motion is executed in open-loop, i.e., without sensor feedback, on the real robot platform. Therefore, it is important that the relative position between the robot and the object(s) to be manipulated or avoided coincides in both worlds. Otherwise the robot may fail to successfully accomplish the assigned task in the real world, i.e., either collides with an obstacle or fails to reach the object or object's handle, although planning has previously been performed successfully.

For manipulation of articulated objects the robot first needs to reach the object's handle with its hand. Since it is intended to obtain a contact between the robot's palm of the hand and the handle after the reaching motion, the handle itself is not modeled as a collision object in the virtual representation of the world. This fact however implies that the initial pose of the robot's hand and thus the approach direction of the hand towards the handle has a strong impact on the success of the task execution. For now, the initial configurations for reaching and manipulating objects has been chosen such that the robot does not need to move its hand around the object's handle before approaching the final hand pose.

6 Conclusion

6.1 Summary

Planning whole-body motions for a humanoid robot is a challenging task due to the high number of degrees of freedom of the mechanical system and the variety of constraints involved. Independent from the planning scenario the robot needs to maintain stability, i.e., keep its CoM inside the support polygon, avoid self-collisions and collisions with obstacles and must respect its geometric and differential capabilities, such as limitations on the joint range and maximum actuator velocities. Due to the high-dimensional configuration space, a probabilistic approach has been chosen in this work to realize efficient whole-body motion and manipulation planning.

As the basis for our planner we used the generic RRT-CONNECT algorithm, which has already proven to be an efficient and fast algorithm to generate solutions for high-dimensional planning problems. With RRT-CONNECT two trees are initially rooted at a start and goal configuration and grown in the configuration space until they are capable of connecting to each other. As opposed to the basic RRT version, the CONNECT function makes the two trees to grow towards each other rather than solely exploring the configuration space at random.

For the expansion of the trees our planner relies on a pre-computed set of statically stable double support configurations. Guiding the expansion in this way accounts for the low probability of generating stable configurations when expanding a tree towards random samples from the configuration space.

To determine whether new configurations are valid, our planner performs a stability and collision check. The former procedure takes the masses of the robot links from a CAD model and evaluates whether the projection of the CoM is inside the boundaries of the scaled support polygon in the given configuration. The latter is based on an external collision checking library, called *FCL*, and relies on a specially designed low-vertex version of the original robot model mesh.

For planning with RRT-CONNECT a start and goal configuration is expected to be given in advance. Whereas the start configuration simply corresponds to the current configuration of the robot, the goal configuration is typically not known beforehand. Since we are considering manipulation actions in this work, we generate whole-body goal configurations with the robot's hand at a desired pose. For doing so, a sampling-

based procedure is applied. As can be see from the experiments, this operation takes up the major part of the overall time required to answer a motion planning query.

For manipulation planning we generate a discrete set of intermediate hand pose along the object trajectory. These hand poses are then used within the tree expansion procedure to obtain new whole-body poses satisfying the manipulation constraints.

In the experiments we have shown that our planner is capable to solve a variety of motion planning queries. Initially, we have illustrated that the range of reachable hand poses can be significantly enlarged by planning motions considering the whole-body kinematic structure. Moreover, we have proven that the planner is capable of generating collision-free statically stable whole-body motions in environments cluttered by obstacles. For the task of reaching and manipulating articulated objects, our planner has shown to reliably produce motion plans within a short amount of time. The same task was also successfully planned in the presence of an additional obstacle. Furthermore, we have presented that our algorithm enables the robot to perform successfully pick and place tasks, at least for objects of limited weight and dimension. All the motions plans generated by our planner were executed reliably on a real humanoid robot.

6.2 Future Work

In our current planning framework goal configurations are obtained by generating double support configurations from sampled whole-body poses and adapting the arm configuration according to the desired 5D hand pose. Afterwards, the configurations found are evaluated considering the manipulability measure for the arm chain and only the best among them is subsequently used for planning. As we have seen in the experiments, this procedure is very time consuming in comparison with the expenditure required for planning and is currently repeated for each new call to the planner. A possible extension to our approach to avoid unnecessary re-computation of goal configurations for 5D hand poses already encountered in previous motion planning problems would be to store all goal configurations generated in a database, so that they can be reused to solve future motion planning queries with the same desired final hand pose. Alternatively, one could think of entirely replacing the sampling-based goal configuration strategy with a procedure that solves the whole-body inverse kinematics problem for a given hand pose numerically through iterative convergence computation or analytically through a decomposition of the robot's kinematic chains [35].

At this stage, our planner uses only the best configuration from the set of goal configurations for planning. The remaining configurations are not used and discarded once a path has been successfully generated. Another approach, proposed by Ichnowski *et al.* in [51], is to keep all goal configurations generated and to root multiple goal trees at these configurations. Afterwards, a parallel search can be run using different threads of a multi-core computer. Following this idea, the time required for planning could be

further reduced. Moreover, the planner may generate multiple solution paths among which the best path can be selected according to some optimal criteria, e.g., considering the length of the paths.

For the expansion of the trees during the search for a path our planner relies on a pre-computed set of statically stable double support configurations. Thus, new candidate configurations are generated along the straight line configuration space path segment connecting a random configuration from the set with the nearest neighbor contained in the respective tree to be expanded. For manipulation of articulated objects, these configurations additionally need to be projected onto the constrained manifold in order to satisfy the manipulation constraints imposed by the object. With this approach, all constraints are satisfied at the waypoints of the solution path, but not necessarily in between them. In this work, this effect is compensated by using a small step size in the tree expansion procedure. Obtaining a continuous constraint satisfaction along the path can be realized by formulating the task specific constraints in the form of a control law [23]. In this way, the local planner could move a point directly on the constraint manifold without the need of applying projection techniques. Another approach was recently proposed by Sucan *et al.* in [52], where an approximation of the constraint manifold, referred to as the *Approximation Graph*, for a given set of geometric task constraints is computed off-line. Afterwards the vertices and edges of the graph are used to directly plan on the constraint manifold instead of planning in the full configuration space. Using the data structures computed off-line, it is said that the constraint manifold can be sampled very quickly and thus the planning time can be significantly reduced.

For the manipulation of articulated objects the position and orientation of the handle was expected to be given in advance in this work. In order to leverage the level of autonomy of the robot in the future we will determine the handle pose by using a depth sensor installed at the head of the NAO robot. Once the handle has been detected, the new vision module will trigger the planner for generating an appropriate whole-body reaching motion. Using the data of this sensor would also allow to perceive changes in the environment. With this information, one could early detect when a motion plan becomes unfeasible due to changes of the position of an object to grasp or obstacles to avoided. Thus, the motion execution could be stopped and re-planning could be performed from the current robot configuration.

A really challenging extension to probabilistic approaches is to include the possibility of the robot to change its support mode (single to double support or vice versa) during planning. This functionality is in particular required when the robot needs to perform footsteps to successfully accomplish a task. Additionally one needs to ensure in such cases that other constraints, e.g., arising from the contact of the robot's hand with an object, remain satisfied while performing the stepping motion. Methods addressing this problem, gathered in the literature so far, rather generate whole-body motions in an on-line fashion with task constraints being formulated within a control framework than relying on probabilistic planning methods [9, 11, 12, 13, 53].

Bibliography

[1] American Honda Motor Co. Inc. Asimo robot. http://asimo.honda.com/asimo-specs/, 2012.

[2] Kawada Industries Inc. HRP-2 robot. http://global.kawada.jp/mechatronics/, 2012.

[3] Aldebaran Robotics. Nao H25 Datasheet. http://www.aldebaran-robotics.com, October 2012.

[4] Korea Advanced Institute of Science and Technology. Hubo robot. http://www.kaist.edu/edu.html, 2012.

[5] Johannes Garimort, Armin Hornung, and Maren Bennewitz. Humanoid navigation with dynamic footstep plans. In *ICRA*, pages 3982–3987, 2011.

[6] American Honda Motor Co. Inc. Asimo opens a water bottle. http://asimo.honda.com/, November 2011.

[7] James J. Kuffner Jr. and Steven M. Lavalle. RRT-Connect: An efficient approach to single-query path planning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 995–1001, 2000.

[8] Oussama Kanoun, Florent Lamiraux, Pierre-Brice Wieber, Fumio Kanehiro, Eiichi Yoshida, and Jean-Paul Laumond. Prioritizing linear equality and inequality systems: application to local motion planning for redundant robots. In *Proceedings of the 2009 IEEE international conference on Robotics and Automation*, ICRA'09, pages 724–729, Piscataway, NJ, USA, 2009. IEEE Press.

[9] Oussama Kanoun, Jean-Paul Laumond, and Eiichi Yoshida. Planning foot placements for a humanoid robot: A problem of inverse kinematics. *Int. J. Rob. Res.*, 30(4):476–485, April 2011.

[10] Shuuji Kajita, Fumio Kanehiro, Kenji Kaneko, Kiyoshi Fujiwara, Kazuhito Yokoi, and Hirohisa Hirukawa. A realtime pattern generator for biped walking. pages 31–37, 2002.

[11] Duong Dang, Florent Lamiraux, and Jean-Paul Laumond. A framework for manipulation and locomotion with realtime footstep replanning. In *Humanoids*, pages 676–681. IEEE, 2011.

[12] E. Yoshida, O. Kanoun, C. Esteves, and J-P. Laumond. Task-driven support polygon reshaping for humanoids.

[13] Nicolas Mansard, Olivier Stasse, François Chaumette, and Kazuhito Yokoi. Visually-guided grasping while walking on a humanoid robot. In *ICRA*, pages 3041–3047, 2007.

[14] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Robotics: Modelling, Planning and Control*. Springer Publishing Company, Incorporated, 1st edition, 2008.

[15] Howie Choset, Kevin M. Lynch, Seth Hutchinson, George A Kantor, Wolfram Burgard, Lydia E. Kavraki, and Sebastian Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, Cambridge, MA, June 2005.

[16] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Norwell, MA, USA, 1991.

[17] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

[18] Lydia Kavraki, Petr Svestka, Jean claude Latombe, and Mark Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. In *IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 566–580, 1996.

[19] James Kuffner, Jr., Satoshi Kagami, Masayuki Inaba, and Hirochika Inoue. Dynamically-stable motion planning for humanoid robots, 2000.

[20] Michael Stilman. Task constrained motion planning in robot joint space. Technical Report CMU-RI-TR-06-43, Robotics Institute, Pittsburgh, PA, September 2006.

[21] Dmitry Berenson, Joel Chestnutt, Siddhartha Srinivasa, James Kuffner, and Satoshi Kagami. Pose-constrained whole-body planning using task space region chains. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids09)*, December 2009.

[22] Niko Vahrenkamp, Christian Scheurer, Tamim Asfour, James J. Kuffner, and Rüdiger Dillmann. Adaptive motion planning for humanoid robots. In *IROS'08*, pages 2127–2132, 2008.

[23] Giuseppe Oriolo and Marilena Vendittelli. A control-based approach to task-constrained motion planning. In *Proceedings of the 2009 IEEE/RSJ international conference on Intelligent robots and systems*, IROS'09, pages 297–302, Piscataway, NJ, USA, 2009. IEEE Press.

[24] Sebastien Dalibard, Alireza Nakhaei, Florent Lamiraux, and Jean-Paul Laumond. Whole-body task planning for a humanoid robot: a way to integrate collision avoidance. *2009 9th IEEERAS International Conference on Humanoid Robots*, pages 355–360, 2009.

[25] J.T Schwartz and M. Sharir. On the piano movers problem: II. General techniques for computing topological properties of real algebraic manifolds. In *Advances in Applied Mathematics*, volume 4, pages 298–351, 2083.

[26] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 2012. To appear.

[27] Bruno Siciliano and Oussama Khatib, editors. *Springer Handbook of Robotics*. Springer, Berlin, Heidelberg, 2008.

[28] C. O'Dunlaing and C. K. Yap. A retraction method for planning the motion of a disc. *Journal of Algorithms*, 6:104–111, 1982.

[29] Nancy M. Amato, O. Burchan Bayazit, and Lucia K. Dale. OBPRM: An obstacle-based PRM for 3D workspaces, 1998.

[30] Mark H. Overmars. The gaussian sampling strategy for probabilistic roadmap planners. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 1018–1023, 1999.

[31] T. Siméon, J.P. Laumond, and C. Nissoux. Visibility based probabilistic roadmaps for motion planning. *Advanced Robotics Journal*, 14(6), December 2000.

[32] J.-P. Laumond. *Robot Motion Planning and Control*. Springer-Verlag, Berlin, 1998. Available online at http://www.laas.fr/∼jpl/book.html.

[33] Petr Svestka. On probabilistic completeness and expected complexity of probabilistic path planning, 1996.

[34] Peter Corke. *Robotics, Vision and Control - Fundamental Algorithms in MATLAB®*, volume 73 of *Springer Tracts in Advanced Robotics*. Springer, 2011.

[35] Fumio Kanehiro, Eiichi Yoshida, and Kazuhito Yokoi. Efficient reaching motion planning and execution for exploration by humanoid robots. In *IEEE International Conference on Intelligent Robots and Systems*, 2012.

[36] Holger Täubig, Berthold Bäuml, and Udo Frese. Real-time swept volume and distance computation for self collision detection. In *IROS*, pages 1585–1592, 2011.

[37] Jia Pan, Sachin Chitta, and Dinesh Manocha. FCL: A general purpose library for collision and proximity queries. 2012.

[38] University of North Carolina. PQP: A proximity query package. GAMMA Research Group, Available from http://www.cs.unc.edu/∼geom/SSV/, 2005.

[39] Russell Smith. Open dynamics engine, 2008. http://www.ode.org/.

[40] L. Han and N. M. Amato. A kinematics-based probabilistic roadmap method for closed chain systems. In B. R. Donald, K. M. Lynch, and D. Rus, editors, *Algorithmic and Computational Robotics: New Directions*, pages 233–246. A.K. Peters, Wellesley, MA, 2001.

[41] J. Cortés. *Motion Planning Algorithms for General Closed-Chain Mechanisms*. PhD thesis, Institut National Polytechnique de Toulouse, Toulouse, France, 2003.

[42] M. Vukobratovic and B. Borovac. Zero-moment point – thirty five years of its life. *Int. Journal of Humanoid Robots*, 1, 2004.

[43] Sébastien Cotton, Andrew Murray, and Philippe Fraisse. Statically equivalent serial chains for modeling the center of mass of humanoid robots. In *Humanoids*, pages 138–144, 2008.

[44] Tsuneo Yoshikawa. Manipulability of robotic mechanisms. *The International Journal of Robotics Research*, 4(2):3–9, 1985.

[45] Stephen L. Chiu. Task compatibility of manipulator postures. *Int. J. Rob. Res.*, 7(5):13–21, October 1988.

[46] Tobias Kunz and Mike Stilman. Time-optimal trajectory generation for path following with bounded acceleration and velocity. In *Proceedings of Robotics: Science and Systems*, Sydney, Australia, July 2012.

[47] R. Smits. KDL: Kinematics and Dynamics Library. http://www.orocos.org/kdl.

[48] J. Sturm, C. Stachniss, and W. Burgard. A probabilistic framework for learning kinematic models of articulated objects. *Journal on Artificial Intelligence Research (JAIR)*, 41:477–626, August 2011.

[49] S. Chitta, I. Sucan, and S. Cousins. MoveIt! [ROS topics]. *Robotics Automation Magazine, IEEE*, 19(1):18 –19, 2012.

[50] Rosen Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute, August 2010.

[51] Jeffrey Ichnowski and Ron Alterovitz. Parallel sampling-based motion planning with superlinear speedup. In *IEEE International Conference on Intelligent Robots and Systems*, Vilamoura, Algarve, Portugal, October 2012.

[52] Ioan A. Sucan and Sachin Chitta. Motion planning with constraints using configuration space approximations. In *IEEE International Conference on Intelligent Robots and Systems*, Vilamoura, Algarve, Portugal, October 2012.

[53] M. Mistry, J. Nakanishi, G. Cheng, and S. Schaal. Inverse kinematics with floating base and constraints for full body humanoid robot control. In *Humanoid Robots, 2008. Humanoids 2008. 8th IEEE-RAS International Conference on*, pages 22 –27, dec. 2008.