Collège Doctoral du Languedoc-Roussillon

THÈSE

Pour obtenir le grade de

Docteur

Délivré par l'**Université de Montpellier**

Préparée au sein de l'école doctorale :
Information Structures Systèmes (I2S)

Et de l'unité de recherche :

Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier

Spécialité :
Systèmes Automatiques et Microélectroniques

Présentée par **Stanislas Brossette**

Viable Multi-Contact Posture Computation for Robots using Nonlinear Optimization on Manifolds

Soutenue le 3 Octobre 2016 devant le jury composé de

M. Nacim Ramdani	Professeur	Université d'Orléans	Rapporteur
M. Ronan Boulic	Docteur	EPFL	Rapporteur
M. Jean-Paul Laumond	Directeur de Recherche	LAAS-CNRS	Examinateur
M. Pierre-Brice Wieber	Chargé de Recherche	INRIA Grenoble	Examinateur
M. Adrien Escande	Chargé de Recherche	CNRS-AIST JRL	Co-encadrant de thèse
M. Abderrahmane Kheddar	Directeur de Recherche	CNRS-UM LIRMM	Directeur de thèse

Université Montpellier

Abstract

Humanoid robots are complex poly-articulated structures with nonlinear kinematics and dynamics. Finding viable postures to realize set-point task objectives under a set of constraints (intrinsic and extrinsic limitations) is a key issue in the planning of robot motion and an important feature of any robotics framework. It is handled by the so called posture generator (PG) that consists in formalizing the viable posture as the solution to a nonlinear optimization problem. We present several extensions to the state-of-the-art by exploring new formulations and resolution methods for posture generation problems. We reformulate the notion of contact constraints by adding variables to enrich the optimization problem and allow the solver to decide the shape of intersection of contact polygons, or of the location of a contact point on a non-flat surface. We present a reformulation of the posture generation problem that encompasses non-Euclidean manifolds natively and presents a more elegant and efficient mathematical formulation of it. To solve such problems, we implemented a new SQP solver that is particularly suited to handle non-Euclidean manifolds structures. By doing so, we have a better mastering in the way to tune and specialize our solver for robotics problems.

Keywords: posture generation; humanoid robotics; nonlinear optimization; manifolds.

Résumé

Un robot humanoïde est un système poly-articulé complexe dont la cinématique et la dynamique sont gouvernées par des équations non-linéaires. Trouver des postures viables qui minimizent une tâche objectif tout en satisfaisant un ensemble de contraintes (intrinsèques ou extrinsèques) est un problème central pour la planification de mouvement robotique et est une fonctionnalité importante de tout logiciel de robotique. Le générateur de posture (PG) a pour rôle de trouver une posture viable en formulant puis résolvant un problème d'optimisation non-linéaire. Nous étendons l'état de l'art en proposant de nouvelles formulations et méthodes de résolution de problèmes de génération de posture. Nous enrichissons la formulation de contraintes de contact par ajout de variables au problème d'optimisation, ce qui permet au solveur de décider automatiquement de la zone d'intersection entre deux polygones en contact ou encore de décider du lieu de contact sur une surface non plane. Nous présentons une reformulation du PG qui gêre nativement les variétés non Euclidiennes et nous permet de formuler des problèmes mathématiques plus élégants et efficaces. Pour résoudre de tels problèmes, nous avons developpé un solveur non linéaire par SQP qui supporte nativement les variables sur variétés. Ainsi, nous avons une meilleure maîtrise de notre solveur et pouvons le spécialiser pour la résolution de problèmes de robotique.

Mots-clés: génération de posture; robot humanoïde; optimisation non-linéaire; variétés.

Contents

Nomenclature

Roman Symbols

c_i	Constraint function
E	Set of index for which constraints are equality constraints
F	The set of linearized feasible directions
f	Objective function
F	a frame
f	a force resultant
I	Set of index for which constraints are inequality constraints
m	a force moment
W	the world frame
w	a wrench or a force
x	Optimization variable
x^*	Solution of the optimization problem

Greek Symbols

Ω	Feasible set

Other Symbols

$\mathbb{I}_n$	Matrix identity of dimension n
$\mathcal{L}(x, \lambda)$	Lagrangian function of the optimization problem

$\mathcal{M}$	A Manifold
$\mathbb{R}$	Real space
$\mathbb{R}_{\geq}0$	The set of positive Reals
S	Variable space
$T_x\mathcal{M}$	The tangent space of manifold $\mathcal{M}$ at point x
$\wedge$	cross product

Acronyms / Abbreviations

DoF	degrees of freedom
IK	Inverse Kinematics
IQP	Inequality constrained Quadratic Programming
KKT	Karush-Kuhn-Tucker first order optimality conditions
LICQ	Linear Independence Constraints' Qualification
PG	Posture Generation
QP	Quadratic Programming
s.t.	subject to
w.r.t	with respect to

Introduction

The ultimate goal of robotics is to make robots realize some tasks. The tasks, as well as the robot used to fulfill them, are various. For example, it can be a robotic arm building a car in a factory, a surgeon robot operating on a human, a submarine robot exploring the wreckage of a ship, or a humanoid robot exploring and fixing a destroyed plant.

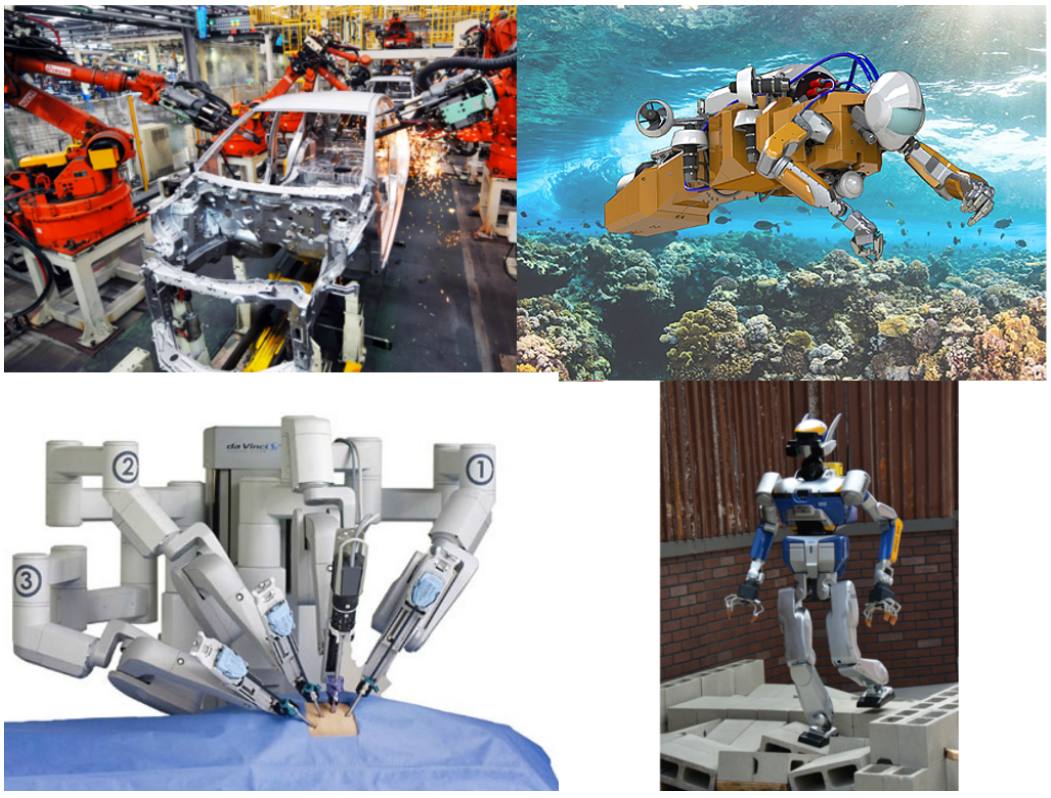


Figure 1 Various robots doing various tasks

The DARPA Robotics Challenge has shone some light on the humanoid robots. This competition brought together a large scope of robotics groups from universities, laboratories, and private companies around a common goal: teleoperate a robot to make it succeed in several challenges without any human physical intervention. The robots were expected to

drive a car, open a door, climb stairs, cross debris, drill a hole in a wall, etc. All those tasks can usually be broken down into sets of elementary tasks in human language. Some typical examples of tasks for a robot can be 'Put hand in contact with target', 'Put foot on next step', 'Avoid collision with that object', 'Maintain stability' or 'Look in that direction'. As is, those tasks do not mean anything for the robot. A robot is made of a collection of bodies that are linked together by joints actuated by motors. A robots configuration consists of the position and orientation of its base body, and the configuration of each of its joints. Satisfying a task requires that the joints of the robot reach a configuration for which the task is satisfied. The action of satisfying a task comes down to moving from an initial configuration to a goal. Computing the trajectory to follow and actually following it, are handled by the trajectory generator and controller of the robot. Trajectory generation and control are by themselves complete sub-fields in robotics, and they both often rely on being given some key postures to help guide the robot's motion(e.g. initial, intermediate and final configurations). Finding those configurations is the job of what we call the Posture Generator (PG).

There are two types of robots. The fixed-base robots and the mobile ones. As the name suggests, the first type of robots have fixed bases, their workspace is predefined by their geometry and they are usually fully actuated. This means that the robot has as many degrees of freedom (DoF) as they have actuators. Thus, any particular joint configuration leads to a unique robot posture. On the contrary, mobile robots are always underactuated such that the robot has more DoF than actuators. Each link of the robot is actuated, but the position of the base of the robot depends on the configuration of the joints as a well as on the locations of the contacts that the robot makes with the environment.

Humanoid robots are expected to move and achieve tasks in ways similar to humans. On flat surfaces, they can walk based on a cyclic motion and the locations of footsteps can be generated by a footstep planner using some simplified models to maintain the robot's stability. In cumbersome and unstructured environments, we humans move in a non-gaited acyclic way: we choose appropriate parts of our body to create contacts with the surrounding environment in order to support the motion of the remaining parts while avoiding obstacles. A whole motion is a sequence of contact creations and releases.

Since we are biped, we mostly use our feet to move. As the environment becomes more difficult to cross, hands may come into play together with feet to help with the motion. Narrow passages may even require other parts of our body (knees, elbows, back...) to make contact in order to support the motion.

Planning a sequence of contacts is thus a necessary step in devising a motion for a robot. Once the key stances of the motion have been identified by the planner, they can be used by the controller or the trajectory planner and finally the motion can be achieved by the robot.

All the aforementioned planning methods rely on the fact that we have a tool to decide if a proposed set of contacts is feasible or not for a given robot. The tool used for finding a robot configuration that satisfies a set of constraints, like the geometric constraints of contact or the stability of the robot, is called a 'Posture Generator' (PG) and the development of this tool is the main topic of this dissertation.

The mission of a posture generator, for a polyarticulated system, is to find a configuration in which the system satisfies a set of constraints. Closed-form solutions may be possible for relatively simple systems and tasks, like a 6 DoF robotic arm that is required to reach a point with its end effector. However, computing robot configurations to meet the requirements of a given set of tasks, within a viable state, is a recurrent problem whose complexity grows with that of the robot. When the robotic problem studied becomes too complex for closed-form solutions, it is formulated as a nonlinear optimization program and solved using optimization algorithms. The PG is a key tool for many robotics applications and as such, it is an important component of any robotics framework. It needs to be efficient at finding a solution when one exists, and at figuring out if a problem is not feasible. The speed of generating a multi-contact sequence is directly related to the quality of the PG, which is directly related to the quality of the optimization algorithm that it uses.

The variables of a robotics problem sometimes naturally belong to a non-Euclidean manifold $\mathcal{M}$, like $SO(3)$ for the 3D rotation of the base of a mobile robot. A manifold is a topological space that locally resembles a Euclidean space near each point. Each point of an n-dimensional manifold has a neighborhood that is homeomorphic to the Euclidean space of dimension n. An n-dimensional non-Euclidean manifold cannot always be globally parameterized over a subset of $\mathbb{R}^n$ without presenting problems of singularity. Most of the optimization solvers available make the assumption that the search space is Euclidean and thus, do not feature the capability of solving problems on non-Euclidean manifolds natively. But it is often possible to parametrize $\mathcal{M}$ over a Euclidean space of higher dimension with added constraints (e.g. unit quaternions, rotation matrices). It is then possible to solve those problems with classical solvers at the cost of additional variables, constraints and special treatments in the optimization problem. There exist some methods and algorithms to solve optimization problems on non-Euclidean manifolds with no substantial extra cost and guaranteeing a good coverage of the manifolds without facing parameterization singularities, and with the minimal number of parameters. However to our knowledge, these are focused on addressing non-constrained optimization.

Through this thesis, we develop our own nonlinear constrained optimization solver on manifolds and use it to solve optimization problems on their native search manifolds. We will exhibit their utility for posture generation problems as well as a few other types of

problems. This has the advantage that, unlike off-the-shelf solvers which are often a black box on which the user has very limited control, we have full control over that solver and can specialize/modify it to fit our needs.

Contributions and plan The main focus of this thesis is the formulation and resolution of problems of posture generation for robotics systems using nonlinear optimization on manifolds. Our contributions are of two types, on one hand, we propose some extensions and improvements on the way to formulate a posture generation problem, and on the other hand, we investigate new ways of solving those problems. The organization of this thesis is as follows:

- In Chapter 1 we give the state of the art of posture generation and optimization on manifolds.
- In Chapter 2, we present the detailed formulation of a posture generation problem.
- In Chapter 3 we present three different and unrelated contributions: two formulation extensions, one allowing to generate non-inclusive contacts between convex surfaces, the other is the exact derivation of the torques in the actuators of a robot. Then we present our endeavor to use the 'Lifted Newton Method' to solve posture generation problems.
- Chapter 4 describes the principles of nonlinear optimization over non-Euclidean manifolds and our implementation of a solver based on those principles.
- In Chapter 5, we present a framework that simplifies and extends the formulation of posture generation problems by formulating and solving the optimization problem over native non-Euclidean manifolds, managing automatically the variables of the problems and proposing a framework that formalizes and simplifies the writing of functions on geometric entities often used in robotics.
- In Chapter 6, we evaluate the performances of our solver on manifolds and of our posture generator, and then present a preliminary work on how to generate postures on a sensory acquired environment.

Chapter 1

State of the art and Problem definition

1.1 State of the art

1.1.1 Inverse Kinematics

Posture generation can be viewed as, and is sometimes called, Generalized Inverse Kinematics. The Inverse Kinematics(IK) problem consists in finding the joint configuration for an articulated multibody to complete a given set of tasks in order to generate believable motions. By definition, it is purely kinematics and has no regards for stability or other physics-related constraints. The IK problem has been widely studied and used in the fields of robotics, computer graphics, computer games, and animation. In some cases, for example with robotic arms that have less than 7 degrees of freedom, a closed-form solution can be found, in [AD03] the redundancy in a robot arm is exploited to devise a closed-form formula for its IK. But for more complicated cases, optimization methods are usually used.

Many approaches to solve IK problems use pseudo-inverse or Gauss-Newton based methods that are the Jacobian inverse and its variations, Jacobian transpose, damped least squares with and without singular values decomposition or selectively damped least squares [BMGS84, TGB00, Bai85, Wam86, NH86, BK05]. Those approaches are all computationally expensive and suffer from singularities [AL09].

In [Pec08] an alternative method based on a control approach is proposed, where no matrix manipulation is required, this approach is as fast as the damped least squares, but outperforms them in terms of handling singularities. The closed-loop inverse kinematics scheme presented in [Sic90] is a famous approach to solve IK problems in a control context. It uses a second order tracking scheme to guarantee satisfactory tracking performances.

Handling conflicting constraints in IK is a challenging problem to which [BB04] and [SK05] propose solutions by enforcing a number of priority levels among constraints in their resolution schemes to generate whole-body control of complex articulated figures.

Some statistical resolution approaches have been proposed in [CA08, HRE+08], with respectively a sequential Monte Carlo method and a particle filtering approach, they have the advantage to only use direct calculations and never require matrix inversions, thus, their strength shows particularly when solving problems with high numbers of degrees of freedom.

In [AL11, ACL16] the forward and backward reaching inverse kinematics(FABRIK) resolution method is introduced. It is based on a geometric iterative heuristic approach, where the bodies of a robot are moved iteratively and separately to reach a target with the end effector while maintaining the integrity of the robot's structure. This approach is simple, does not suffer from singularities and requires fewer iterations than most other IK methods. But it cannot easily be extended to take non-geometric constraints into account.

Finding a solution to an IK problem without regards for the stability of the solution is a common approach in the field of randomized path planning. [CSL02] proposes a method to generate many random configurations for closed loop systems with an increased probability that they be kinematically valid in terms of closure constraints, these configurations are then used in the construction of a Probabilistic RoadMap. In [LYK99], a similar approach is presented where instead of increasing its probability of validation, the closure constraint is enforced for each configuration by additional treatment.

1.1.2 Generalized Inverse Kinematics

The Generalized Inverse Kinematics refers to a problem similar to the Inverse Kinematics in the sense that it searches a joint configuration for an articulated figure to complete a task, but needs to do so while respecting other constraints, like ensuring the stability of the structure, respecting its torque limits, avoiding collisions with the environment or with itself, etc.

All the previously mentioned approaches are used to find a robot configuration solution to the geometric IK problem. To solve a Generalized Inverse Kinematics problem with stability constraints, one can first solve the associated IK with one of the methods presented above, and then test the stability of the IK solution, using methods such as the ones presented in [BL08] or [RMBO08] that determine if the configuration can be statically stable. This gives a rejection criterion for the proposed solution. If the solution can be stable, the optimal contact forces can be computed using the method proposed in [BW07] (which in turn allow computing the joint torques). Otherwise, another IK solution is generated and tested. And the process is repeated until a satisfactory solution is found. This type of sequential approach of posture generation problem resolution features a rejection criterion that can in some cases

be difficult to overcome, making this approach costly, for example, if there is a small number of contacts and very few configurations are stable.

Another way to solve the posture generation problem is to consider the complete problem as one, with the IK targets and all the other constraints (stability, collisions, torque limits, etc.) in a single nonlinear constrained optimization problem. In [ZB94], Zhao uses that approach to solve the IK problem. That is the approach that we and many others use to solve Generalized Inverse Kinematics problems. In [EKM06], that approach is employed to generate postures that are used to grow a tree of usable postures in a multi-contact planning algorithm. It was applied to automatically generate a sequence of contact sets and postures where the HRP-2 robot uses its hand to lean on a table in order to grasp an object otherwise out of reach. Interestingly enough, in that study, the C-code that computes the robot's kinematics is generated through Maple and the HuMAnS toolbox [WBBPG06] provided by INRIA. The constraints of contact, stability and collision are then computed on top of it. And the problem is solved by the CFSQP solver [LZT97]. In [HBL$^+$08], a similar posture generation approach is used to find viable postures for different legged robots on varied terrains in the context of a Probabilistic RoadMap planning, which is a planning approach based on random sampling of the configuration space. In [BK10a], Bouyarmane proposes a generalization of the formulation of posture generation problems for systems of multiple robots and manipulated objects. Also, he generalizes the notion of contacts by allowing them to bear contact forces or not and to not necessarily be coplanar or horizontal. This generic formulation enables the generation of postures with inter-robot contacts. The complete posture generation problem is then solved within a single nonlinear constrained optimization query, computing the contact forces and joint configuration at the same time, while ensuring the stability of all the actors, avoiding collision and respecting the robot's joint and torque limits.

In the past few years, our team has dedicated considerable efforts in proposing a general multi-contact motion planner to solve cases of non-gaited acyclic planning, using a posture generator as a backbone to select valid configurations. Given a humanoid robot, an environment, a start and a final desired postures, the planner generates a sequence of contact stances allowing any part of the humanoid to make contact with any part of the environment to achieve motion towards the goal. The planner's role is to grow a tree of contact stances iteratively; from a given posture, it tries to add or remove contacts one by one. The tree grows, following some heuristics until the solution is reached. For each set of contacts to add to the tree, a posture generation problem is solved in order to validate the feasibility of the set, if it is not feasible, the set is rejected. A typical experiment with an HRP-2 robot achieving such an acyclic motion is presented in [EKMG08], and the planner is thoroughly

described in [EKM13]. Extensions of this multi-contact planner to multi-agent robots and objects gathering locomotion and manipulation are presented in [BK12a], and preliminary validations with some DARPA challenge scenarios, such as climbing a ladder, ingress/egress a utility car or crossing through a relatively constrained pathway are presented in [BK12b]. Another way of planning a multi-contact scenario, which is actually often used when planners fail to find satisfactory solutions, is to do it manually, choosing iteratively which contacts to add and remove until the goal is reached. This type of approach is used when the plan to execute is complex and when a lot of fine tuning of the postures is required, as in [VKA+16], where a sequence of postures allowing the HRP-2 robot to climb a vertical ladder is generated and tuned manually. Those postures are provided as input to a finite state machine that builds additional intermediary tasks and specific grasps procedures to be realized on the real robot by a multi-objective model-based QP controller. Instead of feeding the key postures directly to the controller and trusting that it will find a path to connect them, one can take an additional step and compute a dynamically viable trajectory between successive steps that the controller will then have to follow as presented in [LVYK13]. That allows to take advantage of the dynamic effects and produce motions with better performances in terms of completion time, energy consumption, or other criterions. Although the key postures are guaranteed to be viable by the posture generator, the whole trajectory might not be. Interval analysis methods can be leveraged to ensure the satisfaction of constraints over the whole duration of a motion, as proposed in [LRF11].

In many planning approaches [KNK+05, Che07, HBL+08, KRN08, BK11a] the planning problem is decomposed in two stages, first the sets of contacts and associated postures are planned and then the motions to go from one set to another are computed. Those two stages are loosely coupled, which can result in suboptimal use of the contacts available. [MTP12] presents a different approach to contact planning where some additional terms and variables are added to the optimization problem to decide whether a contact pair should be active(and bear forces) at any given time during the motion. The contact sets, as well as the postures associated are discovered along with the entire movement's trajectories by using contact invariant optimization. This allows exploiting any synergies that might exist between the contact events and the motion trajectory.

In [LME+12], posture generation is used to find the optimal posture and position of contacts to optimally realize a manipulation task along a path while satisfying geometric and kinematic constraints as well as force and torque constraint. The task is defined as a path and force to follow with an end effector. Instead of finding a sequence of unrelated postures along the path, all the postures are found by solving a single optimization problem in which successive postures are coupled by constraints to ensure that the foot positions

remain constant during the task, even though the rest of the body can move. This approach allows the robot or virtual character to apply manipulation forces as strongly as possible while avoiding foot slippage. It also allows taking external perturbations into account to generate more robust postures.

Another utilization of posture generation was presented in [SLL+07], that deals with the problem of object reconstruction. An object is presented to the robot in order to generate automatically its 3D model. To do so, the object is observed from several different angles with the camera that is mounted in the robot's head. The satisfactory postures of the robot to complete this task are obtained by solving a posture generation problem to which a set of constraints defining the direction in which the robot should look and a minimum distance from the object are added. In that work, the observation directions are predefined, [FSEK08] presents an approach where the choice of the best observation direction is left to the posture generator. The representation of the object is iteratively carved into a block of 3D voxels, each new point of view chosen to allow to carve it a little more precisely. It uses a modified cost function to evaluate the amount of information on the object that a given point of view can provide. The posture of the robot that optimizes that cost function is then found by the posture generator under the classical constraints of stability, collision avoidance, and other robot's limitations.

1.1.3 Optimization

The resolution of a posture generation problems is often done by solving a nonlinear optimization problem which consists in finding the optimal point that minimizes a cost function, possibly subject to constraints, the cost and constraints functions being possibly nonlinear.

Optimization algorithms can be derivative-free or not. The computation of the derivatives of the functions involved in the problem is a common source of error. The strength of the derivative-free approaches is that the user does not need to implement the derivatives. But those approaches are much slower than their counterparts using derivatives. One way to avoid implementation errors in the derivatives computation is to use finite differences to compute them. This method is very slow, especially when the dimension of the problem is large. In order to design an efficient optimization algorithm, we will focus on methods using derivative information and will implement those derivatives' computations. The finite difference approach can be used to verify the correctness of the computed derivative.

Nonlinear optimization is a wide topic that has been extensively studied in the past. One can find some excellent reference books about it, such as [NW06, BGLA03, BV04].

Furthermore, several off-the-shelf solvers are available and have been widely used in for solving robotics problems. The CFSQP solver [LZT97] was used in [EK09a] and [EKM13]

where thousands of HRP-2 posture generation queries were made to explore the feasible space. The IPOPT solver [WB06] has been used in [VKA+14], [VKA+16], [BK12a] where the posture generator had been extended to handle multi-robot problems and more complex and various contact models. The SNOPT solver [GMS05] was used in [DVT14] to plan dynamic whole-body trajectories. Several nonlinear optimization solvers are available and have been packaged for use in robotics problems in the Roboptim Framework [MLBY13], such as IPOPT, CFSQP, CMinPack [Dev07], NAG [TNAG], KNITRO [BNW06] and PGSolver(the solver that we develop in this thesis).

Traditionally, optimization problems are solved over Euclidean spaces. When the need comes to find a solution to an optimization problem in a non-Euclidean space $\mathcal{M}$, the commonly used method is to represent the elements of $\mathcal{M}$ in a Euclidean manifold of higher dimension, and enforcing that solution of the optimization problem should lie on $\mathcal{M}$ by adding the necessary constraints to the problem. We will detail the drawbacks of such approaches in Chapter 4. Alternatively, there exist some method to run an optimization algorithm directly on a non-Euclidean manifold, as it is presented in great details in the book [AMS08]. A non-Euclidean manifold, can be thought of as a space that is locally Euclidean, but not globally. For example, a sphere, if one looks in a small enough neighborhood, looks Euclidean, just like the surface of a giant sphere like the earth looks flat for a human being standing on it. Based on the fact that manifolds are locally Euclidean, the classic properties of distance, derivatives, and in general all the Euclidean geometry can be used locally. Based on that property, several optimization methods traditionally used on Euclidean manifolds have been extended to manifolds. The gradient methods have been extended to manifolds in [Lue72, Gab82]. The Newton methods on manifolds which have better convergence rates were extended to manifolds in [Gab82, SH98, Smi13]. And Quasi-Newton methods in [Gab82]. Those approaches are only meant to solve unconstrained optimization problems, which is not enough to solve posture generation problems, where the presence of constraints in unavoidable.

The main idea to optimize on manifolds is that we use a local map between the neighborhood of the current iterate and its Euclidean tangent space in order to run an optimization step and choose an increment in the tangent space that is mapped back to the manifold. Once the iterate has been incremented, the process is repeated with the map and tangent space associated with the new iterate. This comes down to re-parameterizing the problem around the current iterate at each iteration.

In the field of robotics, we are only aware of the work of Schulman *et al.* [SDH+14], where the authors explain the adaptation of their solver to work on $SE(3)$. In the field

of computer vision, and especially for solving pose estimation problems, optimization on manifolds is often used, e.g. [HWFS11, LHM00].

In this thesis, we develop a nonlinear solver on manifolds that can handle constraints, and were largely inspired by the work of Fletcher concerning the notions of Sequential Quadratic Programming without a penalty method [Fle06, Fle10, FL00], along with other optimization approaches that we adapted to work with manifolds.

1.2 Problem Definition

We consider the problem where we have a robotic system and we want it to realize a set of tasks T_i. We denote $\mathbf{q}$ the joint configuration (n joints + base) of the robot and $\mathbf{f} = \{\mathbf{f}_i, i \in [1,m]\}$ represents the m contact forces applied on the robot. Each task T_i can be represented by a set of equality and inequality equations:

$$\begin{cases} g_i(\mathbf{q},\mathbf{f}) = 0 \\ h_i(\mathbf{q},\mathbf{f}) \geq 0 \end{cases} \tag{1.1}$$

For example, the task of making contact between a body of the robot and a part of the environment can be represented in such a way.

In addition to satisfying the tasks T_i, the solution configuration to our problem must describe a viable posture of the robotic system, in the sense that it ensures its integrity. Which translates into the following list of constraints. It must respect the joint limits of the robot (1.2), as well as its torque limits (1.3). Equation (1.4) translates the auto-collision avoidance constraint between a pair of bodies $\{r_i, r_j\}$ given by $\mathcal{I}_{\text{auto}}$ where d is the minimal distance between two bodies. Equation (1.5) denotes the collision avoidance constraint between a robot body r_i and an object of the environment O_k defined in $\mathcal{I}_{\text{coll}}$. ε_{ij} and ε_{ik} denote the smallest acceptable distance for their respective constraints. The stability of the robot is ensured by the respect of equation (1.6), which is the Euler-Newton equation(or a simplification of it). To avoid slippage of the contacts, the contact force must remain in the Coulomb friction cone, which is enforced by equation (1.7). Equations (1.8) and (1.9) translate the constraints to satisfy task T_i.

The set of satisfying configurations can be defined by $\mathcal{Q}$:

$$\mathcal{Q} = \{\mathbf{q}, \mathbf{f}\} : \begin{cases} q_i^- \leq q_i \leq q_i^+ \quad \forall i \in [1,n] & \text{joint limits} \quad (1.2) \\ \tau_i^- \leq \tau_i(\mathbf{q},\mathbf{f}) \leq \tau_i^+ \quad \forall i \in [1,n] & \text{torque limits} \quad (1.3) \\ \varepsilon_{ij} \leq d(r_i(\mathbf{q}), r_j(\mathbf{q})), \quad \forall (i,j) \in \mathcal{I}_{\text{auto}} & \text{auto-collision} \quad (1.4) \\ \varepsilon_{ik} \leq d(r_i(\mathbf{q}), O_k), \quad \forall (i,k) \in \mathcal{I}_{\text{coll}} & \text{collision} \quad (1.5) \\ s(\mathbf{q},\mathbf{f}) = 0 & \text{stability} \quad (1.6) \\ c(\mathbf{f}_i) \geq 0 \quad i \in [1,m] & \text{slippage avoidance} \quad (1.7) \\ g_i(\mathbf{q},\mathbf{f}) = 0 & \text{task i's equalities} \quad (1.8) \\ h_i(\mathbf{q},\mathbf{f}) \geq 0 & \text{task i's inequalities} \quad (1.9) \end{cases}$$

We illustrate those constraints in figure 1.1 and will explicit them in more details in Chapter 2.

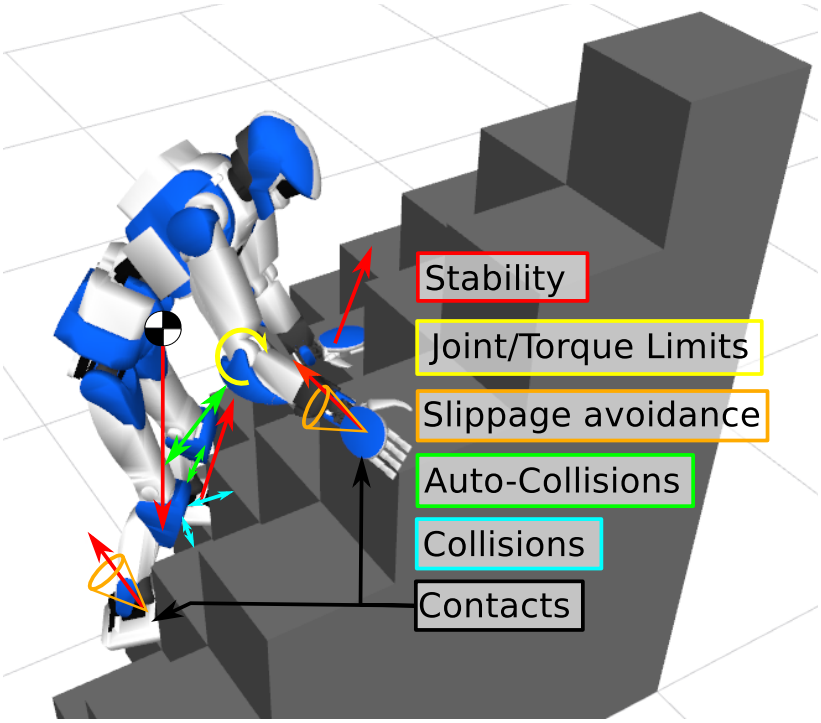


Figure 1.1 HRP4 on a stack of cubes. The color of each box corresponds to the color of the constraint it depicts.

In addition to satisfying the set of constraints T_i while being viable, one can require the searched posture to be optimal in the sense of some cost function f. Defining a cost function can help the user control the result of the problem, by for example minimizing the distance to a reference posture or maximizing the contact force applied by one end-effector. In [EK09b]

the cost function was used to guide the posture toward the end of a planning problem. A sum of different criterions can be used as cost function. The optimization problem that describes the posture generation problem becomes:

$$\begin{aligned} &\min_{\mathbf{q},\mathbf{f}}. f(\mathbf{q},\mathbf{f}) \\ &\text{s.t. } \{\mathbf{q},\mathbf{f}\} \in \mathcal{Q} \end{aligned} \tag{1.10}$$

This type of problem is called a nonlinear constrained optimization problem and can be formulated in a more generic fashion as:

$$\begin{aligned} &\min_{x}. f(x) \\ &\text{s.t. } \begin{cases} c_i(x) = 0, \ \forall i \in E \\ c_i(x) \geq 0, \ \forall i \in I \end{cases} \end{aligned} \tag{1.11}$$

Where x is the optimization variable we want to find that minimizes the cost function $f(x)$ while satisfying the equality constraints $c_i(x) = 0, \ \forall i \in E$ and inequality constraints $c_i(x) \geq 0, \ \forall i \in I$. Such problems can be solved by a nonlinear optimization solver. In Appendix A, we present some principles of nonlinear optimization in unconstrained and constrained cases.

In this thesis, we use the off-the-shelf solvers IPOPT [WB06] in the beginning and later we tackle the development of our own nonlinear solver and then use it. From this point forward, we formulate and solve posture generation problems as nonlinear constrained optimization problem. And in the next chapter, we focus on formulating all the basic functions and algorithms used to describe robotics constraints and cost function in the formalism of nonlinear optimization.

Chapter 2

Posture Generation: Problem Formulation

In this Chapter, we present in detail the formulation of a posture generation problem. We present the algorithms used to compute the kinematics of a robot and its derivatives as well as the joint torques. Then we formulate some classical functions that are often used in posture generation: joint limits, contact constraints, collision avoidance, stability, torque limits and friction cones. In Figure 1.1, we illustrate those constraints with the result of a posture generation problem where the HRP4 robot must climb on a stack of cubes while being statically stable, respecting its joint and torque limits, the contact forces must remain in the friction cones and the robot must avoid auto-collisions and collisions with the environment.

2.1 Forward Kinematics

In this section, we present a formulation of robotic systems that allows specifying most of the typical constraints encountered in robotics problems.

We consider a robotic system made of n_B bodies and $n_J (= n_B + 1)$ joints. The global structure of the robot is described by an ordered graph called multibody graph. The base body (World) has index 0 and other bodies get different positive integer index. We denote the body of index i, B_i. B_0 refers to the World. Each body B_i has its reference frame F_i attached to it. F_0 denotes the World frame. Bodies are linked together by joints that also are indexed by positive integers, we denote the joint of index i, J_i, and the body that comes after it is B_i. Each joint defines the relation between its predecessor and successor bodies. For joint J_i, they are respectively denoted $pred(i)$ and $succ(i)$, and $B_{pred(i)}$ is called the parent body of $B_{succ(i)}$. We denote $\lambda(j)$ the index of the parent body of B_j. The number of degrees of

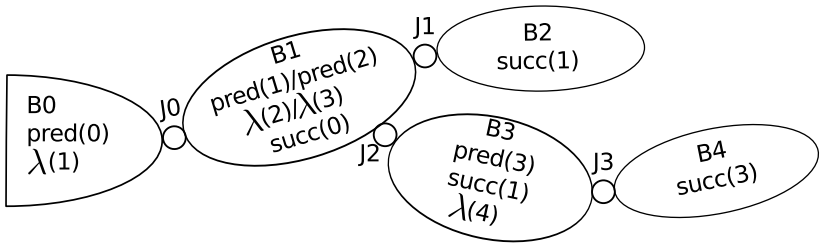


Figure 2.1 MultiBody graph

freedom of J_i is denoted dof_i^J and the number of degrees of freedom of the whole robot is denoted dof. Figure 2.1 illustrates this numbering system for a simple robot with 4 joints and 5 bodies (including the basis)

The geometric relations between bodies and joints are described through transformations between their reference frames. We use transformations as described in the Spatial Vector Algebra chapter of 'Rigid Body Dynamics Algorithm' by Roy Featherstone [Fea07]. Motion vectors (vectors describing motion quantities as positions, velocities and accelerations) and their force counterpart are defined in [Fea07].

For any 3D vector $v \in \mathbb{R}^3$, $\hat{v}$ denotes the 3×3 skew-symmetric matrix such that $\hat{v}u = v \wedge u$. Where $\wedge$ denotes the cross product operator.

Let A and B be Cartesian frames with origins O and P respectively. Let $\mathbf{t}$ be the coordinate vector expressing $\overrightarrow{OP}$ in A. And $\mathbf{R}$ be the rotation matrix that transforms 3D vectors from A to B coordinates. The transformation from A to B for a motion vector is defined by:

$$ {}^{B}X_{A} = \begin{bmatrix} \mathbf{R} & \mathbf{0} \\ -\mathbf{R}\hat{\mathbf{t}} & \mathbf{R} \end{bmatrix} \tag{2.1} $$

Its inverse is:

$$ {}^{B}X_{A}{}^{-1} = {}^{A}X_{B} = \begin{bmatrix} \mathbf{R}^T & \mathbf{0} \\ \hat{\mathbf{t}}\mathbf{R}^T & \mathbf{R}^T \end{bmatrix} \tag{2.2} $$

The transformation from A to B for a force vector is defined by:

$$ {}^{B}X_{A}^{*} = \begin{bmatrix} \mathbf{R} & -\mathbf{R}\hat{\mathbf{t}} \\ \mathbf{0} & \mathbf{R} \end{bmatrix} \tag{2.3} $$

Its inverse is:

$$ {}^{B}X_{A}^{-*} = {}^{A}X_{B}^{*} = \begin{bmatrix} \mathbf{R}^T & \hat{\mathbf{t}}\mathbf{R}^T \\ \mathbf{0} & \mathbf{R}^T \end{bmatrix} \tag{2.4} $$

Each joint J_i is defined by a static transformation $X_i^x = \{\mathbf{R}_i^x, \mathbf{t}_i^x\}$ between the reference frame of its predecessor body and its own reference frame. Each joint J_i is associated with a motion subspace which representation matrix is denoted $\mathbf{S}_i$. Each column of $\mathbf{S}_i$ described a degree of freedom of J_i its upper part for the rotations and lower for translations (see Section 2.2).

$$\mathbf{S}_i = \begin{bmatrix} S_{i,0}^R & \cdots & S_{i,j}^R & \cdots & S_{i,dof}^R \\ S_{i,0}^t & \cdots & S_{i,j}^t & \cdots & S_{i,dof}^t \end{bmatrix} \tag{2.5}$$

For a given joint configuration q, the transformation due to the joint J_i current configuration from its reference frame to the reference frame of its successor body is denoted $X_i^J(q) = \{\mathbf{R}_i^J(q), \mathbf{t}_i^J(q)\}$.

The transformation between $B_{\lambda(i)}$ and B_i is denoted $X_i^{PtS}(q) = \{\mathbf{R}_i^{PtS}, \mathbf{t}_i^{PtS}\}$ (PtS stands for 'Parent to Son') can then be computed as:

$$X_i^{PtS}(q) = {}^iX_{\lambda(i)}(q) = X_i^J(q)X_i^x \tag{2.6}$$

Let $\kappa(i) = \{0, i_1, i_2 \dots i\}$ be the list of indexes of successive joints going from B_0 to B_i. It can easily be computed by adding iteratively the parent of the current body:

Algorithm 1 Joint Path to B_i

$j \leftarrow i$, $\kappa(i) = [i]$
while $j \neq 0$ **do**
  $j \leftarrow \lambda(j)$
  $\kappa(i) \leftarrow [\kappa(i),\ j]$
end while

The transformation from the World base to B_i is denoted ${}^iX_0(q) = \{{}^i\mathbf{R}_0(q), {}^i\mathbf{t}_0(q)\}$. The formula (2.6) can be used iteratively on all bodies of the robot to obtain the expression of ${}^iX_0(q)$.

We obtain the full expression of ${}^iX_0$ as:

$${}^iX_0(q) = \prod_{j \in \kappa(i)} X_j^J(q)X_j^x = \prod_{j \in \kappa(i)} {}^jX_{\lambda(j)} = {}^iX_{\kappa(1)}\ {}^{\kappa(1)}X_{\kappa(2)} \dots {}^{\kappa(\text{end}-1)}X_W \tag{2.7}$$

Which can be computed recursively by a Forward Kinematics algorithm:

In the following section, we provide some detailed description of how to compute $X_J(q)$ for a variety of useful joints. Using the joint descriptions and the Forward Kinematics algorithm, we are able to explicit a relation between q the joint parameters of the robot and the 3D position and orientation of any geometric quantity defined in the reference frame of

Algorithm 2 Forward Kinematics

for $i = 0 : n_J$ **do**
  if $\lambda(i) \neq -1$ **then** ${}^iX_0 = {}^iX_{\lambda(i)}\ {}^{\lambda(i)}X_0$
  else ${}^iX_0 = X_i^{PtS}$
  end if
end for

a body of the robot. Given a transformation ${}^pX_i$ defined in the frame of B_i, its value in the world frame is given by ${}^pX_0(q) = {}^pX_i\ {}^iX_0(q)$

2.2 Joints formulations

The entire geometry of our system is described by the list of static transformations X_j^x and of joint transformations $X_j^J(q)$. In this section, we explicit the descriptions and formulations of several useful joints.

Let us consider a joint J that governs the transformation between two frames $F_1 = \{O_1, x_1, y_1, z_1\}$ and $F_2 = \{O_2, x_2, y_2, z_2\}$. The most common type of joint encountered in robotics systems is the revolute joint, that allows a rotation around a fixed axis. If J is a revolute joint around the axis (O_1, z_1) with parameter q, its motion subspace, rotation, and translation are as follows:

Joint type	S	*Rotation*	*translation*
Revolute (O_1, z_1)	$\begin{bmatrix}0\\0\\1\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}1 & 0 & 0\\0 & \cos(q) & \sin(q)\\0 & -\sin(q) & \cos(q)\end{bmatrix}$	$\mathbf{0}_{3\times1}$

Similar formulas can be devised for rotations around any other axis, provided that R describes the rotation of angle q around that axis.

In the case of a prismatic joint, all rotations are blocked, and only one translation along a given axis is allowed. A prismatic joint along x_1 is described by the following formulas:

Planar joints are also frequently used in robotics. A planar joint describes a plan sliding on another plan, assuming that the normal to both plans is $z_1 = z_2$ this type of joint allows free rotation of F_2 around z_1 and translations along x_1 and y_1. We denote $q = \{q_1, q_2, q_3\}$ the joint parameters, q_1 corresponding to the rotation and $q_2,\ q_3$ to the translations. We get:

Joint type	S	*Rotation*	*translation*
Prismatic (x_1)	$\begin{bmatrix}0\\0\\0\\1\\0\\0\end{bmatrix}$	$\mathbf{1}_{3\times3}$	$\begin{bmatrix}q\\0\\0\end{bmatrix}$

Joint type	S	*Rotation*	*translation*
Planar (z_1)	$\begin{bmatrix}0&0&0\\0&0&0\\1&0&0\\0&1&0\\0&0&1\\0&0&0\end{bmatrix}$	$\begin{bmatrix}1&0&0\\0&\cos(q_1)&\sin(q_1)\\0&-\sin(q_1)&\cos(q_1)\end{bmatrix}$	$\begin{bmatrix}\cos(q_1)q_2-\sin(q_1)q_3\\\sin(q_1)q_2+\cos(q_1)q_3\\0\end{bmatrix}$

A spherical joint blocks all translations and allows all rotations. This joint must be parameterized by a 3D rotation. The space of 3D rotations $SO(3)$ can be represented in many different ways. The simplest and most intuitive way to parameterize $SO(3)$ is to use Euler Angles. It comes down to decomposing the 3D rotation into a succession of three 1D rotations around different axes. For example, the roll, pitch, yaw is a succession of a rotation of F_1 around its x axis, followed by a rotation of the resulting frame around its y axis and a rotation of the resulting frame around its z axis. The rotation matrix for such a rotation is given by:

$$\mathbf{R}=\begin{bmatrix}1&0&0\\0&\cos(q_3)&\sin(q_3)\\0&-\sin(q_3)&\cos(q_3)\end{bmatrix}\cdot\begin{bmatrix}\cos(q_2)&0&-\sin(q_2)\\0&1&0\\\sin(q_2)&0&\cos(q_2)\end{bmatrix}\cdot\begin{bmatrix}\cos(q_1)&\sin(q_1)&0\\-\sin(q_1)&\cos(q_1)&0\\0&0&1\end{bmatrix} \quad (2.8)$$

Euler Angle formulations have the advantage to be simple and intuitive. There are many other possible choices of axes to define a Euler Angle 3D rotation. But they all suffer from singularities (the gimbal lock), which happens when two of the three rotation axes become aligned. In such a configuration, the only rotations possible are one rotation around the two aligned axis and one rotation around the third axis. Thus, one degree of freedom is lost. Those singularities are prohibitive for the use of that type of formulation in a posture generation. In [Gra98], Grassia states that any attempt to parameterize the entire set of 3D rotations by an open subset of Euclidean space(as do Euler angles) will suffer from gimbal lock. Note that this singularity is only due to the user's choice of parameterization, it is not

intrinsic to the manifold $SO(3)$. It is possible to parameterize $SO(3)$ without having to face singularities by parameterizing it over another non-Euclidean manifold. The most common ones are the set of unit quaternion embedded in $\mathbb{R}^4$ and the set of rotation matrices embedded in $\mathbb{R}^{3\times3}$. With the unit quaternion parameterization, a variable on $SO(3)$ is represented by 4 parameters $q = [q_w, q_x, q_y, q_z]$, and it is necessary to ensure that the quaternion is of norm 1, $\{q \in \mathbb{R}^4 : ||q|| = 1\}$. Similarly, if a variable is parameterized by a rotation matrix, then the matrix M representing it has 9 parameters and M must be orthogonal and have determinant 1: $\{M \in \mathbb{R}^{3\times3} : M^T M = \mathbb{I}_3 \ \& \ \det(M) = 1\}$. Similar issues can be found with the parameterization of other non-Euclidean manifolds, like $S2$ for example.

A quaternion $q = [q_w, q_x, q_y, q_z]$ is a unit quaternion iff $q_w^2 + q_x^2 + q_y^2 + q_z^2 = 1$. It represents a rotation of angle θ around an axis $\mathbf{u}$ such that:

$$q_w = \cos(\theta/2) \tag{2.9}$$
$$q_x = \sin(\theta/2)u_x \tag{2.10}$$
$$q_y = \sin(\theta/2)u_y \tag{2.11}$$
$$q_z = \sin(\theta/2)u_z \tag{2.12}$$
$$\tag{2.13}$$

The rotation matrix associated with this quaternion is:

$$\mathbf{R} = 2\begin{bmatrix} \frac{1}{2} - {q_y}^2 - {q_z}^2 & q_x q_y - q_z q_w & q_x q_z + q_y q_w \\ q_x q_y + q_z q_w & \frac{1}{2} - {q_x}^2 - {q_z}^2 & q_y q_z - q_x q_w \\ q_x q_z - q_y q_w & q_y q_z + q_x q_w & \frac{1}{2} - {q_x}^2 - {q_y}^2 \end{bmatrix} \tag{2.14}$$

That formulation does not suffer from singularities, but it requires to maintain 4 parameters for a 3D rotation. And those 4 parameters must satisfy the unit norm constraint which in turn would become an additional constraint in the optimization formulation. Given a parameter set $q = \{q_w, q_x, q_y, q_z\}$, we get the following table.

Joint type	S	*Rotation*	*translation*
Spherical	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$2\begin{bmatrix} \frac{1}{2} - {q_y}^2 - {q_z}^2 & q_x q_y - q_z q_w & q_x q_z + q_y q_w \\ q_x q_y + q_z q_w & \frac{1}{2} - {q_x}^2 - {q_z}^2 & q_y q_z - q_x q_w \\ q_x q_z - q_y q_w & q_y q_z + q_x q_w & \frac{1}{2} - {q_x}^2 - {q_y}^2 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$

Finally, a free joint allows free motion of its successor body with respect to its predecessor body. It can be viewed as a combination of a spherical joint and 3 perpendicular prismatic joints. Given a parameter set $q = \{q_w, q_x, q_y, q_z, t_x, t_y, t_z\}$, we get the following table.

Joint type	S	*Rotation*	*translation*
Spherical	$\mathbb{I}_6$	$2\begin{bmatrix} \frac{1}{2} - {q_y}^2 - {q_z}^2 & q_x q_y - q_z q_w & q_x q_z + q_y q_w \\ q_x q_y + q_z q_w & \frac{1}{2} - {q_x}^2 - {q_z}^2 & q_y q_z - q_x q_w \\ q_x q_z - q_y q_w & q_y q_z + q_x q_w & \frac{1}{2} - {q_x}^2 - {q_y}^2 \end{bmatrix}$	$\mathbf{R}^{-1}\begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$

2.3 Jacobian computation

For the sake of solving our problem with a nonlinear optimization algorithm, it is useful to compute the derivatives of every function used as constraint or cost with respect to any variable of the problem. The transformations ${}^iX_0$ are used in many functions, therefore, having an efficient algorithm to compute their derivatives and the derivatives of any transformation defined in B_i is necessary.

Given a static transformation ${}^pX_i$ defined in body B_i. Its expression in the world frame is ${}^pX_0 = {}^pX_i\ {}^iX_0$ and its expression in the frame of B_j is ${}^pX_j = {}^pX_i\ {}^iX_0\ {}^jX_i^{-1}$.

We denote Jac_i the Jacobian of body i, and $\text{Jac}_i(X)$ the jacobian of the frame defined by X in the referential of body i. q_i is the part of q that corresponds to the degrees of freedom of joint J_i. We denote $\text{Jac}_i.\text{cols}(j)$ the columns of Jac_i associated with joint J_j. The jacobian of the frame defined by ${}^pX_i$ in B_i with respect to q_j is given by

$$\text{Jac}_i({}^pX_i).\text{cols}(j) = {}^pX_j\ S_i \tag{2.15}$$

The complete jacobian of a body Jac_i is a $6 \times \text{dof}$ matrix that can be computed by using the formula 2.15 on every index j in $\kappa(i)$ and filling the rest of Jac_i with zeros.

The algorithm that we use to compute $\text{Jac}_i({}^pX_i)$ writes as follows:

Algorithm 3 Jacobian Computation

$\text{Jac}_i({}^pX_i) = \mathbf{0}_{6\times\text{dof}}$
${}^pX_0 = {}^pX_i\ ({}^iX_0)^{-1}$
for $j = 0 : \text{size}(\kappa(i))$ **do**
  $k \leftarrow \kappa(j)$
  ${}^pX_k = {}^pX_0\ ({}^kX_0)^{-1}$
  $\text{Jac}_i({}^pX_i).\text{cols}(k) = {}^pX_k\ S_k$
end for

We write the jacobian of each body at its origin as follows:

$$\mathbf{Jac}_i^0 = \begin{bmatrix} \frac{\partial^i \mathbf{R}_0}{\partial q_0} & \cdots & \frac{\partial^i \mathbf{R}_0}{\partial q_j} & \cdots & \frac{\partial^i \mathbf{R}_0}{\partial q_{dof}} \\ \frac{\partial^i \mathbf{t}_0}{\partial q_0} & \cdots & \frac{\partial^i \mathbf{t}_0}{\partial q_j} & \cdots & \frac{\partial^i \mathbf{t}_0}{\partial q_{dof}} \end{bmatrix} = \begin{bmatrix} \omega_{i,0} & \cdots & \omega_{i,j} & \cdots & \omega_{i,dof} \\ v_{i,0} & \cdots & v_{i,j} & \cdots & v_{i,dof} \end{bmatrix} \tag{2.16}$$

2.4 Joint Limits

Most robotic joints have geometric limits which define the range of value that can be accessed by the joint variables. The joint limits for 1D joints like revolute and prismatic joints are trivial to formulate: We denote q^- and q^+ the lower and upper values accessible and add a boundary constraint to the optimization problem:

$$\boxed{q^- \leq q \leq q^+} \tag{2.17}$$

Most joints are easy to limit because their variables are independent. Limiting the movements of a spherical joint, and by extension, of a free joint, is more complicated. In humanoid robotics, spherical joints can be used to model the shoulder or hip joint of the robot. A common approach to limit shoulder joint, inspired from the biomechanics field, considers the spherical motion (or swing) and the axial motion (or twist) separately as shown in Fig. 2.2.

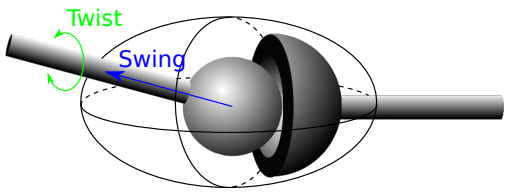


Figure 2.2 Swing and Twist in ball and socket joint

The spherical motion can be parameterized by a vector of the 3D unit sphere $S2$ and constrained to lay within a limit cone, the axial motion can be parameterized in $\mathbb{R}$ and limited by equation (2.17). This type of formulation is presented in [BB01].

2.5 Contact constraints

Humanoid robots evolve in their environment by making and breaking contacts with it. A contact can be defined between 2 surfaces of different bodies of robots. The most usual

types of contact constraint encountered are the planar contact and the fixed contact. A planar contact constraint is used when a planar surface of a robot is put in contact with a planar surface of the environment. We denote $F_1 = \{O_1, \vec{x_1}, \vec{y_1}, \vec{z_1}\}$ a frame defined on S_1, the surface of the first body involved in the contact, such that the 3D point O_1 is on S_1 and the vector $\vec{z_1}$ is normal to S_1 and pointing toward the inside the body. $F2 = \{O_2, \vec{x_2}, \vec{y_2}, \vec{z_2}\}$ is a frame on S_2, the surface of the second body involved, such that O_2 is on S_2 and the vector $\vec{z_2}$ is normal to S_2 and pointing away from the body.

Constraining S_1 and S_2 to be coplanar comes down to aligning $\vec{z_1}$ with $\vec{z_2}$ and to ensure that the projection of the distance between O_1 and O_2 along $\vec{z_1}$ is null. Note that we avoid using the dot product of two vectors that are meant to be aligned e.g. $\vec{z_1} \cdot \vec{z_2} = 1$ because when that constraint is satisfied, its gradient is zero, which implies that in the optimization context it is unqualified. That is why we prefer imposing orthogonality constraints. This translates into adding the following set of constraints to our problem:

$$\boxed{\begin{cases} \overrightarrow{O_1O_2} \cdot \vec{z_1} = 0 \\ \vec{x_1} \cdot \vec{z_2} = 0 \\ \vec{y_1} \cdot \vec{z_2} = 0 \\ \vec{z_1} \cdot \vec{z_2} \geq 0 \end{cases}} \tag{2.18}$$

This set of constraints leaves free the displacements of F_2 along $\vec{x_1}$ and $\vec{y_1}$ as well as its rotation around $\vec{z_1}$. We call this a floating planar contact, the optimization algorithm will be able to choose the location of F_2 in the plane $\{O_1, \vec{x_1}, \vec{y_1}\}$. This contact has 3 degrees of freedom.

If we constrain the location of F_2 in $\{O_1, \vec{x_1}, \vec{y_1}\}$ such that O_1 and O_2 are superimposed and $\vec{x_1}$, $\vec{y_1}$, $\vec{z_1}$ are aligned with respectively $\vec{x_2}$, $\vec{y_2}$, $\vec{z_2}$, we obtain a fixed contact with zero degrees of freedom. This translates into adding the following set of constraints to our problem:

$$\boxed{\begin{cases} \overrightarrow{O_1O_2} = \vec{0} \\ \vec{x_1} \cdot \vec{z_2} = 0 \\ \vec{y_1} \cdot \vec{z_2} = 0 \\ \vec{x_1} \cdot \vec{y_2} = 0 \\ \vec{z_1} \cdot \vec{z_2} \geq 0 \\ \vec{x_1} \cdot \vec{x_2} \geq 0 \end{cases}} \tag{2.19}$$

We illustrate those two types of contacts in Fig. 2.3

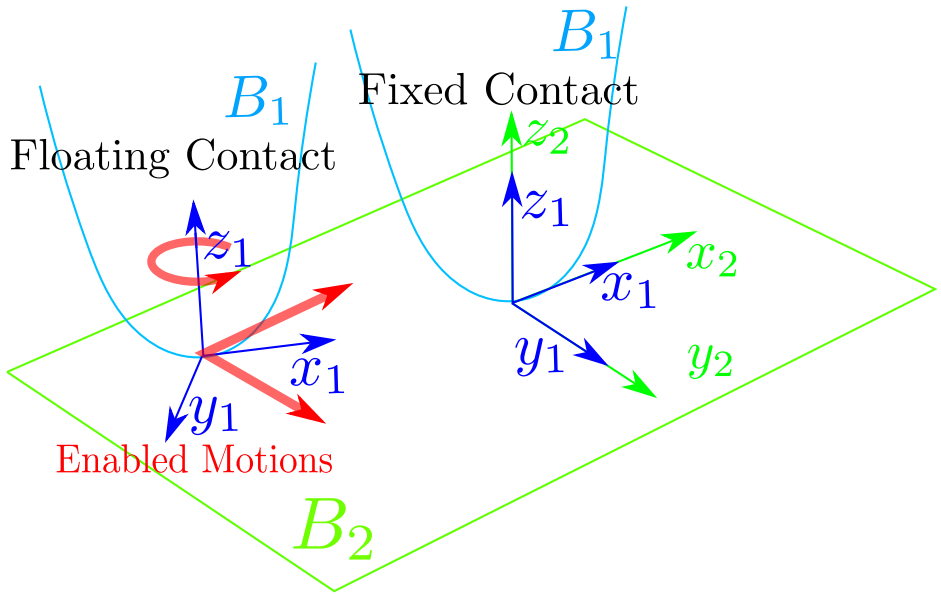


Figure 2.3 Floating and Fixed Contacts

2.6 Collision avoidance

In order to avoid unwanted collisions between bodies of robots, for two bodies B_1 and B_2, we want to define a continuously differentiable function $d_{\{B_1,B_2\}}(q)$ that has the properties of a pseudo-distance:

- $d_{\{B_1,B_2\}}(q) > 0$ when the bodies are not touching each other
- $d_{\{B_1,B_2\}}(q) = 0$ when the bodies are in collision without interpenetration
- $d_{\{B_1,B_2\}}(q) < 0$ when the bodies are in collision with interpenetration

Using the cartesian distance between the exact surfaces of B_1 and B_2 might result in a discontinuous gradient of $d_{\{B_1,B_2\}}$ if the surfaces of B_1 and B_2 are not strictly convex. A conservative approach is to associate to each body, a strictly convex bounding volume and to compute the distance between those volumes. [EMK07] and [EMBK14] proposes a method to generate a strictly convex Sphere-Torus-Patch Bounding Volumes (STP-BV) that guarantees the gradient continuity of the proximity distance. The distance between the STP-BV of B_1 and B_2 computed by an enhanced GJK [GJK88] collision-detection algorithm is a continuously differentiable pseudo-distance. Thus, we can use this function in our optimization algorithm to ensure that the distance between the bodies is greater than a safety distance ε_{12}:

$$\boxed{d_{\{B_1,B_2\}}(q) \geq \varepsilon_{12}} \tag{2.20}$$

This function can be used to avoid collisions between a robot and the environment as well as auto collisions between bodies of the same robot. We denote *Coll* the list of triplets $\{B_i, B_j, \varepsilon_{ij}\}$ defining each collision that we want to avoid.

Then the set of constraints to add to our problem is:

$$\boxed{\forall \{B_i, B_j, \varepsilon_{ij}\} \in Coll,\ d_{\{B_i,B_j\}}(q) \geq \varepsilon_{ij}} \tag{2.21}$$

In many cases, it is possible to avoid the collision between two bodies of a robot by modifying the joint limits and reducing them to a span where the collision of interest cannot happen. That approach is conservative and ad-hoc but can save some precious computation time.

2.7 External Forces

For a robot to interact with the real world, its geometric description is not enough. The robot is subject to forces applied on its bodies by the exterior world, which can be generated by contacts with the environment or with another actor (human, another robot, manipulated object...), by the effect of physical forces like gravitation or magnetism, or by contacts between two bodies of the robot. Our posture generator must take those 'External forces' into account, to be able to estimate the stability of the robot and compute the internal torques generated in the joints.

An external force applied on a rigid body can also be called a wrench and is composed of a resultant part f (sometimes called force) and a moment part (sometimes called couple). Let w be a wrench, $w|_F^O$ is the expression of w calculated at the point O expressed in the frame F. We denote $\vec{f}$ the resultant part of w, and $\vec{f}|_F$ the expression of $\vec{f}$ in F. $\vec{m}$ is the moment part of w and $\vec{m}|_F^O$ the expression of $\vec{m}$ in F calculated at the point O.

$$w|_F^O = \left\{ \begin{array}{c} \vec{m} \\ \vec{f} \end{array} \right\}_F^O = \left\{ \begin{array}{c} \vec{m}|_F^O \\ \vec{f}|_F \end{array} \right\} \tag{2.22}$$

The expression of the moment part on a different point P is given by the following formula:

$$\vec{m}|_F^P = \vec{m}|_F^O + \overrightarrow{PO} \wedge \vec{f}|_F \tag{2.23}$$

The resultant part is invariant with respect to the point at which the wrench is calculated.

We drop the frame subscript when the choice of the frame does not matter and all quantities are computed in the same frame.

2.8 Static stability

We denote g the acceleration of gravity on earth $g = 9.81 m.s^{-2}$. The wrench associated with the action of gravity on a body of mass M which center of mass is denoted G with $\vec{z}$ the upward vertical vector in the world frame F_0 is:

$$w_g|^G_{F_0} = \left\{ \begin{array}{c} \vec{0} \\ -Mg\vec{z} \end{array} \right\}^G_{F_0} \tag{2.24}$$

A solid is statically stable if it satisfies the Euler-Newton Equation. We consider a robot on which m external wrench $w_i = \left\{ \begin{array}{c} \vec{m}_i \\ \vec{f}_i \end{array} \right\}^{P_i}$ are applied. We denote P the application point at which the equation and all its terms are calculated:

$$\sum_i w_i|^P + w_g|^P = 0 \tag{2.25}$$

which is equivalent to:

$$\left\{ \begin{array}{c} \sum_i \vec{m}_i|^P + \overrightarrow{GP} \wedge Mg\vec{z} = 0 \\ \sum_i \vec{f}_i - Mg\vec{z} = 0 \end{array} \right. \tag{2.26}$$

This equation can be simplified by applying it at the center of mass of the body as:

$$\left\{ \begin{array}{c} \sum_i \vec{m}_i|^G = 0 \\ \sum_i \vec{f}_i - Mg\vec{z} = 0 \end{array} \right. \tag{2.27}$$

Satisfying equation (2.27) ensures the stability of a rigid body. If the robot's actuators are powerful enough to maintain its posture under any external perturbation, namely, when they can generate infinite or at least large enough torques, then the robot can be approximated as a rigid body and satisfying equation (2.27) is enough to ensure its stability. Otherwise, it is necessary to verify that the robot's actuators can generate large enough torques to maintain that posture. The details of torque computation are discussed in Section 2.10

Equation (2.27) can be used in an optimization problem. We consider that each wrench w_i applied on the system is defined by the position of its application point P_i and the values m_i and f_i that represent the moment and resultant of w_i at P_i. P_i depends on q the joint parameter of the robot. m_i and f_i are new variables that need to be added to the problem. In summary, w_i depends on q, m_i and f_i. We denote f the concatenation of all the variables m_i and f_i.

$$\boxed{s(q,f) = \left\{ \begin{array}{c} \sum_i \vec{m}_i + \overrightarrow{P_iG} \wedge \vec{f}_i \\ \sum_i \vec{f}_i - Mg\vec{z} \end{array} \right\} = 0} \tag{2.28}$$

The optimization problem (1.11) becomes (we denote m the dimension of the force variables):

$$\left\{ \begin{array}{l} \min\limits_{q \in \mathcal{C}, f \in \mathbb{R}^m} f(q) \\ \text{s.t.} \left\{ \begin{array}{l} s(q,f) = 0 \\ c_i(q) = 0, \ \forall i \in E \\ c_i(q) \geq 0, \ \forall i \in I \end{array} \right. \end{array} \right. \tag{2.29}$$

The derivation of the static stability constraint is straightforward. All the terms of equation (2.27) are components of wrenches. A wrench is completely defined by the frame in which it is expressed and its values of resultant and moment in that frame. Deriving the stability condition comes down to deriving each term w.r.t its components values and w.r.t the transformation of its frame.

$$\left\{ \begin{array}{c} \partial \left(\sum_i m_i|^G \right) = \sum_i \partial (m_i|^G) \\ \partial \left(\sum_i f_i \right) = \sum_i \partial (f_i) \end{array} \right. \tag{2.30}$$

We will explicit a method to automatically compute those derivatives in a further chapter.

2.9 Center of mass projection

When all the wrenches applied to the body are due to unilateral punctual contacts on the same horizontal plane $H = \{O, \vec{x}, \vec{y}\}$, the stability criterion (2.27) can be simplified. The wrench w_i generated by a unilateral punctual contact is a pure force resultant, its moment part is zero on the contact point.

$$w_i|^{P_i} = \left\{ \begin{array}{c} \vec{0} \\ \vec{f}_i \end{array} \right\}^{P_i}$$

Equation (2.27) becomes:

$$\begin{cases} \sum_i \overrightarrow{OP_i} \wedge \vec{f}_i - \overrightarrow{OG} \wedge Mg\vec{z} = 0 \\ \sum_i \vec{f}_i - Mg\vec{z} = 0 \end{cases} \tag{2.31}$$

We can write $\overrightarrow{OG} = \overrightarrow{OG_p} + z_G\vec{z}$ with G_P the projection of G on H. Replacing in the moment equation gives:

$$\sum_i \overrightarrow{OP_i} \wedge \vec{f}_i - \sum_i \overrightarrow{OG_P} \wedge \vec{f}_i = 0 \tag{2.32}$$

With $f_i = f_i^x\vec{x} + f_i^y\vec{y} + f_i^z\vec{z}$, G and P_i can be written as $\overrightarrow{OG_P} = G_x\vec{x} + G_y\vec{y}$ and $\overrightarrow{OP_i} = P_{ix}\vec{x} + P_{iy}\vec{y}$. The two first lines of equation (2.32) give:

$$\sum_i \begin{Bmatrix} P_{iy}f_i^z \\ -P_{ix}f_i^z \end{Bmatrix} = \begin{Bmatrix} G_y \\ -G_x \end{Bmatrix} \sum_i f_i^z \tag{2.33}$$

$$\overrightarrow{OG_P} = \frac{\sum_i \overrightarrow{OP_i} f_i^z}{\sum_i f_i^z} \tag{2.34}$$

Since all the contacts are unilateral, all the f_i^z are positive. For any set of $f_i^z \geq 0$, G_P is a barycenter with positive coefficients of the P_i. Any point G_P that is included in the convex hull of all the P_i is a solution.

Thus, we have the property: a rigid body that has all its contacts with the environment being punctual, unilateral and all lying on the same horizontal plane H is stable if and only if the projection of its center of mass on H is inside the convex hull of all its contact points.

2.10 Torque limits

In general, satisfying equation (2.27) is not enough to ensure that a robot can be statically stable. The joint torques that are required to hold the posture must be within the physical capabilities of the robot, namely, its torque limits. We denote τ_i^- and τ_i^+ the minimal and maximal torques that can be generated by the robot's actuators on joint J_i. In some cases, the torque limits can depend on the joint parameters $\tau_i^-(q)$ and $\tau_i^+(q)$. For example, it is the case with the Atlas robot that is hydraulically actuated.

Featherstone [Fea07] proposes a recursive algorithm to compute the torques, accelerations, and velocities generated in a multi-articulated system by a set of external forces called the Inverse Dynamics Algorithm. For the purpose of generating statically stable postures, the

velocities and accelerations are useless. Thus, we devise a specialized algorithm to fit our needs and call it the Inverse Static Algorithm.

We denote a_g the gravity acceleration vector with $\vec{a}_c$ its rotation part and $\vec{a}_f$ its translation part:

$$a_g = \left\{ \begin{array}{c} \vec{a}_c \\ \vec{a}_f \end{array} \right\} = \left\{ \begin{array}{c} \vec{0} \\ g\vec{z} \end{array} \right\} \tag{2.35}$$

The algorithm first computes the generalized forces f_i^G applied to each body. It is the sum of the action of gravity and of the external forces applied on a body calculated at the origin of the world frame, expressed in the world frame. Then, the generalized forces are used to compute the torques. We denote $\mathbf{I}_i$ the inertia matrix of body i.

Algorithm 4 Inverse Static Algorithm

for $i = 0 : n_B$ **do**
  $f_i^G = \mathbf{I}_i {}^i\mathbf{X}_0 a_g - {}^i\mathbf{X}_0{}^* f_i^{ext}$
end for
for $i = n_J - 1 : 0$ **do**
  $\tau_i = f_i^{G^T} S_i$
  if $pred(i) \neq -1$ **then**
    $f_{pred(i)}^G += \mathbf{X}_i^{PtS}(q)^{-*} f_i^G$
  end if
end for

We can write the torques as a function of the joint parameters and the external forces $\tau(q, f)$. The torque limit constraint writes as:

$$\boxed{\tau^- \leq \tau(q, f) \leq \tau^+} \tag{2.36}$$

We will detail the derivation of this constraint in Section 3.2.

2.11 Contact Forces and Friction Cones

The contacts involved in a posture generation problem can be separated into two types: Geometric Contacts and Stability Contacts.

The Geometric Contact is a contact where the position of contact is reached, but that contact does not support any contact forces (this is a necessary step to generate a sequence of quasi-static transitions). Physically, that correspond to the transition state between a configuration without contact and a configuration with contact on which forces are applied.

We defined the Geometric Contacts in Section 2.5. The Stability Contact is a Geometric Contact that bears interaction forces.

We denote $w_{1\to2}$ the force applied by body B_1 on body B_2. Then the force applied by B_2 on B_1 is $w_{2\to1} = -w_{1\to2}$.

The interaction force resulting from a punctual contact (see Fig. 2.4) on a point P can be modeled as a pure force resultant along the z_1 direction $\vec{f}_n = f_z\vec{z}_1$ and the tangential efforts due to the friction in that contact can be modeled as $\vec{f}_t = f_x\vec{x}_1 + f_y\vec{y}_1$.

$$w_{2\to1}|^P = \left\{ \begin{array}{l} \vec{0} \\ \overrightarrow{f_{2\to1}} = \vec{f}_t + \vec{f}_n \end{array} \right\}^P \tag{2.37}$$

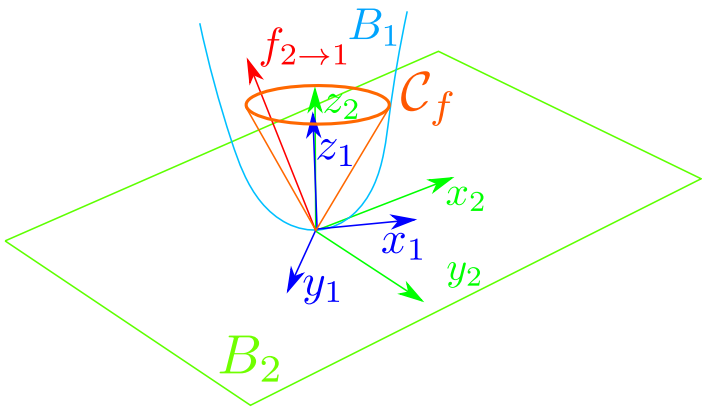


Figure 2.4 Punctual unilateral stability contact with friction

This formulation of the interaction force defines a bilateral contact, in the sense that the force can be in any direction, B_1 can push as well as pull on B_2.

To model a unilateral contact, we must constrain the normal component of $f_{2\to1}$ to be oriented toward the inside of B_1. This means that only pushing actions can be generated, not pulling actions. This translates into:

$$\overrightarrow{f_{2\to1}} \cdot \vec{z}_1 = f_z \geq 0 \tag{2.38}$$

Furthermore, to avoid slippage, the Coulomb friction law must be respected for each contact force $\vec{f}$. Which translates into the following equation, with $\vec{f}_n$ and f_t, respectively the normal and tangential parts of $\vec{f}$ and μ, the friction coefficient:

$$\mu\|\vec{f}_n\| \geq \|\vec{f}_t\| \tag{2.39}$$

Given the decomposition of $f_{2\to1}$ in F_1, $f_{2\to1} = f_x\vec{x}_1 + f_y\vec{y}_1 + f_z\vec{z}_1$, for any punctual contact in a posture generation problem, we can add the following set of constraint to our optimization problem:

$$\begin{cases} f_z \geq 0 \\ \mu^2 f_z^2 - f_x^2 + f_y^2 \geq 0 \end{cases} \tag{2.40}$$

When it comes to planar contacts on surface S with $\vec{n}$ the outbound normal to S, the interaction force can have components of forces and moments in all directions. The forces components intrinsic to the planar contact model are a resultant part aligned with $\vec{n}$ $\vec{f}_n = f_z\vec{z}_1$ and a moment part tangential to S: $\vec{m}_t = m_x\vec{x}_1 + m_y\vec{y}_1$. The forces due to friction are a tangential friction resultant part $\vec{f}_t = f_x\vec{x}_1 + f_y\vec{y}_1$ and a normal friction moment $\vec{m}_n = m_z\vec{z}_1$.

This type of force can be modeled by a set of unilateral punctual efforts applied on each vertex of a polygon describing the contact area. And ensuring that each of them lay in their respective friction cone, thus satisfying the equation (2.40). As depicted in Fig. 2.5

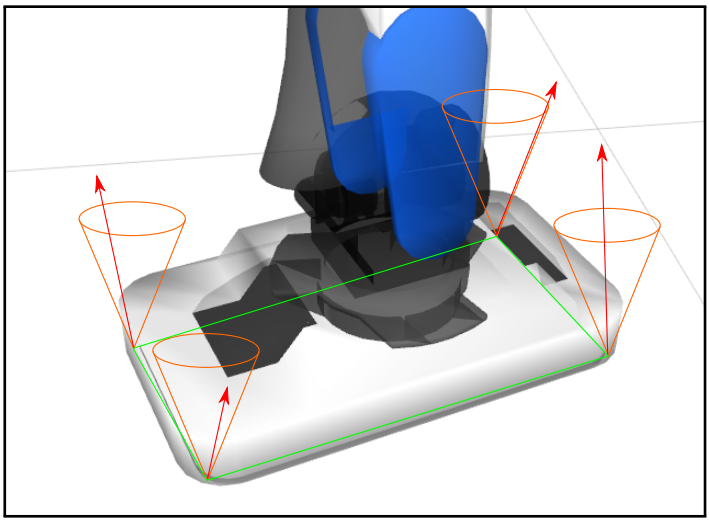


Figure 2.5 Modelization of a Planar Contact between the foot on HRP4 and the ground

2.12 Cost Functions

In addition to constraints, it is often useful to add a cost function to our optimization problem. The submanifold of feasible configurations $\mathcal{C}_F$ can contain an infinity of solutions and even some continuous solution areas in which all points are solutions. The cost function helps to choose the 'best' candidate solution. Various types of cost functions can be chosen, for example, we can minimize the distance to a reference posture q_R:

$$f_{\text{posture}}(q) = \|q - q_R\|^2$$

The effect of that type of cost function is illustrated in Fig. 2.6. On both images, the HRP-2 Kai robot is stable, respects its joints and torques limits. The only difference is that the right one minimizes the distance to a reference posture (standing straight with bent knees) while the left result does not use a cost function.

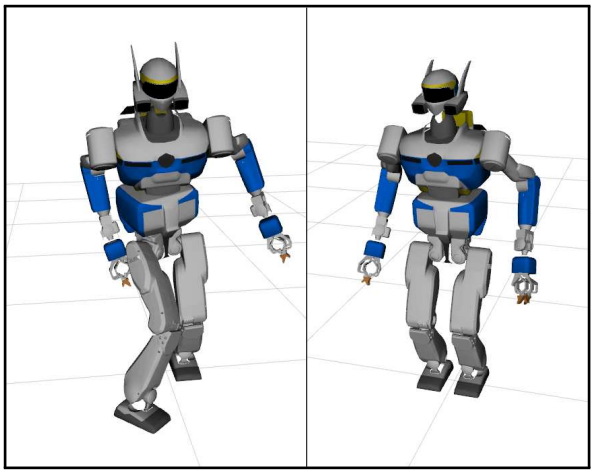


Figure 2.6 Effect of cost function. Left: no cost function. Right: Distance to posture

We can also want to minimize the sum of norms of contact forces:

$$f_{\text{forces}}(f) = \sum_i \|f_i(f)\|^2$$

or the torques in the robot's joints:

$$f_{\text{torques}}(q, f) = \sum_i \|\tau_i(q, f)\|^2$$

Some more custom cost functions can also be considered, for example, we may want a point P_i on body B_i with to be as far as possible in a direction $\vec{d}$

$$f_{\text{point}}(q) = -\overrightarrow{O_0 P_i} \cdot \vec{d}$$

Any positively weighted combination of cost function can be used, in which case it is important to choose the weights p_i carefully to scale all the costs so that they all can influence

the result and none is completely dominated by another.

$$f_{\text{cost}}(q,f) = \sum_i p_i f_i(q,f) = p_0 f_{\text{posture}}(q) + p_1 f_{\text{forces}}(f) + p_2 f_{\text{torques}}(q,f) + \cdots \tag{2.41}$$

2.13 Conclusion

In this section, we have seen how to formulate a robotics problem with several different tasks and objectives as an optimization problem.

We denote $\mathcal{T}_i$ the additional tasks added to the problem, which is described by the set of equations $g_i(q,f) = 0$ and inequations $h_i(q,f) \geq 0$. The contact tasks are included in those and the equations describing them must encompass the geometric contact constraint equation like (2.18) as well as the unilaterality and friction equations (2.40) in the case of a unilateral contact.

A typical robotics problem can be written as a combination of all those costs and constraints:

$$\min_{q,f} \quad f_{\text{cost}}(q,f)$$
$$\text{s.t.} \begin{cases} q^- \leq q \leq q^+ \\ s(q,f) = 0 \\ \tau^- \leq \tau(f,q) \leq \tau^+ \\ \forall \{B_i, B_j, \varepsilon_{ij}\} \in Coll,\ d_{\{B_i,B_j\}}(q) > \varepsilon_{ij} \\ g_i(q,f) = 0 \ \ \forall \mathcal{T}_i, \\ h_i(q,f) \geq 0 \ \ \forall \mathcal{T}_i. \end{cases} \tag{2.42}$$

In the next chapter, we present an extension to the contact constraint formulation that allows generating non-inclusive contacts, an algorithm to compute the exact derivatives of the torques in robot's joints, and our endeavor to apply a different optimization approach to solving posture generation problems.

Chapter 3

Extensions of Posture Generation

In this chapter, we present three different and unrelated contributions to the state of the art of posture generation. First, we present a novel method called Integration of Non-Inclusive Contacts in Posture Generation that was published in [BEV+14]. Second, we present a generic algorithm to compute efficiently the derivative of the joint torques of a robot. Third, we present our endeavor to apply an optimization method called 'Lifted Newton Method', presented in [AD10], to problems of inverse kinematics.

3.1 Integration on Non-Inclusive Contacts in Posture Generation

In Section 2.5, we defined a set of constraints (equations (2.18)) that ensures the coplanarity of two planar surfaces in geometric planar contact. Those equations guarantee the coplanarity of two infinite planar surfaces. Another set of constraints needs to be added to ensure the validity of a contact between two finite surfaces S_1 and S_2, to ensure that the intersection of those two surfaces is not empty and that it is big enough to support the contact.

In this section, we propose a simple way to formulate geometric contact formation to have an arbitrary intersection shape in a robotic (humanoid) posture generation problem, when searching for a mobile contact between two surfaces. We start by defining convex areas of contact on the robot's body and the environment, that we call contact patches. We can generate contacts with an arbitrary intersection of a pair of any of these predefined contact patches. Virtually any planar surface can be a contact patch. In our previous works, the constraints formulating a new contact in the posture generator, states that a contact patch is totally included into another, which is obviously a limitation of the posture generation. A simple example is when a human climbs stairs. He usually contacts only the front half of

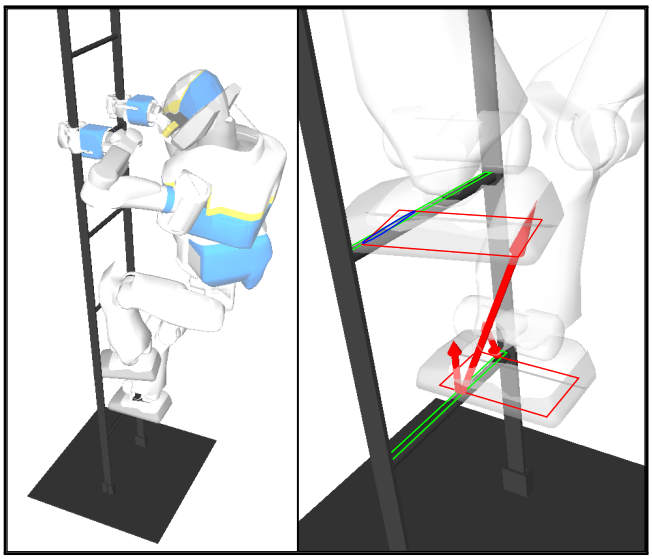


Figure 3.1 Using non-inclusive contacts for ladder climbing (green/red: contact polygons; blue: contact ellipse; red arrows: contact forces resultants)

its foot with each step, the heel almost never touches the steps. Our new geometric contact modeling writes very simply as additional constraints and variables added to the optimization problem, translating the search for an ellipse inscribed in the intersection of the pair of patches we want in contact. The result of our posture generator is then a configuration where contact patches are not necessarily included in one another and the shape and area of their intersection can be monitored. This allows our posture generator to propose contacts of different shapes with a non-predefined number of contact points (used later to compute reaction/contact forces). We illustrate the efficiency of our method in multi-contact posture generation with the HRP-2 and ATLAS humanoid robots with results that can not be generated automatically by existing methods.

A contact is generally defined by a pair of points (one on each object in contact) and a pair of normal vectors. A contact constraint boils down to finding a posture in which the predefined authorized contact points and normal of each body match [ZLH+13][HBL+08]. Likewise, in [OGHB11], the position of the feet of the NAO robot is manually tuned in order to obtain statically stable position during the climbing of a spiral staircase. In [CTS+09], the surface in contact is chosen according to two criteria: the position of the force sensors of the feet, and the type of contact desired. In [SK10], the problem of contact discovery in not considered. In [MTP12], two surfaces are considered in contact as soon as the center point of one of them touches the other one and their normals match. Although used in many papers, it is not difficult to see that this definition of contact excludes a series of possibilities that would have been obtained if the predefined points were placed in different configurations

within their respective patches. Table 3.1 presents some diagrams describing some typical methods used for constraining the relative positions of contact surfaces. On each figure, the blue rectangle describes a surface S_2 with which we want to make another surface S_1 contact. S_1 is drawn in green if the contact is valid and in red otherwise.

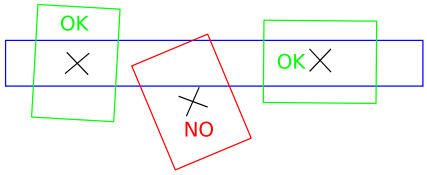
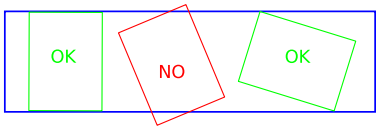
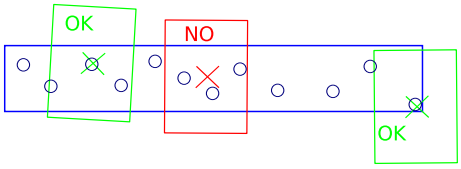
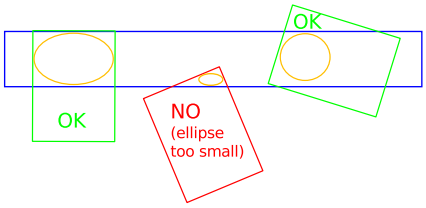
	The center point of S_1 is included in S_2 [HBL05] • Easy and cheap to compute • Eliminates valid solutions
	S_1 is fully included in S_2 [BK12a] • Eliminates many valid solutions
	S_1 contains a set of sampled points, the center of S_2 has to match one of them [CKNK03] • Discrete set of possible solutions • Eliminates many valid solutions
	The contact is valid if an ellipse of sufficient size is found in the intersection $S_1 \cap S_2$ [BEV+14] • Smooth • Does not eliminate valid solutions

Table 3.1 Contact descriptions

In planning, once a geometric contact is found for a posture, the next posture will make use of it as a stability contact and apply forces on it. Once the geometric contact is established, one determines the intersection of the contacting surfaces in order to find the points on which the reaction forces are to be computed when the geometric contact becomes a stability contact. Several approaches require fixing the number of contact points or to have inclusive contact (i.e. one patch is fully included in the other) [BK12a].

We provide a simple solution that relaxes contact constraints and gets rid of predefining the contact points. We consider that a contact is valid if the intersection between two distinct patches has an area greater than a given threshold. To enforce this, we require this intersection

to contain an ellipse whose surface can possibly be maximized. Convex patches allow writing the inscription constraints easily by means of half-spaces.

3.1.1 Contact geometry formulation

Let us consider a contact defined by two flat surfaces S_1 and S_2 which are respectively delimited by two convex polygons P_1 and P_2. For this contact to be valid, it is obviously necessary that the intersection $P_1 \cap P_2$ is not empty.

An important remark is that the number of sides of the intersection polygon is not known a priori and, as shown in Fig. 3.2, this number can change depending on the configuration. Each time this number changes, the gradient of the area of the intersection is discontinuous. This is an issue for integrating any constraint or objective based on the area because we use a solver for smooth optimization problems. This issue could be dealt with by using non-smooth optimization algorithms, but such algorithms are slower and less available, and our posture generator is not designed to use them. Moreover, supposing that we want to write constraints

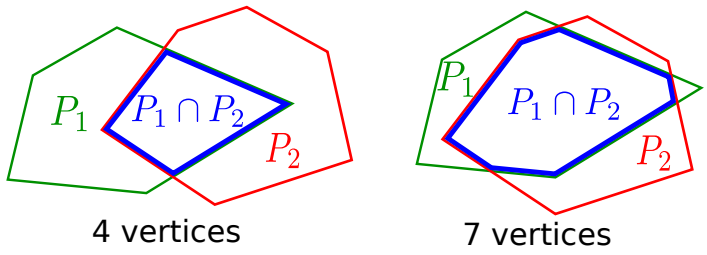


Figure 3.2 Topological instability of $P_1 \cap P_2$

based on the sides of the contact area, then, the number of constraints would change with the number of vertex of $P_1 \cap P_2$. The large majority of the optimization software cannot deal with a non-constant number of constraints. The solution proposed in Section 3.1.2 overcomes these issues by defining a set of constraints that is independent of the topology of the intersection area.

3.1.2 Non-inclusive contact constraints

Main Idea

We present a smooth formulation of the non-empty intersection between two contact surfaces. We assume that co-planarity of S_1 and S_2 is obtained by using the constraints presented in (2.18). Here we focus on the intersection of the two polygons P_1 and P_2, respectively describing the contours of S_1 and S_2.

To avoid dealing with changes of topology, we consider using an ellipse $\mathcal{E}$ included in $P_1 \cap P_2$ to estimate the area of the intersection. Since P_1 and P_2 are convex polygons, then $P_1 \cap P_2$ is also a convex polygon. A convex polygon can be seen as an intersection of half-planes based on the lines supporting its edges. Thus, an ellipse is inside of a convex polygon if it lies entirely in the corresponding half-planes. Having the ellipse be included in the intersection of two polygons is equivalent to having it included in both polygons:

$$\mathcal{E} \subset P_1 \cap P_2 \iff \mathcal{E} \subset P_1 \ \wedge \ \mathcal{E} \subset P_2 \tag{3.1}$$

Even if the number of edges of $P_1 \cap P_2$ can change, the numbers of edges of P_1 and P_2 respectively are fixed. To assert that an ellipse lies in a half-plane, we need a function that is positive when the ellipse is in it (with zero value when the ellipse is on the edge) and negative if not. The signed distance to the line defining the half-space is a good candidate (distance ellipse-line if the ellipse is in the half-plane, opposite of the penetration distance if not), but actually, any pseudo-distance does the job. And a sufficient condition for the ellipse to be inside the polygons intersection is that the pseudo-distance between the ellipse and each edge of both polygons is positive (see Fig. 3.3). By considering each edge separately as opposed to the pseudo-distance of the ellipse to a whole polygon, we can write smooth constraints with a simple pseudo-distance function. We develop such a pseudo-distance in the next subsection.

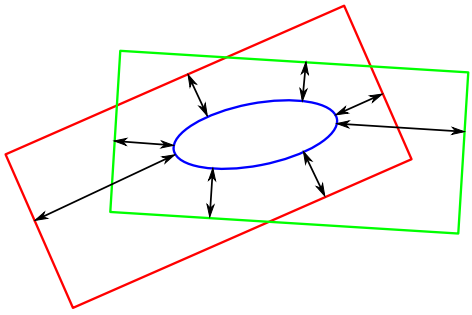


Figure 3.3 Distance between $\mathcal{E}$ and $P_1 \cap P_2$

Pseudo-distance

Computing the distance between an ellipse and a line is not straightforward, whereas the distance between a line and a circle is very easy to compute. Also, we note that in the frame $F_\mathcal{E}$ defined by the ellipse's axes and pseudo-radius, the ellipse is a circle of radius $r_\mathcal{E} = 1$ (The x-unit along the first axis of the ellipse is r_x, the first radius of the ellipse, the y-unit along the second axis of the ellipse is r_y, the second radius). The transformation from the

original frame F_0 in which the ellipse and the polygons are described to the ellipse's frame $F_\mathcal{E}$ is just the composition of a rotation and a scaling of the space along the axes of the ellipse with a scaling vector $[\frac{1}{r_x}, \frac{1}{r_y}]$. The effect of such a transformation applied to an ellipse and two polygons is shown in Fig. 3.4. We thus defined the following pseudo-distance from an ellipse to a half-plane as the signed Euclidean distance from the corresponding unit circle to the transformed half-plane in the frame $F_\mathcal{E}$.

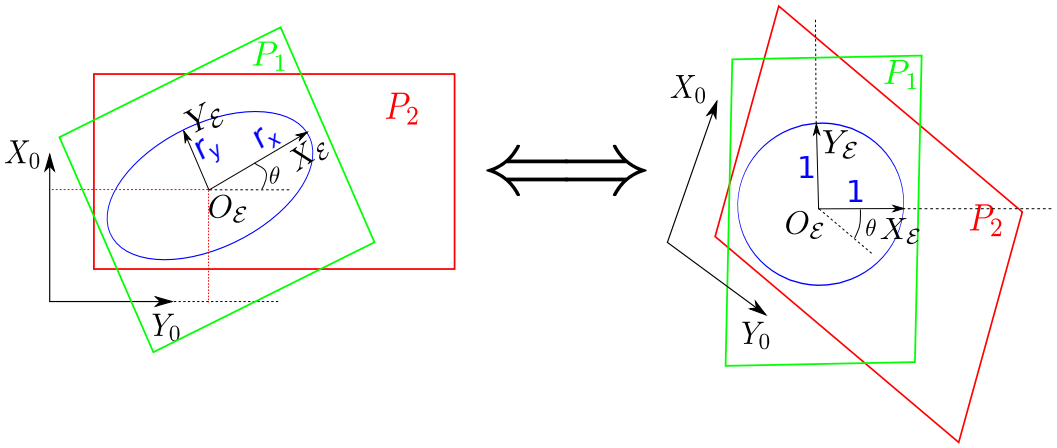


Figure 3.4 Transformation from $F_0(O_0, X_0, Y_0)$ to $F_\mathcal{E}(O_\mathcal{E}, X_\mathcal{E}, Y_\mathcal{E})$

Now let us consider a single segment $p_i p_j$ and an ellipse $\mathcal{E}$ defined in F_0. The expression of a vector ${\mathbf{v}_{F_0}}^T = [v_x, v_y]_{F_0}$ in $F_\mathcal{E}$ is obtained by applying the formula (3.2).

$$\mathbf{v}_{F_\mathcal{E}} = \begin{bmatrix} \frac{1}{r_x} & 0 \\ 0 & \frac{1}{r_y} \end{bmatrix} \cdot \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \cdot \mathbf{v}_{F_0} \tag{3.2}$$

In $F_\mathcal{E}$, the distance between the circumference of $\mathcal{E}$ and the segment $p_i p_j$ is:

$$d_s(\mathcal{E}, p_i p_j) = \frac{\overrightarrow{p_i p_j}\big|_{F_\mathcal{E}} \wedge \overrightarrow{p_i O_\mathcal{E}}\big|_{F_\mathcal{E}}}{\|\overrightarrow{p_i p_j}\|_{F_\mathcal{E}}} - 1 \tag{3.3}$$

where $\wedge$ denotes the cross product and $O_\mathcal{E}$ is the center of the ellipse.

To avoid numerical problems when the segment $p_i p_j$ is small, it is preferable to multiply this distance by $\|\overrightarrow{p_i p_j}\|_{F_\mathcal{E}}$ before using it as a constraint. Then we get the following constraint:

$$-\overrightarrow{p_i p_j}\big|_{F_\mathcal{E}} \wedge \overrightarrow{p_i O_\mathcal{E}}\big|_{F_\mathcal{E}} + \|\overrightarrow{p_i p_j}\|_{F_\mathcal{E}} \leq 0 \tag{3.4}$$

The combination of these equations (3.2) and (3.4) applied for each edge of the polygons gives us all the necessary tools to develop a set of constraints that ensures that an ellipse is inside the intersection of two polygons.

Modification of the optimization problem

To include the above idea in our posture generation, we need to modify the optimization problem (2.42) as follows. Each non-inclusive geometrical contact adds five variables to the optimization vector, corresponding to the position, orientation, and radiuses of the ellipse (x, y, θ, r_x and r_y). One constraint of ellipse inclusion (as described above) is added to the problem for each edge of the polygons. The parameters r_x and r_y are given lower positive bounds to ensure that the ellipse is not empty. The existence of a contact between S_1 and S_2 is thus transformed into the existence of r_x and r_y respecting their bounds. In summary, this kind of constraint adds 5 variables and $card(P_1) + card(P_2)$ constraints to the optimization problem, while the 'usual' inclusion constraint adds 0 variable and $card(P_1)card(P_2)$ constraints. We denote $card(P)$ the number of edges of polygon P. The existence of the contact can alternatively be enforced by imposing a minimum area for the ellipse.

Maximization of the contact area

The formulation in the above section only ensures the existence of a contact of minimal size. However, one may want to find a contact area as large as possible, so that it is more likely to be able to support strong forces and have strong friction forces, which is helpful to guarantee the stability of the robot. Therefore, it seems appropriate to try and maximize the area of contact between two polygons. As explained before, computing the area of the intersection surface is not a good practice in our case. But the ellipse computed as above gives a lower bound of the contact area.

$$\mathcal{E} \subset P_1 \cap P_2 \Longrightarrow \mathcal{A}(\mathcal{E}) \leq \mathcal{A}(P_1 \cap P_2) \tag{3.5}$$

with $\mathcal{A}(X)$ being the area of X.
Therefore, we can maximize the area of the ellipse in order to maximize the contact area. This is readily obtained by minimizing the value of $f_{\text{ellipse}}(r_x, r_y) = -\pi r_x r_y$ in the modified problem (2.42). In case there are other cost functions, the above cost can be added to them with a desired weight. This requires, however, to scale properly the cost so as to have a meaningful and easy-to-tune weight: the range of value of the ellipse's area goes from 0 to $\mathcal{A}(P_1 \cap P_2) \leq \min(\mathcal{A}(P_1), \mathcal{A}(P_2))$. This latter quantity can be small (a typical area of contact of a humanoid robot is about $0.01m^2$, some environment surfaces can be smaller). To get a

basic cost (before weighting) of magnitude around 1, we use the following scaling:

$$f_{\text{ellipse}}(r_x, r_y) = -\frac{\pi r_x r_y}{\min(\mathcal{A}(P_1), \mathcal{A}(P_2))} \tag{3.6}$$

This cost's absolute value will always be less than 1, but not much less around the optimum, in most cases.

Using a non-inclusive contact to maintain stability

The method we presented so far allows finding a configuration in which a new non-inclusive contact is added, but this contact does not bear any force. It is found as a geometrical contact, but will eventually have to bear some forces, and thus, become a stability contact. Usually, for a stability contact, each vertex of the contact area is considered as the application point of a force that has to be in a friction cone. Since our method allows dealing with surfaces that are intersecting each other, the contact surface is not known beforehand. Therefore, as soon as a non-inclusive contact is going to be used for the stability, we compute the intersection of the two polygons P_1 and P_2 that are involved, and that intersection $P_1 \cap P_2$ is the contact surface, and its vertices will bear the forces. We do not present here the algorithm to compute the intersection of two convex polygons, as it can be found in the literature easily.

Extension to singular cases

Our method can be extended to be used to approximate singular situations, such as finding an optimal contact with a linear or even punctual surface. This is done by giving a slight width to the point or the line. This approximation is physically grounded: in terms of real contacts, linear or punctual contacts do not exist. In fact, since all objects are deformable, even slightly, the contact area between two objects cannot be a perfect line, and must have a non-null area, which justifies that linear and punctual contacts can be modeled as thin contact surfaces. By defining such a surface, we impose partly the orientation of the contact. Here again, one must be careful with numerical issues. Dealing with small numbers (here we would like to take the width of a fraction of a centimeter) may induce conditioning problems. Also, having two close parallel constraints of opposite direction (i.e. $g(x) \leq \alpha$ and $-g(x) \leq \alpha$ with α small) is not a good practice in optimization as it will lead the solver to take small steps. Therefore, it is best to apply a scaling to the constraints by applying a geometrical scaling to P_1 and P_2 in the appropriate direction.
Likewise, constraints on the area of the ellipse should be based on the same formulation as in equation (3.6).

3.1.3 Simulation results

In order to illustrate our method, we present some examples starring the HRP-2 and ATLAS humanoid robots, that are typical posture generations encountered in multi-contact planning. For the implementation of our posture generator, we use the RobOptim optimization framework [MLBY13] relying on the IPOPT solver [WB06].

Inclined ladder climbing

In this first example, we generate a posture that is part of an inclined ladder climbing planning. We consider that the robot HRP-2 reached a posture in which its right foot is on the first step and its right hand is grasping the right guardrail, both of those contacts are bearing forces. Those contacts are fixed, and we search a posture that adds to it a geometrical contact between the left foot and the second step. We require the contact to include an ellipse with both radiuses bigger than 40% of the ladder step's width. The resulting posture and a close-up view of the contact areas are shown in Fig. 3.5. The latter shows clearly how an ellipse of sufficient size is found, included in the contact area between the left foot and the second step. It also shows that the contact forces on the right foot are located on the vertex of the intersection of the contact surfaces between the right foot and the first step, which was also generated with our method, in a prior posture generation. One can note that we use a contact area slightly smaller than the actual surface under the foot of the robot. We use indeed safety margins to account for modeling errors so that the obtained posture is achievable by the real robot.

Vertical ladder climbing

In the second example, we generate a posture in which the robot climbs a vertical ladder. In this particular step, the robot is using its right foot and both hands to maintain its stability on top of the first rung of the ladder. We search a posture that keeps those previous contacts and adds a geometrical contact between the left foot and the second rung of the ladder. The result of that optimization can be observed on figure 3.1, with the robot posture on the left and a close-up look at the contact areas on the right. The difficulty of this situation is that a contact has to be made with a very thin surface of the environment (the ladder rung). The contact chosen by our software includes an ellipse which first axis is the width of the robot's foot and second axis is as thin as the ladder rung. This example also illustrates one limitation of our method: it only considers planar contacts and if one wants to model a purely linear contact another contact model must be used, since our modeling of those singular cases is approximative.

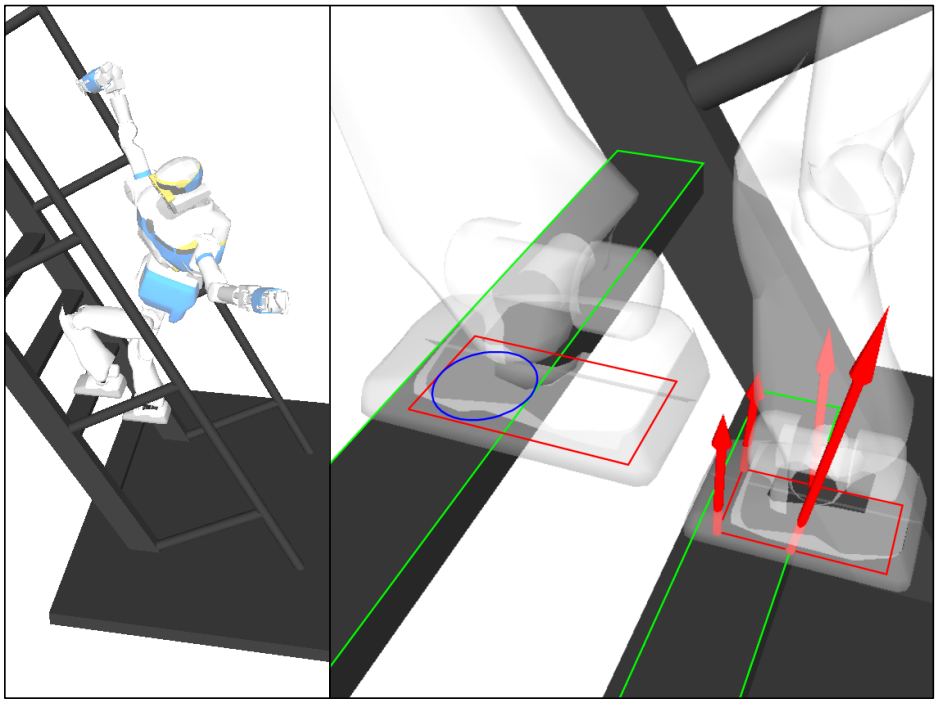


Figure 3.5 HRP2 ladder climbing posture and up close view of the contact areas (green/red: contact polygons; blue: contact ellipse; red arrows: contact forces resultants)

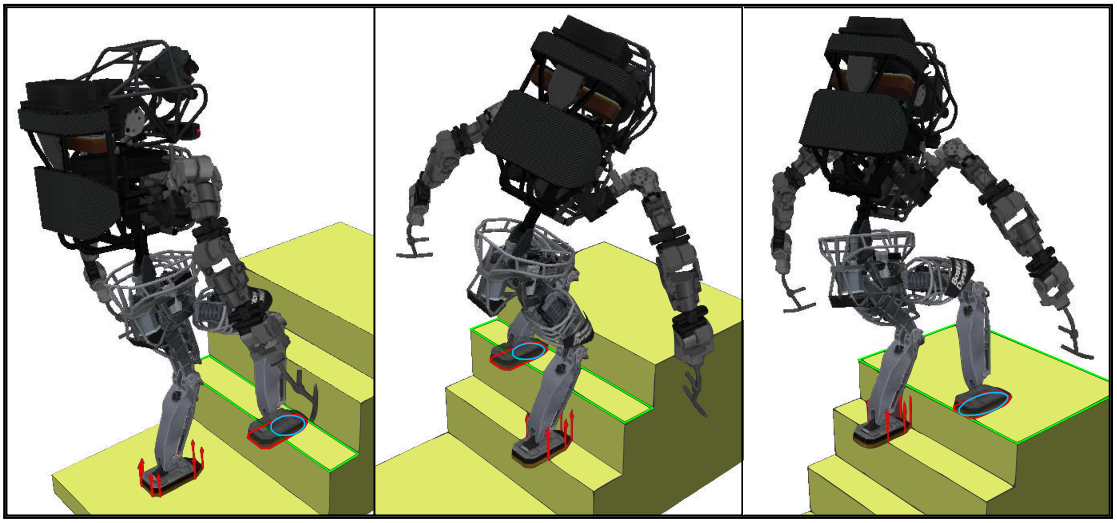


Figure 3.6 Atlas climbing stairs with small steps by maximizing the size of the contact areas (green/red: contact polygons; blue: contact ellipse; red arrows: contact forces resultants)

Climbing Stairs

In a third simulation, the ATLAS robot climbs a flight of stairs. All the steps are too small for the robot to put its entire foot on. Therefore, it has to make a non-inclusive contact and we propose to maximize the size of the contact area with the ellipse included in it, as explained in Section 3.1.2. The size of the contact area is limited by the fact that the foot cannot penetrate the wall behind each step. On Fig. 3.6, we present 3 postures generated in this environment. On each of those postures, we see that the ellipse's size is maximized until the foot enters in collision with the vertical wall behind each step. And when possible, like on the last step, the contact area is maximized without collision limitation and the foot is positioned as fully included in the support surface. We can see here that even when the size of the ellipse is maximized while competing with other nonlinear constraints like collision avoidance, our method still works well and leads us to a satisfactory solution. It has been used to actually make the robot climb a vertical ladder and presented in [VKA$^+$16] and [VKA$^+$14].

Walking along a path made of small objects

In the last simulation, the HRP-2 robot has to cross a gap by making contacts with small surfaces. As for the previous simulation, the two objects with which the feet of the robot will be in contact are too small for making a complete contact, instead, non-inclusive geometric contacts are found by maximizing the area of an ellipse that fits in the intersection of the polygons involved in the contact. Results are presented in figure 3.7. As we can see, the feet turns slightly, to be more aligned with the support surfaces, yet they do not become completely aligned with these supports, which would permit to get the biggest ellipse area. This is due to the fact that a posture cost is competing with the area cost, yielding this compromise. Under each simulation result, an image presents the disposition of the foot (in green) that is in geometrical non-inclusive contact with a step (in blue) and the optimal ellipse that has been found.

3.1.4 Discussion and conclusion

Generating arbitrarily shaped contact areas proved to be doable very simply in an optimization based posture generation module. We focused on writing constraints that have continuous gradients since the posture generation problem is dominantly smooth. Hence, our geometric contact model can be useful for other optimization based purposes, for example, control or trajectory optimization, and any gradient-based descent scheme which handles inequalities (e.g. [EMW10]). While we were expecting an increase of computation time due to the addition of new variables in the problem (5 more for a total of 80–100 variables) we noticed

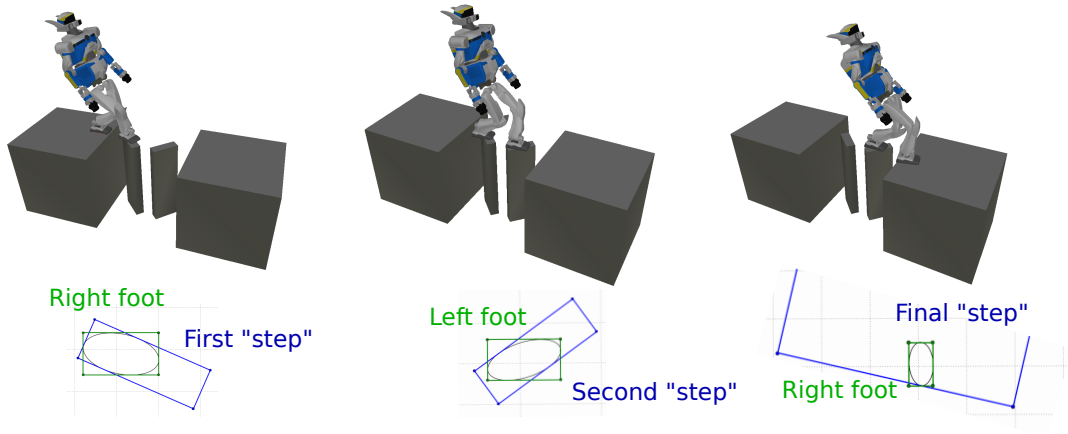


Figure 3.7 Simulation results for crossing a gap by walking on small items

that the timings obtained with this method are sensibly the same that with our previous version of the posture generator with full contact surface inclusion. Consequently, this method offering a richer contact search (exploration) during planning comes without degrading computation time. In fact, it truly allows us to substantially reduce the time spent by the user in ad-hoc tuning the shapes of the contact patches, or fixing the contact positions that were previously done by hand. Also, it is fairly easy to implement and extends multi-contact planning algorithms like the one described in [EKM13] to give it richer planning possibilities. This method extends straightforwardly to point cloud data as far as polygonal convex patches can be extracted. But one of its limitation is that in its current form, it cannot handle contacts with non-convex surfaces.

3.2 Torque derivation

In this section, we present an efficient algorithm to compute the Jacobian matrix of the joint torques in a robot with respect to all the articular variables of the robot and the variables involved in the expression of the external forces applied on the robot. The evaluation of the joint torques Jacobian matrix is useful to conduct an optimization algorithm in which the torques are involved in constraints or cost functions. The computation of the joint torques jacobian matrix is done by differentiating algorithm 4. In most cases, the acceleration of gravity could be replaced by its value on earth $\mathbf{a_c} = \vec{0}$ and $\mathbf{a_f} = g\vec{z}$. For the sake of genericity, we keep it as $\mathbf{a_c}$ and $\mathbf{a_f}$ in the algorithm description, which could be useful if the robot is in an accelerating vehicle, for example. We first write algorithm 4 in its matrix form(which is more convenient for the purpose of derivating it).

Algorithm 5 Inverse Static algorithm on Matrix Form

for $i = 0 : n_B$ **do**

$$f_i^G = \begin{bmatrix} \mathbf{m}_i^G \\ \mathbf{f}_i^G \end{bmatrix} = \mathbf{I}^W \begin{bmatrix} {}^i\mathbf{R}_W & \mathbf{0} \\ -{}^i\mathbf{R}_W \widehat{{}^i\mathbf{t}_W} & {}^i\mathbf{R}_W \end{bmatrix} \begin{bmatrix} \mathbf{a_c} \\ \mathbf{a_f} \end{bmatrix} - \begin{bmatrix} {}^i\mathbf{R}_W & -{}^i\mathbf{R}_W \widehat{{}^i\mathbf{t}_W} \\ \mathbf{0} & {}^i\mathbf{R}_W \end{bmatrix} \begin{bmatrix} \mathbf{m}_i^{ext} \\ \mathbf{f}_i^{ext} \end{bmatrix}$$

end for

for $i = n_J - 1 : 0$ **do**

$\tau_i = f_i^{G^T} S_i$

if $pred(i) \neq -1$ **then**

$$f_{pred(i)}^G \leftarrow \begin{bmatrix} \mathbf{m}_{pred(i)}^G \\ \mathbf{f}_{pred(i)}^G \end{bmatrix} + \begin{bmatrix} \mathbf{R}_i^{PtS^T} & \widehat{\mathbf{t}_i^{PtS}} \mathbf{R}_i^{PtS^T} \\ \mathbf{0} & \mathbf{R}_i^{PtS^T} \end{bmatrix} \begin{bmatrix} \mathbf{m}_i^G \\ \mathbf{f}_i^G \end{bmatrix}$$

end if

end for

The variables of this algorithm w.r.t. which we need to differentiate it are the configuration of the robot q and the variables of the external wrenches which can themselves depend on q and some other variables y, for example, in the case of a contact with another robot, the application point and value of those forces depend on q and on the configuration of the other robot. We denote $\dim(q)$ and $\dim(y)$ the dimensions of q and y respectively. We assume that the derivatives of m_i^{ext} and f_i^{ext}: $\frac{\partial m_i^{ext}}{\partial q}$, $\frac{\partial m_i^{ext}}{\partial y}$, $\frac{\partial f_i^{ext}}{\partial q}$, and $\frac{\partial f_i^{ext}}{\partial y}$ are known.

Besides m_i^{ext} and f_i^{ext}, all the quantities of the algorithm depend only on q. Therefore, the derivation w.r.t y is trivial and is automatically computed by our final algorithm. From here we will focus on the derivation w.r.t q.

Recall the expression of the Jacobian of a robot's body:

$$\mathbf{Jac}_i^W = \begin{bmatrix} \frac{\partial^i \mathbf{R}_W}{\partial q_0} & \cdots & \frac{\partial^i \mathbf{R}_W}{\partial q_j} & \cdots & \frac{\partial^i \mathbf{R}_W}{\partial q_{dof}} \\ \frac{\partial^i \mathbf{t}_W}{\partial q_0} & \cdots & \frac{\partial^i \mathbf{t}_W}{\partial q_j} & \cdots & \frac{\partial^i \mathbf{t}_W}{\partial q_{dof}} \end{bmatrix} = \begin{bmatrix} \omega_{i,0} & \cdots & \omega_{i,j} & \cdots & \omega_{i,dof} \\ v_{i,0} & \cdots & v_{i,j} & \cdots & v_{i,dof} \end{bmatrix} \tag{3.7}$$

Note the following relations that make use of the robot's Jacobian to compute derived quantities:

$$\frac{\partial^i \mathbf{R}_W \mathbf{u}}{\partial q_j} = {}^i\mathbf{R}_W \mathbf{u} \wedge \omega_{i,j} = {}^i\mathbf{R}_W \widehat{\mathbf{u}} \omega_{i,j} \tag{3.8}$$

$$\frac{\partial^i \mathbf{R}_W {}^i\mathbf{t}_W \wedge \mathbf{u}}{\partial q_j} = {}^i\mathbf{R}_W \mathbf{v}_{i,j} \wedge \mathbf{u} + {}^i\mathbf{R}_W \left({}^i\mathbf{t}_W \wedge \mathbf{u} \right) \wedge \omega_{i,j} \tag{3.9}$$

$$= -{}^i\mathbf{R}_W \widehat{\mathbf{u}} \mathbf{v}_{i,j} + {}^i\mathbf{R}_W \widehat{\left({}^i\widehat{\mathbf{t}_W} \mathbf{u} \right)} \omega_{i,j} \tag{3.10}$$

Let's differentiate the first equation of 5 w.r.t q_j.

$$A = \begin{bmatrix} {}^{i}\mathbf{R}_W & \mathbf{0} \\ -{}^{i}\mathbf{R}_W \widehat{{}^{i}\mathbf{t}_W} & {}^{i}\mathbf{R}_W \end{bmatrix} \begin{bmatrix} \mathbf{a_c} \\ \mathbf{a_f} \end{bmatrix} \tag{3.11}$$

$$B = \begin{bmatrix} {}^{i}\mathbf{R}_W & -{}^{i}\mathbf{R}_W \widehat{{}^{i}\mathbf{t}_W} \\ \mathbf{0} & {}^{i}\mathbf{R}_W \end{bmatrix} \begin{bmatrix} \mathbf{m}_i^{ext} \\ \mathbf{f}_i^{ext} \end{bmatrix} \tag{3.12}$$

$$f_i^G = \mathbf{I}^W A - B \tag{3.13}$$

$$\frac{\partial A}{\partial q_j} = \begin{bmatrix} {}^{i}\mathbf{R}_W \widehat{\mathbf{ac}} \omega_{i,j} \\ -{}^{i}\mathbf{R}_W \left(\widehat{\widehat{{}^{i}\mathbf{t}_W} \mathbf{ac}} \right) \omega_{i,j} + {}^{i}\mathbf{R}_W \widehat{\mathbf{ac}} v_{i,j} + {}^{i}\mathbf{R}_W \widehat{\mathbf{af}} \omega_{i,j} \end{bmatrix} \tag{3.14}$$

$$\frac{\partial B}{\partial q_j} = \begin{bmatrix} -{}^{i}\mathbf{R}_W \left(\widehat{\widehat{{}^{i}\mathbf{t}_W} \mathbf{f_i^{ext}}} \right) \omega_{i,j} + {}^{i}\mathbf{R}_W \widehat{\mathbf{f_i^{ext}}} v_{i,j} + {}^{i}\mathbf{R}_W \widehat{\mathbf{m_i^{ext}}} \omega_{i,j} \\ {}^{i}\mathbf{R}_W \widehat{\mathbf{f_i^{ext}}} \omega_{i,j} \end{bmatrix} \tag{3.15}$$

$$+ \begin{bmatrix} {}^{i}\mathbf{R}_W & -{}^{i}\mathbf{R}_W \widehat{{}^{i}\mathbf{t}_W} \\ \mathbf{0} & {}^{i}\mathbf{R}_W \end{bmatrix} \begin{bmatrix} \frac{\partial \mathbf{m}_i^{ext}}{\partial q} \\ \frac{\partial \mathbf{f}_i^{ext}}{\partial q} \end{bmatrix} \tag{3.16}$$

We rearrange the equations in order to make the jacobian appear. Putting all the pieces of ω and v together allows grouping all the j-derivatives to obtain the derivative of f_i^G with a single equation.

$$\mathbf{M} = \begin{bmatrix} {}^{i}\mathbf{R}_W \widehat{\mathbf{ac}} & 0 \\ -{}^{i}\mathbf{R}_W \left(\widehat{\widehat{{}^{i}\mathbf{t}_W} \mathbf{ac}} \right) + {}^{i}\mathbf{R}_W \widehat{\mathbf{af}} & {}^{i}\mathbf{R}_W \widehat{\mathbf{ac}} \end{bmatrix} \tag{3.17}$$

$$\mathbf{N} = \begin{bmatrix} -{}^{i}\mathbf{R}_W \left(\widehat{\widehat{{}^{i}\mathbf{t}_W} \mathbf{f_i^{ext}}} \right) + {}^{i}\mathbf{R}_W \widehat{\mathbf{m_i^{ext}}} & {}^{i}\mathbf{R}_W \widehat{\mathbf{f_i^{ext}}} \\ {}^{i}\mathbf{R}_W \widehat{\mathbf{f_i^{ext}}} & 0 \end{bmatrix} \tag{3.18}$$

$$\frac{\partial f_i^G}{\partial q} = (\mathbf{I}^W \mathbf{M} - \mathbf{N}) \mathbf{Jac}_i^W + \begin{bmatrix} {}^{i}\mathbf{R}_W & -{}^{i}\mathbf{R}_W \widehat{{}^{i}\mathbf{t}_W} \\ \mathbf{0} & {}^{i}\mathbf{R}_W \end{bmatrix} \begin{bmatrix} \frac{\partial \mathbf{m}_i^{ext}}{\partial q} \\ \frac{\partial \mathbf{f}_i^{ext}}{\partial q} \end{bmatrix} \tag{3.19}$$

The derivation of the second equation is obvious:

$$\tau_i = f_i^{G^T} S_i \tag{3.20}$$

$$\frac{\partial \tau_i}{\partial q} = S_i^T \frac{\partial f_i^G}{\partial q} \tag{3.21}$$

$$\tag{3.22}$$

Note the following relations:

$$\frac{\partial \mathbf{R}_i^{JT}\mathbf{u}}{\partial q_j} = -S_{i,j}^R \wedge (\mathbf{R}_i^{JT}\mathbf{u}) = \widehat{(\mathbf{R}_i^{JT}\mathbf{u})} S_{i,j}^R \tag{3.23}$$

$$\frac{\partial \mathbf{t}_i^J \wedge \mathbf{R}_i^{JT}\mathbf{u}}{\partial q_j} = S_{i,j}^t \wedge \left(\mathbf{R}_i^{JT}\mathbf{u}\right) - \mathbf{t}_i^J \wedge \left(S_{i,j}^R \wedge \left(\mathbf{R}_i^{JT}\mathbf{u}\right)\right) \tag{3.24}$$

$$= -\widehat{(\mathbf{R}_i^{JT}\mathbf{u})} S_{i,j}^t + \left(\left(\mathbf{R}_i^{JT}\mathbf{u}\right)\cdot \mathbf{t}_i^{JT} - \left((\mathbf{R}_i^{JT}\mathbf{u})^T \cdot \mathbf{t}_i^J\right)\mathbf{I}_3\right) S_{i,j}^R \tag{3.25}$$

$$\tag{3.26}$$

The last equation's derivation goes as follows:

$$f_{pred(i)}^G \leftarrow f_{pred(i)}^G + \begin{bmatrix} \mathbf{R}_i^{PtS^T} & \widehat{\mathbf{t}_i^{PtS}}\mathbf{R}_i^{PtS^T} \\ \mathbf{0} & \mathbf{R}_i^{PtS^T} \end{bmatrix} \begin{bmatrix} \mathbf{m}_i^G \\ \mathbf{f}_i^G \end{bmatrix} \tag{3.27}$$

$$= f_{pred(i)}^G + \begin{bmatrix} \mathbf{R}_i^{xT} & \widehat{\mathbf{t}_i^x}\mathbf{R}_i^{xT} \\ \mathbf{0} & \mathbf{R}_i^{xT} \end{bmatrix} \begin{bmatrix} \mathbf{R}_i^{JT} & \widehat{\mathbf{t}_i^J}\mathbf{R}_i^{JT} \\ \mathbf{0} & \mathbf{R}_i^{JT} \end{bmatrix} \begin{bmatrix} \mathbf{m}_i^G \\ \mathbf{f}_i^G \end{bmatrix} \tag{3.28}$$

$$\tag{3.29}$$

We define K_i as:

$$K_i = \begin{bmatrix} \widehat{(\mathbf{R}_i^{JT}\mathbf{m}_i^G)} + \left(\mathbf{R}_i^{JT}\mathbf{f}_i^G\right)\cdot \mathbf{t}_i^{JT} - \left((\mathbf{R}_i^{JT}\mathbf{f}_i^G)^T \cdot \mathbf{t}_i^J\right)\mathbf{I}_3 & -\widehat{(\mathbf{R}_i^{JT}\mathbf{f}_i^G)} \\ \widehat{(\mathbf{R}_i^{JT}\mathbf{f}_i^G)} & 0 \end{bmatrix} \tag{3.30}$$

$$\frac{\partial f_{pred(i)}^G}{\partial q} \leftarrow \frac{\partial f_{pred(i)}^G}{\partial q} - \begin{bmatrix} \mathbf{R}_i^{xT} & \widehat{\mathbf{t}_i^x}\mathbf{R}_i^{xT} \\ \mathbf{0} & \mathbf{R}_i^{xT} \end{bmatrix} K_i \mathbf{S_i} \tag{3.31}$$

In the following algorithm, we use the following notation x to combine the variables q and y:

$$x = \begin{bmatrix} q & y \end{bmatrix} \tag{3.32}$$

$$\frac{\partial f_{ext}}{\partial x} = \begin{bmatrix} \frac{\partial \mathbf{m}_i^{ext}}{\partial q} & \frac{\partial \mathbf{m}_i^{ext}}{\partial y} \\ \frac{\partial \mathbf{f}_i^{ext}}{\partial q} & \frac{\partial \mathbf{f}_i^{ext}}{\partial y} \end{bmatrix} \tag{3.33}$$

The entire algorithm for calculating the torque jacobian is presented in Algorithm 6.

This algorithm efficiently computes the exact joint torques Jacobian for a robotic system subject to any type of external forces.

Algorithm 6 Torque Jacobian Calculation

Compute the Jacobian of generalized forces

for $i = 0 : n_B$ **do**

$$f_i^G = \mathbf{I}^W \begin{bmatrix} {}^i\mathbf{R}_W & \mathbf{0} \\ -{}^i\mathbf{R}_W \widehat{{}^i\mathbf{t}_W} & {}^i\mathbf{R}_W \end{bmatrix} \begin{bmatrix} \mathbf{a_c} \\ \mathbf{a_f} \end{bmatrix} - \begin{bmatrix} {}^i\mathbf{R}_W & -{}^i\mathbf{R}_W \widehat{{}^i\mathbf{t}_W} \\ \mathbf{0} & {}^i\mathbf{R}_W \end{bmatrix} \begin{bmatrix} \mathbf{m}_i^{ext} \\ \mathbf{f}_i^{ext} \end{bmatrix}$$

$$\mathbf{M} = \begin{bmatrix} {}^i\mathbf{R}_W \widehat{\mathbf{ac}} & 0 \\ -{}^i\mathbf{R}_W \widehat{\left(\widehat{{}^i\mathbf{t}_W}\mathbf{ac}\right)} + {}^i\mathbf{R}_W \widehat{\mathbf{af}} & {}^i\mathbf{R}_W \widehat{\mathbf{ac}} \end{bmatrix}$$

$$\mathbf{N} = \begin{bmatrix} -{}^i\mathbf{R}_W \widehat{\left(\widehat{{}^i\mathbf{t}_W}\mathbf{f_i^{ext}}\right)} + {}^i\mathbf{R}_W \widehat{\mathbf{m_i^{ext}}} & {}^i\mathbf{R}_W \widehat{\mathbf{f_i^{ext}}} \\ {}^i\mathbf{R}_W \widehat{\mathbf{f_i^{ext}}} & 0 \end{bmatrix}$$

$$\frac{\partial f_i^G}{\partial x} = (\mathbf{I}^W\mathbf{M} - \mathbf{N}) \begin{bmatrix} \mathbf{Jac_i^W} & \mathbf{0}_{6\times\dim(y)} \end{bmatrix} + \begin{bmatrix} {}^i\mathbf{R}_W & -{}^i\mathbf{R}_W \widehat{{}^i\mathbf{t}_W} \\ \mathbf{0} & {}^i\mathbf{R}_W \end{bmatrix} \frac{\partial f_{ext}}{\partial x}$$

end for

Compute the Jacobian of torques

for $i = n_J - 1 : 0$ **do**

$$\frac{\partial \tau_i}{\partial x} = \begin{bmatrix} \mathbf{S_i} & \mathbf{0}_{6\times\dim(y)} \end{bmatrix}^T \frac{\partial f_i^G}{\partial x}$$

if $pred(i) \neq -1$ **then**

$$\mathbf{K} = \begin{bmatrix} \widehat{(\mathbf{R}_i^{JT}\mathbf{m}_i^G)} + \left(\mathbf{R}_i^{JT}\mathbf{f}_i^G\right)\cdot\mathbf{t}_i^{JT} - \left(\left(\mathbf{R}_i^{JT}\mathbf{f}_i^G\right)^T\cdot\mathbf{t}_i^J\right)\mathbf{I}_3 & -\widehat{(\mathbf{R}_i^{JT}\mathbf{f}_i^G)} \\ \widehat{(\mathbf{R}_i^{JT}\mathbf{f}_i^G)} & 0 \end{bmatrix}$$

$$f_{pred(i)}^G \leftarrow f_{pred(i)}^G + \mathbf{X}_i^{PtS}(q)^{-*} f_i^G$$

$$\frac{\partial f_{pred(i)}^G}{\partial x} \leftarrow \frac{\partial f_{pred(i)}^G}{\partial x} - \begin{bmatrix} \mathbf{R}_i^{xT} & \widehat{\mathbf{t}}_i^x \mathbf{R}_i^{xT} \\ \mathbf{0} & \mathbf{R}_i^{xT} \end{bmatrix} \mathbf{K}\cdot\begin{bmatrix} \mathbf{S_i} & \mathbf{0}_{6\times\dim(y)} \end{bmatrix}$$

end if

end for

3.3 On the use of lifted variables for Robotics Posture Generation

In this section, we present our work on the use of lifted variables for robotics posture generation. The goal of this thesis is, on one hand, to enable writing more complex and various problems in simpler ways by improving the formulation of posture generation problems, and on the other hand, to improve and adapt the resolution methods for those problems. This section focuses on the later. We formulate posture generation problems in an optimization friendly way in order to solve them using nonlinear constrained optimization algorithms. In the following, we propose to apply the methods of lifted optimization presented in [AD10] to posture generation problems.

The idea of lifting variables consists in taking a complicated equation $F(u) = 0$ and decomposing it into simpler functions $\{f_1, f_2, \ldots, f_m\}$ that all depend only on u and on the output of functions of lower index 3.34.

$$x_i = f_i(u, x_1, x_2, \ldots, x_{i-1}),\ \forall i \in [1, \ldots, m] \tag{3.34}$$

Such that they can be used in a chain to recompose F:

$$\begin{aligned} F(u) &= f_F\,(u, x_1, x_2, \ldots, x_m) && (3.35)\\ &= f_F\,(u, f_1(u), f_2(u, x_1), f_3(u, x_1, x_2), \ldots, f_m(u, x_1, \ldots, x_m - 1)) && (3.36)\\ &= f_F\,(u, f_1(u), f_2(u, f_1(u)), f_3(u, f_1(u), f_2(u, f_1(u))), \ldots, f_m(u, \ldots, f_{m-1}(u, \cdots))) && (3.37) \end{aligned}$$

With this formulation and the additional 'lifted' variables $x_i,\ i \in [1, \ldots, m]$, the problem $F(u) = 0$ can be rewritten as a lifted problem:

$$G(u,x) = \begin{pmatrix} f_1(u) - x_1 \\ f_2(u, x_1) - x_2 \\ f_3(u, x_1, x_2) - x_3 \\ \vdots \\ f_m(u, x_1, \ldots, x_{m-1}) - x_m \\ f_F(u, x_1, \ldots, x_{m-1}, x_m) \end{pmatrix} = 0 \tag{3.38}$$

This modified formulation can be used for any function of our optimization problem to reduce the complexity of individual equations at the cost of adding extra variables and

constraints. By reducing the complexity of the functions, we ought to improve the quality of their quadratic approximations, which should lead to better convergence properties of the Newton scheme.

One obvious drawback of this approach for solving a nonlinear problem is the increased size of the linearized problem to solve at each iteration of the optimization. Fortunately, [AD10] proposes an algorithmic trick leveraging the triangularity of the jacobian of $G(u,x)$ to condense the lifted problem into a problem of the same size as the original one. Thus, the resolution of each iteration should take approximately the same time as with the original problem.

The lifted variable approach is known to be superior to the non-lifted one in the problem of shooting methods for boundary value problems [Osb69]. With good intuition, the authors of [AD10] state that the lifting idea could be directly transferred in the context of kinematic chain arising in robotics. Indeed, as we state in Section 2.1, the equation that governs the transformation of a robot's end effector on body i, ${}^{i}X_{W}(q)$ is constructed iteratively and seems to be a good candidate for lifting. If we consider a single chain robot with m bodies, such that $\kappa(m) = \{0,1,2,\dots m\}$ the ${}^{m}X_{W}(q) = X_{\text{goal}}$ can naturally be lifted as follows:

$$G(u,x) = \begin{pmatrix} {}^{1}X_{W}(q) - x_1 \\ {}^{2}X_{1}(q) \cdot x_1 - x_2 \\ {}^{3}X_{2}(q) \cdot x_2 - x_3 \\ \vdots \\ {}^{m}X_{m-1}(q) \cdot x_{m-1} - x_{m-1} \\ x_{m-1} - X_{\text{goal}} \end{pmatrix} = 0 \tag{3.39}$$

The transformation of a body is used in almost all the constraints and cost functions of posture generation problems, thus, we can use this lifting approach in most functions of our problems. This formulation can be generalized to more complex robots by considering/lifting only the bodies in the direct chain from the base to the end effector.

3.3.1 Lifting Algorithm

In order to explore the capabilities of optimization on lifted problems, we developed and studied several lifting approaches based on different lifting criterion. A lifting criterion is a criterion that should be satisfied as best as possible by all the functions present at the end of the lifting process. For example, the degree of a polynomial can be a used, one could require that all the functions are polynomials of at most degree m. In 3.39, we lifted based one the criterion that each function should only contain a single joint transformation.

Let us consider an example with a simple planar 2 axis robot, with 2 successive links of length 1 attached by revolute joints. Its articular variables are denoted $q = (q_1, q_2)$ and the expression of the end-effector's (EE) position is the following:

$$\begin{pmatrix} \cos(q_1) + \cos(q_1)\cos(q_2) - \sin(q_1)\sin(q_2) \\ \sin(q_1) + \cos(q_1)\sin(q_2) + \cos(q_2)\sin(q_1) \end{pmatrix} = \begin{pmatrix} \mathrm{EE}_x(q) \\ \mathrm{EE}_y(q) \end{pmatrix} \tag{3.40}$$

Those equations are polynomials in the variables $\cos(q_1)$, $\sin(q_1)$, $\cos(q_2)$, $\sin(q_2)$. Furthermore, they are polynomials of degree 1 in each of those variables separately. This observation extends to the case of any robot composed of the joints presented in Section 2.2 and that does not contain kinematic loops (kinematic loops are not considered in this thesis). The operators 'addition', 'subtraction', 'multiplication', 'sinus' and 'cosinus' are enough to describe the kinematics of a robot.

We developed a generic lifting algorithm that allows to automatically lift symbolic functions, and in particular, polynomials of trigonometric functions, based on a given criterion. To do so, we use symbolic variables. A symbolic equation can be seen as a graph of atomic operations (addition, multiplication, and other functions), which can, in turn, be explored and truncated at appropriate places where we insert additional variables. Figure 3.8 presents the graph of atomic operations for equation (3.40), where the green circles represent the original variables, the yellow ones represent the lifted variables, the pink ones represent the atomic operations and the red circles represent the final atomic operations that lead to the end effector equations.

To lift a set of equations, we explore the graph from left to right. At each node of the graph, we consider the sub-graph that lies on its left. If this sub-graph violates our criterion, a lifting variable is added and replaces the sub-graph. Then we iterate this process on the remaining graph until it does not violate the criterion. Given that the equations to solve are polynomials of trigonometric functions, we decided to lift them based on a criterion of maximum degree of the lifted polynomials. We define the degree function as the degree of a polynomial with the specificity that trigonometric functions are considered to be polynomials of infinite degree (And thus are always lifted, no matter the maximum degree criterion). We also consider the option of eliminating or not the trigonometric functions. By eliminating the trigonometric functions, we mean that we reformulate the problem so that, given a variable x that appears in our equations in the forms $\cos(x)$ and/or $\sin(x)$, we introduce 2 new variables s_x and c_x, replace all the occurrences of $\cos(x)$ by c_x and all the occurrences of $\sin(x)$ by s_x, and to ensure the equivalence of the two formulations, we add the equation ${c_x}^2 + {s_x}^2 = 1$ to our problem.

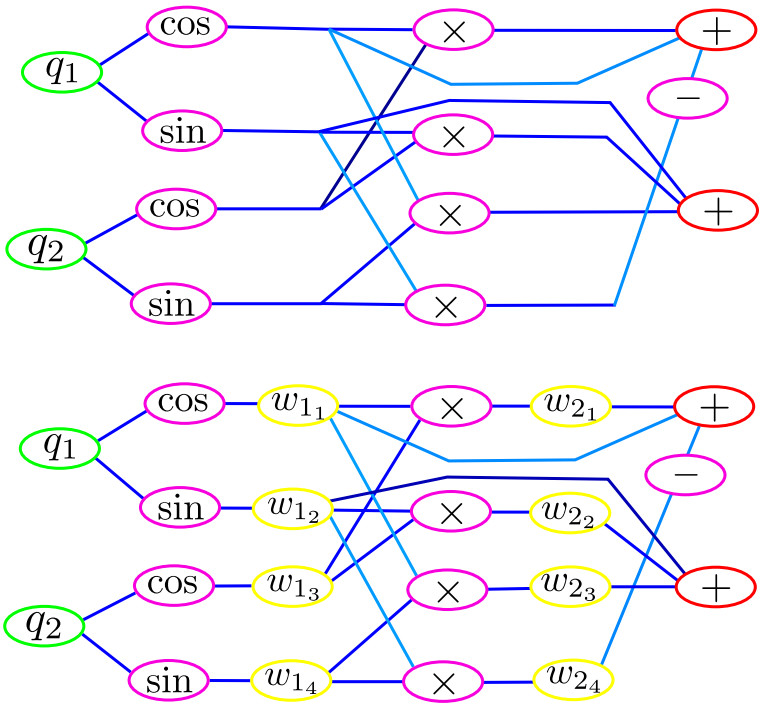


Figure 3.8 Computation graph for end effector position of a 2 axis planar robotic arm. Top: Direct equations. Bottom: Lifted equations. Green circles represent the original variables, yellow the lifted ones, pink represent intermediary operations and red represent the final operation for the end effector equations

For example, lifting the set of equation (3.40) with a maximum degree of 2 and without eliminating the trigonometric functions gives the following system, with all the w_{i_j} being the additional lifted variables:

$$\begin{array}{c}\textbf{Lifted Equations}\\ \left(\begin{array}{c}\cos(q_1) = w_{1_1}\\ \sin(q_1) = w_{1_2}\\ \cos(q_2) = w_{1_3}\\ \sin(q_2) = w_{1_4}\\ w_{1_1}.w_{1_3} = w_{2_1}\\ w_{1_2}.w_{1_3} = w_{2_2}\\ w_{1_1}.w_{1_4} = w_{2_3}\\ w_{1_2}.w_{1_4} = w_{2_4}\end{array}\right)\\ \textbf{End-effector's position}\\ \left(\begin{array}{c}w_{1_1} + w_{2_1} - w_{2_4} = EE_x\\ w_{1_2} + w_{2_2} + w_{2_3} = EE_y\end{array}\right)\end{array} \tag{3.41}$$

Adding to it the elimination of trigonometric functions gives the following system. Note that here, all the equations are polynomials of degree at most 2, no trigonometric function is present, and the optimization variables are $\left(c_{q_1}, c_{q_2}, s_{q_1}, s_{q_2}\right)$ instead of (q_1, q_2).

$$\begin{array}{c}\textbf{Lifted Equations}\\ \left(\begin{array}{c}c_{q_1}.c_{q_2} = w_{2_1}\\ s_{q_1}.c_{q_2} = w_{2_2}\\ c_{q_1}.s_{q_2} = w_{2_3}\\ s_{q_1}.s_{q_2} = w_{2_4}\end{array}\right)\\ \textbf{Additional equations}\\ \left(\begin{array}{c}{c_{q_1}}^2 + {s_{q_1}}^2 = 1\\ {c_{q_2}}^2 + {s_{q_2}}^2 = 1\end{array}\right)\\ \textbf{End-effector's position}\\ \left(\begin{array}{c}c_{q_1} + w_{2_1} - w_{2_4} = EE_x\\ s_{q_1} + w_{2_2} + w_{2_3} = EE_y\end{array}\right)\\ \textbf{Eliminated equations (Not used in further calculations)}\\ \left(\begin{array}{c}\cos(q_1) = c_{q_1}\\ \sin(q_1) = s_{q_1}\\ \cos(q_2) = c_{q_2}\\ \sin(q_2) = s_{q_2}\end{array}\right)\end{array} \tag{3.42}$$

This lifting methodology can be extended to most robotic systems. We also developed a lifting method that follows the principle described in (3.39), in which case, we get another set of equations similar to (3.3.1).

3.3.2 Optimization on lifted variables

We want to estimate the gain of performances yielded by the use of a lifted formulation on a posture generation problem. In order to do so, we implemented and/or used several optimization algorithms.

Given the lifted equations and their derivatives, it is straightforward to solve and compare the results obtained by the different solvers provided in Matlab (trust-region-reflective; interior-point; active-set and SQP) for the set of equations with or without lifting. A limitation of that approach is that it does not allow us to easily use the condensed approach presented in [AD10] in the resolution of the problems because it is necessary to de-condensate and then update the problem between 2 iterations of the optimization. The lifted condensed problem is meant to be equivalent to the lifted full-space problem (lifted non-condensed problem), in the sense that they generate the same optimization steps. Solving lifted problems with or without using the condensation trick gives the same result in the same number of iterations. The only difference is that with the lifted full-space problem, each iteration takes more time to compute. In our study, this is not a problem, we want to evaluate the performances in terms of number of iterations first, and if it appears that the lifted problem resolution outperforms the non-lifted, then we will implement the condensation trick for our problems and solvers, which will bring the per iteration time to approximately the same as in the direct problem case. Therefore, we only tried those algorithms on the direct problems (non-lifted problem) and the lifted full-space problems.

In addition to the Matlab solvers, we implemented the optimization algorithms proposed in [AD10], namely: the Lifted Newton, the Lifted Gauss-Newton and the Lifted SQP. With the only difference that we use the symbolic expressions of all our function and compute the symbolic expressions of the Jacobians and Hessians of our problems, instead of using auto-differentiation methods.

We implemented the Newton and Gauss-Newton methods for condensed lifted problems and obtained the same results as the ones presented in paragraph 5.2 of [AD10] that show the efficiency of a lifted approach in a specific kind of root finding problems. Our problems require the use of nonlinear cost and constraints, thus, a Newton method is not sufficient to solve them, whereas an SPQ method is.

We implemented the lifted and direct SQP approaches as described in paragraph 3.2.3 of [AD10]. The hessian matrices in our problems are usually not symmetric positive definite,

so we need to use a globalization method as well as a regularization of the Hessian to enforce convergence. For globalization, we implemented some line search methods with Armijo and Wolfe conditions, as well as a filter with line search (see A.5.2). And for regularization, we approximate the full-space hessian matrix through BFGS updates (with Powell's correction) or SR1 updates. We implemented different ways to initialize the lifted variables. They can either be all initialized with zero value, or all random, or their initial value can be computed such that all the lifter equations are satisfied(m first lines of 3.38). Those methods are presented in [BGLS02].

3.3.3 Results, experimentation

In order to evaluate the performances of the lifting approach in the resolution of posture generation problems, we conducted many experiments in which we solved simple inverse kinematics problems with the different solvers and lifting methods available.

A difficulty that arises in that endeavor lays in the inherent combinatorial aspect on the choice of formulation and resolution method. Indeed, we need to choose one item amongst each or the categories cited below:

- Type of lifting method
  - None, direct problem
  - Lifted with maximum degree of 2
  - Lifted based on Joints(one joint transformation per equation)
- Treatment of trigonometric equations
  - Keep trigonometric equations
  - Eliminate trigonometric equations
- Type of Solver
  - Our custom SQP
  - fmincon (active-set, interior point, SQP)
- Type of initialization of the lifted variables
  - Zeros
  - Random
  - Satisfying the lifted equations

- Type of Hessian update
  - None
  - BFGS
  - SR1
- Type of globalization
  - Wolfe
  - Armijo
  - Filter
- Cost function
  - None
  - Norm 2 of articular parameters

The number of possible combination of choices is large and grows with any new possibility added.

We devised a set of simple robots to be the basis of our experimentation. For simplicity, we focused on fixed base robots with only revolute joints. To study the influence of the number of degrees of freedom of a robot on the optimization performance, we studied planar robots composed of n successive links of length 1 attached to one another by revolute joints. We also studied a 7 axis 3D manipulator arm robot and a planar upper body robot with 2 arms. Illustration of those different types of robots are given in figure 3.9, where blue lines represent the links, red circles represent the revolute joints, green squares represent the origin of links, the biggest green square being the origin of the base link of the robot, and the red cross is the end effector.

We propose to use the following test to compare the performances of different lifting methods for all our experiments:

1. Choose a robot and a resolution method
2. Generate 1000 pairs of initial configuration and end effector goal
3. Solve the problem of finding a configuration in which the end effector reaches the goal for all the available formulations:
   - Direct: Direct formulation

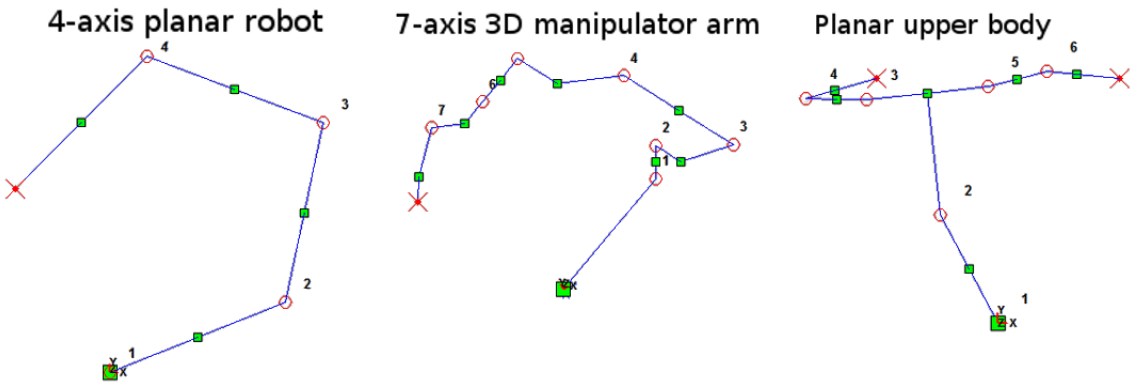


Figure 3.9 Examples of robots used

- Direct Trigo: Direct with trigonometric elimination
- Lifted: Lifted with maximum degree 2
- Lifted Trigo: Lifted with maximum degree 2 and trigonometric elimination
- Lifted Joint: Lifted based on joints
- Lifted Joint Trigo: Lifted based on joints and trigonometric elimination

This approach allows to fairly compare the performances of all lifting approaches on a given problem. And to observe the influence of the number of degrees of freedom of the robot on those performances, we use the n-axis planar robot model. We plot the average number of iterations needed to reach convergence against the number of degrees of freedom of the planar robot. In figure 3.10 we present the results obtained for the resolution with our custom SQP, using BFGS updates and a filter line-search, without cost function, initializing all the variables randomly. In figure 3.11 we present the best case in favor of the lifted approach that we encountered, those results are obtained for the resolution with our custom SQP, using SR1 updates and a filter line-search, without cost function, initializing all the variables such that the lifted equations are satisfied.

The results presented in figure 3.10 show that the direct formulation outperforms every form of lifting for almost all the robots tested. This is the typical result that we observed in most cases.

In figure 3.11, we show one specific case where, with long kinematic chains ($\geq$ 10 DOF), the lifted methods are slightly faster to solve.

We studied many of the possible configurations of resolution methods (22 in total), but not all of them. Still, it seems possible to draw some conclusions out of it, because in most cases, the direct approach outperforms the lifted one.

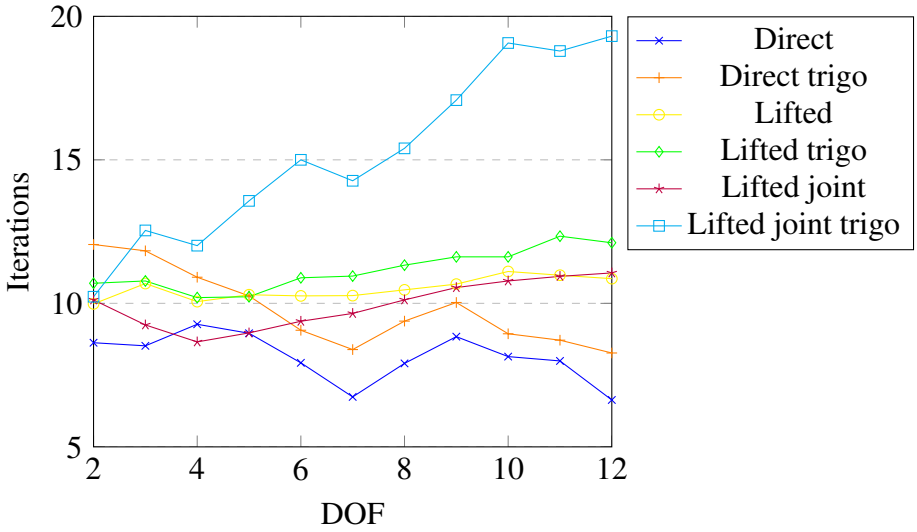


Figure 3.10 Number of iterations to reach convergence against number of DOF of the n-axis planar robot for the different lifting methods solved with our Custom SQP, BFGS, filter line-search, random initialization

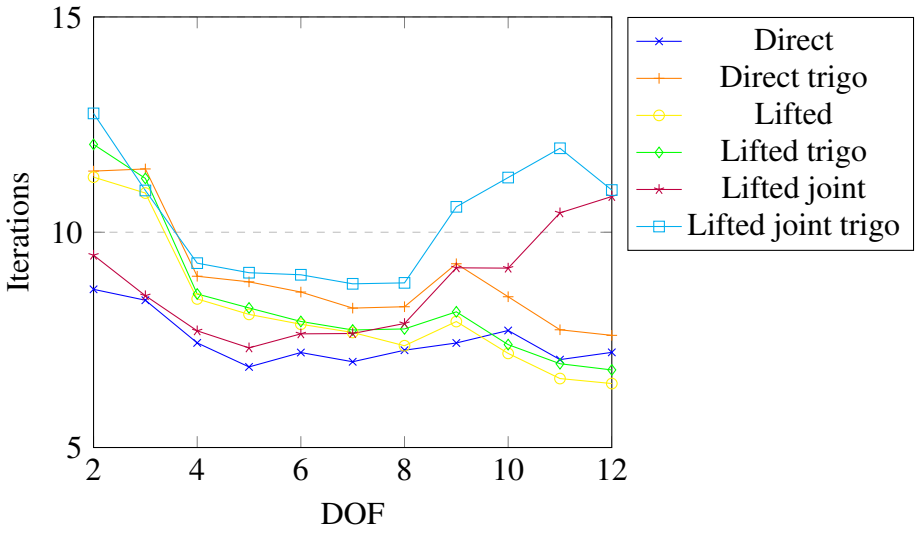


Figure 3.11 Number of iterations to reach convergence against number of DOF of the n-axis planar robot for the different lifting methods solved with our Custom SQP, SR1, filter Line-Search, initialization satisfying lifted equations

The overall results we extracted from those experiments is that the best performing method is: solving the direct formulation with our custom solver, using BFGS regularization and either Armijo or filter line search and no cost function.

We observed that when the lifted variables are initialized randomly the direct problem is usually solved faster than the lifted one. Whereas when the lifted variables are initialized so that the lifted equations are initially satisfied, the results are sensibly the same for lifted and direct approaches.

After comparison of all the results that we obtained, for all the planar robots with 2 to 12 degrees of freedom and across all the resolution methods that we tried, the best performing formulation is the direct one, except for the 7 DOF robot, where the resolution of the lifted problem with maximum degree 2 outperforms slightly all the other methods.

Overall, it seems that the idea of lifting does not provide better performances than the direct formulation for solving inverse kinematics problem. In the view of that observation, we decided to not go further in this research direction. Even though intuitively, the idea of lifting seems to apply naturally to robotics equations, the results obtained were not satisfying enough to justify investing more efforts in this approach so we decided to move on.

3.4 Conclusion

In this chapter, we presented three different and unrelated contributions to the field. First, we proposed a convenient formulation of contact constraints that allows generating non-inclusive contacts between two surfaces while ensuring that the size of the intersection is satisfactory. Second, we presented a generic differentiation of the torque computation algorithm, allowing to compute the joint torque jacobian of a robotic system. And finally, we described our endeavor to use the idea of variable lifting in optimization to solve posture generation problems, which unfortunately proved inefficient to accelerate the convergence of our solvers on those problems.

In the next chapter, we will dive deeper in the field of nonlinear optimization on manifolds, study how an SQP algorithm can be modified to be able to deal with manifolds and finally present our own implementation of such an algorithm.

Chapter 4

Optimization on non-Euclidean Manifolds

4.1 Introduction to optimization on Manifolds

Posture generation has been traditionally formulated as a problem over a Euclidean space. Robots variable may, however, be more naturally expressed over non-Euclidean manifolds. The archetypes for this in robotics are the rotation part of the root body of a robot, and ball joints, whose variables live in $SO(3)$. Some typical tasks are also naturally formulated on different manifolds. For example, for making contact with any object that can be mapped on a sphere, the contact point position for this object can be parametrized in $S2$ [EBK16]. The human shoulder can be elegantly parametrized on $S2 \times \mathbb{R}$, as proposed in [BB01]. A non-Euclidean manifold can be thought of as a space that is locally Euclidean, but not globally. Like a sphere, if one looks in a small enough neighborhood, it looks Euclidean, just like the surface of a giant sphere like the earth looks flat for a human standing on it.

Formulating the problem over $\mathbb{R}^n$ leads either to singularities that can prevent the convergence of the optimization solver, or cumbersome writing to specify that the variable is actually living on a manifold (see [BK12b]). In addition, since constraints can be violated during the optimization process, the iterates may not be elements of the search manifold.

For example, let us consider an optimization problem over the $SO(3)$ manifold:

$$\begin{aligned} &\min_{x \in SO(3)} \quad f(x) \\ &\text{subject to } \; l \leq c(x) \leq u \end{aligned} \tag{4.1}$$

$SO(3)$ is a 3-dimensional manifold. As such, it can be parametrized *locally* by 3 variables, for example, a choice of Euler angles. But any such parametrization necessarily exhibits singularities when taken as a global map (e.g. gimbal lock for Euler angles), which can be detrimental to our optimization process.

For this reason, when addressing $SO(3)$ with classical optimization algorithms, it is often preferred to use one of the two following parametrizations:

- unit quaternion, *i.e.* an element q of $\mathbb{R}^4$ with the additional constraint $\|q\| = 1$,
- rotation matrix, *i.e.* an element R of $\mathbb{R}^{3\times3}$ (or equivalently $\mathbb{R}^9$) with the additional constraints $R^T R = I$ and $\det R \geq 0$.

Then, if we use the unit quaternion parametrization, the problem (4.1) becomes:

$$\begin{aligned} &\min_{q\in\mathbb{R}^4} \quad f(q) \\ &\text{subject to} \begin{cases} l \leq c(q) \leq u \\ \|q\|^2 - 1 = 0 \end{cases} \end{aligned} \tag{4.2}$$

The problem to solve has 4 dimensions (to represent a 3-dimensional manifold), and has an additional constraint that is entirely due to our formulation choice. With this formulation, it is guaranteed that the solution q^* is a unit quaternion, but not that all the iterates q_k along the optimization process have a unit norm. During the optimization process, at each iteration, an increment $\mathbf{p}$ is computed by solving a quadratic problem that approximates (4.2) locally around q_k. In particular, this quadratic problem approximates the constraints linearly, thus, for any step $\mathbf{p}$ not null, even if iterate q_k if of unit norm, the next iterate $q_{k+1} = q_k + \mathbf{p}$ does not respect the unit-norm constraint (it respects the *linearization* of the unit-norm constraint). So the quaternion q_{k+1} does not represent a rotation, while the objective and constraints functions might expect it to do so. In general, with non-manifold formulations, at any given iteration, the parametrization-related constraints can be violated, thus, the iterate might not lie in the manifold. It is then needed to project them on it. Denoting $\pi_{\mathcal{M}}$ the projection (for example $\pi_{\mathcal{M}} = \frac{q}{\|q\|}$ in the unit quaternion formulation), to evaluate a function f on a manifold, we need to compute $f \circ \pi_{\mathcal{M}}$. If further the gradient is needed, that projection must also be accounted for (authors in [BK12b] explain that issue in great details for robotics problems with free-floating basis).

Similar issues can be found with the $\mathbb{R}^{3\times3}$ matrix representation.

The alternative is to use optimization software working natively with manifolds like the ones presented in [BED+15] or [AMS08] and solve the optimization problem as it is written in (4.1). It has an immediate advantage: we can write directly the problem without the need

to add any parametrization-related constraints. Working directly with manifolds also has the advantage that at each iteration, the variables of the problem represent an element of the manifold. Having intermediate values naturally staying on the manifold can be useful to evaluate additional functions that pre-suppose it (additional constraints, external monitoring ...). It can also be leveraged for real-time applications where only a short time is allocated repeatedly to the optimization, so that when the optimization process is stopped after a few iterations, the output is still meaningful in the sense that it is always a point of the manifold.

In this chapter, we present a new nonlinear optimization solver able to work on generic smooth manifolds. We take inspiration from the approach used for unconstrained optimization on manifolds [AMS08] and adapt it to constrained optimization. To the best of our knowledge, constrained optimization on manifolds has drawn few research for now. This is likely due to the fact that in most problems the only constraint is to be on the manifold.We are only aware of the work of Schulman *et al.* [SDH+14], where the authors explain the adaptation of their solver to work on $SE(3)$. This adaptation is however not valid for general manifolds without more care about hessian computation.

A background motivation for this work is to have our own optimization solver, instead of a black box. We will now be able to specialize the solver for robotic problems, by leveraging modeling properties and approximations, for a gain in time and robustness.

4.2 Optimization on Manifolds

In this section, we describe a Sequential Quadratic Programming (SQP) approach [NW06] to solve the following nonlinear constrained optimization program

$$\begin{aligned} &\min_{x \in \mathcal{M}} \quad f(x) \\ &\text{subject to } \; l \leq c(x) \leq u \end{aligned} \tag{4.3}$$

where $\mathcal{M}$ is a n-dimensional smooth manifold and c is a m-dimensional real-valued function. The inequality constraints can be replaced with an equality one when the upper and lower bounds are equal.

4.2.1 Representation problem

When $\mathcal{M} = \mathbb{R}^n$, the problem (4.3) is solved iteratively, starting from an initial guess x_0 and performing successive steps $x_{i+1} = x_i + \mathbf{p_i}$ where $\mathbf{p_i}$ is the increment found at the i-th iteration, until convergence is achieved. The strategy to compute $\mathbf{p_i}$ depends on the solver.

This classical scheme cannot be readily applied to optimization over non-Euclidean manifolds. First of all, only (a subset of) the real numbers can be stored in computers. To manipulate elements of $\mathcal{M}$ we need to choose a way to represent them in memory. This boils down to choosing a representation space $\mathbb{E} = \mathbb{R}^r$ (with $r \geq n$) and a function

$$\psi : \begin{array}{ccc} x & \mapsto & \mathbf{x} \\ \mathcal{M} & \to & \psi(\mathcal{M}) \subseteq \mathbb{E} \end{array}$$

$\psi(\mathcal{M})$ is the subset of $\mathbb{E}$ containing the representation of all the elements of $\mathcal{M}$. The projection operator $\pi_{\mathcal{M}}$ that we mentioned in Section 4.1 is actually a projection from $\mathbb{E}$ to $\psi(\mathcal{M})$.

With this representation, it is tempting to simply transform problem (4.3) as an optimization over $\mathbb{R}^r$ with objective $f \circ \psi^{-1}$ and constraint $c \circ \psi^{-1}$, and solve it with a classical solver. But, as we stated in Section 4.1, if $r = n$ we will get singularities, and if $r > n$ we will add parametrization-related variables and constraints to our problem and the iterates may not lie on the manifold and need to be projected on it. So another approach needs to be taken.

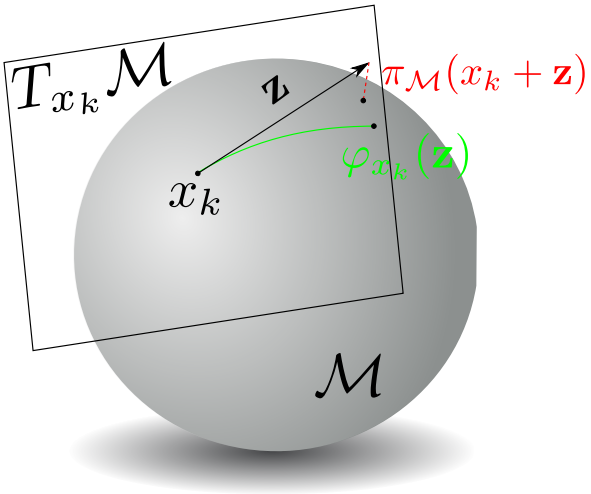


Figure 4.1 StepOnSphere

4.2.2 Local parametrization

By definition, there is always, at a point x of a smooth n-dimensional manifold $\mathcal{M}$, a smooth function φ_x between an open set of $T_x\mathcal{M}$, the tangent space to $\mathcal{M}$ at x(which is isomorphic to $\mathbb{R}^n$), and a neighborhood of x in $\mathcal{M}$, with $\varphi_x(0) = x$.

$$\varphi_x : \begin{array}{ccc} \mathbf{z} & \mapsto & \varphi_x(\mathbf{z}) \\ T_x\mathcal{M} & \to & \mathcal{M} \end{array}$$

φ_x gives us a local parametrization for $\mathcal{M}$. Figure 4.1 illustrates the difference between a step through φ_x in optimization on manifolds and a step followed by a projection $\pi_{\mathcal{M}}$ as it can be done in classical optimization.

$T_x\mathcal{M}$ can be identified with $\mathbb{R}^n$, but in some cases, it needs to be considered as a hyperplane of a higher dimensionality space. For example, in figures 4.1 and 4.2, $T_x\mathcal{M}$ is a 2-dimensional hyperplane embedded in $\mathbb{R}^3$. We denote $T_x\mathbb{E}$ the representation space of $T_x\mathcal{M}$. The driving idea of the optimization on manifolds is to change the parametrization of the problem by using a local function φ_{x_i} at the current iterate x_i at each iteration. Applying this idea, we can reformulate Problem (4.3) around x_i as:

$$\begin{aligned} \min_{\mathbf{z}\in T_{x_i}\mathcal{M}} \quad & f \circ \varphi_{x_i}(\mathbf{z}) \\ \text{subject to } \; l \leq \quad & c \circ \varphi_{x_i}(\mathbf{z}) \quad \leq b \end{aligned} \tag{4.4}$$

This is an optimization problem on $\mathbb{R}^n$. If we perform one iteration of a classical solver starting from x_i, we compute an increment $\mathbf{z_i}$, which leads to the next iterate $x_{i+1} = \varphi_{x_i}(\mathbf{z_i})$. We can then reformulate Problem (4.3) around x_{i+1}, perform a new iteration and repeat the process until convergence.

However, convergence cannot be achieved without care on the choice of φ_{x_i}. Once the optimization algorithm chooses an increment $\mathbf{z}_i$, we want to move from x_i in the direction of $\mathbf{z}_i$ on the manifold. In Euclidean spaces($\mathbb{R}^n$), moving in the direction of a vector is straightforward. On a manifold, the notion of moving in the direction of a tangent vector, while staying on the manifold, is generalized by the notion of retractation function.

A retractation R at x, denoted R_x is a function from $T_x\mathcal{M}$ to $\mathcal{M}$ with a local rigidity condition that preserves gradients at x. Absil [AMS08] defines a retractation as follows:

Definition A retractation on a manifold $\mathcal{M}$ is a smooth function R from the tangent bundle $T\mathcal{M}$ onto $\mathcal{M}$ with the following properties. Let R_x denote the restriction of R to $T_x\mathcal{M}$.

1. $R_x(0_x) = x$, where 0_x denotes the zero element of $T_x\mathcal{M}$.

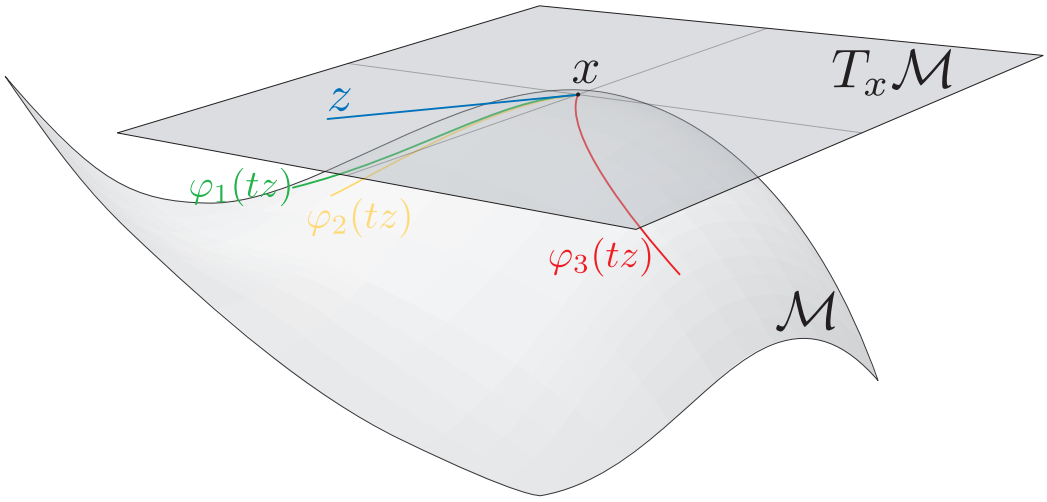


Figure 4.2 There are many possible choices for φ_x but not all yield a curve $\varphi_x(t\mathbf{z})$ which is going in the same direction as $\mathbf{z}$: φ_1 and φ_2 are correct choices, φ_3 is not.

2. With the canonical identification $T_{0_x}T_x\mathcal{M} \approx T_x\mathcal{M}$, R_x satisfies

$$DR_x(0_x) = \mathbb{I}_{T_x\mathcal{M}} \tag{4.5}$$

   Where $DR_x(0_x)$ denotes the gradient of R_x at 0_x and $\mathbb{I}_{T_x\mathcal{M}}$ denotes the identity function on $T_x\mathcal{M}$.

This signifies that for any $\mathbf{z}$, the curve $t \mapsto \varphi_{x_i}(t\mathbf{z})$ is tangent to $\mathbf{z}$, see Fig. 4.2, so that the update $x_{i+1} = \varphi_{x_i}(\mathbf{z_i})$ is made in the direction given by $\mathbf{z_i}$.

The exponential map is a good theoretical candidate, but it is often impractical or expensive to compute. Depending on the manifold, cheaper functions can be chosen.

With the iterative formulation approach described above, we do not have any parametrization issue, do not need additional constraints, and have the minimum number of optimization parameters. We can use the function $\psi : \mathcal{M} \to \psi(\mathcal{M})$, which is surjective, to represent the x_i and keep track of them in a global way. Also, the programmer can write the function $f' = f \circ \psi^{-1}$ as if it was a function from $\mathbb{E}$ to $\mathbb{R}$ without the need to project on $\psi(\mathcal{M})$ first (same goes for $c' = c \circ \psi^{-1}$). For example, if $\mathcal{M} = SO(3)$ and $\mathbb{E} = \mathbb{R}^{3\times3}$, $\mathbf{x_i} = \psi(x_i)$ is always a rotation matrix and can be used directly as such when writing the function.

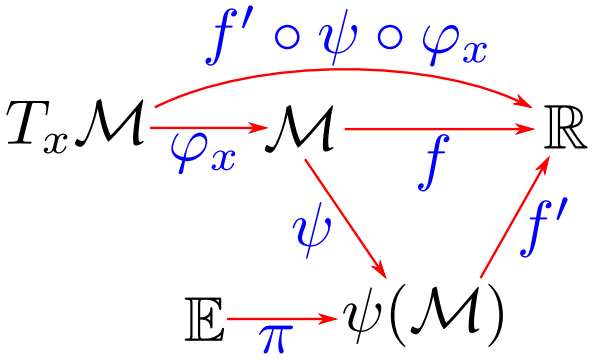


Figure 4.3 Diagram summarizing the different functions(in blue) and spaces(in black) used to represent a function on a manifold

4.2.3 Local SQP on manifolds

We choose to adopt an SQP approach to solve our problem. We first define the Lagrangian function

$$\mathcal{L}_x(\mathbf{z},\lambda) = f \circ \varphi_x(\mathbf{z}) - {\lambda_u}^T (c \circ \varphi_x(\mathbf{z}) - u) - {\lambda_l}^T (c \circ \varphi_x(\mathbf{z}) - l) \tag{4.6}$$

with $\lambda_l \in \mathbb{R}^m$ and $\lambda_u \in \mathbb{R}^m$ the vector of Lagrange multipliers respectively associated with the lower and upper bound constraints. The values of a Lagrange multiplier translate the activity status of the constraint it is associated with(see 4.3.5).

We denote H_k the Hessian matrix $\nabla^2_{zz}\mathcal{L}_{x_k}$. Taking $\mathbf{z_0} = 0$, the k-th SQP step for Problem (4.4) is computed by solving the following quadratic program:

$$\begin{aligned} \min_{\mathbf{z}\in\mathbb{R}^\mathbf{n}} \quad & \frac{\partial f \circ \varphi_{x_k}}{\partial \mathbf{z}}(0)^T \mathbf{z} + \frac{1}{2}\mathbf{z}^T H_k \mathbf{z} \\ \text{subject to } & 1 \leq c \circ \varphi_{x_k}(0) + \frac{\partial c \circ \varphi_{x_k}}{\partial \mathbf{z}}(0)\mathbf{z} \leq \mathrm{u} \end{aligned} \tag{4.7}$$

The basic SQP approach adapted to manifolds can be summarized as follows

1. set $k = 0$ and x_k to the initial value
2. compute $\mathbf{z}$ from Problem (4.7) for current x_k
3. set $x_{k+1} = \varphi_{x_k}(\mathbf{z})$ and $k = k+1$
4. if convergence is not yet achieved go to step 2

Computations of function values and derivatives are based on the fact that $f \circ \varphi = f' \circ \psi \circ \varphi$ (and same for c), and

$$\begin{aligned} f' &: \mathbb{E} \to \mathbb{R} \\ \psi \circ \varphi &: T_x\mathcal{M} \to \mathbb{E} \end{aligned}$$

are representable functions (whereas f, ψ and φ_x are not, due to the fact that they feature $\mathcal{M}$ as input or output). The gradient of $f \circ \varphi_x$ is

$$\frac{\partial f \circ \varphi_x}{\partial \mathbf{z}} = \frac{\partial f'}{\partial y}(\psi \circ \varphi_x) \times \frac{\partial (\psi \circ \varphi_x)}{\partial \mathbf{z}} \tag{4.8}$$

$\frac{\partial f'}{\partial y}$ denotes the gradient of f' with respect to an element of $\mathbb{E}$, which is the derivative that is usually calculated for use in classical optimization schemes.

In figure 4.3, we present a summary of the different functions used in our approach to represent functions on manifolds.

4.2.4 Vector transport

Nonlinear optimization algorithms such as the SQP rely on the second order information on the problem that is contained in the Hessian. The exact value the Hessian is not always available, or might be too expensive to compute. In those cases, we approximate the second order derivative by comparing first order information (tangent vectors) taken on distinct points of the manifold. Much like in the case of the retractation operation, comparing tangent vectors is straightforward on a Euclidean space is straightforward, but not on a manifold.

Given $x_1 \in \mathcal{M}$ and $\mathbf{z} \in T_{x_1}\mathcal{M}$, we denote $x_2 = \varphi_{x_1}(\mathbf{z})$. We consider two vectors $\mathbf{v}_1 \in T_{x_1}\mathcal{M}$ and $\mathbf{v}_2 \in T_{x_2}\mathcal{M}$ that we want to compare, it is necessary to transport $\mathbf{v}_1$ into $T_{x_2}\mathcal{M}$.

Absil [AMS08] gives a formal definition of the vector transport in chapter 8.

One can describe the vector transport function from a point $x \in \mathcal{M}$ along an increment $\mathbf{z}$ as:

$$\mathcal{T}_{x,z} : \begin{array}{ccc} \mathbf{v} & \mapsto & \mathcal{T}_{x,\mathbf{z}}(\mathbf{v}) \\ T_x\mathcal{M} & \to & T_{\varphi_x(\mathbf{z})}\mathcal{M} \end{array}$$

Figure 4.4 illustrates that transport of a vector $\mathbf{v}_1$ from $T_{x_1}\mathcal{M}$ to $T_{x_2}\mathcal{M}$.

4.2.5 Description of non-Euclidean manifolds

In order to develop some optimization algorithm on manifolds, we need to define a set of elements and operations for each elementary manifold $\mathcal{M}$ that will enable us to handle its

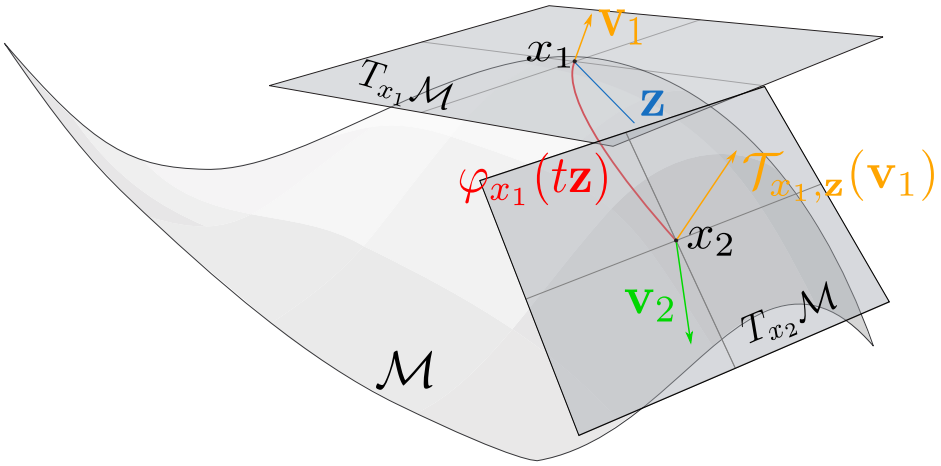


Figure 4.4 Vector transport on a non-Euclidean manifold

elements with ease. Those need only to be implemented once (it is then trivial to get those functions for Cartesian products of manifolds). The composition with f' and c' is done automatically. The expression of those functions is adapted from [BMAS14].

Retractation: We need a retractation operation $\phi_x = \psi \circ \varphi_x$ and its derivative. During the optimization process, we need the expression of the derivatives of ϕ_x in order to compute the gradients of the cost function and constraints at the beginning of each iteration. Because we change the parametrization of our problem to be centered on x_k at each iteration, we only ever need to evaluate the gradient of ϕ_x for $\mathbf{z} = 0$, $\frac{\partial \phi_x}{\partial \mathbf{z}}(0)$. In some cases, this quantity is invariant w.r.t x and can be computed once and for all.

Pseudo-logarithm and distances: It is interesting to compute distances on manifolds. For that, we define the pseudo-logarithm (denoted pseudolog), which is the inverse of the retractation operator.

$$\zeta_x : \mathcal{M} \to T_x\mathcal{M}$$

$$\forall (x,y) \in \mathcal{M} \times \mathcal{M},\ \mathbf{z} = \zeta_x(y) \text{ is such that } \phi_x(\mathbf{z}) = y$$

The pseudolog operator gives the vector of $T_x\mathcal{M}$ to go from x to y. It is used to compute the (pseudo-) distance between two points of $\mathcal{M}$

$$\text{dist}(x,y) = \|\zeta_x(y)\|$$

Vector transport: To compare two vectors living in the tangent spaces of different points of $\mathcal{M}$, it is necessary to use the vector transport operation to transport one of them in the

space of the other one before comparing them. This operation will come in handy for the computation of Hessian approximations explained later in this chapter.

Projections: It is useful to define a projection operator on $\mathcal{M}$, as well as one on $T_x\mathcal{M}$, especially to help eliminate some numerical errors when necessary. The projection operator on $\mathcal{M}$ projects an element of $\mathbb{E}$ onto $\psi(M) \subseteq \mathbb{E}$ while the one on T_xM projects an element of $\mathbb{E}$ onto $T_x\mathcal{M}$.

Limits of validity of the retractation: The retractation is only valid locally, and we need to give a (conservative) approximation of its validity region.

To summarize, for each elementary manifold $\mathcal{M}$, we need to implement the following elements:

- Tangent space at point x, $T_x\mathcal{M}$
- Embedding spaces $\mathbb{E}$ and $T_x\mathcal{M}$
- Retractation operator $\phi : (x, \mathbf{z}) \to \phi_x(\mathbf{z})$
- Gradient of the retractation operator at zero $\partial\phi(x) :\to \frac{\partial \phi_x}{\partial \mathbf{z}}(0)$
- Pseudo-logarithm operator $\zeta : (x, y) \to \zeta_x(y)$
- Gradient of pseudo-logarithm operator at the iterate $\frac{\partial \zeta_x}{\partial y}(x)$
- Transport operator $\mathcal{T} : (x, \mathbf{z}, \mathbf{v}) \to \mathcal{T}_{x,\mathbf{z}}(v)$
- Projection from $\mathbb{E}$ on $\mathcal{M}$, $\pi_{\mathcal{M}}$
- Projection from $T_x\mathbb{E}$ on $T_x\mathcal{M}$, $\pi_{T_x\mathcal{M}}$
- Limits of validity of the retractation on $T_x\mathcal{M}$, lim

We provide the detailed formulas for those operations in Appendix B for different elementary manifolds:

- The Real space of dimension n in B.1
- The 3D rotations manifold SO(3) with matrix formulation in B.2
- The 3D rotations manifold SO(3) with quaternion formulation in B.3
- The unit sphere manifold S2 in B.4

Cartesian Product of Manifolds

Given two manifolds $\mathcal{M}_1$ and $\mathcal{M}_2$, we denote $\mathcal{M} = \mathcal{M}_1 \times \mathcal{M}_2$ their cartesian product. Any operation on an element of $\mathcal{M}$ can simply be computed term by term for each manifold composing $\mathcal{M}$. For example, the retractation is computed as follows, and that scheme can be reproduced for all other operations:

$$x_1 \in \mathcal{M}_1,\ \mathbf{z_1} \in T_{x_1}\mathcal{M}_1,\ x_2 \in \mathcal{M}_2,\ \mathbf{z_2} \in T_{x_2}\mathcal{M}_2 \tag{4.9}$$

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \mathcal{M},\ \mathbf{z} = \begin{bmatrix} \mathbf{z_1} \\ \mathbf{z_2} \end{bmatrix} \in T_x\mathcal{M} \tag{4.10}$$

$$\phi_x(z) = \begin{bmatrix} \phi_{x_1}(\mathbf{z_1}) \\ \phi_{x_2}(\mathbf{z_2}) \end{bmatrix} \tag{4.11}$$

4.2.6 Implementation of Manifolds

In order to use the manifold formulation described above in other software, and particularly in a numerical solver, we wrote an independent C++ project. This implementation is open-source and available at https://github.com/stanislas-brossette/manifolds. The implementation consists of 3 types of classes: the Manifold class, the elementary manifold classes, and the Point class. The Manifold class describes the abstract mathematical structure of a non-Euclidean manifold and defines a common interface for all elementary manifolds to implement (retractation, pseudoLog,...). Elementary Manifold classes ($\mathbb{R}^n$, $SO(3)$, $S2$, and the Cartesian Product) are the concrete manifolds. They inherit from the Manifold class and implement all their mathematical operations. The Cartesian Product class is used to build compound manifolds by being 'multiplied' with other elementary manifolds. The Point class represents a point on a manifold, it contains the data that represents its numerical value. It can only be constructed by a manifold, and provides some proxy to its manifolds operations, in particular, it is equipped with an increment method, that applied a retractation on it. Figure 4.5 presents a simplified class diagram of this project, omitting all the settor, gettor, bookkeeping mechanics and accessory functions.

4.3 Practical implementation

The SQP algorithm presented in Section 4.2.3 works locally, *i.e.* it is guaranteed to converge when starting close enough to the solution. In practice, various refinements are made to

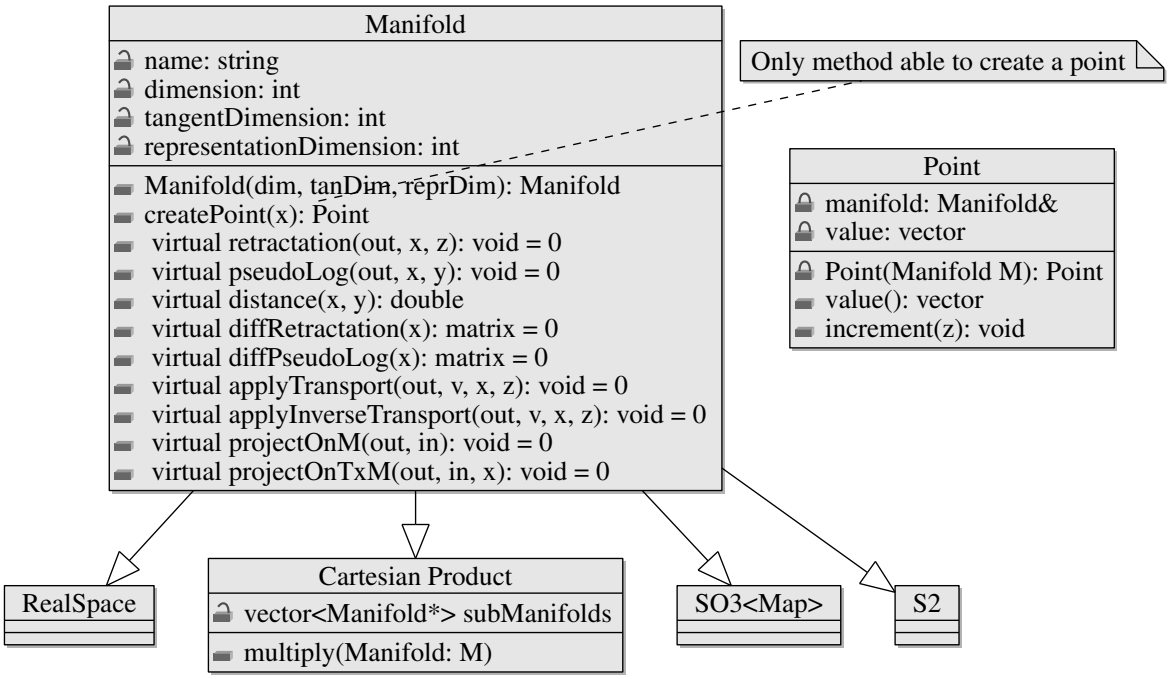


Figure 4.5 Simplified class diagram of the Manifold project

ensure convergence from any starting point, it is called the globalization. We detail hereafter the choices we made for the implementation of our solver called `PGSolver`.

We summarize the entire SQP algorithm in Diagrams 4.6, 4.7 and 4.8, which represent respectively the main SQP loop, the restoration phase, and the second order correction algorithm.

4.3.1 Linear and quadratic problems resolution

The central idea of an SQP algorithm is that it solves a series of QP iteratively until a solution is reached. There are many off-the-shelf QP solvers available and the state of the art is mature in that field, thus we decided to use the `LSSOL` solver [GHM+86]. `LSSOL` provides resolution methods for several types of problems and we are interested in using the FP (Feasibility Problem) and QP ones. The `LSSOL` framework formulates problems as follows (Note that there are no equality constraints in this formulation):

$$\min_{x \in \mathbb{R}^n} F(x) \tag{4.12}$$

$$\text{subject to } l \leq \begin{Bmatrix} x \\ Cx \end{Bmatrix} \leq u \tag{4.13}$$

With the F function taking different forms depending on the problem to solve:

FP:	None	(find a feasible point for the constraints)
QP2:	$F(x) = c^T x + \frac{1}{2} x^T A x$	A symmetric and positive semi-definite
QP4:	$F(x) = c^T x + \frac{1}{2} x^T B^T B x$	B $m \times n$ upper-trapezoidal

The QP4 type of problem is used when a decomposition of the matrix A of QP2, $A = B^T B$ is already available.

4.3.2 Problem Definition

As stated in Eq (4.1) we want to solve a nonlinear constrained optimization problem on manifold that takes the following form:

$$\begin{aligned} &\min_{x \in \mathcal{M}} \quad f(x) \\ &\text{subject to } \; l \leq c(x) \leq u \end{aligned} \tag{4.14}$$

The list of constraints can be separated into 3 categories: Bounds, Linear, and Nonlinear. For convenience, we formulate our optimization problem in a way that is compatible with `LSSOL` by changing all equality constraints into inequality constraints:

$$c(x) = a \Leftrightarrow a \leq c(x) \leq a \tag{4.15}$$

Our problem can be written as:

$$\begin{aligned} &\min_{x \in \mathcal{M}} \quad f(x) \\ &\text{subject to} \begin{cases} L_B \leq x \leq U_B \\ L_L \leq Ax \leq U_L \\ L_N \leq c(x) \leq U_N \end{cases} \end{aligned} \tag{4.16}$$

With L_B and U_B being respectively the lower and upper bounds for Bounds constraints, L_L and U_L the lower and upper bounds for Linear constraints, and L_N and U_N the lower and upper bounds for Nonlinear constraints. That formulation is conveniently used to define our problem. Since we are solving this problem on a non-Euclidean manifold, we need to re-formulate it around the current iterate x_i at each iteration. This is done automatically behind the scene and the user only has to provide the information to build problem (4.16).

At iteration i, the problem becomes (for clarity, we drop the subscript i that goes with each appearance of x):

$$\min_{\mathbf{z}\in T_x\mathcal{M}} \quad f(\phi_x(\mathbf{z})) \tag{4.17}$$
$$\text{subject to} \begin{cases} \mathbf{z}^-_{\text{map}} \leq \mathbf{z} \leq \mathbf{z}^+_{\text{map}} \\ L_B \leq \phi_x(\mathbf{z}) \leq U_B \\ L_L \leq A\phi_x(\mathbf{z}) \leq U_L \\ L_N \leq c(\phi_x(\mathbf{z})) \leq U_N \end{cases}$$

With $\mathbf{z}^-_{\text{map}}$ and $\mathbf{z}^+_{\text{map}}$ being respectively the lower and upper bounds of validity of the tangent map of $\mathcal{M}$ around x. After this reformulation, we note that if ϕ_x is nonlinear, then the bounds and linear constraints of problem (4.16) become nonlinear in problem (4.17). It is then necessary to treat the problem and evaluate which constraint is linear and which is not, in order to go back to a formulation of the same type as in problem (4.16). Basically, for each constraint c_j in problem (4.17), if the submanifold of $\mathcal{M}$ on which that constraint is applied is a real space $\mathbb{R}^m$ then the constraint maintains its bound, linearity or nonlinearity status, otherwise, it becomes a nonlinear constraint[1]. It is, however, unusual to write linear constraints on non-Euclidean manifolds.

In the particular case of linear constraints (and by extension bound constraints) on a submanifold that is a real space, the constraints on x are transformed into constraints on z as follows:

$$L_L \leq Ax \leq U_L \;\Rightarrow\; L_L - Ax \leq A\mathbf{z} \leq U_L - Ax \tag{4.18}$$

For bounds constraints, the same goes with A being the identity matrix on the submanifold.

Once those substitutions are done, we obtain a new problem of the following form (where L'_B, L'_L, L'_N, U'_B, U'_L, U'_N are the lower and upper bounds for the bounds, linear and nonlinear constraint, A' is the matrix of linear constraints and c' is the list of nonlinear functions):

$$\min_{\mathbf{z}\in T_x\mathcal{M}} \quad f \circ \phi_x(\mathbf{z}) \tag{4.19}$$
$$\text{subject to} \begin{cases} L'_B \leq \mathbf{z} \leq U'_B \\ L'_L \leq A'\mathbf{z} \leq U'_L \\ L'_N \leq c' \circ \phi_x(\mathbf{z}) \leq U'_N \end{cases}$$

[1] At the time of writing those lines. This treatment of the constraints is a planned work, it has not been implemented yet.

This problem is an exact reformulation of problem (4.16) on $T_x\mathcal{M}$. At each step of the optimization process, we solve the QP approximating this problem (4.19) around $\mathbf{z} = 0$. The Taylor development of $c \circ \phi_x(z)$ to the first order around $\mathbf{z} = 0$ gives:

$$c' \circ \phi_x(\mathbf{z}) \approx c' \circ \phi_{x_k}(0) + \frac{\partial c' \circ \phi_{x_k}}{\partial \mathbf{z}}(0)\mathbf{z} = c'(x_k) + \left(\nabla \phi_{x_k}(0) \nabla c'(x_k)\right)^T \mathbf{z} \tag{4.20}$$

The trust-region constraint (see Section 4.3.3) is added to the problem to limit the length of a step, ρ denotes the size of the trust-region. H_k is an estimation of the Hessian computed as shown in Section 4.3.8. The QP to be solved at each iteration is the following:

$$\begin{aligned} &\min_{\mathbf{z} \in T_x\mathcal{M} = \mathbb{R}^n} \quad (\nabla \phi_{x_k}(0) \nabla f(x_k))^T \mathbf{z} + \frac{1}{2} \mathbf{z}^T H_k \mathbf{z} \\ &\text{subject to} \begin{cases} L'_B \leq \mathbf{z} \leq U'_B \\ L'_L \leq A'\mathbf{z} \leq U'_L \\ L'_N \leq c'(x_k) + (\nabla \phi_{x_k}(0) \nabla c'(x_k))^T \mathbf{z} \leq U'_N \\ -\rho \leq \mathbf{z} \leq \rho \quad \text{(trust-region constraint)} \end{cases} \end{aligned} \tag{4.21}$$

The resolution of this QP (4.21) (with `LSSOL`) gives the optimal solution $\mathbf{z}^*$ and Lagrange multipliers λ^*. The new iterate that will be considered is:

$$x_{k+1} \leftarrow \phi_{x_k}(\mathbf{z}^*) \tag{4.22}$$

$$\lambda_{k+1} \leftarrow \lambda^* \tag{4.23}$$

Note that the only additional computation due to the formulation on manifolds is the multiplication of the gradients of constraints and cost by the map's gradient at 0. The computation time related to those products can be reduced by taking advantage of the fact that $\nabla \phi_x(0)$ is block-diagonal.

4.3.3 Trust-region and limit map

Maps φ_x are only valid locally, and we need to account for this: a step $\mathbf{z}$ found by solving Problem (4.21) should not be outside the validity region of the map. We can enforce this by adding the limits of the manifold map to the problem as bound constraints. In fact, this comes down to intersecting the original problem's bounds with the limits of the map and imposing the resulting bounds as constraints. This leads naturally to trust region methods that we, therefore, favor over line-search approaches. We present the classical trust-region strategy in Section A.5.2.

In the case of a robotics problem, different variables often have different orders of magnitude, for example, contact forces can be of the order of $100N$ while joint angles are of the order of $1rad$. Let x be the variable vector of such problem with $x = [\theta_0, \theta_1, f_0, f_1, f_2]^T$ where θ_i represent angular variables and f_i forces. The shape of the trust-region should reflect those differences, but we want to keep the simplicity of the classical trust-region strategy. For that, we propose to separate the trust-region ρ in two parts: its scale ρ_{scale} which is a scalar, and its shape ρ_{shape} which in a vector of dimension n (that value is actually stored in the instance of manifold). ρ_{shape} is constant and set at the beginning of the optimization, the i-th element of ρ_{shape} is the order of magnitude of the i-th dimension of the optimization variable. In the previous example, we'd get $\rho_{\text{shape}} = [1, 1, 100, 100, 100]^T$ And ρ_{scale} is the scalar that gets updated in the trust-region strategy.

Finally, the constraints related to the trust-region added to the problem come down to:

$$-\rho = -\rho_{\text{scale}}\rho_{\text{shape}} \leq \mathbf{z} \leq \rho_{\text{scale}}\rho_{\text{shape}} = \rho \tag{4.24}$$

4.3.4 Filter method

To know if a step $\mathbf{z}$ is acceptable or not, one usually uses a penalty-based merit function as we present in Section A.4. In our early tests, the update of the penalty parameters proved to be difficult with our types of problems. We now use a filter approach instead, as presented in Section A.5.2.

Our algorithm is an adaptation of Fletcher's filter SQP [FL00] to the case of manifolds: we use an adaptive trust-region that is intersected with the validity region of ϕ_{x_i}, and a new iterate $x_{i+1} = \phi_{x_i}(\mathbf{z})$ is accepted if either the cost function or the sum of constraint violations is made better than for any previous iterates.

4.3.5 Convergence criterion

An iterate $\{x, \lambda\}$ is considered to be a solution of the optimization problem if it satisfies the first-order optimality conditions (presented in Section A.3.1) to within certain tolerances. We use the same criterion as the one used in `SNOPT` and presented in [GMS02]. It has the advantage of using two different tolerance constants: τ_P monitors the primal optimality conditions, which means the satisfaction of the constraints, and τ_D monitors the dual conditions, which means the optimality of the cost function and of the Lagrange multipliers. That gives the user more control over the quality of the solution.

Dropping the distinction between Linear and NonLinear Constraints, the problem can be rewritten as:

$$\begin{aligned} &\min_{\mathbf{x}\in\mathcal{M}} \quad f(\mathbf{x}) \\ &\text{subject to } l \leq c(\mathbf{x}) \leq u \end{aligned} \tag{4.25}$$

Each constraint is considered as a double inequality, thus, is associated with two Lagrange multipliers: λ_l for the lower bound and λ_u for the upper bound. The basic KKT condition for that problem writes as:

$$\begin{cases} \nabla\mathcal{L} = \nabla f(\mathbf{x}) + \lambda_l.\nabla c(\mathbf{x}) + \lambda_u.\nabla c(\mathbf{x}) = 0 \\ \quad \forall i \begin{cases} c_i(\mathbf{x}) - l_i \geq 0 \\ c_i(\mathbf{x}) - u_i \leq 0 \\ \lambda_{li} \leq 0 \\ \lambda_{ui} \geq 0 \\ \lambda_{li}.(c_i(\mathbf{x}) - l_i) = 0 \\ \lambda_{ui}.(c_i(\mathbf{x}) - u_i) = 0 \end{cases} \end{cases} \tag{4.26}$$

For each constraint, there are 3 possible situations:

Lower bound violated or active	$c_i(x) - l_i \leq 0$	$\lambda_{li} \leq 0$	$\lambda_{ui} = 0$
Constraint satisfied	$l_i \leq c_i(x) \leq u_i$	$\lambda_{li} = 0$	$\lambda_{ui} = 0$
Upper bound violated or active	$c_i(x) - u_i \geq 0$	$\lambda_{li} = 0$	$\lambda_{ui} \geq 0$

Both λ_l and λ_u cannot be nonzero at the same time. So for each constraint, we can use a single Lagrange multiplier that is negative when the lower bound is violated or active, null when the constraint is satisfied, and positive when the upper bound is active or violated. This allows to reduce the KKT system to the following:

$$\begin{cases} \nabla\mathcal{L} = \nabla f(\mathbf{x}) + \lambda.\nabla c(\mathbf{x}) = 0 \\ \quad \forall i \begin{cases} c_i(\mathbf{x}) = l_i & \text{and } \lambda_i \leq 0 \\ \text{OR} \\ l_i \leq c_i(\mathbf{x}) \leq u_i & \text{and } \lambda_i = 0 \\ \text{OR} \\ c_i(\mathbf{x}) = u_i & \text{and } \lambda_i \geq 0 \end{cases} \end{cases} \tag{4.27}$$

To evaluate the satisfaction of this system, we approximate it with the tolerance constants. First, we scale the tolerance constants with respect to the values of the iterate x and λ:

$$\begin{aligned} \tau_x &= \tau_P(1+\|x\|_\infty) \\ \tau_\lambda &= \tau_D(1+\|\lambda\|_\infty) \end{aligned} \tag{4.28}$$

And we get the following convergence criterion:

$$\begin{cases} \|\nabla\mathcal{L}\|_\infty \leq \tau_\lambda \\ \quad \forall i \begin{cases} |c_i(\mathbf{x}) - l_i| \leq \tau_x & \text{and } \lambda_i \leq -\tau_\lambda \\ \text{OR} \\ -(c_i(\mathbf{x}) - l_i) \leq -\tau_x & \text{and } -(c_i(\mathbf{x}) - u_i) \geq -\tau_x \quad \text{and } |\lambda| \leq \tau_\lambda \\ \text{OR} \\ |c_i(\mathbf{x}) - u_i| \leq \tau_x & \text{and } \lambda_i \geq \tau_\lambda \end{cases} \end{cases} \tag{4.29}$$

4.3.6 Feasibility restoration

Along the optimization process, the set of linearized constraints in the QP Problem (4.21) can become unfeasible, for example after a reduction of the trust-region, see Section A.5.2. In that case the QP (4.21) cannot be solved and the resolution as presented above cannot proceed. To cope with this issue, the algorithm enters a phase called of restoration, that aims at finding a feasible point without regards for the value of the cost function. The basic idea is the following: at the beginning of an iteration, the list of unfeasible constraints is computed and stored in $\mathcal{U}_l$ if the lower bound in unfeasible and in $\mathcal{U}_u$ if the upper bound is unfeasible and the list of feasible constraints is stored in $\mathcal{F}$. The unfeasible constraints are removed from the restoration problems' constraints list and their violation is added to its cost function (that becomes the sum of all constraints violation). We get the following restoration cost function:

$$f^{\text{rest}} = \sum_{i\in\mathcal{U}_l} (l_i - c_i(x)) + \sum_{i\in\mathcal{U}_u} (c_i(x) - u_i) \tag{4.30}$$

Then the restoration problem only contains feasible constraints and has to minimize the sum of constraint violation. The problem to solve becomes:

$$\min_{x\in\mathcal{M}} \quad \sum_{i\in\mathcal{U}_l} (l_i - c_i(x)) + \sum_{i\in\mathcal{U}_u} (c_i(x) - u_i) \tag{4.31}$$

$$\text{s.t.} \begin{cases} \forall i \in \mathcal{F} & l_i \leq c_i(x) \leq u_i \\ \forall i \in \mathcal{U}_l & -\infty \leq c_i(x) \leq u_i \\ \forall i \in \mathcal{U}_u & l_i \leq c_i(x) \leq +\infty \end{cases}$$

To simplify the writing of that QP, we denote l^{rest} and u^{rest} two vectors that represent respectively the lower and upper bounds of the constraints of the restoration problem:

$$\forall i \in \mathcal{F},\, l_i^{\text{rest}} = l_i,\; u_i^{\text{rest}} = u_i \tag{4.32}$$

$$\forall i \in \mathcal{U}_l,\, l_i^{\text{rest}} = -\infty,\; u_i^{\text{rest}} = u_i \tag{4.33}$$

$$\forall i \in \mathcal{U}_u,\, l_i^{\text{rest}} = l_i,\; u_i^{\text{rest}} = +\infty \tag{4.34}$$

$$\tag{4.35}$$

The problem becomes:

$$\begin{aligned} \min_{x \in \mathcal{M}} \quad & \sum_{i \in \mathcal{U}_l} (l_i - c_i(x)) + \sum_{i \in \mathcal{U}_u} (c_i(x) - u_i) \\ \text{s.t.} \begin{cases} \forall i \quad l_i^{\text{rest}} \leq c_i(x) \leq u_i^{\text{rest}} \end{cases} & \end{aligned} \tag{4.36}$$

That is solved by iterating just like in the SQPs main loop with the difference that at each iteration, we update the problem based on new lists of unfeasible constraints. An approximation H_k^{rest} of the Hessian is computed especially for the restoration phase. The restoration phase has and updates its own trust-region ρ_{rest}. Dropping the differences between bounds, linear and nonlinear constraints, the QP to solve at each iteration of the restoration phase is the following:

$$\begin{aligned} \min_{\mathbf{z} \in T_x \mathcal{M}} \quad & \left(\nabla \phi_{x_k}(0) \nabla f^{\text{rest}}(x_k)\right)^T \mathbf{z} + \frac{1}{2} \mathbf{z}^T H_k^{\text{rest}} \mathbf{z} \\ \text{s.t.} \begin{cases} \forall i \quad l_i^{\text{rest}} \leq c_i(x_k) + (\nabla \phi_{x_k}(0) \nabla c_i(x_k))^T \mathbf{z} \leq u_i^{\text{rest}} \\ -\rho^{\text{rest}} \leq \mathbf{z} \leq \rho^{\text{rest}} \end{cases} & \end{aligned} \tag{4.37}$$

The restoration phase has its own filter called the restoration filter. For more details on the restoration phase, see Section A.5.3.

Note that each iteration of the main SQP algorithm starts with the resolution of an FP (Feasibility Problem), that consists of the same linearized constraints as the QP problem (4.21) without cost function.

$$\text{find } \mathbf{z} \text{ such that: } \begin{cases} L_B \leq \mathbf{z} \leq U_B \\ L_L \leq A\mathbf{z} \leq U_L \\ L_N \leq c(x_k) + (\nabla \phi_{x_k}(0) \nabla c_i(x_k))^T \mathbf{z} \leq U_N \end{cases} \tag{4.38}$$

Its role is to determine whether the set of linearized constraints is feasible. If it is, the main SQP continues, otherwise, the restoration phase is entered. This feasibility problem is also solved at the beginning of each restoration iteration to determine whether or not to exit the restoration phase. As soon as a feasible point is found, the restoration is exited. Once a feasible point x_F is found by the restoration phase, it is used as the new iterate in the main SQP phase. Since during the restoration no care is taken about the value of the cost function, it is possible that x_F is refused by the main filter, which is not an acceptable behavior. So x_F is forced in the filter, and any pair dominating it is removed. Then the main phase of the optimization can continue.

4.3.7 Second Order Correction

In the event where a step proposed in the restoration process by the resolution of the restoration QP is refused by the restoration filter, instead of immediately reducing the size of the trust region, we can perform a Second Order Correction Step.

The idea is to re-solve a QP after its solution $\mathbf{z}_k$ has been refused by the restoration filter, but with a better approximation (second order) of the constraints:

$$c_i(x_k+\mathbf{z}) = c_i(x_k) + \nabla c_i(x_k)^T\mathbf{z} + \frac{1}{2}\mathbf{z}^T\nabla^2 c_i(x_k)\mathbf{z} \tag{4.39}$$

The restoration QP problem becomes:

$$\begin{aligned} \min_{\mathbf{z}\in T_x\mathcal{M}} \quad & \left(\nabla\phi_{x_k}(0)\nabla f^{\text{rest}}(x_k)\right)^T\mathbf{z} + \frac{1}{2}\mathbf{z}^T H_k^{\text{rest}}\mathbf{z} \\ \text{s.t.} \quad & \begin{cases} \forall i \quad l_i^{\text{rest}} \le c_i(x_k) + \left(\nabla\phi_{x_k}(0)\nabla c_i(x_k)\right)^T\mathbf{z} + \frac{1}{2}\mathbf{z}^T\nabla^2 c_i(x_k)\mathbf{z} \le u_i^{\text{rest}} \\ -\rho^{\text{rest}} \le \mathbf{z} \le \rho^{\text{rest}} \end{cases} \end{aligned} \tag{4.40}$$

Using $\frac{1}{2}\mathbf{z}^T\nabla^2 c_i(x_k)\mathbf{z} \approx c_i(x_k+\mathbf{z}_k) - c_i(x_k) - \nabla c_i(x_k)^T\mathbf{z}_k$ we get:

$$\begin{aligned} \min_{\mathbf{z}\in T_x\mathcal{M}} \quad & \left(\nabla\phi_{x_k}(0)\nabla f^{\text{rest}}(x_k)\right)^T\mathbf{z} + \frac{1}{2}\mathbf{z}^T H_k^{\text{rest}}\mathbf{z} \\ \text{s.t.} \quad & \begin{cases} \forall i \quad l_i^{\text{rest}} \le \left(\nabla\phi_{x_k}(0)\nabla c_i(x_k)\right)^T(\mathbf{z}-\mathbf{z}_k) + c_i(\phi_{x_k}(\mathbf{z}_k)) \le u_i^{\text{rest}} \\ -\rho^{\text{rest}} \le \mathbf{z} \le \rho^{\text{rest}} \end{cases} \end{aligned} \tag{4.41}$$

Denoting $g_k = (\nabla\phi_{x_k}(0)\nabla f(x_k))$ and $A_k = (\nabla\phi_{x_k}(0)\nabla c_i(x_k))$, we can rewrite 4.41 as follows:

$$\begin{aligned} &\min_{\mathbf{z}\in T_x\mathcal{M}} \quad g_k^T\mathbf{z}+\frac{1}{2}\mathbf{z}^T H_k^{\text{rest}}\mathbf{z} \\ &\text{s.t.} \begin{cases} \forall i \quad l_i^{\text{rest}}+A_k\mathbf{z}_k-c_i(\phi_{x_k}(\mathbf{z}_k)) \le A_k^T\mathbf{z} \le u_i^{\text{rest}}+A_k\mathbf{z}_k-c_i(\phi_{x_k}(\mathbf{z}_k)) \\ -\rho^{\text{rest}} \le \mathbf{z} \le \rho^{\text{rest}} \end{cases} \end{aligned} \tag{4.42}$$

This system is solved iteratively by updating $\mathbf{z}_k$ (but never changing x) with the solution of the previous system and updating the values of $c_i(\phi_{x_k}(\mathbf{z}_k))$ until a satisfactory $\mathbf{z}$ is found.

4.3.8 Hessian update on manifolds

Aside from the manifold adaptation, our main departure from Fletcher is in the Hessian computation where we used an approximation because the exact one is too expensive to compute in our problems. After testing several possibilities, we settled for a self-scaling damped BFGS update [NY93, NW06], adapted to the manifold framework. More precisely, given the Hessian approximation H_k at iteration k, we compute the approximation H_{k+1} as follows: Note that as explained in Section 4.2.4, the gradients are transformed through vector transport before being compared.

$$\begin{aligned} &s_k = \mathcal{T}_{\mathbf{z}}(z), \quad y_k = \nabla_z\mathcal{L}_{x_{k+1}}(0,\lambda_{k+1}) - \mathcal{T}_{\mathbf{z}}(\mathcal{L}_{x_k}(0,\lambda_k)) \\ &\theta_k = \begin{cases} 1 & \text{if } s_k^T y_k \ge 0.2 s_k^T\tilde{H}_k s_k \\ \frac{0.8 s_k^T\tilde{H}_k s_k}{s_k^T\tilde{H}_k s_k - s_k^T y_k} & \text{otherwise} \end{cases} \quad \text{(Powell update)} \\ &r_k = \theta_k y_k + (1-\theta_k)\tilde{H}_k s_k \quad \text{(damped update)} \\ &\tau_k = \min\left(1, \frac{s_k^T r_k}{s_k^T\tilde{H}_k s_k}\right) \quad \text{(self-scaling)} \\ &H_{k+1} = \tau_k\left(\tilde{H}_k - \frac{\tilde{H}_k s_k s_k^T\tilde{H}_k}{s_k^T\tilde{H}_k s_k}\right) + \frac{r_k r_k^T}{s_k^T r_k} \end{aligned}$$

where $\mathcal{T}_{\mathbf{z}}$ is a vector transport along $\mathbf{z}$ (see [AMS08]) and $\tilde{H}_k$ is such that for $\mathbf{u}\in T_{x_{k+1}}\mathcal{M}$, $\tilde{H}_k\mathbf{u} = \mathcal{T}_{\mathbf{z}}\left(H_k\mathcal{T}_{\mathbf{z}}^{-1}(\mathbf{u})\right)$.

4.3.9 Hessian Regularization

Despite Powell's update, H_k might not be positive definite (but still symmetric). We regularize it as follows: we first perform a Bunch-Kaufman factorization $P_k H_k P_k^T = L_k B_k L_k^T$ where P_k is a permutation matrix, L_k is unit lower triangular and B_k is block diagonal with blocks of

size 1×1 or 2×2 (obtaining B_k as a diagonal matrix is not numerically stable for Cholesky-like decomposition of indefinite matrices), see [GVL96]. The eigenvalue decomposition $B_k = Q_k D_k Q_k^T$ is immediate and cheap to compute. From the diagonal matrix D_k, we form D'_k such that $d'_{ii} = \max(d_{ii}, \mu_{\min})$ where $\mu_{\min} > 0$ is user-defined (we typically set it to 0.1). Defining $L'_k = L_k Q_k (D'_k)^{1/2}$, we get a regularized matrix $H'_k = P_k^T L_k L_k^T P_k = (L_k^T P_k)^T L_k^T P_k$. We take advantage of `LSSOL`'s capability to receive H in the factorized form $H = A^T A$ with $A = L_k^T P_k$ upper-trapezoidal as an input. This avoids an internal Cholesky factorization so that our regularization does not add too much time to the overall process of building and solving the QP.

4.3.10 An alternative Hessian Approximation Update

In [Fle06], Fletcher presents a new Hessian update method that maintains and update the Hessian H in the form $H = UU^T$, which is obviously always symmetric and it ensures that it is always positive semi-definite. Since a decomposition of H is readily available with this method, we can get rid of the regularization of the Hessian in our algorithm. The matrix U of size (n,m) is smaller than H and contains only information from the last m iteration, this means that this method has a built-in limited memory capability. In our robotics problems, using limited memory Hessian updates is useful because along the iteration process, the variables may change a lot, and the Hessian value near the solution may be very different from the one approximated around the first iterations. Thus, it can be helpful to 'forget' about the old components of the approximation to leave room for the latest ones. At each iteration, the matrix U is updated with either BFGS or SR1, or a hybrid method, based on some tests on the value of the latest step. When possible, the SR1 update is preferred, because some results suggest that it allows faster convergence when the SR1 denominator is positive, otherwise, a BFGS or the hybrid method are used.

Although this method provides a decomposition of H as UU^T, U is not trapezoidal, so we need to apply a QR on U^T it so that we get $U^T = QR$, then $H = R^T Q^T QR = Q^T Q$ with Q trapezoidal and R an orthogonal matrix. Then Q can be fed to `LSSOL` directly.

For all those reasons, this update method is very attractive. We implemented it in our solver, but so far the results have not been very conclusive and some more work will be dedicated to that issue in the future. As of now, the best results were observed when using the self-scaling BFGS update method.

4.3.11 Hessian Update in Restoration phase

The hessian H computed through whichever method presented above is meant to approximate the value of $\nabla^2\mathcal{L}_{xx}(x,\lambda) = \nabla^2 f(x) + \lambda^T \nabla^2 c(x)$, with f the cost function of the current problem to solve, and c its constraints. But during the restoration phase, the definitions of f and c change at each iteration, depending on the sets of unfeasible and feasible constraints ($\mathcal{I}$ and $\mathcal{F}$). In order to account for that change, we propose to compute an approximation of each constraints second derivative separately, denoting H_i the hessian of c_i, and correctly combining the Hessians based on the current set of feasible constraints.

$$\begin{aligned} H_{c_i} &\approx \nabla^2 c_i \\ H &= \sum_{i\in\mathcal{I}} H_{c_i} + \sum_{i\in\mathcal{F}} \lambda_i H_{c_i} \end{aligned} \tag{4.43}$$

With this definition of H, we can ensure the symmetry of H, but nothing guarantees its positive definiteness. It is then necessary to regularize H after combining the H_{c_i}. This also allows to have an approximation of the Hessian when exiting the restoration phase, by computing $H = H_f + \sum \lambda_i H_{c_i}$ with $H_f \approx \nabla^2 f$ and regularizing it.

4.3.12 Solver Evaluation

We integrated our solver with the Roboptim framework (http://roboptim.net/) to have a common interface with other solvers that are already interfaced with roboptim and used in our team, like `IPOPT`, `CFSQP`, and `NAG`. The roboptim framework provides a list of 72 canonical optimization problems coming from the Hock-Schittkowski-Collection [HS80] that allows to evaluate the rate of success and speed of each solvers. Although those problems are all on $\mathbb{R}^n$, thus, do not try the non-Euclidean capabilities of our solver, it is a good way compare its performances with those of other solvers. As of now, `PGSolver` finds a solution for 51 tests out of 72, using its default parameters. In the same conditions, `IPOPT` solves 65 problems, `CFSQP` 46 and `NAG` 58. This result serves only as an indication of the correct behavior of a solver because default parameters are usually not good and a solver reaches its full potential once its parameters have been tuned for a type of problem. Those tests can also be used in a non-regression criterion for our software.

4.4 Diagrams of the algorithms

In Figures 4.6, 4.7, and 4.8, we summarize the functioning of the algorithms that are respectively the main loop of the SQP, the restoration phase, and the second order correction phase.

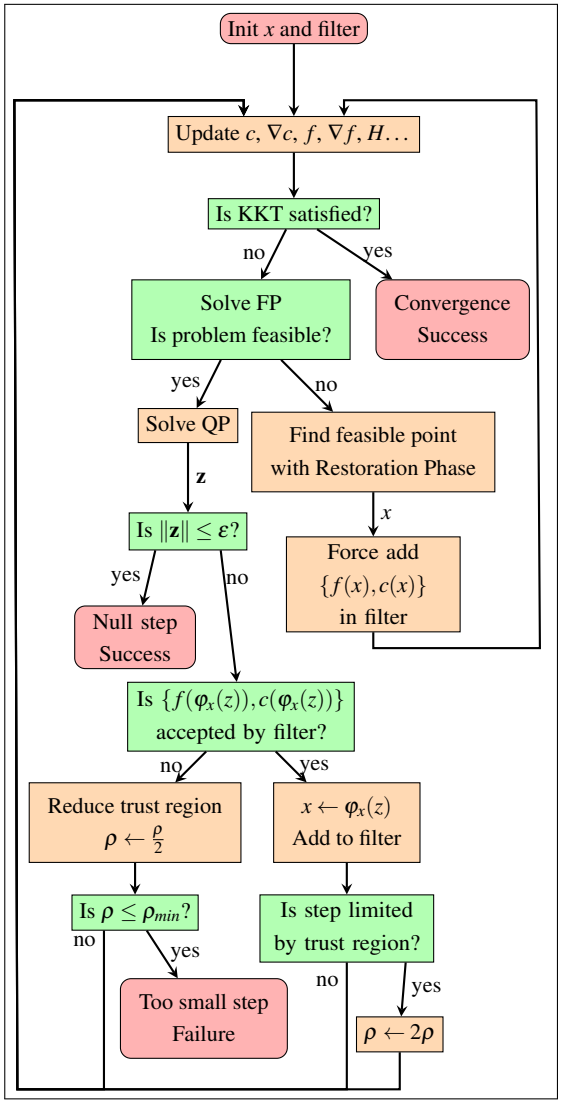


Figure 4.6 Main SQP Loop

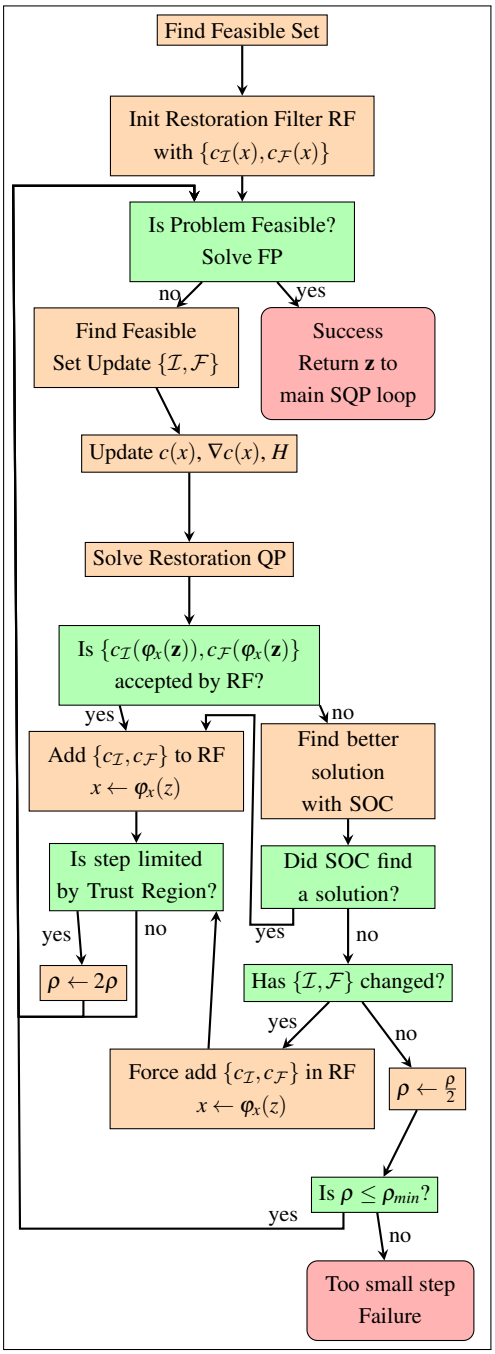


Figure 4.7 Restoration Loop

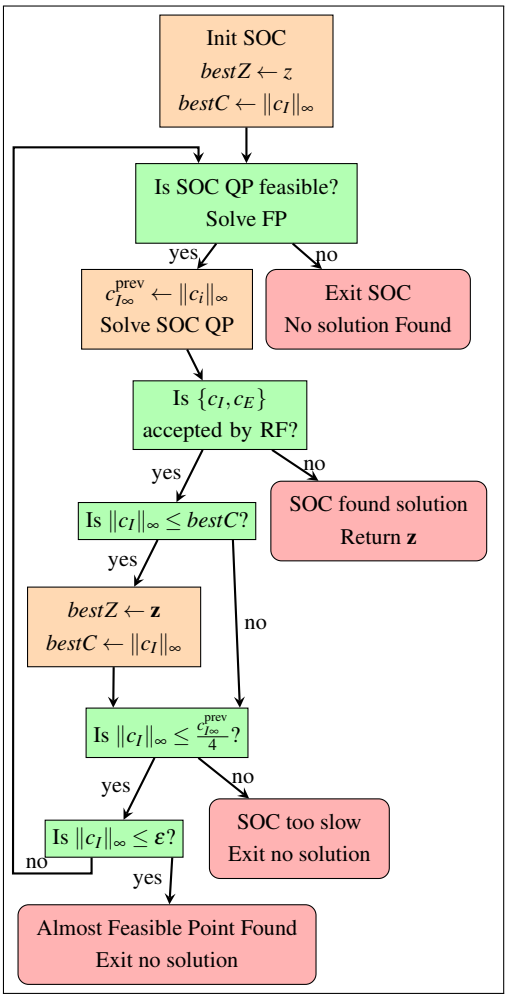


Figure 4.8 Second Order Correction

4.5 Conclusion

In this chapter, we presented the principle of nonlinear optimization on non-Euclidean manifolds, and detailed the choices we made to implement an SQP algorithm that solves those

problems efficiently. In addition to being used in our posture generation problems, `PGSolver` has been used successfully in inertia identification problems as presented in [TBEN16]. In the next section, we will focus on the formulation of optimization problems for posture generation.

Chapter 5

Posture Generator

Writing a posture generation problem can easily become cumbersome without the appropriate tools. Common pitfalls are for example writing the derivative of a function, managing how the Jacobian matrices of the already implemented functions are modified when a variable is added to the problem, adding a new type of constraint, or correctly writing a function on a sub-manifold of the problem manifold. A fair amount of bookkeeping is always necessary, which should not be the charge of the user writing the constraints. In our PG, we propose an architecture automating most of the problematic tasks so that the user can focus on the mathematical formulation of the problem:

- A system of geometrical and mathematical expression trees that automatically computes the mathematical expressions behind geometric relations
- An automatic mapping between the submanifold of each function and the global manifold of the problem
- A problem generator that aggregates all the information from the above-mentioned items to generate an optimization problem that can be passed to the solver

We then propose an extension to classic posture generation by developing a constraint of contact with parametrized surfaces which takes advantage of our framework.

5.1 Geometric expressions

Most constraints are geometric. In order to simplify the writing of functions, we use a dedicated system of expression graph encapsulated in a set of geometric objects. The main idea is to separate the purely mathematical logic from the geometric one. As an example if P_r and $\overrightarrow{V_r}$ are a point and a vector attached to the camera of the robot, and P_e is a fixed point

in the environment, the constraint $(P_e - P_r) \cdot \overrightarrow{V_r} = 0$ can be used to have the robot look at P_e. With our system, the user creates only those objects and write the code `(Pe-Pr).dot(Vr)` to create the needed function. The geometric layer takes care that all the quantities are expressed in the correct frame, the mathematical layer performs the corresponding operations. If q is a variable object, `(Pe-Pr).dot(Vr).diff(q)` returns automatically the differential of the expression w.r.t. q. This makes the writing of the constraints very easy.

At the mathematical level, we consider 5 types of expressions which can be either variables or constants:

- Scalar, a 1-dimensional element of $\mathbb{R}$
- Coordinates, a 3-dimensional element of $\mathbb{R}^3$
- Rotation, a 3×3 matrix representing a 3D rotation
- Transformation, a 4×4 matrix representation of a 3D isometry
- Array, a dynamic size array

The meaningful unary (inverse, opposite, norm...) and binary (multiplication, addition, subtraction, dot product...) operations (with their derivatives by chain rule) are implemented. We also have a Function class for more complicated expressions, for example expressing $q \mapsto T_i(q)$ where T_i is the transformation between the reference frame of the robot and the frame of its i-th body [1]. The combinations of those elementary operations define a computation graph, just like in many symbolic calculation frameworks.

Each expression is able to compute its own value and the value of its derivative with respect to another expression. For example, let A and B be two unrelated expressions. We denote $\dim(x)$ the dimension of expression x. The value of expressions A or B are obtained by respectively calling the methods `A.value()` and `B.value()`. The derivative of A with respect to itself is obtained by calling the method `A.diff(A)` and returns an Identity matrix of size $\dim(A)$. Whereas the derivative of A w.r.t B returns a zero matrix with $\dim(A)$ rows and $\dim(B)$ columns.

$$\frac{\partial A}{\partial A} = \mathbf{1}_{\dim(A)}, \quad \frac{\partial A}{\partial B} = \mathbf{0}_{\dim(A),\dim(B)}$$

Each operator on expressions defines its resulting expression, thus, it defines a `value` and a `diff` methods. For example, let us consider that A and B are 3D coordinate expressions and a new expression $C = A \cdot B$ that is defined as the dot product of A and B, in code, one can write C's definition as simply:

[1] The kinematics of rigid body systems is handled by the RBDyn library (git@github.com:jrl-umi3218/RBDyn.git)

```
Scalar C = dot(A,B).
```

The value and the derivative of C w.r.t any x expression are then automatically computed as follows:

$$C = A \cdot B, \quad \frac{\partial C}{\partial x} = A^T \frac{\partial B}{\partial x} + B^T \frac{\partial A}{\partial x} \tag{5.1}$$

The code of the dot function that translates eq: (5.1) is pretty straightforward:

```
C.value() = A.value().dot(B.value())
C.diff(x) = A.value().transpose()*B.diff(x)
  + B.value().transpose()*A.diff(x)
```

It is easy to extend the capabilities of this framework by implementing additional operations. By following this approach, we can easily compute the values and derivatives of expressions that are the outcome of complex arithmetic trees. Currently, the following binary operators are available: cross product, dot product, classic product, addition, subtraction, transformation. And the following unary operators are available: component extraction, inverse, norm, rotation, square root, transformation, and vectorization.

The geometric layer consists of physical or geometric objects, called features, which exist independently of their mathematical expression in a given reference frame. We have so far four objects:

1. A Frame, defined by a Transformation expression and a reference frame.
2. A Point, defined by a Coordinates expression and a reference frame.
3. A Vector, defined by a Coordinates expression and a reference frame.
4. A Wrench, defined by a pair of Coordinates expressions and a reference frame.

We have a special World Frame object to serve as starting reference frame.

For each feature, one can get its expression in a given frame. Basic operations are defined between those features (when applicable). For example, the subtraction between two Points gives a Vector. The geometric logic resides in the change of frame and those operations.

Based on that expression system, the robot's geometry can simply be represented by a set of frames representing all its links, and functions that keep track and update the transformations of the frames of all its bodies, with respect to an Array expression on entry, its articular parameters. The forward kinematics algorithm 2 and the jacobian computation algorithm 3 are used to define those functions. For a given articular parameter array, the user can query the frame of any body on the robot, and by composition, any feature defined on a frame of the robot, as well as its derivatives. This tool allows for a simplified writing of

robotics constraints as a combination of operations on geometric features, without worrying about the vectors and matrices and without having to write the derivatives.

Let us consider a simple constraint and its expression in this framework: we define a point P_h on a body of a robot and a point P_e in the environment. We want to write a constraint that ensures that P_h and P_e are superimposed. Given a frame F, we can simply write that as:

```
P_h.coord(F) - P_e.coord(F)=0
```

(the "$= 0$" part of the equation is here for readability only; In the code, the value of the resulting expression is constrained by boundaries)

In the case of a scalar expression like $(P_e - P_r) \cdot \overrightarrow{V_r} = 0$, it can be written without any consideration about frames as

```
(P_h - P_e).dot(V) = 0.
```

This mathematical and geometric expression framework simplifies the task of the constraints developers. And it helps to avoid calculation mistakes because we only ever need to write the code for elementary functions and their combination is computed automatically by chain rule.

5.2 Automatic mapping

The manifold $\mathcal{M}$, on which the optimization takes place, is a Cartesian product of several sub-manifolds. Same goes for their representation spaces:

$$\begin{aligned} \mathcal{M} &= \mathcal{M}_1 \times \mathcal{M}_2 \times \mathcal{M}_3 \times \dots \\ \mathbb{E} &= \mathbb{E}_1 \times \mathbb{E}_2 \times \mathbb{E}_3 \times \dots \end{aligned} \tag{5.2}$$

From the solver's viewpoint, the entry space of each function is the complete manifold. But for the developer, writing a function on the complete $\mathbb{E}$ is cumbersome because (i) of the need to manage indexes, and (ii) when the function is implemented, the complete $\mathbb{E}$ may not be known. A user-written function f is usually defined on a subset of $\mathbb{E}$, say $\mathbb{E}_I = \mathbb{E}_i \times \mathbb{E}_j \times \mathbb{E}_k \dots$, that is minimalistic for that function, and should not account for unrelated manifolds. One does not want to think about the values of the forces when writing a geometric constraint for example.

We developed an automatic mapping tool that keeps track of all the necessary mappings for each function added and upon the instantiation of the problem, generates the correct projection functions π_I such that the developer can write a function f on $\mathbb{E}_I$ while the solver receives it as a function $f \circ \pi_I$ on $\mathbb{E}$.

This tool was built as an add-on to the RobOptim [MLBY13] framework called **roboptim-core-manifold** and its goal is to extend the notion of function to take into account its application manifold and map the correspondence between the values in the input of the problem and the input of the function. We define a class called DescriptiveWrapper that contains a RobOptim function and a description of its application manifold. Once a function is wrapped in a DescriptiveWrapper and the optimization problem is set, we need to establish the mapping between the variable of the problem, which is defined on the global manifold of the problem, and the input of the function that must match the description of its application manifold. That is the role fulfilled by the WrapperOnManifold layer, that compares the two manifolds, matches them and computes an input mapping from the problem's input vector to the function's, which we denote π_I. This mapping is computed once during the instantiation of the problem and then used at each call of the function. The use of the input mapping is illustrated by the example in Fig. 5.1

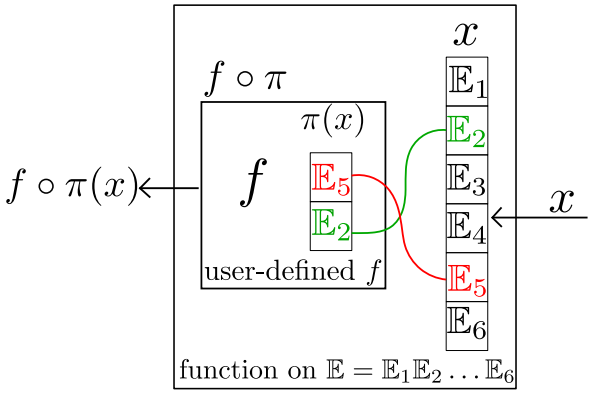


Figure 5.1 automatic variable mapping

5.3 Problem formulations

Let $q = [q_F^T; q_r^T] \in SE(3) \times \mathcal{M}_r$ be the combination of the free-flyer of the robot $q_F \in SE(3) = \mathbb{R}^3 \times SO(3)$ and the articular parameters $q_r \in \mathcal{M}_r$. Let $\mathcal{W}_i(p) = \{f_i, m_i(p)\}$ be the wrench (force+moment) applied by the environment onto the robot at contact i and expressed on point p. A frame F is composed of a reference point and an orthonormal basis of 3 vectors $F = \{O, (\vec{x}, \vec{y}, \vec{z})\}$. For frames define on a surface, with one vector of $(\vec{x}, \vec{y}, \vec{z})$ being the outbound normal to the surface, and the two others tangent, we rename $\vec{n}$ the normal one and $\vec{t}, \vec{b}$(tangent and bitangent) the two other ones, ensuring that $(\vec{n}, \vec{t}, \vec{b})$ is an orthonormal direct basis. When a frame is denoted with the subscript i, F_i, all its components are added the same subscript.

Here is a list of constraints that we consider in our problems:

- Joint limits $q^- \leq q_r \leq q^+$:
  These cannot be directly translated on manifolds other than $\mathbb{R}^n$. For example, spherical joints can be parametrized on $S2 \times \mathbb{R}$, then the $S2$ part can be limited by a cone, and the $\mathbb{R}$ part can have real bounds(in that case the limit constraint is not a simple bound constraint, but writes as $f(q_r) \leq 0$).

- The Frame constraint is a generic type of constraint that consists in blocking a predefined set of degrees of freedom of one frame F_2 w.r.t. another frame F_1. By default, the frame F_2 has 6 degrees of freedom w.r.t. F_1, three translational degrees of freedom TX, TY, and TZ, respectively along $\vec{x_1}$, $\vec{y_1}$ and $\vec{z_1}$ and three rotational ones RX, RY, and RZ, respectively around $(O_1, \vec{x_1})$, $(O_1, \vec{y_1})$ and $(O_1, \vec{z_1})$. We can write constraints such that any combination of the translational degrees of freedom is blocked:

$$\text{If TX is blocked: } \overrightarrow{O_1O_2} \cdot \vec{x_1} = 0 \tag{5.3}$$

$$\text{If TY is blocked: } \overrightarrow{O_1O_2} \cdot \vec{y_1} = 0 \tag{5.4}$$

$$\text{If TZ is blocked: } \overrightarrow{O_1O_2} \cdot \vec{z_1} = 0 \tag{5.5}$$

Similarly, any pair of rotational degrees or the three of them can be blocked (blocking only one rotation is not possible with that formalism). If the three rotations RX, RY, RZ are blocked, we get the set of constraints 5.6:

$$\begin{cases} \vec{z1} \cdot \vec{y2} = 0 \\ \vec{z1} \cdot \vec{x2} = 0 \\ \vec{x1} \cdot \vec{y2} = 0 \\ \vec{z1} \cdot \vec{z2} \geq 0 \\ \vec{y1} \cdot \vec{y2} \geq 0 \end{cases} \tag{5.6}$$

When two rotational degrees of freedom are blocked, we get the set of constraints 5.7 To generalize the following formulas, we rename the axis X, Y, and Z with respect to their respective roles in this constraint. The one rotational degree that remains free is denoted N with vector $\vec{n}$, and the two others are denoted T and B, with vectors $\vec{t}$, and $\vec{b}$. Those 3 vectors are ordered such that $(\vec{n}, \vec{t}, \vec{b})$ is an orthonormal direct basis.

$$\begin{cases} \vec{n_2} \cdot \vec{n_1} \geq 0 \\ \vec{n_2} \cdot \vec{t_1} = 0 \\ \vec{n_2} \cdot \vec{b_1} = 0 \end{cases} \tag{5.7}$$

In theory, one could simply write that constraint as $\vec{n_1} = \vec{n_2}$, but in practice, this formulation leads to bad numerical behavior of the optimization, thus we prefer 5.7.

All those independent sets of constraints can be combined to create a wide variety of geometric constraints. For example, one can create a mobile planar contact constraint with normal $\vec{z_1}$ by blocking TZ, RX and RY. A punctual contact constraint can be devised by blocking TX, TY and TZ. To completely fix the position of F_2 w.r.t. F_1, one can block TX, TY, TZ, RX, RY, and RZ. And many other types of custom constraints can be devised with this formulation.

In the event that the two frames do not have a satisfying configuration to write the desired constraint with this formalism (for example, if one wants to write a planar contact between two surfaces S_1 and S_2, where $\vec{z_1}$ and $\vec{z_2}$ are outbound normal vectors, then equation 5.7 is not satisfactory because we want to oppose $\vec{z_1}$ and $\vec{z_2}$, not identify them.), then user is required to define a new frame F_2' that is a constant 3D transformation away from F_2 (in that case a rotation of π around $(O_1, \vec{x_1})$ would suffice).

- The stability constraint ensures that the Euler-Newton equation (5.8) is balanced for the set of external wrenches applied to the robot (gravity $\mathcal{W}_G$ and contact forces $\mathcal{W}_i$).

$$\sum_i \mathcal{W}_i(p) + \mathcal{W}_G(p) = 0 \tag{5.8}$$

For each contact that bears forces ('stability' contact), a wrench applied on the robot at the contact point is added to the problem. If the contact is added through a frame constraint between F_1 and F_2, the wrench can be formed by a combination of elementary components that are:

  - FX: force along $\vec{x_1}$, value f_x
  - FY: force along $\vec{y_1}$, value f_y
  - FZ: force along $\vec{z_1}$, value f_z
  - MX: moment around $(O_1, \vec{x_1})$, value m_x
  - MY: moment around $(O_1, \vec{y_1})$, value m_y
  - MZ: moment around $(O_1, \vec{z_1})$, value m_z

That wrench is parametrized on a $\mathbb{R}^m$ with m the number of components used. We model the resultant wrench of planar contacts as a combination of punctual forces applied at each vertex of the contact polygon. In the case of interaction forces between 2 robots, only one wrench is created and it is used as is in the stability equation of one robot and its opposite is used for the stability of the second robot. That way, we are guaranteed that the forces in between the robots are balanced and we can evaluate the stability of each robot separately.

- The friction cone constraint can be used to limit the tangential part of any force to avoid slippage. We write it as (5.9) (with μ the friction coefficient)

$$\begin{aligned} \mu^2 f_z^2 - f_x^2 - f_y^2 &\geq 0 \\ f_z &\geq 0 \end{aligned} \tag{5.9}$$

- The rotational friction limit constraint can be used to limit the friction torque of any force to avoid slippage. We write it as (5.10) (with σ the rotational friction coefficient)

$$\begin{aligned} \sigma^2 f_z^2 - m_z^2 &\geq 0 \\ f_z &\geq 0 \end{aligned} \tag{5.10}$$

- The collision avoidance constraint can be defined between any two bodies. A convex mesh is attached to each body involved in the constraint. We denote them C_1 and C_2. We use the `sch-core` library described in [BEMK09] that implements an efficient GJK algorithm to compute the distance $d(C_1, C_2)$ between 2 convex shapes and its derivative. A description of strictly convex hulls that can be used for this purpose is detailed in [EMBK14]. For each collision avoidance constraint, we require that the distance between the two convex remains superior to a minimal distance d_{min}

$$d(C_1, C_2) \geq d_{\text{min}} \tag{5.11}$$

- The torque limit constraint makes use of the Inverse Statics algorithm 5 to compute the torques in all the joints of every robots and their derivatives are computed by algorithm 6. Denoting τ_i the torque in joint i resulting from the robot's configuration and the external forces applied on it, and its lower and upper limits as τ_i^- and τ_i^+. The torque limit constraint writes as follows:

$$\forall i,\ \tau_i^- \leq \tau_i \leq \tau_i^+ \tag{5.12}$$

Most often, the frame's configuration depends on a part of the optimization variables, that must be accounted for in computing the constraints' Jacobian. Our framework computes such dependencies automatically.

Another important part is the problem's cost function. Any function of the problem's variables that returns a scalar value can be used as a cost function. In our formulation, we take a sum of cost functions into account, several different functions f_i over different variables can be defined separately. And then combined in a weighted sum as the problem's cost function (with w_i the weight of function f_i):

$$cost = \sum_i w_i f_i \tag{5.13}$$

Taking advantage of our framework, the set of variables over which the cost function is defined, its value and its derivatives are then computed automatically.

A typical cost function is the distance to a reference posture q_0. On a robot that has all its articulations parametrized on $\mathbb{R}$ the distance can be expressed simply with the Euclidean norm $d = \|q_r - q_0\|^2$. Since we work on non-Euclidean manifolds, the logarithm function on the manifold must be used. It gives the distance vector between two points in the tangent space, the norm of this vector can be used as a distance. So we get $d = \left\|\log_{q_0}(q_r)\right\|^2$ (note that on $\mathbb{R}^n$, $\left\|\log_{q_0}(q_r)\right\|^2 = \|q_r - q_0\|^2$)

We can also use the cost function to set a target value v for the value of a contact force $\vec{F}$ in a given direction d. By simply implementing the following function:

$$f = \left\|\vec{F} \cdot \vec{d} - v\right\| \tag{5.14}$$

The list of constraints and cost functions that can be used in robotics posture generation problems can be extended, and this process is made easier by our framework.

5.4 Implementation

In this section, we provide an overview of some of the key elements of the framework:

Problem's configuration: The implementation of the core structure of a posture generation problem is written in `C++`. A text file in YAML format can be used to configures the solver and some custom parameters of the problem; It is parsed at runtime to avoid having to recompile the problem for every configuration change. Also, some `Python` bindings have been developed to allow writing the core structure of the problem without compilation.

The Robot class: The RBDyn library contains all the low-level tools and algorithm to compute the kinematics and statics of a robot. The robot class is built on top of RBDyn to provide high-level functionalities while adapting it to our manifold formalism (which RBDyn does not handle natively). The basic structure of the robot is extracted from its URDF description file. The main functionality of the robot class is to create a Frame attached to each of its bodies and to update their transformations and derivatives based on the robots articular variables. It also maintains the transformations of all the collision meshes and contains the joint and torque limits, as well as the manifold on which its variables live. Additionally, it contains some useful objects used to maintain and update the external forces applied on the robot, as well as the joint torques.

The Function Class: The functions that we use derives from the Roboptim's functions. As such, they are able to compute their return value and derivative for a given input. Functions can conveniently be defined through our expression framework to take advantage of their automatic derivation.

The Constraint class: A constraint object contains all the necessary elements for the description of a mathematical constraint. That is, a function h and the manifold $\mathcal{M}_c$ on which it should be applied, as well as its lower bound l, upper bound u and the scaling factor of the constraint. Once a constraint is instantiated by the problem, the input mapping π from the problem's complete manifold $\mathcal{M}$ to the function's application manifold is computed. The constraint that will then be used in the optimization for a given iterate x can be written as:

$$l \leq h \circ \pi(x) \leq u \tag{5.15}$$

Constraints can be equipped with an 'addStabilityToProblem' method that is used to specify the forces generated by the constraint to add to the stability constraint of the problem.

The Frame Constraint class: It describes a very useful type of constraint between two frames F_1 and F_2. It implements the Frame constraint that we described in Section 5.3 with some additional features. This specific constraint is composed of four distinct parts:

- The geometric constraint generates the constraints associated with the user's choice of combination of degrees of freedom to block amongst TX, TY, TZ, RX, RY, RZ.
- The static constraint part (optional) instantiates the wrench to be added to the stability constraint of the problem based on the user's choice of components FX, FY, FZ, MX, MY, MZ.

- A custom geometric additional function field (optional) allows the user to add extra geometry constraints to the problem (through a `C++11` lambda that is executed after the geometric constraints are added to the problem).
- A custom static additional function field (optional) allows the user to add extra static constraints to the problem (through a `C++11` lambda that is executed after the constraint's wrench has been created and added to the problem).

The custom geometric additional function can typically be used to in the context of a planar contact to limit the position of O_2 to the inside of a polygon defined in the plan $(O_1, \vec{x_1}, \vec{x_2})$. As for the static additional function, it can be used to add constraints on the values of the constraint's wrench, for example, enforcing the friction cone constraints.

The PostureProblem class: The PostureProblem class is in charge of building the complete optimization problem that represents our posture generation problem. It creates and owns the robots present in the problem. It registers all the variables and constraints of the problem as well as all the functions involved in the cost function. Each function is likely to bring additional variables with it. For each contact contributing to the balance, a variable representing the contact force is added to the problem. The associated wrench is added to the stability constraints. Once the registration is complete, the complete manifold of the problem is generated and all the functions on manifold used in the problem will generate their input mapping that will project the complete variable of the problem onto the submanifold that the function is interested in. Subsequently, the optimization problem can be generated and passed to the solver. The communication between the solver and the generated problem is made through the RobOptim framework[2] [MLBY13, MCY14]. After creating the complete manifold, the PostureProblem class can be used to facilitate the individual initialization of all the separate variables of the problem without the user having to worry about the indexes. Finally, it can query the resolution of the problem by the solver and return the result.

5.5 Examples of postures generation

In this section, we present some scenarios solved with our posture generator and solver.

Application of a desired force In the context of using a robot to help humans in the construction of an aircraft, we formulate a problem where the HRP-4 robot is required to apply a desired force on a given point of the airplane's structure. We denote f_g the target

[2] http://www.roboptim.net/

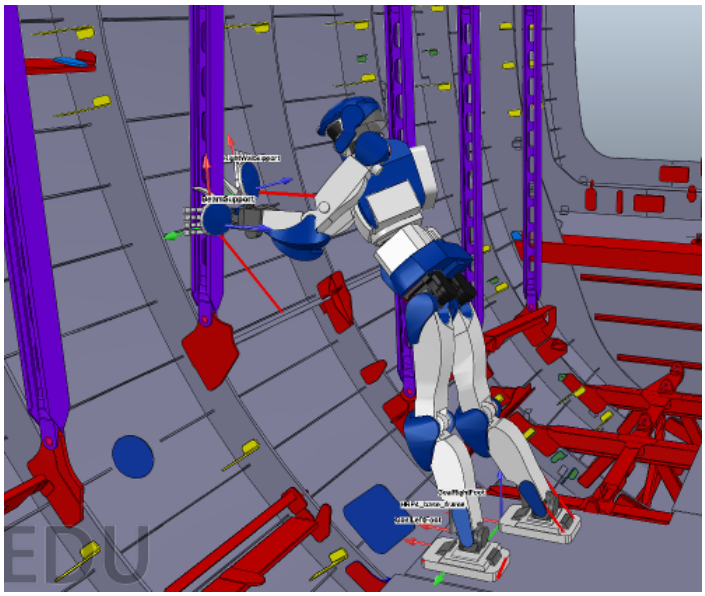


Figure 5.2 HRP-4 applying a desired force on a contact point with its right hand

value of the force in direction $\vec{d}$ and $\vec{f}_c$ the actual force at the contact point and the could simply implement the following function $f_{target}(f_c)$ to minimize to get the desired result:

$$f_{target}(f_c) = \left(\vec{f}_c \cdot \vec{d} - f_g\right)^2 \tag{5.16}$$

In addition to that cost function, the robot needs to keep its foot on the ground, its left hand is used to lean on a beam of the structure to allow a longer reach with the right hand that is in contact with the wall. Those constraints must be satisfied while respecting the joint and torque limits of the robot, maintaining balance and avoiding auto-collisions. The result is depicted in figure 5.2

Generating postures in highly constrained environment In this scenario, the HRP-4 robot is required to evolve in a highly constrained environment, in the sense that many obstacles limit its displacement, and it must avoid collision with them. The goal is to real a panel of circuit breakers (in green on figure 5.3) to activate them. We generated a sequence of steps to reach that panel and then some postures to reach different points of this panel. In figure 5.3, we show some extracts of this sequence of postures. In each of them the two foot are in planar contact with the ground, alternatively bearing forces, while always respecting the joint, torque limits, collision avoidance, and stability constraints. On the last posture, the top left corner of the panel is reached with the tip of the right hand of the robot.

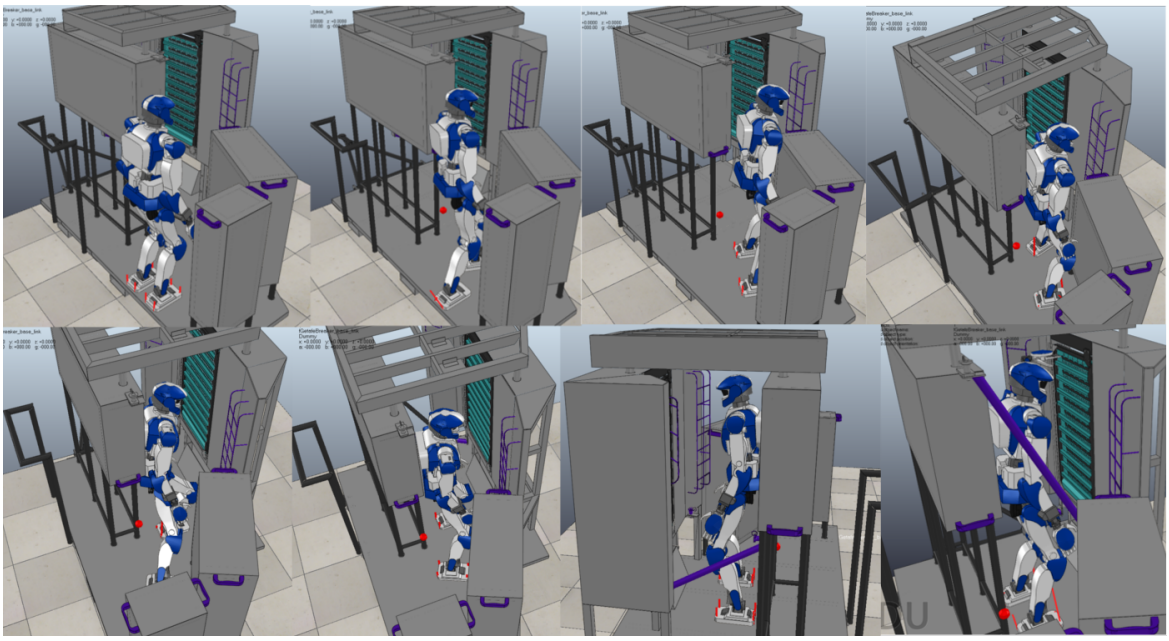


Figure 5.3 Extracts of a sequence of postures with HRP-4 entering a constrained environments to reach a point on a panel of circuit breakers (in green)

5.6 Contact on non-flat surfaces

Writing relations between geometric quantities defined in frames allows to describe a large variety of contacts. In particular, it allows to write most types on geometric constraints between canonical geometric shapes (point, line, plane). The most common type of contact constraint encountered is the contact between two surfaces S_1 and S_2. The set of equations used to define that type of constraint can be obtained through a frame constraint blocking TZ, RX and RY. Provided that the correct frames are defined, F_1 on S_1 with $\vec{z_1}$ normal outbound and F_2 on S_2 with $\vec{z_2}$ normal inbound, the constraint writes as:

$$\begin{cases} \overrightarrow{O_1O_2} \cdot \vec{z_1} = 0 \\ \vec{z_2} \cdot \vec{z_1} \geq 0 \\ \vec{z_2} \cdot \vec{x_1} = 0 \\ \vec{z_2} \cdot \vec{y_1} = 0 \end{cases} \tag{5.17}$$

When the two surfaces are flat, a contact between them can be written simply with the set of equations (5.17). In that case, the quantities involved in the constraint are invariant with the location of the contact point, thus, the equations are valid for any point of the surface (in particular, the normal vector is the same for any point on the surface). When it comes to

non-flat surfaces, the location of the frames used in equations (5.17) matters, as the quantities in the constraint equations depend on it.

We propose to add an additional variable u_S to our optimization problem to represent the location of the contact frame on a non-flat surface. This gives the ability to our solver to choose the optimal contact position on a non-flat surface. Depending on its shape, the manifold $\mathcal{M}_S$ on which the surface is parameterized can be $\mathbb{R}$, $\mathbb{R}^2$ or $S2$. Then we can write the contact constraint on that parametrized frame with ease.

$$\mathcal{F}(u_S \in \mathcal{M}_S) = \{O(u_S), (x(u_S), y(u_S), z(u_S))\} \tag{5.18}$$

In this section, we will successively present several ways to parameterize various non-flat surfaces to generate contacts with them.

5.6.1 Application to plan-sphere contact

We consider the contact between a body's flat surface S_B of normal n_B, with the surface of a sphere S_s of center c_s, radius r_s, and let p_s and n_s be a point and its normal on S_s. The most general way to express such a constraint is to ensure that p_s is on S_B and that n_s and n_B are opposite. To do that, we create a variable v_{S2} on $S2$, add it to our optimization problem and map p_S and n_S on it. In our framework, this constraint is expressed exactly as the contact between 2 planar surfaces, once the mappings of $p_s(v_{S2})$ and $n_s(v_{S2})$ are done. In a framework that does not handle manifolds, it would require to setup a specific constraint, ensuring that the distance between c_s and S_B is equal to r_S.

In Fig. 5.4 we show the results obtained by solving a problem where the HRP2-Kai robot has to keep its feet in contact with the ground at fixed positions, touch a sphere with a side of its right wrist and point as far as possible in a given direction d with its left hand, under balance constraints. The top row of Fig. 5.4 shows the results for this problem with several different d. In every situation, the projection of the CoM is outside the polygon of support, meaning that such postures would not be reached without leaning on the sphere.

5.6.2 Contact with parametrized wrist

Being able to choose the location of the contact point on the sphere is interesting, but a limitation of this formulation is that the contact point on the wrist of the robot is restricted to one single user-defined face. Instead, we describe the shape of the wrist body as a parametric function and let the contact point on the wrist as well as its counterpart on the sphere, result from the optimization process. The section of HRP2-Kai's wrist is a square with rounded

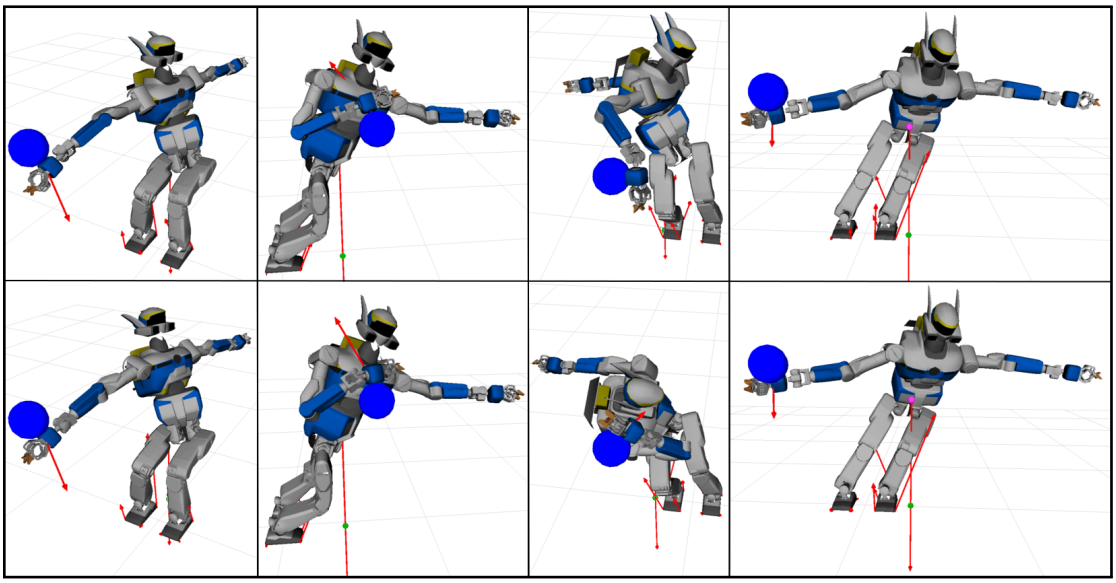


Figure 5.4 HRP2-Kai leaning on a sphere with its right wrist to point its left gripper as far as possible in 4 cardinal directions. Top row: semi-predefined contact; Bottom row: free contact with parametrized wrist. Projection of the CoM on the ground (green dots)

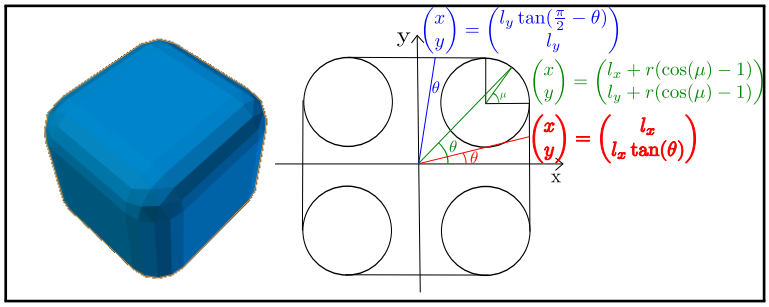


Figure 5.5 Parametrization of the wrist of HRP2-Kai

edges. We parametrize this shape as shown in Fig. 5.5: we consider the angular coordinate θ of the point on the section. It is added as a variable on $\mathbb{R}$ to the problem. The shape of the quarter of section $[0;\pi/2]$ is a succession of a vertical line, a quarter of a circle and a horizontal line. This pattern is repeated for the 3 other quarters. The equations are given in Fig. 5.5. In our framework, we define the function describing the shape of the wrist, create a frame parametrized by that function and then define the contact between that frame and the point and normal on the sphere. This formulation not only is very easy to implement but most importantly, allows for richer posture generations. The optimization algorithm chooses the contact point on the sphere as well as the contact point on the wrist, which leads to a wider accessibility range, and a better satisfaction of the cost function.

The bottom row of Fig. 5.4 displays the results of this simulation for the robot pointing in 4 directions. Notice that on the 2nd and the 4th (pointing forward and to the left) images, the results for the 2 types of models are nearly identical, whereas in the 1st and 3rd images, different faces of the wrist have been chosen (On the 1st, the wrist is rotated by 180°, and 90° on the 3rd). In these 4 cases, the contact with parametrized wrist gives a better value of the objective function (the left hand points further). This observation scales: we solved this problem for 5000 random pointing directions, and in average, the contact with parametrized wrist allows to reach 5mm further. The success rate of the solver is 98.5% in the parametrized wrist case against 99.9% when the face is fixed. The numbers of iterations are similar.

This method is certainly scalable, and can be used for any kind of humanoid robot and environment. Yet, it requires having a parametric equation of the surface.

5.6.3 Contact with an object parametrized on S2

In this case, we want the HRP-4 robot to carry a cube with its two hands. The most general way to do it is to select a face of the cube for each contact and to enforce the contact between that face and the hand's surface. We propose to approximate the cube with a superellipsoid and to parametrize the resulting shape on $S2$, thus adding a variable on $S2$ to our problem. The implicit equation of a superellipsoid is $S(x,y,z) = 0$, with

$$S(x,y,z) = \left(\left|\frac{x}{A}\right|^r + \left|\frac{y}{B}\right|^r\right)^{\frac{t}{r}} + \left|\frac{z}{C}\right|^t - 1 \tag{5.19}$$

A point in $S2$ is represented by a vector $v = (x,y,z)$ in $\mathbb{E} = \mathbb{R}^3$. To a given unit vector v we associate a point αv on the surface of the superellipsoid by solving $S(\alpha v) = 0$ for α. At this point, the normal is given by $\frac{\nabla S(\alpha v)}{\|\nabla S(\alpha v)\|}$ which simplifies into $\frac{\nabla S(v)}{\|\nabla S(v)\|}$. Given this parametrization, we write a contact constraint between the frame of the hand of the robot and the point and normal on the surface of the superellipsoid.

In Fig. 5.6 we present some results for a posture generation problem with manipulation: On the left side, the feet are free to move on a sphere, and, on the right side, the left foot position is fixed and the right foot is free to move on the ground. The hands must be in contact with the cube. The cube is free to move (parametrized by $\mathbb{R}^3 \times SO(3)$) and has it own set of Euler-Newton equations, which must be fulfilled.

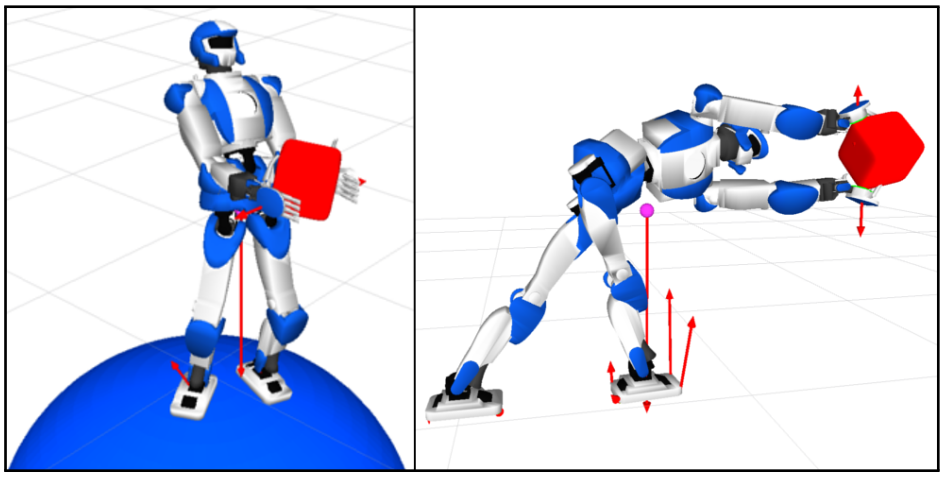


Figure 5.6 HRP4 carrying a 2 kg cube. Left: the objective function is to maintain the cube at a given position. Right: it is to put the cube as far as possible in a given direction

5.7 Contact with Complex Surfaces

We have presented some methods to formulate contact constraints with parameterized surfaces. The surfaces that we used so far could all be approximately represented by closed-form equations (sphere, superellipsoid, wrist of HRP-2). This is sufficient to tackle a large number of interesting scenarios, especially for robots in structured environment [VKA+14] [VKA+16]. However, contacts with more complex objects are needed too, for unstructured environment or for manipulating objects.

To work with derivative-based optimization algorithms, such as SQP or Interior Points, the functions involved in the optimization problem need to be at least C^1 and gradients must be full rank at the solution (for ensuring constraints qualification, see [NW06]). Since the contact point can be anywhere on our parametrized surface, the gradients must be full rank everywhere.

In the previously presented parametrizations of typical geometric shapes (sphere, superellipsoid...) those continuity requirements are not a problem. But for a given object, a parametrization of its surface is rarely available, especially a parametrization conforming with the continuity requirements. On the contrary, we usually have an approximation of the object as a mesh.

We can consider two types of surface parametrizations, the surfaces of closed topology, like a sphere, which are a large part of the objects of interest in our application, that we parametrize on $S2$ (because there is no diffeomorphism between the unit sphere and a subset of $\mathbb{R}^2$) and the surfaces of open topology, that generally can be parametrized on $\mathbb{R}^2$. The

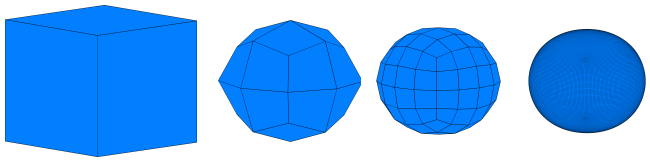


Figure 5.7 Several Catmull-Clark subdivisions of a cube. From left to right: original mesh, after 1, 2 and 6 iterations

problem of finding a parametrization satisfying the continuity requirements exists for both of those when the surface's representation comes as a mesh.

In [EBK16], we provide a systematic way to obtain a parametric surface closely approximating a mesh and meeting the continuity requirements mentioned above, for a large class of objects, namely star-convex objects. To that end, we combine the use of Catmull-Clark subdivision surfaces with a tailored ray casting algorithm, and propose an efficient and robust numerical method to evaluate a point, a normal and their derivatives at a given set of parameters.

The Catmull-Clark algorithm is used to create a smooth surface modeling a given *control mesh*. It iteratively subdivides a control mesh by following a set of rules; The limit surface of this subdivision process is called the Catmull-Clark Surface(CCS), see 5.7. It was shown by Stam in a seminal paper [Sta98] that points on CCS can be computed by direct analytical formula without explicitly subdividing the control mesh. A map between each face of the control mesh and the patch of the CCS associated to it $s(u,v)$ can be devised(with (u,v) a 2D parametrization of a face). To find the point of the CCS associated with a value $d \in S2$, we use a raycast algorithm: given c, a center point of the star-convex control mesh(a point seeing all the surface of the control mesh), we solve numerically $c+td = s(u,v)$ in (u,v,t) with a Newton method. This gives us a mapping from $S2$ to the CCS. From there, we propose methods to compute the normal to the CCS point associated with an element of $S2$ and we devise the derivations of both those mappings.

As a result, we get a smooth parametrization of the input mesh that satisfies the continuity requirements. In Fig:5.8, we show the Catmull-Clark surface (in blue) corresponding to the mesh (in green) of an HRP-2 link, for two different levels of smoothness. On the right picture, we first apply twice a simple subdivision scheme, before using Catmull-Clark subdivisions, thus obtaining a surface closer to the original mesh but less smooth. The red line is the image of a great circle of the unit sphere through our parametrization.

Because this parametrization respects the continuity requirements for being used efficiently in optimization, we can use it freely in our posture generation problems.

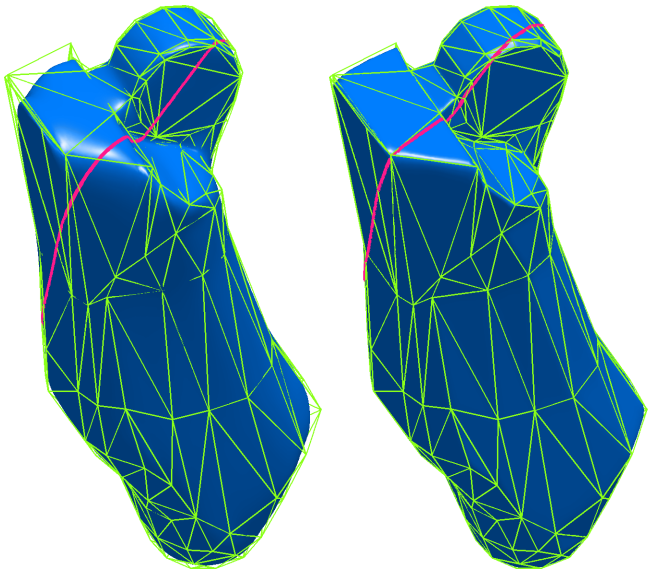


Figure 5.8 Catmull-Clark Surfaces (in blue) approximating the mesh of a link on HRP-2 (in green)

We now show some examples of PG using the proposed approach to generate contacts with complex objects. We make use of the capacity of our optimization solver to directly work with a variable $d \in S2$ to parametrize our surfaces.

In the first example (Fig. 5.9, left), a HRP-4 robot is asked to grasp an object (here a leg part of HRP-2) and to hold it as far as possible in front of him, with the following constraints: contacts must occur between predefined points on the hands of HRP-4 and points free to move on the surface of the object, modeled as a parametric surface with the approach of [EBK16]; furthermore the robot has to keep its left foot at a given fixed position and has its right foot anywhere on the floor. The forces applied by the robot on the object must be sufficient to counter the gravity, and the system (robot,object) must be stable with all forces within their friction cones. Additionally, the robot must stay within its joint limits. We do not check here for collisions or torque limits. The solver finds a solution in about 70 iterations.

Contacts also occur in manipulation. Actually, manipulation and locomotion are closely related. Locomotion can be seen as a manipulation of the environment, it is only a matter of reference frame (see [BK12a]). The second example (Fig. 5.9, right) demonstrates our PG applied to grasping an object. The base of the hand is fixed, and the contact points on the fingers are given. The contact points on the objects are parametrized with our method and are free to move on the entire (approximated) surface of the object: the optimization finds automatically how to grasp the object so as to be able to maintain its stability, taking about 50 iterations. Note that the object is non-convex.

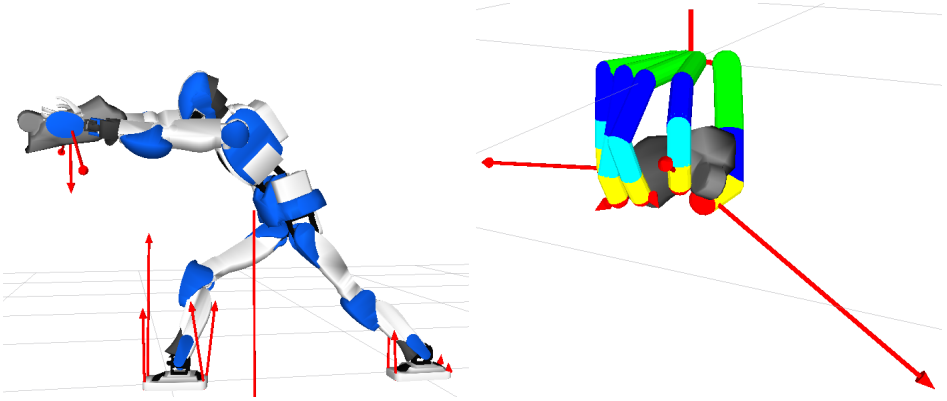


Figure 5.9 Left: HRP-4 holding an object, modeled with CCS, as far as possible in front of it. The red arrows depict the contact forces and the gravity forces applied on each object. Right: example of grasp generation.

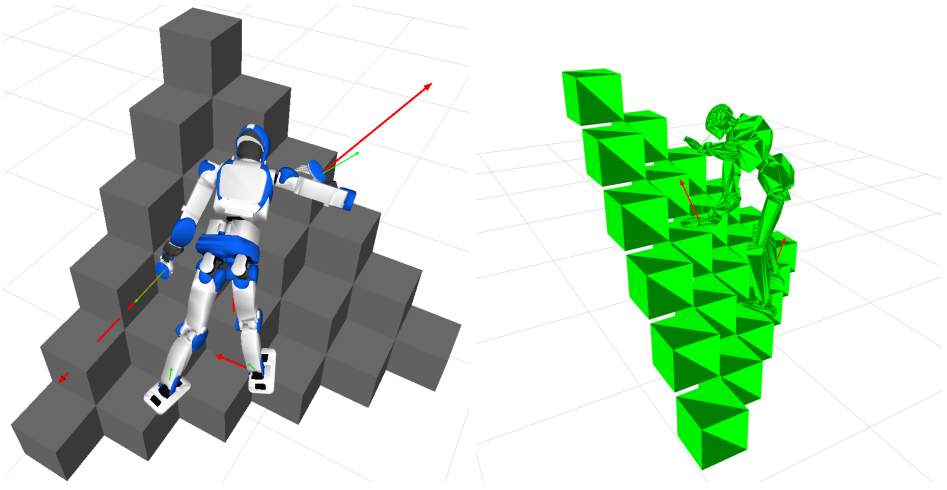


Figure 5.10 HRP-4 climbing a pile of cubes modeled as a single object with a single surface. (Right) Convex shapes used for collision avoidance

The third example (Fig. 5.10) involves a stack of cubes on which the robot must stand, using its hands and feet to maintain its stability and respect its torque limits. The stack is modeled as a single object. Once again, contact points are fixed on the robot and free on the surface. The surface is modeled by a single object. That object is not convex, thus it cannot be used as is for collision avoidance. For collisions, we model the stack of cubes with a set of 21 cubes all slightly smaller than the actual cubes see: 5.10 (Right), and we require the robot to avoid collision with those cubes. The optimization takes around 115 iteration to converge to a solution. Since there is a single surface to contact with, we do not need to specify with which faces, edges or corners the robot needs to be in contact with. The optimization decides it automatically. This is a very attractive feature of our approach as it allows to include discrete choices directly into our PG. This can be used to alleviate the work to be done by the user, be it a human or a planning algorithm: the user still has to specify the bodies in contact, but part of the combinatorics relative to the matching of a body with a particular surface or object is handled directly by the PG.

5.8 Conclusion

In this section, we presented our approach for formulating a posture generation problem and generating an optimization problem that represents it. We presented several tools that are meant to simplify this process for the developer, with the automatic variable management, the use of non-Euclidean manifolds and the geometric expressions framework.

Leveraging those tools, we developed methods to formulate constraints involving non-flat surfaces, by adding an extra variable to our optimization problem to parameterize the location of a frame on the surface. To avoid optimization issues, we make sure that our formulations have satisfying continuity properties. This addition allows formulating much richer posture generation problems, where the location of a contact on a non-flat surface is abstracted and chosen by the optimization solver. We were able to use this approach to represent several kinds of surfaces that could be described by closed-form formulas, such as the wrist of HRP-2, a sphere, and a superellipsoid. We extended that method to approximate surfaces for which the only available representation is a (star-convex) mesh. This allows formulating constraints on the surface of a very large variety of objects.

In the next section, we evaluate the performances of our solver and our posture generator before presenting some use cases and extensions to real world situations.

Chapter 6

Evaluation and Experimentation

In this chapter, we study the performances of our solver to solve problems formulated on manifolds as well as the performances of our posture generator. We then present an approach to make use of posture generation in real-environment with data acquired directly from the robot.

6.1 On the performance of formulation with Manifolds

In Chapter 4 we presented our approach to develop a nonlinear SQP solver on non-Euclidean Manifolds. We then proposed to take advantage of it in the posture generator presented in Chapter 5. The choice of using optimization on manifolds was driven by the intuition that such approach would lead to generally faster and more robust convergence of resolution when the search space is a non-Euclidean manifold (due to the reduction of the numbers of constraints and variables).

In a typical robotic posture generation problem, the search manifold is a composition of several instances of $\mathbb{R}^n$ and one $SO(3)$ per robot, and the equations involving the $SO(3)$ variables are quite complex. Thus it may be difficult to extract the actual influence of the use of manifold formulation on the resolution. In order to evaluate the influence of the formulation and resolution with manifolds, we choose to study a problem different from a typical robotic posture generation.

We choose to study a problem of cube stacking. Given a set of cubes, which pose is defined on $\mathbb{R}^3 \times SO(3)$, we want to find a configuration in which the cubes are all inside a box, while not penetrating each other. For any cube C_i, we denote $V_i = \{v_0, v_1, \ldots, v_7\}$ the set of all its vertex, $\vec{t} \in \mathbb{R}^3$ and $R \in SO(3)$ respectively represent its translation and rotation w.r.t the world frame. To ensure that the cubes are inside the box, we write a constraint that forces each corner of each cube to be inside the box. For each plane composing the box, we

denote $\vec{n}$ its normal toward the inside of the box, and d the signed distance from the plane to the origin along $\vec{n}$. The constraint for each cube C_i being 'above' a plane defined by $\{d, \vec{n}\}$ is of dimension 8 (1 per vertex) and can be written as:

$$\forall v \in V,\ (\vec{t} + Rv) \cdot \vec{n} \geq d \tag{6.1}$$

To avoid interpenetration of the cubes, we could use the usual collision avoidance constraints as presented in Section 5.3. In that case, the use of the exact mesh of the cubes would generate gradient discontinuities of the constraints. Approximating the mesh with STPBV would allow avoiding those discontinuities, but then, if the STPBV approximates the exact mesh closely, we would have a gradient close to discontinuous. Instead, we propose another approach that uses non-Euclidean manifolds: for each pair of cubes $C_i,\ C_j$, we require a plane P_{ij} to fit in-between them. The plane's location can be parametrized by its normal $\vec{n} \in S2$ and $d \in \mathbb{R}$, the signed distance from the plane to the origin along $\vec{n}$. Each plane's pose can be represented by a variable on $\mathbb{R} \times S2$. Thus we can write a constraint of dimension 16 (1 per vertex) such that C_i is above P_{ij} and C_j is below P_{ij} as follows:

$$\begin{aligned} &\forall v \in V_i,\ (\vec{t} + Rv) \cdot \vec{n} \geq d \\ &\forall v \in V_j,\ (\vec{t} + Rv) \cdot (-\vec{n}) \geq -d \end{aligned} \tag{6.2}$$

In order to simulate gravity, we minimize the sum of all the cubes' vertical coordinates:

$$f = \sum_i \vec{t}_i \cdot \vec{z} \tag{6.3}$$

We consider the problem of stacking n cubes in an open-top box composed of 5 plans (the ground and 4 walls). Each cube's pose is parametrized on $\mathbb{R}^3 \times SO(3)$, while each plane is parametrized on $\mathbb{R} \times S2$. In figure 6.1, we illustrate the case of stacking 3 cubes. With the initial condition on the left side and the solution on the right.

There is one plane for each pair of cubes, so, $n(n-1)/2$ plans. Thus the search manifold is:

$$\mathcal{M} = \left(\mathbb{R}^3 \times SO(3)\right)^n \times (\mathbb{R} \times S2)^{\frac{n(n-1)}{2}}$$

and the problem contains 5 constraints of dimension 8 per cube to fit them in the box and $n(n-1)/2$ constraints of dimension 16 to avoid the interpenetration of cubes. We have a problem of dimension $4.5n + 1.5n^2$ with $32n + 8n^2$ constraint.

In order to compare the resolution with and without the use of manifolds, we also formulate this problem over $\mathbb{R}^n$. To do so, each variable on $SO(3)$ is replaced by a variable

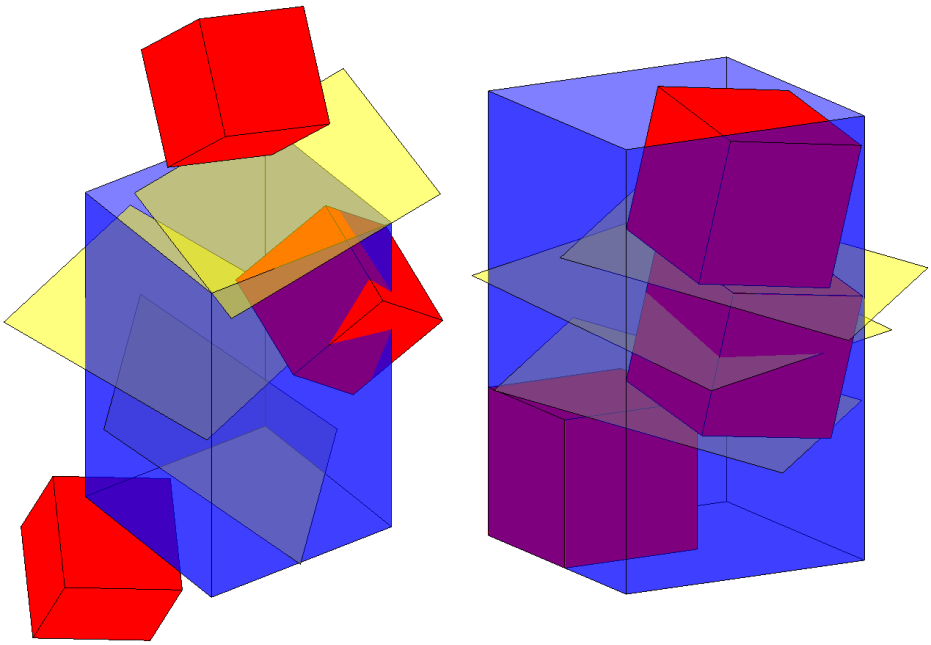


Figure 6.1 Problem of stacking 3 red cubes in a blue box, separating each pair of cubes by a yellow plane. Initial (left) and final (right) configurations

on $\mathbb{R}^4$, while each variable on $S2$ is replaced by one on $\mathbb{R}^3$. In both cases, a norm-1 constraint on the variable is added to the problem to force those variables on the manifolds. This gives rise to a problem on

$$\mathcal{M} = \left(\mathbb{R}^3 \times \mathbb{R}^4\right)^n \times \left(\mathbb{R} \times \mathbb{R}^3\right)^{\frac{n(n-1)}{2}} = \mathbb{R}^{5n+2n^2}$$

We have a problem of dimension $5n+2n^2$ with ,$32.5n+8.5n^2$ constraints, which is $\frac{n(n+1)}{2}$ more variables and constraints than with the manifold formulation.

We solve those problems for different numbers of cubes and compare the results in terms of number of iterations before convergence, time taken to converge and time spent per iterations. In each resolution, both problems are initialized with the same randomized initial guess, and the solver is set up with the same set of parameters. The initial positions of the cubes are chosen randomly, and each plane is initialized at a position between the two cubes it separates. We display the results of these tests in figure 6.2.

With 300 resolutions per case, approximately 98% converged when using a manifold formulation versus 99.5% with a non-manifold formulation. We observe that the numbers of iterations are sensibly similar for the two types of resolutions, but the time spent per iteration is consistently smaller in the case of a resolution with manifolds, which is in agreement

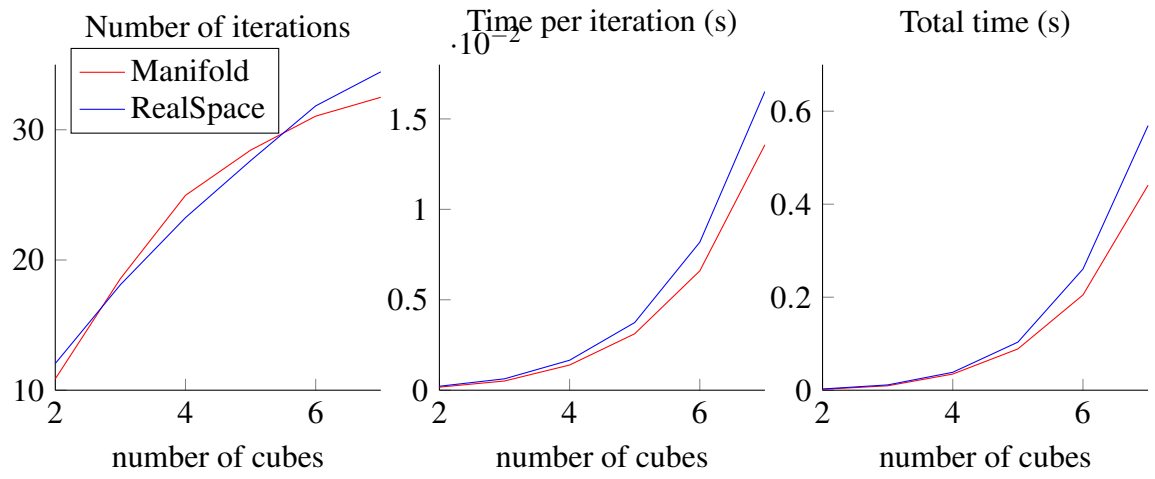


Figure 6.2 Comparison of resolutions with and without using manifolds

with our expectations and subsequently, the convergence time is consistently shorter for the formulation with the manifold formulation.

During our experimentations, we noticed that the regularization applied on the Hessian of the problem plays an important role in the convergence speed. In particular, the minimum value for the diagonal terms μ is important. We observed that for values of μ between 10^{-8} and 10^{-2}, the behaviors of both resolutions are quite consistent (the influence of μ is small in that span), with the best results observed for 10^{-8}, and the formulation with manifolds is consistently faster than the one without manifolds. Values outside of that span tend to degrade both resolutions. Too high values tend to damp the hessian, which translates into bigger numbers of iterations. Too small values make the hessian closer to being rank deficient, in that case, we observe larger numbers of internal iterations of the QP solver, which translates into longer times per iteration.

This study shows that on a simple example that makes heavy use of non-Euclidean manifolds, solving a problem on non-Euclidean manifolds with optimization on manifolds not only allows the user to profit of a simpler and more intuitive formulation of the problem but also outperforms the classical approach.

6.2 Evaluation of the Posture Generation

In addition to evaluating our solver, we want to evaluate the efficiency of our posture generator. We propose to study its ability to find a solution to problems that are known to be feasible.

First, we generate a list of feasible problems. To do so, we set the HRP-2 Kai robot in a random configuration q_{rand} within its joint limits and record the 3D poses of its end-effectors'

frames: at the tip of its grippers and below its foot,we denote them $F_{\text{rightGripper}}$, $F_{\text{leftGripper}}$, $F_{\text{rightFoot}}$, F_{leftFoot}. Then we generate a problem with the constraints that both grippers are in fixed bilateral contact (forces can be applied in any direction) with their associated frames $F_{\text{rightGripper}}$ and $F_{\text{leftGripper}}$, while the feet are in unilateral frictional contact (forces in friction cones) with their associated frames $F_{\text{rightFoot}}$ and F_{leftFoot} and the stability, torque limits, and auto-collision avoidance constraints must be respected. To help the resolution, we initialize the robot's configuration variable to q_{rand}, which is a solution to the geometric part of the problem. If a solution is found, the problem is known to be feasible and is recorded for further use.

We evaluate the performance of our posture generator by solving the recorded problems, starting from the half-sitting configuration $q_{\text{half-sitting}}$ of the robot. This approach tests the ability to find a feasible posture when starting from a configuration remote from the solution. Since there is no cost function, the optimization algorithm is exited as soon as the restoration phase terminates. The resolution is successful for approximately 35% of the problems. Which is not surprising, as the configuration to reach is very remote from the initial posture. We noticed that most of the randomly generated configuration put the robot in a very twisted posture that seems difficult to reach for the optimization algorithm when starting from the half-sitting configuration.

We ran another set of tests in which the value of q_{rand} is modified to be closer to the half-sitting, to do so, we simply take the average between the random configuration and the half-sitting:

$$q_{\text{rand}} \leftarrow \frac{q_{\text{rand}} + q_{\text{half-sitting}}}{2} \tag{6.4}$$

We generate the set of feasible problems from those configurations. In that case, a solution is found in 85% of the tests. This shows that the initial guess and its distance from the solution which we coin the *initial distance* are of crucial importance and should be chosen wisely by the user.

To study that influence, we propose to estimate the effect of the initial distance on the success rate of the posture generator. The distance between 2 configurations q_0 and q^* is defined as $d = \|q_0 - q^*\|^2$. In that experiment, each feasible problem is solved several times, with a value of the initial guess increasingly close to the solution. Given an initial guess chosen randomly q_0, and q^* being the solution of the problem found when generating the list of feasible problems, we solve the problem starting from a configuration $q_0^{n\%}$ defined as:

$$q_0^{n\%} = nq_0 + (1-n)q^* \tag{6.5}$$

With n taking successively the values 1.0, 0.8, 0.6, 0.4,and 0.2. After aggregating the results, we are able to estimate the influence of the initial distance on the success rate of the optimization. During all our work with our solver, we observed that the choice of hessian approximation method influences a lot the resolution. We take here the opportunity to verify this observation by solving series of problems with the solver using different hessian update methods. As we explained in Chapter 4, the Hessian can be updated as a whole, or individually, in which case a list of Hessians approximating each constraint's second-order derivative is maintained and combined in the global hessian matrix. We compare the results obtained with BFGS with or without self-scaling and using whole or individual updates. We gather and report those results in figure 6.3:

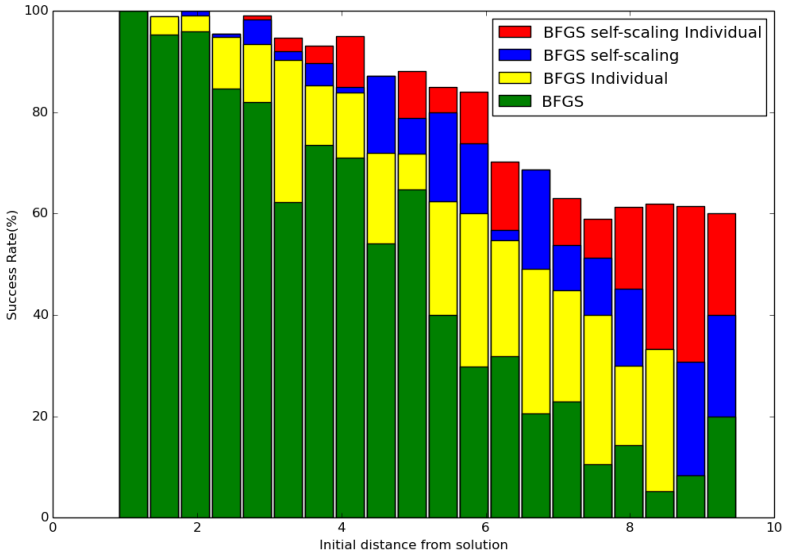


Figure 6.3 Rate of success of the posture generation compared to the distance between initial guess and solution.

This experiment shows that the closer the initial guess is to the solution, the more likely the solver is to converge to a solution and gives us a quantitative estimate of the expected success rate with respect to the distance. Overall, solving those problems with our solver using the BFGS with self-scaling and individual updates approach is more successful than with other methods. With that Hessian update method, for any distance between initial configuration and solution, the resolution converges in more than 60% of the cases.

It would be interesting to run more tests like those to study the influence of the different solver options and find a set of options that is optimal to solve posture generation problems.

6.3 Application to Inertial Parameters Identification

Although it was developed with the idea of solving posture generation problems, our solver and the formulation on manifolds can be used for different types of problems. In this section, we present an application of our optimization algorithm to solve a problem of inertial parameters identification that was published in [TBEN16].

6.3.1 Physical Consistency of Inertial Parameters

The dynamics of a rigid body is completely defined by its mass distribution:

$$\rho(.) : \mathbb{R}^3 \mapsto \mathbb{R}_{\geq 0} \tag{6.6}$$

That function defines the density of a rigid body in the 3D space, it is strictly positive on points that belong to the rigid body and null everywhere else. It can be reduced to 10 inertial parameters $\pi \in \mathbb{R}^{10}$ to describe the dynamics of the rigid body [HKG08]. We denote $\text{vech}(.)$ the serialization operation on symmetric matrices.

$$\pi = \begin{bmatrix} m \\ mc \\ \text{vech}(I_B) \end{bmatrix} \tag{6.7}$$

- $m \in \mathbb{R}$ is the mass of the rigid body
- $\hat{c}$ denotes the skew-symmetric matrix associated with $c \in \mathbb{R}^3$, the position of the center of mass of the rigid body
- $I_B \in \mathbb{R}^{3\times 3}$ is its 3D inertia matrix

By definition, the inertial parameters are functionals of the mass distribution $\rho(.)$, and can be written as $\pi_d : (\mathbb{R}^3 \mapsto \mathbb{R}) \mapsto \mathbb{R}^{10}$

$$\pi_d(\rho(.)) = \begin{bmatrix} \iiint_{\mathbb{R}^3} \rho(r)dr \\ \iiint_{\mathbb{R}^3} r\rho(r)dr \\ \text{vech}\left(\iiint_{\mathbb{R}^3} \hat{r}^\top \hat{r} \rho(r)dr \right) \end{bmatrix} \tag{6.8}$$

The identification of those parameters can in some cases be obtained from the Computer-Aided Design (CAD) model of the body, but it is often necessary to evaluate those parameters

experimentally. Their identification can be done by measuring the body acceleration A^g, twist V and the external wrench applied to it F for N different situation and finding π^* that minimizes the error in the Newton-Euler equation for those values:

$$\pi^* = \underset{\pi \in \mathbb{R}^{10}}{\arg\min.} \|Y(A_i^g, V_i)\pi - F_i\|^2 \tag{6.9}$$

Where $Y(A^g, V)$ is such that:

$$Y(A^g, V)\pi = MA^g + V\bar{\times}^* MV$$

M is the spatial inertia matrix:

$$M = \begin{bmatrix} m\mathbb{I}_3 & -m\hat{c} \\ m\hat{c} & I_B \end{bmatrix} \tag{6.10}$$

and $V\bar{\times}^*$ is the 6D force cross product operator [Fea07] that, for $V = \begin{bmatrix} v^\top & \omega^\top \end{bmatrix}^\top \in \mathbb{R}^6$, gives:

$$V\bar{\times}^* = \begin{bmatrix} \omega^\top & 0_{3\times 3} \\ v^\top & \omega^\top \end{bmatrix}$$

However, this optimization problem 6.9 does not take into account the physical properties of the inertial parameters. For this, the *physical consistency* constraint is introduced [YOMO94] and added to the problem:

Definition A vector of inertial parameters $\pi \in \mathbb{R}^{10}$ is called *physical consistent* if:

$$\begin{aligned} m(\pi) &\geq 0 \\ I_C(\pi) &\succeq 0 \end{aligned} \tag{6.11}$$

Where $I_C(\pi)$ is the 3D inertial matrix at the center of mass. Inertial parameters satisfying the *physical consistency* conditions have nice properties (M is invertible), but it is still possible to find some *physical consistent* inertial parameters that cannot be generated by a mass distribution of a rigid body. In [TBEN16], we propose the *full physical consistent* condition that assesses that a vector of inertial parameters can be generated from a physical rigid body.

Definition A vector of inertial parameters $\pi \in \mathbb{R}^{10}$ is called *fully physical consistent* if:

$$\exists \rho(.) : \mathbb{R}^3 \mapsto \mathbb{R}_{\geq 0} \text{ s.t. } \pi = \pi_d(\rho(.)) \tag{6.12}$$

We propose a new parametrization of the inertial parameters by an element $\theta \in \mathfrak{P} = \mathbb{R}_{\geq 0} \times \mathbb{R}^3 \times SO(3) \times \mathbb{R}^3_{\geq 0}$ that ensures the *full physical consistency*. In particular the components of θ are:

- $m \in \mathbb{R}_{\geq 0}$ the mass of body
- $c \in \mathbb{R}^3$ the center of mass of the body
- $Q \in SO(3)$ the rotation matrix between the body frame and the frame of principal axis at the center of mass
- $L \in \mathbb{R}^3_{\geq 0}$ the second central moment of mass along the principal axes

And a function $\pi_p(\theta) : \mathfrak{P} \mapsto \mathbb{R}^{10}$ that maps this new parametrization to the corresponding inertial parameters.

$$\pi_p(\theta) = \begin{bmatrix} m(\theta) \\ mc(\theta) \\ \text{vech}\,(I_B(\theta)) \end{bmatrix} = \begin{bmatrix} m \\ mc \\ \text{vech}\left(Q\,\text{diag}\,(\begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} L) Q^\top - m\hat{c}\hat{c}\right) \end{bmatrix}$$

In [TBEN16], we prove that every $\theta \in \mathfrak{P}$ generates *fully physically consistent* inertial parameters. However, the optimization variable now lives on a non-Euclidean manifold, because $\mathfrak{P}$ includes $SO(3)$. Thus, we propose to solve that problem with our solver on manifolds.

6.3.2 Resolution with optimization on Manifolds

The formulation of the problem becomes:

$$\begin{aligned} \underset{\theta \in \mathbb{R} \times \mathbb{R}^3 \times SO(3) \times \mathbb{R}^3}{\arg\min.} & \sum_{i=1}^{N} \left\| Y(A_i^g, V_i)\pi(\theta) - F_i \right\|^2 \\ & \text{s.t. } m \geq 0,\ L_x \geq 0,\ L_y \geq 0,\ L_z \geq 0 \end{aligned} \tag{6.13}$$

It has an immediate advantage: we can write directly the problem (6.13) without the need to add any parametrization-related constraints. Because there are fewer variables and fewer constraints, it is also faster to solve. To check this, we compared the resolution of (6.13) formulated with each of the three parametrizations (native $SO(3)$, unit quaternion, rotation matrix). We solved the three formulations with the solver presented in [BED+15], and the two last with an off-the-shelf solver (CFSQP [LZT97]), using the dataset presented in section 6.3.3. The formulation with native $SO(3)$ was consistently solved faster. We observed

timings around 0.5s for it, and over 1s for non-manifold formulations with CFSQP. The mean time for an iteration was also the lowest with the native formulation (at least 30% when compared to all other possibilities).

Working directly with manifolds has also an advantage that we do not leverage here, but could be useful for future work: at each iteration, the variables of the problem represent a *fully physically consistent* set of inertial parameters. This is not the case with the other formulations we discussed, as the (additional) constraints are guaranteed to be satisfied only at the end of the optimization process. Having physically meaningful intermediate values can be useful to evaluate additional functions that presuppose it (additional constraints, external monitoring ...). It can also be leveraged for real-time applications where only a short time is allocated repeatedly to the inertial identification, so that when the optimization process is stopped after a few iterations, the output is physically valid.

6.3.3 Experiments

We ran some experiments with the iCub robot. It is a full-body humanoid with 53 degrees of freedom [MNN+10]. For validating the presented approach, we used the six-axis force/torque (F/T) sensor embedded in iCub's right arm to collect experimental F/T measurements. We locked the elbow, wrist and hands joints of the arm, simulating the presence of a rigid body directly attached to the F/T sensor, a scenario similar to the one in which an unknown payload needs to be identified [KKW08].

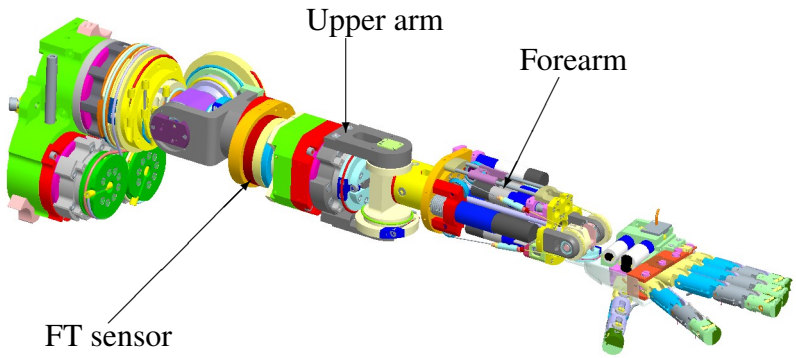


Figure 6.4 CAD drawing of the iCub arm used in the experiments. The six-axis F/T sensor used for validation is visible in the middle of the upper arm link.

We generated five 60 seconds trajectories in which the three shoulder joints were reaching successive random joint positions using minimum-jerk like trajectories. Each trajectory

Trajectory Time	Optimization Manifold	m	mc_x	mc_y	mc_z	I_{xx}	I_{xy}	I_{xz}	I_{yy}	I_{yz}	I_{zz}
10s	$\mathbb{R}^{10}$	1.836	0.062	0.001	0.208	0.580	0.593	-0.541	1.022	0.190	-0.129
	$\mathfrak{P}$	1.836	0.062	0.001	0.208	0.215	0.012	-0.064	0.227	0.038	0.028
5s	$\mathbb{R}^{10}$	1.842	0.061	0.000	0.206	0.128	-0.018	-0.125	0.125	0.026	-0.001
	$\mathfrak{P}$	1.842	0.060	0.000	0.206	0.166	0.001	-0.089	0.216	0.001	0.050
2s	$\mathbb{R}^{10}$	1.852	0.060	0.001	0.206	0.065	0.001	-0.035	0.066	0.006	0.007
	$\mathfrak{P}$	1.852	0.060	0.001	0.206	0.067	0.001	-0.030	0.086	0.003	0.014
1s	$\mathbb{R}^{10}$	1.820	0.060	0.002	0.205	0.032	0.0014	-0.017	0.036	0.002	0.008
	$\mathfrak{P}$	1.820	0.060	0.002	0.205	0.034	0.001	-0.015	0.042	0.001	0.009
0.5s	$\mathbb{R}^{10}$	1.843	0.060	0.005	0.204	0.033	0.003	-0.014	0.035	0.000	0.008
	$\mathfrak{P}$	1.844	0.059	0.004	0.204	0.037	0.001	-0.013	0.039	0.000	0.008

Figure 6.5 Inertial parameters identified with the different datasets and the different optimization problems. Inertial parameters identified on $\mathbb{R}^{10}$ optimization manifold that are not *fully physically consistent* are highlighted. Masses are expressed in kg, first moment of masses in $kg.m$, inertia matrix elements in $kg.m^2$.

is decomposed is sub-trajectories to travel between two consecutive random positions in respectively 10s, 5s, 2s, 1s and 0.5s (As a result, the trajectory are ordered from the slowest to the fastest). We played those trajectories on the robot and we sampled at 100Hz the F/T sensors and joint encoders output. We used joint positions, velocities and accelerations with the kinematic model of the robot to compute A^g and V of the F/T sensor for each time sample. We solved the inertial identification problem (6.9) using a classical linear algorithm and the one using the proposed *fully physically consistent* parametrization (6.13) with our solver on manifolds. We report the identified inertial parameters in Table 6.5. It is interesting to highlight that for slow datasets (sub-trajectory times of 10s or 5s) the unconstrained optimization problem (6.9) results in inertial parameters that are not *fully physically consistent*. In particular, this is due to the low values of angular velocities and acceleration, that do not properly excites the inertial parameters, which are then *numerically not identifiable*.

The proposed optimization problem clearly cannot identify those parameters anyway, as the identified parameters are an order of magnitude larger than the ones estimated for faster datasets, nevertheless, it always estimates inertial parameters that are *fully physically consistent*. For faster datasets (trajectory time of 1s or 0.5s) the results of the two optimization problems are the same because the high values of angular velocities and accelerations permit to identify all the parameters perfectly. While this is possible to identify all the inertial parameters of a single rigid body, this is not the case when identifying the inertial parameters of a complex structure such as a humanoid robot, for which both structural [AVN14] and numerical [PG91] not identifiable parameters exists. In this later application, the enforcement of *full physical consistency* will always be necessary to get meaningful results.

6.4 Application to contact planning on real-environment

In this section, we present our approach to generate multi-contact planning scenarios in a sensory acquired environment. This work was done prior to the development of our posture generator and therefore make use of a previous version of it that is presented in [BK10b].

Generating isolated postures is not sufficient to make a robot move or achieve tasks in ways similar to humans. It needs to plan entire motions, with sequences of contact creations and releases, and trajectories in-between. That's the role of the Multi-Contact Planner (MCP). The core of the planner consists of the multi-contact search and the posture generator. The former incrementally builds a tree of contact sets and is presented in [EKM13]. The children of a node are obtained by adding or removing exactly one contact to its set. At each iteration of the search, the best leaf, according to a potential field, is expanded this way. The tentative sets of contacts are tested by a posture generator. Upon success, the contact set is validated, and a new leaf is created. The goal is written as a set of specific constraints. A node is a final node if its associated posture generation problem augmented by these constraints remains feasible. By backtracking from this final node to the initial root node, we obtain a sequence of nodes and thus a sequence of contact sets, that can be executed on the robot by a whole-body controller such as the one based on a quadratic programming (QP) formulation presented in [BK11b].

The potential field is derived from a crude path, made of a few key postures, that does not take contacts into account. Such a path is either user-defined or can be the output of a first dedicated planner [BELK09].

The MCP relies largely on the 3D geometric models of the environment and robotic agents. In our previous work [EKM13, BK12a], the geometric models are provided by the user. The contact transition for the robot are planned off-line and later executed by the robot assuming exactness of the models and their relative positioning. We aim at extending our MCP to deal directly with real data acquired by the robot. Subsequently, we must deal with two kinds of situations:

1. the models of the objects in the environment are known: in this case adapting the MCP consists mainly in dealing with recognition, model superposition, and handling uncertainties. In brief, once model superposition is achieved, we can use the 3D model for MCP as in [EKM13, BK12a], yet some adjustments are needed.

2. the models of the hurdles and the environment are not known (e.g. disaster or outdoor environments, for example, related to the Fukushima disaster that inspired the DARPA Robotic Challenge), MCP is to be achieved in an egocentric way with models built from the robot's embedded sensors. This chapter deals with this case and we describe

how the MCP is modified to achieve this goal. In a nutshell, we construct planar surfaces from the 3D point clouds data and feed them to the MCP.

In robotics, the use of 3D-based vision for recognition and navigation in environments known or partially unknown has first been used on mobile robots, evolving in a flat environment, for example by coupling it with a SLAM system [WCD+10]. Another approach consists in extracting the surfaces from the point cloud, and then link them to the known environment or simply consider them as obstacles to be avoided [PVBP08]. Since working on raw point clouds is costly because of the high number of data points, this extraction has also been enhanced [BV12] in order to be run in real-time. This approach has recently been experimented on a humanoid robot in [MHB12], where two methods are combined: the surface extraction from a point cloud, and the voxel-based 3D decomposition of the environment [NL08]. Still, since the robot only navigates in a flat environment, and does not realize manipulation tasks, the surfaces extracted from the 3D point cloud are down projected to a 2D plan, on which are based the navigation and collision avoidance processes.

The use of humanoid robots allows to navigate in more complex environments, some work has been done to make a humanoid robot go down a ramp [LAHB12] or climb stairs [OHB12]. Yet, those methods use laser-based vision rather than point-cloud-based vision, so as to have a precise analysis of a known environment.

In this work, we aim at enabling a robot to analyze and plan a motion into a 3D environment. Hence, we use the surface extraction of a point cloud to directly have a global picture of the environment and determine the convex planar surfaces that the robot can use at its advantage to progress using the MCP. In our approach to make such an extension, we intentionally seek for technical solutions that minimize changes to be done on the existing MCP software.

6.4.1 Building an understandable environment

Our first concern is to build an environment that our multi-contact planner is able to "understand" and that can be extracted from a point cloud scene.

The simplest entity that our planner would be able to deal with and that could correctly describe the robot's environment is a set of convex polygonal planar surfaces. Therefore, starting from an acquired point cloud, we extract a relevant set of such geometrical entities that will be used as of the surroundings of the robot.

The different steps we follow to create a set of relevant convex polygonal plane surfaces out of an acquired point cloud are the following:

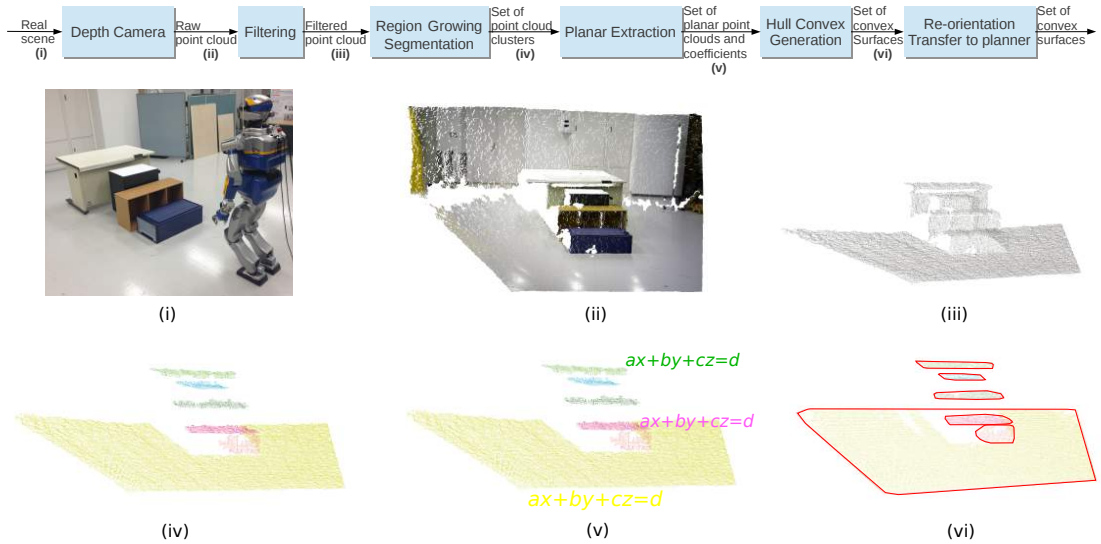


Figure 6.6 Top: flowchart describing the main elements of our algorithm and the type of data that is passed between them. Bottom: the data throughout the process, illustrated in the case of our first experiment.

The Figure 6.6 illustrates the major steps of this point cloud treatment. We use Willow Garage's Point Cloud Library[1] (PCL) [RC11] extensively for processing the point cloud.

Acquisition of point cloud from an RGB and depth sensor The point cloud representing the scene is acquired by an Asus Xtion Pro camera. The points are defined by their space coordinates and colors. We do not use the color information except for display purpose. It may, however, be useful for future developments in matching object models with sensor data and to perform color-based segmentation algorithms.

Filtering In order to reduce the computation time and improve the efficiency of our point cloud treatment algorithm, we filter out the points that are too far and use a voxelized grid approach to downsample the remaining point cloud. This consists of creating a 3D voxel grid over the point cloud and replacing the points in each voxel by their centroid. (We use this step to reduce the number of points to treat by a factor 5 to 6)

Region growing segmentation We divide a global point cloud scene into clusters of points that belong to the same flat area, by using a region growing algorithm [PVBP08] that regroups the neighboring points that have similar normals into clusters.

[1] http://pointclouds.org/

Planar extraction For each cluster, we use a plane segmentation to find the plane model that fits the highest number of points. The outlying points can then be either filtered, or we can try to find another plan to fit them.

Planar projection and hull convex generation We extract the convex hull of the projection of each cluster on their respective fitting plan. After this step, each plane surface of the scene is represented by a frame composed of the barycentre of the set of points and the convex hull of the surface.

Re-orientation and transfer to the planner After re-orienting each frame to get their expression with respect to the world frame, the list of {frame + convex hull} can be sent to the planner as a list of contact surface candidates.

6.4.2 Constraints for surfaces extracted from point clouds

Generating postures in which a contact between convex polygonal plane surfaces requires ensuring that the intersection area between the two surfaces is large enough to support the contact. One could use the formulation presented in Section 3.1. Or more simply a constraint of convex polygon inclusion, which is sufficient for cases such as the ones we study here, where the support surfaces are wide enough for the robot to lean on.

Such constraint can be written by enforcing that all the points of polygon S_i are located on the left side of all the segments of polygon S_j (provided that the points of S_j are ordered counterclockwise around its normal). Given S_i and S_j two surfaces which contours are defined respectively by the sets of points $p_0, p_1, \ldots, p_n$ and $q_0, q_1, \ldots, q_m$ and $\vec{n}$ a vector normal to S_i, the inclusion of S_i in S_j can be written as the following constraint:

$$\forall k \in [0,n],\ \forall l \in [0,m],\ [\overrightarrow{q_l p_k} \times \overrightarrow{q_l q_{l+1}}] \,.\vec{n} \leq 0 \tag{6.14}$$

Once the surfaces are defined, we can choose which ones are suitable for the robot to make contact with. Although it is not a mandatory step, it allows to reduce the exploration during the planning phase by removing undesired or inappropriate pairs of robot/environment surfaces. For the time being, this is determined by heuristics that are defined depending on the situation. If we want the robot to walk on various surfaces, only surfaces that have a normal vector closely aligned with the gravity field would be selected as potential candidates (so as to eliminate the walls and other surfaces on which the robot cannot walk). Similarly, only surfaces located at a certain height can be considered for hand contact, etc.

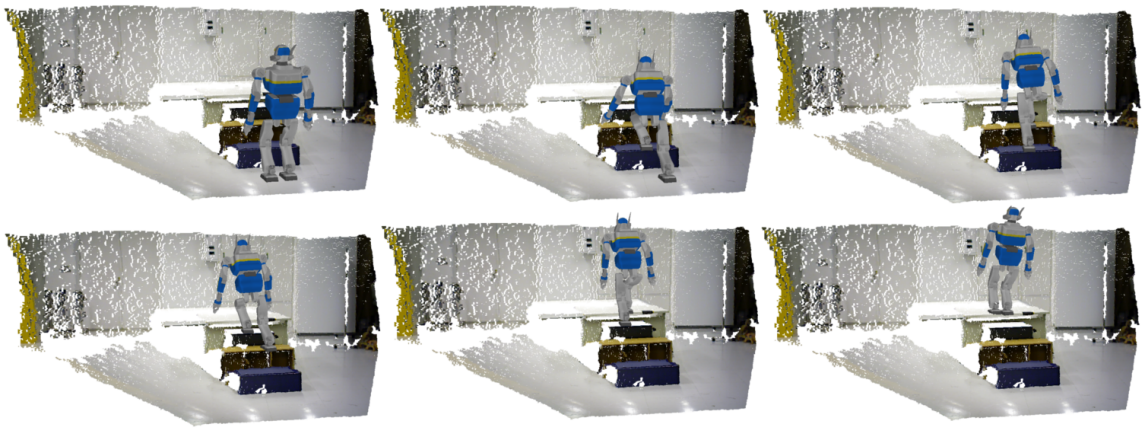


Figure 6.7 Table climbing simulation using irregular stairs. Of the 11 nodes of the path, we depicted the nodes 1, 4, 6, 8, 10 and 11.

For collision avoidance with the environment, we consider each surface generated by our point cloud treatment algorithm as a thin 3D body. Basically, we extrude each surface by few centimeters in the direction opposite to its normal (provided that this normal is pointing toward the outside of the real body) and create a convex hull surface using `QHull` [BDH96].

6.4.3 Results

We illustrate this approach with two experiments where the HRP-2 robot is required to move 2m forward. In both scenarios, this results in climbing on a table (The first one is 71cm-high and the second one is 53cm-high) with the help of various surrounding objects. The knowledge of the environment and surrounding objects is obtained from a point cloud captured with an RGBD camera.

All the computations of the following experiments are performed on a single thread of an Intel (R) Core (TM) i7–3840QM CPU at 2.80GHz, with 16Go of RAM.

Plan 1: irregular stairs In this first experiment, the robot has to walk up some irregular stairs made of several random pieces of furniture to reach its goal. The filtered point cloud was split into 6 plane surfaces. The whole cloud processing was done in 2.7 s. The planner generates a sequence of 11 nodes, some of which are depicted in Fig. 6.7. We notice that the robot climbs the stairs one by one without ever putting its two feet on the same step and without any noticeable problem. In total, 23 nodes were generated and the planning time was 98.4 s.

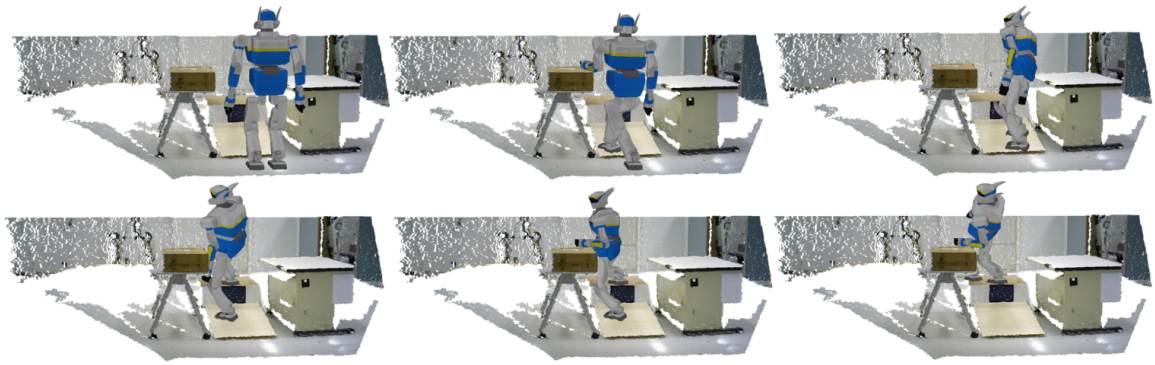


Figure 6.8 Slope and step climbing simulation. Of the 19 nodes of the path, we depicted the nodes 1, 7, 12, 15, 17 and 18.

Plan 2: helping motion with the hand This second experiment was designed to showcase a more complex plan involving the use of the HRP-2 upper limbs. In this experiment, we extracted 11 plan surfaces from the point cloud. The point cloud processing took 2.7 s. The planner computation generates a path of 19 nodes, some of which are depicted in Fig. 6.8. To climb on the table, the robot uses its left arm and walks on an inclined slope before climbing a step at the end of the slope, once again, with the help of its left arm as support. In total, 40 nodes were generated and the planning time was 122.3 s.

6.4.4 Discussion

This work is a first step toward a fully sensory-perception-based multi-contact planner. It raises several interesting questions on the way to adapt our MCP.

One advantage of our approach is to avoid having to precisely position the robot in the environment prior to the plan execution. Yet, for now, the positioning is done once and for all with the acquisition of the point cloud, before planning. When executing the plan, the robot might still deviate from it, for example, a support might move, or a foot might contact a few centimeters away from where it was planned to, or the support might not be at its expected position because of measurement errors in the acquired point cloud. We thus need to close the loop between the planning and its execution. To do so, we need robust detection of discrepancies to be made by the robot. This can be achieved by combining point-cloud-based SLAM with contact sensing. Once the robot has acquired knowledge of its deviation or of the position error of the support, it has to adapt to it. This adaptation should not be time-consuming so as to not interrupt the execution for too long. A slight deviation can be recovered by simply positioning with care the next contacts and the closed-loop multi-contact controller shall work on guarded-motions basis. However, a bigger deviation might make the

next contact stances infeasible. In this later case, re-planning the next contacts is necessary to go back to the plan. How many contacts have to be re-planned depends on the context. In difficult situations, changes in contacts might cascade up to requiring an entire re-planning. Recognizing the situation should be the task of a local planner that re-plans as few steps as possible. In case too much contacts must be reprocessed, the re-planning phase can be stopped before it ends and resumed at the next step.

This partial planning approach can also be seen in the context of semi-autonomous motion: an operator gives the overall direction with, for example, a joystick, and the planner reactively finds a sequence of a few contact sets to move as closely as possible in this desired direction. The operator is thus in charge of preventing the robot from getting stuck while the planner only concentrates on finding the correct contacts over a short time window.

Another question stems from the partial knowledge of the environment: it is not possible to give a guide path as we used to do with the 3D models. This guide path will necessarily be very crude, either a straight line to the desired position or a plan in a known environment before it was changed (for example in the case of a disaster in a plant). The planning must then be driven also by the need of getting information about the environment, for example reaching a viewpoint allowing to see parts of the environment that were hidden before, filling empty spaces and possibly adding new supports on-the-fly. Planning is then only partial since necessary part of the environment might be unknown.

Later on, the discovery of the environment might be improved by the use of other sensors. One can then imagine having the robot test a contact to ensure a given surface is fit for support or that it is precisely at the position measured by vision. If the surface is not, this is another kind of discrepancy in the plan that needs to be handled by re-planning.

The simulation results revealed that our MCP does not require major adjustments to handle egocentric sensory data.

Although we choose to treat the case of not having 3D models, we believe that the implementation of an MCP with knowledge of the 3D model is necessary. For example, even in a disaster situation as in Fukushima nuclear power plants, the inside exploration videos available show that many objects kept their shape and were not totally destroyed (e.g. door, stairs, ladders, etc...). So having their models would then still permit our MCP to rely on 3D models to plan contacts for motion. PCL provides only partial information, it is then necessary to drive the planner by the mission objective and also a perceptual one (e.g. SLAM).

6.5 Conclusion

In this section, we presented several evaluations and applications of our solver and posture generator. We first evaluated the influence of the formulation on manifolds on a problem making heavy usage of them and showed that as expected, this formulation outperforms the classical approach in terms of convergence times. Then we propose an approach to estimate the success rate of our posture generator with respect to the distance between the initial guess and the solution. We then present a different application of our solver to the problem of inertial identification of a rigid body. In which case, working with manifolds ensures the *full physical consistency* of the inertial parameters in any situation and all along the optimization process. There again, the formulation with manifolds outperforms the classical approach. We then present an application of posture generation to the problem of planning in a real environment, where we propose a method to segment a sensory acquired environment to allow the generation of postures on it. Finally, we present some use case scenarios in which our posture generator has been used efficiently.

Conclusion

In this Ph.D. work, we contributed to the formulation and the resolution of posture generation problems for robotics. Such problems aim at finding a robot configuration that satisfies some high-level requests while ensuring its viability, in the sense that, in this configuration, the robot is stable, avoid collisions and respects its intrinsic limitations. This problem is traditionally formulated and solved as an optimization problem with a collection of geometric and static constraints to satisfy while minimizing a cost function.

We presented the formulations of the basic building blocks of such problems before proposing some extensions, among which is a smooth formulation of non-inclusive contact constraints between two polygons; this allows to find optimal contact configurations in complex situations where the two surfaces in contact cannot be included in each other.

Robotics problems often contain variables that live in non-Euclidean spaces, we present a generic way to handle such variables in our formulation, and most importantly, we propose an approach to adapt existing optimization techniques to solve constrained nonlinear optimization problems defined on non-Euclidean manifolds. We then detail our implementation of such an algorithm, based on an SQP approach. Not only does it allow us to formulate mathematical problems more elegantly and ensures the validity of our variables at every step of the optimization process, it also enables us to have a better understanding and mastering in the way to tune our solver for robotics problems, and allows us to try and discover new ways of solving those problems.

We present a formulation of the posture generation problem that takes into account the inherent structure of its configuration space as a cartesian product of submanifolds representing different quantities(translations, rotations, joint angles, contact forces, etc.). Each submanifold of the search space is considered as a separate entity, and variables on it can be considered separately from each other. We take advantage of that structure to propose a posture generation framework where the variables, submanifolds, and functions derivations are managed automatically, and geometric expressions can be written intuitively, which simplifies the work of the developer of new functions. This allows to easily and elegantly

write custom constraints on selected sets of variables defined on the submanifolds of our search space.

We exploit the capabilities of our framework to generate viable configurations with contacts on non-flat surfaces by parametrizing the location of the contact point on additional variables. We proposed a generic way to parametrize the surface of a solid represented by a mesh, based on Catmull-Clark subdivision algorithm, and used it to compute configurations where the optimal location of contact on a complex object is chosen by the solver. The genericity of our framework allows to write a wide range of functions using any variables of the problem, should it be the joint angles, the forces, the torques or some additional variables, and to use those in problems that can then be solved by our solver to compute viable postures that satisfy the user-defined tasks.

We then present some evaluations of our solver and posture generator: we solved a cube stacking problem that relies heavily on the manifold formulation to showcase that, the manifold formulation performs better than the traditional one in that problem and in particular, the time spent per iteration is, as expected, shorter. We studied the influence of the distance between the initial guess and the problem's solution on the success rate of the posture generation problem, this showed that when starting close to the solution, a solution is almost always found whereas when starting remotely, it is more difficult to reach the solution, and more work on the solver could help increase this success rate. This approach allows to compare results for different solver options and showed that regarding hessian update options, the BFGS update with self-scaling on individual Hessians gives us the best results.

Although it was made to solve posture generation problems, we show that our solver's capabilities can be leveraged to solve other kinds of problems such as inertial parameters identification, where the manifold formulation allows to guarantee that the optimization iterates are always physically valid. For this problem, our solver and formulation proved more efficient than the traditional formulation solved with an off-the-shelf solver. Finally, we presented our preliminary work in using posture generation in multi-contact planning in real sensory acquired environment.

Although we manage to get some satisfying results out of our posture generator, it still often requires some tuning of the solver options and of the initial guess. We believe that improvements could be made to make our solver more reliable and to specialize it for the resolution of posture generation problems. In particular, we believe that the restoration phase and especially the treatment of the Hessian updates could be improved, we could also try using other QP solvers than LSSOL, a sparse one could take advantage of the structure of our problem. But most importantly, future works should be oriented in the direction of specializing the solver for posture generation problems, either by finding optimal solver

options for that kind of problems or by modifying the optimization algorithm. This is made possible by the fact that we now have an open solver that we can modify to suit our needs.

It would be interesting to use our posture generator in a higher level software, like a multi-contact stance planner, to automatically generate sequences of viable configuration to use to guide the control of the robot. Ideally, we would like to integrate the posture generation with our multi-contact control framework to allow on-the-fly replanning of postures when the initial plan is not feasible anymore because of some drift of the robot from the original plan.

Bibliography

[ACL16] Andreas Aristidou, Yiorgos Chrysanthou, and Joan Lasenby. Extending fabrik with model constraints. *Computer Animation and Virtual Worlds*, 27(1):35–57, January 2016.

[AD03] Tamim Asfour and Rüdiger Dillmann. Human-like motion of a humanoid robot arm based on a closed-form solution of the inverse kinematics problem. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 2, pages 1407–1412. IEEE, 2003.

[AD10] Jan Albersmeyer and Moritz Diehl. The lifted newton method and its application in optimization. *SIAM J. on Optimization*, 20(3):1655–1684, January 2010.

[AL09] Andreas Aristidou and Joan Lasenby. *Inverse kinematics: a review of existing techniques and introduction of a new fast iterative solver*. University of Cambridge, Department of Engineering, 2009.

[AL11] Andreas Aristidou and Joan Lasenby. FABRIK: A fast, iterative solver for the inverse kinematics problem. *Graphical Models*, 73(5):243–260, 2011.

[AMS08] P.-A. Absil, R. Mahony, and R. Sepulchre. *Optimization Algorithms on Matrix Manifolds*. Princeton University Press, 2008.

[AVN14] Ko Ayusawa, Gentiane Venture, and Yoshihiko Nakamura. Identifiability and identification of inertial parameters using the underactuated base-link dynamics for legged multibody systems. *The International Journal of Robotics Research*, 33(3):446–468, 2014.

[Bai85] John Baillieul. Kinematic programming alternatives for redundant manipulators. In *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, volume 2, pages 722–728. IEEE, 1985.

[BB01] Paolo Baerlocher and Ronan Boulic. Parametrization and range of motion of the ball-and-socket joint. In *Deform. Avatars*, pages 180–190. 2001.

[BB04] Paolo Baerlocher and Ronan Boulic. An inverse kinematics architecture enforcing an arbitrary number of strict priority levels. *The visual computer*, 20(6):402–417, 2004.

[BDH96] C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. The Quickhull algorithm for convex hulls. In *ACM Transactions on Mathematical Software, Vol. 22*, pages 469–483, University of Minnesota, December 1996.

[BED+15] Stanislas Brossette, Adrien Escande, Grégoire Duchemin, Benjamin Chretien, and Abderrahmane Kheddar. Humanoid posture generation on non-euclidean manifolds. In *Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on*, pages 352–358, Nov 2015.

[BELK09] Karim Bouyarmane, Adrien Escande, Florent Lamiraux, and Abderrahmane Kheddar. Collision-free contacts guide planning prior to non-gaited motion planning for humanoid robots. In *IEEE International Conference on Robotics and Automation*, 2009.

[BEMK09] M. Benallegue, A. Escande, S. Miossec, and A. Kheddar. Fast c1 proximity queries using support mapping of sphere-torus-patches bounding volumes. In *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pages 483–488, May 2009.

[BEV+14] Stanislas Brossette, Adrien Escande, Joris Vaillant, François Keith, Thomas Moulard, and Abderrahmane Kheddar. Integration of non-inclusive contacts in posture generation. In *IROS'14: International Conference on Robots and Intelligent Systems*, Chicago, United States, September 2014.

[BGLA03] J. Frédéric Bonnans, J. Charles Gilbert, Claude Lemaréchal, and Claudia A.Sagastizábal. *Numerical Optimization*. Springer, 2003.

[BGLS02] Frédéric Bonnans, Charles Gilbert, Claude Lemaréchal, and Claudia A. Sagastizábal. *Numerical optimization- Theoretical and Practical Aspects*. Springer, September 2002.

[BK05] Samuel R Buss and Jin-Su Kim. Selectively damped least squares for inverse kinematics. *journal of graphics, gpu, and game tools*, 10(3):37–49, 2005.

[BK10a] Karim Bouyarmane and Abderrahmane Kheddar. Static multi-contact inverse problem for multiple humanoid robots and manipulated objects. In *2010 10th IEEE-RAS International Conference on Humanoid Robots*, pages 8–13. IEEE, 2010.

[BK10b] Karim Bouyarmane and Abderrahmane Kheddar. Static multi-contact inverse problem for multiple humanoid robots and manipulated objects. In *IEEE-RAS International Conference on Humanoid Robots*, pages 8–13, Nashville, TN, 6-8 December 2010.

[BK11a] Karim Bouyarmane and Abderrahmane Kheddar. Multi-contact stances planning for multiple agents. In *IEEE International Conference on Robotics and Automation*, Shanghai, China, 9-13 May 2011.

[BK11b] Karim Bouyarmane and Abderrahmane Kheddar. Using a multi-objective controller to synthesize simulated humanoid robot motion with changing contact configurations. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, San Fransico, CA, 25-30 September 2011.

[BK12a] Karim Bouyarmane and Abderrahmane Kheddar. Humanoid robot locomotion and manipulation step planning. *Advanced Robotics*, 26, 2012.

[BK12b] Karim Bouyarmane and Abderrahmane Kheddar. On the dynamics modeling of free-floating-base articulated mechanisms and applications to humanoid whole-body dynamics and control. In *IEEE-RAS International Conference on Humanoid Robots*, 2012.

[BL08] Tim Bretl and Sanjay Lall. Testing static equilibrium for legged robots. *IEEE Transactions on Robotics*, 24:794–807, August 2008.

[BMAS14] Nicolas Boumal, Bamdev Mishra, P.-A. Absil, and Rodolphe Sepulchre. Manopt, a matlab toolbox for optimization on manifolds. *Journal of Machine Learning Research*, 15:1455–1459, 2014.

[BMGS84] A Balestrino, De MARIA G, and L Sciavicco. Robust control of robotic manipulators. In *9th World Congress of the IFAC*, 1984.

[BNW06] Richard H Byrd, Jorge Nocedal, and Richard A Waltz. Knitro: An integrated package for nonlinear optimization. In *Large-scale nonlinear optimization*, pages 35–59. Springer, 2006.

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[BV12] Joydeep Biswas and Manuela M. Veloso. Depth camera based indoor mobile robot localization and navigation. In *IEEE International Conference on Robotics and Automation*, pages 1697–1702, 2012.

[BVK+13] Stanislas Brossette, Joris Vaillant, François Keith, Adrien Escande, and Abderrahmane Kheddar. Point-Cloud Multi-Contact Planning for Humanoids: Preliminary Results. In *CISRAM: Cybarnetics and Intelligent Systems Robotics, Automation and Mechatronics*, volume 1, Manila & Pico de Loro Beach, Philippines, November 2013.

[BW07] Stephen P Boyd and Ben Wegbreit. Fast computation of optimal contact forces. *IEEE Transactions on Robotics*, 23(6):1117–1132, 2007.

[CA08] Nicolas Courty and Elise Arnaud. Inverse kinematics using sequential monte carlo methods. In *International Conference on Articulated Motion and Deformable Objects*, pages 1–10. Springer, 2008.

[CF03] Choong Ming Chin and Roger Fletcher. On the global convergence of an slp-filter algorithm that takes eqp steps. *Mathematical Programming*, 96(1):161–177, 2003.

[Che07] Joel Chestnutt. *Navigation planning for legged robots*. ProQuest, 2007.

[CKNK03] Joel Chestnutt, James Kuffner, Koichi Nishiwaki, and Satoshi Kagami. Planning biped navigation strategies in complex environments. 2003.

[CSL02] Juan Cortes, Thierry Siméon, and Jean-Paul Laumond. A random loop generator for planning the motions of closed kinematic chains using prm methods. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 2, pages 2141–2146. IEEE, 2002.

[CTS+09] Joel Chestnutt, Yutaka Takaoka, Keisuke Suga, Koichi Nishiwaki, James Kuffner, and Satoshi Kagami. Biped navigation in rough environments using on-board sensing. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, (IROS'09)*, pages 3543–3548, Piscataway, NJ, USA, 2009. IEEE Press.

[Dev07] Frédéric Devernay. C/c++ minpack. http://devernay.free.fr/hacks/cminpack/, 2007.

[DVT14] Hongkai Dai, Andrés Valenzuela, and Russ Tedrake. Whole-body motion planning with centroidal dynamics and full kinematics. In *2014 IEEE-RAS International Conference on Humanoid Robots*, pages 295–302. IEEE, 2014.

[EBK16] Adrien Escande, Stanislas Brossette, and Abderrahmane Kheddar. Parametrization of catmull-clark subdivision surfaces for posture generation. In *ICRA: International Conference on Robotics and Automation*, Stockholm, Sweden, February 2016.

[EK09a] Adrien Escande and Abderrahmane Kheddar. Contact planning for acyclic motion with tasks constraints. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 435–440, St. Louis, USA, October 11-15 2009.

[EK09b] Adrien Escande and Abderrahmane Kheddar. Planning contact supports for acyclic motion with task constraints and experiment on hrp-2. In *IFAC 9th International Symposium on Robot Control (SYROCO'09)*, pages 259–264, Gifu, Japan, September 9-12 2009.

[EKM06] Adrien Escande, Abderrahmane Kheddar, and Sylvain Miossec. Planning support contact-points for humanoid robots and experiments on HRP-2. In *IEEE/RSJ International Conference on Robots and Intelligent Systems*, pages 2974–2979, Beijing, China, 9-15 October 2006.

[EKM13] Adrien Escande, Abderrahmane Kheddar, and Sylvain Miossec. Planning contact points for humanoid robots. *Robotics and Autonomous Systems*, 61(5):428 – 442, 2013.

[EKMG08] Adrien Escande, Abderrahmane Kheddar, Sylvain Miossec, and Sylvain Garsault. Planning support contact-points for acyclic motions and experiments on HRP-2. In *International Symposium on Experimental Robotics*, Athens, Greece, 14-17 July 2008.

[EMBK14] Adrien Escande, Sylvain Miossec, Mehdi Benallegue, and Abderrahmane Kheddar. A strictly convex hull for computing proximity distances with continuous gradients. *Robotics, IEEE Transactions on*, 30(3):666–678, June 2014.

[EMK07] Adrien Escande, Sylvain Miossec, and Abderrahmane Kheddar. Continuous gradient proximity distance for humanoids free-collision optimized-postures. In *IEEE-RAS Conference on Humanoid Robots*, Pittsburg, Pennsylvania, November 29 - December 1 2007.

[EMW10] Adrien Escande, Nicolas Mansard, and Pierre-Brice Wieber. Fast resolution of hierarchized inverse kinematics with inequality constraints. In *IEEE International Conference on Robotics and Automation*, pages 3733 – 3738, Anchorage, USA, 3-7 May 2010.

[Fea07] Roy Featherstone. *Rigid Body Dynamics Algorithms*. Springer, 2007.

[FL00] Roger Fletcher and Sven Leyffer. Nonlinear programming without a penalty function. *Mathematical Programming*, 91:239–269, 2000.

[Fle06] R. Fletcher. A new low rank quasi-{N}ewton update scheme for nonlinear programming. (August):275–293, 2006.

[Fle10] Roger Fletcher. The sequential quadratic programming method. In *Nonlinear optimization*, pages 165–214. Springer, 2010.

[FSEK08] Toréa Foissotte, Olivier Stasse, Adrien Escande, and Abderrahmane Kheddar. A next-best-view algorithm for autonomous 3d object modeling by a humanoid robot. In *IEEE-RAS International Conference on Humanoid Robots*, 2008.

[Gab82] Daniel Gabay. Minimizing a differentiable function over a differential manifold. *Journal of Optimization Theory and Applications*, 37(2):177–219, 1982.

[GHM+86] P. E.Philip E. Gill, Sven J. Hammarling, Walter Murray, Michael A. Saunders, and Margaret H. Wright. User's guide for lssol (version 1.0): a fortran package for constrained linear least-squares and convex quadratic programming. Technical Report 86-1, Standford University, Standord, California 94305, January 1986.

[GJK88] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal of Robotics and Automation*, 4(2):193–203, 1988.

[GMS02] Philip E. Gill, Walter Murray, and Michael A. Saunders. Snopt: An sqp algorithm for large-scale constrained optimization, 2002.

[GMS05] Philip E Gill, Walter Murray, and Michael A Saunders. Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM review*, 47(1):99–131, 2005.

[Gra98] F Sebastian Grassia. Practical parameterization of rotations using the exponential map. 3:1–13, 1998.

[GVL96] G. Golub and C. Van Loan. *Matrix computations*. John Hopkins University Press, 3rd edition, 1996.

[HBL05] Kris Hauser, Tim Bretl, and Jean-Claude Latombe. Non-gaited humanoid locomotion planning. In *IEEE-RAS International Conference on Humanoid Robots*, pages 7–12, December 5-7 2005.

[HBL+08] K. Hauser, T. Bretl, J.-C. Latombe, K. Harada, and B. Wilcox. Motion planning for legged robots on varied terrain. In *Intl. J. of Robotics Research 27(11-12):1325-1349*, 2008.

[HKG08] John Hollerbach, Wisama Khalil, and Maxime Gautier. Model identification. In *Springer Handbook of Robotics*, chapter 14, pages 321–344. Springer, 2008.

[HRE+08] Chris Hecker, Bernd Raabe, Ryan W Enslow, John DeWeese, Jordan Maynard, and Kees van Prooijen. Real-time motion retargeting to highly varied user-created morphologies. *ACM Transactions on Graphics (TOG)*, 27(3):27, 2008.

[HS80] W. Hock and K. Schittkowski. Test examples for nonlinear programming codes. *Journal of Optimization Theory and Applications*, 30(1):127–129, 1980.

[HWFS11] Christoph Hertzberg, René Wagner, Udo Frese, and Lutz Schröder. Integrating Generic Sensor Fusion Algorithms with Sound State Representations through Encapsulation of Manifolds. (3), July 2011.

[KKW08] Daniel Kubus, Torsten Kröger, and Friedrich M Wahl. On-line estimation of inertial parameters using a recursive total least-squares approach. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 3845–3852. IEEE, 2008.

[KNK+05] James Kuffner, Koichi Nishiwaki, Satoshi Kagami, Masayuki Inaba, and Hirochika Inoue. Motion planning for humanoid robots. In *Robotics Research. The Eleventh International Symposium*, pages 365–374. Springer, 2005.

[KRN08] J Zico Kolter, Mike P Rodgers, and Andrew Y Ng. A control architecture for quadruped locomotion over rough terrain. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 811–818. IEEE, 2008.

[LAHB12] C. Lutz, F. Atmanspacher, A. Hornung, and M. Bennewitz. Nao walking down a ramp autonomously. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5169–5170, Oct. 2012.

[LHM00] C-P Lu, Gregory D Hager, and Eric Mjolsness. Fast and globally convergent pose estimation from video images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(6):610–622, 2000.

[LME+12] Mingxing Liu, Alain Micaelli, Paul Evrard, Adrien Escande, and Claude Andriot. Task-driven posture optimization for virtual characters. In *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation (SCA)*, pages 155–164, 2012.

[LRF11] Sébastien Lengagne, Nacim Ramdani, and Philippe Fraisse. Planning and fast replanning safe motions for humanoid robots. *IEEE Transactions on Robotics*, 27(6):1095–1106, 2011.

[Lue72] David G Luenberger. The gradient projection method along geodesics. *Management Science*, 18(11):620–631, 1972.

[LVYK13] Sébastien Lengagne, Joris Vaillant, Eiichi Yoshida, and Abderrahmane Kheddar. Generation of whole-body optimal dynamic multi-contact motions. *The International Journal of Robotics Research*, 32(9-10):1104–1119, 2013.

[LYK99] Steven M LaValle, Jeffery H Yakey, and Lydia E Kavraki. A probabilistic roadmap approach for systems with closed kinematic chains. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 3, pages 1671–1676. IEEE, 1999.

[LZT97] Craig Lawrence, Jian L. Zhou, and André L. Tits. User's guide for CFSQP version 2.5: A C code for solving (large scale) constrained nonlinear (minimax) optimization problems, generating iterates satisfying all inequality constraints, 1997.

[MCY14] Thomas Moulard, Bejamin Chrétien, and Eichii Yoshida. Software tools for nonlinear optimization - modern solvers and toolboxes for robotics -. *Journal of the Robotics Society of Japan*, 32(6):536–541, 2014.

[Mer09] Xavier Merlhiot. *Une contribution algorithmique aux outils de simulation mécanique interactive pour la maquette numérique industrielle*. PhD thesis, Paris VI University, 2009.

[MHB12] Daniel Maier, Armin Hornung, and Maren Bennewitz. Real-time navigation in 3D environments based on depth camera data. In *Humanoids'12: 12th IEEE-RAS International Conference on Humanoid Robots*, Osaka, Japan, November 2012.

[MLBY13] Thomas Moulard, F Lamiraux, K Bouyarmane, and E Yoshida. Roboptim: an optimization framework for robotics. In *Japan Society for Mechanical Engineers: Robotics and Mechatronics Conference*, 2013.

[MNN+10] Giorgio Metta, Lorenzo Natale, Francesco Nori, Giulio Sandini, David Vernon, Luciano Fadiga, Claes Von Hofsten, Kerstin Rosander, Manuel Lopes, José Santos-Victor, et al. The icub humanoid robot: An open-systems platform for research in cognitive development. *Neural Networks*, 23(8):1125–1134, 2010.

[MTP12] Igor Mordatch, Emanuel Todorov, and Zoran Popović. Discovery of complex behaviors through contact-invariant optimization. *ACM Trans. Graph.*, 31(4):1–8, July 2012.

[NH86] Yoshihiko Nakamura and Hideo Hanafusa. Inverse kinematic solutions with singularity robustness for robot manipulator control. *Journal of dynamic systems, measurement, and control*, 108(3):163–171, 1986.

[NL08] A. Nakhaei and F. Lamiraux. Motion planning for humanoid robots in environments modeled by vision. In *Humanoids'08: 8th IEEE-RAS International Conference on Humanoid Robots*, pages 197–204, Dec 2008.

[NW06] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, 2nd edition, 2006.

[NY93] Jorge Nocedal and Ya-xiang Yuan. Analysis of a self-scaling quasi-newton method. *Mathematical Programming*, 61(1-3):19–37, 1993.

[OGHB11] S. Osswald, A. Gorog, A. Hornung, and M. Bennewitz. Autonomous climbing of spiral staircases with humanoids. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 4844–4849, Sept 2011.

[OHB12] Stefan Oßwald, Armin Hornung, and Maren Bennewitz. Improved proposals for highly accurate localization using range and vision data. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1809–1814, Vilamoura, Portugal, October 2012.

[Osb69] Mike R Osborne. On shooting methods for boundary value problems. *Journal of mathematical analysis and applications*, 27(2):417–433, 1969.

[Pec08] Alexandre N Pechev. Inverse kinematics without matrix inversion. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 2005–2012. IEEE, 2008.

[PG91] CM Pham and Maxime Gautier. Essential parameters of robots. In *Decision and Control, 1991., Proceedings of the 30th IEEE Conference on*, pages 2769–2774. IEEE, 1991.

[PVBP08] J. Poppinga, N. Vaskevicius, A. Birk, and K. Pathak. Fast plane detection and polygonalization in noisy 3d range images. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3378–3383, Sept 2008.

[RC11] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *IEEE International Conference on Robotics and Automation*, Shanghai, China, May 9-13 2011.

[RMBO08] Elon Rimon, Richard Mason, Joel W Burdick, and Yizhar Or. A general stance stability test based on stratified morse theory with application to quasi-static locomotion planning. *IEEE Transactions on Robotics*, 24(3):626–641, 2008.

[SDH+14] J. Schulman, Y. Duan, J. Ho, a. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel. Motion planning with sequential convex optimization and convex collision checking. *Int. J. Rob. Res.*, 2014.

[SH98] Andrew Stuart and Anthony R Humphries. *Dynamical systems and numerical analysis*, volume 2. Cambridge University Press, 1998.

[Sic90] Bruno Siciliano. A closed-loop inverse kinematic scheme for on-line joint-based robot control. *Robotica*, 8(03):231–243, 1990.

[SK05] Luis Sentis and Oussama Khatib. Synthesis of whole-body behaviors through hierarchical control of behavioral primitives. *International Journal of Humanoid Robotics*, 2(04):505–518, 2005.

[SK10] Luis Sentis and Oussama Khatib. Compliant control of multicontact and center-of-mass behaviors in humanoid robots. *IEEE Trans. Robot.*, 26(3):483–501, June 2010.

[SLL+07] Olivier Stasse, Diane Larlus, Baptiste Lagarde, Adrien Escande, Francois Saidi, Abderrahmane Kheddar, Kazuhito Yokoi, and Frederic Jurie. Towards autonomous object reconstruction for visual search by the humanoid robot hrp-2. In *IEEE RAS/RSJ Conference on Humanoids Robots*, page Oral presentation, Pittsburg, USA, 30 Nov. - 2 Dec. 2007.

[Smi13] Steven Thomas Smith. Geometric optimization methods for adaptive filtering. *arXiv preprint arXiv:1305.1886*, 2013.

[Sta98] Jos Stam. Exact evaluation of catmull-clark subdivision surfaces at arbitrary parameter values. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, pages 395–404, New York, NY, USA, 1998. ACM.

[TBEN16] Silvio Traversaro, Stanislas Brossette, Adrien Escande, and Francesco Nori. Identification of fully physical consistent inertial parameters using optimization on manifolds. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, To appear in IROS 2016.

[TGB00] Deepak Tolani, Ambarish Goswami, and Norman I Badler. Real-time inverse kinematics techniques for anthropomorphic limbs. *Graphical models*, 62(5):353–388, 2000.

[TNAG] United Kingdom The Numerical Algorithms Group(NAG), Oxford. The nag library. www.nag.com.

[VKA+14] Joris Vaillant, Abderrahmane Kheddar, Hervé Audren, Francois Keith, Stanislas Brossette, Kenji Kaneko, Mitsuharu Morisawa, Eiichi Yoshida, and Fumio Kanehiro. Vertical Ladder Climbing by HRP-2 Humanoid Robot. In *IEEE-RAS Int. Conf. Humanoid Robot.*, pages 671–676, Madrid, Spain, 2014.

[VKA+16] Joris Vaillant, Abderrahmane Kheddar, Hervé Audren, François Keith, Stanislas Brossette, Adrien Escande, Karim Bouyarmane, Kenji Kaneko, Mitsuharu Morisawa, Pierre Gergondet, Eiichi Yoshida, Suuji Kajita, and Fumio Kanehiro. Multi-contact vertical ladder climbing with an HRP-2 humanoid. *Autonomous Robots*, 40(3):561–580, 2016.

[Wam86] Charles W Wampler. Manipulator inverse kinematic solutions based on vector formulations and damped least-squares methods. *IEEE Transactions on Systems, Man, and Cybernetics*, 16(1):93–101, 1986.

[WB06] Andreas Wächter and Lorentz Biegleri. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Math. Program.*, 106(1):25–57, 2006.

[WBBPG06] Pierre-Brice Wieber, Florence Billet, Laurence Boissieux, and Roger Pissard-Gibollet. The humans toolbox, a homogenous framework for motion capture, analysis and simulation. In *International Symposium on the 3D Analysis of Human Movement*, 2006.

[WCD+10] MA. Whitty, S. Cossell, KS. Dang, J. Guivant, and J. Katupitiya. Autonomous navigation using a real-time 3d point cloud. In *ACRA'10: Australasian Conference on Robotics & Automation*, Brisbane, 1 - 3 December 2010.

[YOMO94] Koji Yoshida, Koichi Osuka, Hirokazu Mayeda, and Toshiro Ono. When is the set of base parameter values physically impossible? In *Intelligent Robots and Systems' 94.'Advanced Robotic Systems and the Real World', IROS'94. Proceedings of the IEEE/RSJ/GI International Conference on*, volume 1, pages 335–342. IEEE, 1994.

[ZB94] Jianmin Zhao and Norman I. Badler. Inverse kinematics positioning using nonlinear programming for highly articulated figures. *ACM Trans. Graph.*, 13(4):313–336, oct 1994.

[ZLH+13] Y. Zhang, J. Luo, K. Hauser, R. Ellenberg, P. Oh, H.A. Park, M. Paldhe, and C.S.G. Lee. Motion planning of ladder climbing for humanoid robots. In *IEEE Conf. on Technologies for Practical Robot Applications*, 2013.

Academic contributions

- Silvio Traversaro, Stanislas Brossette, Adrien Escande, and Francesco Nori. Identification of fully physical consistent inertial parameters using optimization on manifolds. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, To appear in IROS 2016

- Adrien Escande, Stanislas Brossette, and Abderrahmane Kheddar. Parametrization of catmull-clark subdivision surfaces for posture generation. In *ICRA: International Conference on Robotics and Automation*, Stockholm, Sweden, February 2016

- Joris Vaillant, Abderrahmane Kheddar, Hervé Audren, François Keith, Stanislas Brossette, Adrien Escande, Karim Bouyarmane, Kenji Kaneko, Mitsuharu Morisawa, Pierre Gergondet, Eiichi Yoshida, Suuji Kajita, and Fumio Kanehiro. Multi-contact vertical ladder climbing with an HRP-2 humanoid. *Autonomous Robots*, 40(3):561–580, 2016

- Stanislas Brossette, Adrien Escande, Grégoire Duchemin, Benjamin Chretien, and Abderrahmane Kheddar. Humanoid posture generation on non-euclidean manifolds. In *Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on*, pages 352–358, Nov 2015

- Stanislas Brossette, Adrien Escande, Joris Vaillant, François Keith, Thomas Moulard, and Abderrahmane Kheddar. Integration of non-inclusive contacts in posture generation. In *IROS'14: International Conference on Robots and Intelligent Systems*, Chicago, United States, September 2014

- Joris Vaillant, Abderrahmane Kheddar, Hervé Audren, Francois Keith, Stanislas Brossette, Kenji Kaneko, Mitsuharu Morisawa, Eiichi Yoshida, and Fumio Kanehiro. Vertical Ladder Climbing by HRP-2 Humanoid Robot. In *IEEE-RAS Int. Conf. Humanoid Robot.*, pages 671–676, Madrid, Spain, 2014

- Stanislas Brossette, Joris Vaillant, François Keith, Adrien Escande, and Abderrahmane Kheddar. Point-Cloud Multi-Contact Planning for Humanoids: Preliminary Results. In

CISRAM: Cybarnetics and Intelligent Systems Robotics, Automation and Mechatronics, volume 1, Manila & Pico de Loro Beach, Philippines, November 2013

Appendix A

Numerical Optimization: Introduction

A.1 Introduction

In modern science, optimization has an important place. Mechanical engineers optimize the shape of structural parts. Investors optimize the profit of a portfolio while minimizing the risks of loss. Chemists optimize the efficiency and speed of reactions. When it comes to robotics, optimization is widely used. From the design of a robot to its actuation. Any positioning of a robot requires the computation of the articular parameters of each joint of the robot, finding such parameters might be possible by using analytical methods for simple robots, but for robots as complex as humanoid robots, it is not possible. Most often, an optimization process is used.

The goal of an optimization algorithm is to find an optimal solution x^* to a problem. Optimal in the sense that the solution is an optimum of a given objective function f. And solution of a problem in the sense that it satisfies a set of m constraints $\{c_i,\ i \in [1,m]\}$. Both the constraints and the objective function are defined on the variable space S, which is the space in which the variable x lives and in which we search a solution x^* to our problem.

In order to present the principles of optimization, in this chapter, we will always consider that the variable space is $S = \mathbb{R}^n$. We first consider unconstrained problems. That type of problem will not be used in the rest of our dissertation, therefore, we will just present them shortly. We will then focus on constrained optimization problems and the numerous methods used in their resolution. We will particularly detail one specific constrained problem resolution algorithm that is the Sequential Quadratic Program (SQP). The extension of those methods to solving problems on more complex variable spaces that are the non-Euclidean manifolds is the topic of Chapter 4.

This Appendix does not pretend to give a fully detailed overview of numerical optimization methods, it gives the reader a quick overview of some key principles. For more detailed

information, we invite the reader to refer to the excellent books that we took inspiration from to write this chapter [NW06, BGLA03, BV04].

A.2 Unconstrained Optimization

An unconstrained optimization problem consists of an objective function to minimize, without any constraint. This problem, denoted $\mathcal{P}$, can be formulated as follows:

$$\min_{x \in \mathbb{R}^n} f(x) \tag{A.1}$$

The classical approach to solve $\mathcal{P}$ is to use an iterative algorithm, starting from an initial guess x_0, and converging toward the solution x^*. A very basic optimization scheme is presented in Algorithm 7.

Algorithm 7 Basic optimization scheme

```
Starting from an initial guess x_0
while Convergence condition not met do
    Compute descent direction p_k
    Update x_{k+1} ← x_k + p_k
end while
```

The objective function is not necessarily completely known, in the sense that we cannot always have an explicit formula. Often, the function f is computed by another program that is able to compute $f(x)$, $\nabla_x f(x)$ and sometimes $\nabla_{xx} f(x)$ for a given value of x. In order to have an efficient algorithm, we need to avoid any unnecessary computation of $f(x)$ and its derivatives. We denote the values taken by x along the iterations as $x_0, x_1, x_2, \ldots x_i$. And $f(x_i)$ is denoted f_i.

Since our knowledge of the objective function is only partial, without further assumptions, it is not possible to guarantee that a point x^* is a global solution:

$$x^* \text{ is a global solution of } \mathcal{P} \text{ on } S \equiv \forall x \in S, f(x^*) \leq f(x) \tag{A.2}$$

Though, we can find a local solution x^* of $\mathcal{P}$ (a local minimizer of f): such that there exist a neighborhood N of x^* where $\forall x \in N$, $f(x^*) \leq f(x)$. Or even a strict local minimizer of f: such that there exist a neighborhood N of x^* where $\forall x \in N$, $f(x^*) < f(x)$. That is the kind of solution that we are looking for and that our algorithms should find.

Under the assumption that the objective function is smooth and sufficiently continuous ($\mathcal{C}^2$), we have the following sufficient conditions for the optimality of x^*:

Theorem A.2.1 *If $\nabla^2 f$ is continuous in an open neighborhood of x^*, $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive definite. Then x^* is a strict local minimizer of f.*

The same definition can be given for a local minimizer with $\nabla^2 f(x^*)$ positive semi-definite.

There are several ways to choose a descent direction p_k from an iterate x_k. The most obvious one is probably the steepest descent direction $-\nabla f_k$. This method provides the direction along which f decreases most rapidly and only requires the evaluation of the first derivative of f, but that method can become extremely slow on complicated problems. Another popular approach is the Newton method, in which the objective function is approximated to the second order

$$f(x_k + p) = f_k + p^T \nabla f_k + p^T \nabla^2 f_k p \tag{A.3}$$

Then the chosen descent direction is the optimum of that approximated function, the Newton direction.

$$p_k^N = -(\nabla^2 f_k)^{-1} \nabla f_k \tag{A.4}$$

The choice of this descent direction implies that the $\nabla^2 f_k$ is positive definite, in which case an adaptation of the definition of p_k is required. Or an approximation B_k of $\nabla^2 f_k$ that guarantees definite positiveness can be used.

$$p_k = -B_k^{-1} \nabla f_k \tag{A.5}$$

p_k is then added to x_k to compute $x_{k+1} \leftarrow x_k + p_k$ and the process is repeated until convergence is reached.

Many methods and refinements exist to solve this problem and some are presented in [NW06]. Our main focus here is on methods to solve constrained problem as they arise in robotics.

A.3 Constrained Optimization

Solving a constrained optimization problem consists in minimizing a cost function while satisfying a set of constraints.

A general formulation for such problems (that we denote $\mathcal{P}$) is:

$$\mathcal{P} \equiv \begin{cases} \min\limits_{x \in \mathbb{R}^n} f(x) \\ \text{s.t.} \begin{cases} c_i(x) = 0, \ \forall i \in E \\ c_i(x) \geq 0, \ \forall i \in I \end{cases} \end{cases} \tag{A.6}$$

Where f and c_i are real-valued functions on a subset of $\mathbb{R}^n$. f is the objective function, and the c_i are the constraint functions. I and E are sets of index such that $c_i,\ i \in E$ are the equality constraints, and $c_i,\ i \in I$ are the inequality constraints.

We define the feasible set Ω that contains all the feasible points (points satisfying the constraints) of $\mathcal{P}$.

$$\Omega = \{x \in \mathbb{R}^n : \ \forall i \in E,\ c_i = 0,\ \forall i \in I,\ c_i \geq 0\} \tag{A.7}$$

The formulation A.6 can then be rewritten as:

$$\mathcal{P} \equiv \min_{x \in \Omega} f(x) \tag{A.8}$$

In the general case, the problem A.8 has multiple local solutions. And finding the global minimizer of f in Ω is a difficult problem that we do not treat in this thesis. Our goal is to find a local minimizer x^* of f in Ω.

A.3.1 Optimality conditions

First-Order Optimality Conditions The First Order Optimality Condition (Karush-Kuhn-Tucker Condition) is a necessary condition verified by all solutions of problem A.6.

We introduce the Lagrangian function of A.6:

$$\mathcal{L}(x, \lambda) = f(x) + \sum_{i \in E \cup I} \lambda_i c_i(x) \tag{A.9}$$

Any given constraint c_i is said to be active at x if $c_i(x) = 0$. In particular, for any feasible point x, all equality constraints $c_i,\ i \in E$ are active. An inequality constraint $c_i,\ i \in I$ is inactive if $c_i(x) > 0$ and active if $c_i(x) = 0$

Definition The active set $A(x)$ at a feasible point x is the set of all the indexes of active constraint.

$$A(x) = E \cup \{i \in I : c_i(x) = 0\} \tag{A.10}$$

Most optimization algorithms make the assumption that at the solution, the constraints satisfy the following Linear Independence Constraints' Qualification (LICQ) definition:

Definition Given a point x and an active set $A(x)$, we say that the LICQ holds if the set of active constraint gradient $\{\nabla c_i(x),\ i \in A(x)\}$ is linearly independent

In general, if LICQ holds, none of the active constraints gradients can be zero.

Theorem A.3.1 *First-Order Necessary Conditions*
Suppose that x^* is a local solution of A.6, that the functions f and c_i are continuously differentiable, and that the LICQ holds at x^*. Then there is a Lagrange multiplier vector λ^*, with components λ_i^*, $i \in E \cup I$, such that the following conditions are satisfied at (x^*, λ^*)

$$\begin{array}{ll} \nabla_x \mathcal{L}(x^*, \lambda^*) = 0 & , \\ c_i(x^*) = 0 & , \forall i \in E \\ c_i(x^*) \geq 0 & , \forall i \in I \\ \lambda_i^* \geq 0 & , \forall i \in I \\ \lambda_i^* c_i(x^*) = 0 & , \forall i \in E \cup I \end{array} \tag{A.11}$$

In many cases, the main goal of an optimization algorithm is to find a point that satisfies the KKT conditions.

Second-Order Optimality Conditions

Definition Given a feasible point x, and the active set $A(x)$, the set of linearized feasible directions $F(x)$ is:

$$F(x) = \left\{ d \;\middle|\; \begin{array}{ll} d^T \nabla c_i(x) = 0 & , \forall i \in E \\ d^T \nabla c_i(x) \geq 0 & , \forall i \in A(x) \cap I \end{array} \right\} \tag{A.12}$$

And at a KKT solution (x^*, λ^*), the critical cone $C(x^*, \lambda^*)$ is:

$$C(x^*, \lambda^*) = \{w \in F(x^*) | \nabla c_i(x^*)^T w = 0, \forall i \in A(x^*) \cap I \text{ with } \lambda_i^* > 0\} \tag{A.13}$$

Once the KKT condition is reached for a point (x^*, λ^*), for a direction w of $F(x^*)$ for which $w^T \nabla f(x^*) = 0$, it is impossible to tell from first derivative information only, if an increment in this direction will have a positive effect on the problem resolution. Thus it is possible that the KKT constraint is not enough to determine if a point is a solution or just a stationary point.

The Second-Order Sufficient Conditions allow to discriminate local solutions from stationary points:

Theorem A.3.2 *Suppose that for some feasible point $x^* \in \mathbb{R}^n$ there is a Lagrange multiplier vector λ^* such that the KKT conditions are satisfied. Suppose also that*

$$w^T \nabla^2_{xx} \mathcal{L}(x^*, \lambda^*) w > 0, \ \forall w \in C(x^*, \lambda^*), \ w \neq 0 \tag{A.14}$$

Then x^* is a strict local solution for A.6

In particular, this condition is always verified if $\nabla^2_{xx} \mathcal{L}(x^*, \lambda^*)$ is positive definite.

A.4 Resolution of a NonLinear Constrained Optimization Problem

In the previous section, we presented some theoretical tools to decide if a point is a solution to a nonlinear constrained optimization problem based on the values of the first and second order derivatives of its functions. Here we present some methods for actually solving such problems. There exist many different algorithms to solve a nonlinear constrained optimization problem. But all have in common to be iterative processes that follow the model of Algorithm 8.

Algorithm 8 Basic scheme for solving a nonlinear constrained optimization problem

```
Starting from an initial guess x_0
repeat
    Compute an increment z_k such that x_k + z_k is closer to the solution
    Update x_{k+1} ← x_k + z_k
until Optimality conditions are met with a precision ε
```

The resolutions algorithms differ mostly by the method chosen to generate a satisfactory increment z. We will list here some of the most popular methods:

The penalty method combines the cost and constraints function in a penalty function that, as its name indicates, penalizes the violation of constraints, without completely proscribing it. The penalty function can be written as follows, using the notation $[y]^- = \max\{0, -y\}$:

$$p(\mu, x) = f(x) + \mu \sum_{i \in E} |c_i(x)| + \mu \sum_{i \in I} [c_i(x)]^- \tag{A.15}$$

With this approach, an unconstrained optimization approach can be used. The minimum of $p(\mu, x)$ varies with the penalty parameter μ. By increasing μ to ∞, we penalize the violation of the constraints with increasing severity until reaching a solution x^*.

The interior point method generates steps by solving a relaxed constrained problem where slack variables are introduced to relax inequality constraints. The problem solved is the following:

$$\begin{aligned} &\min_{x,s} f(x) \\ &\text{subject to } \begin{cases} c_i = 0,\ i \in E \\ c_i - s_i = 0,\ i \in I \\ s_i \geq 0,\ i \in I \end{cases} \end{aligned} \tag{A.16}$$

One way to solve this problem is to consider the following associated barrier problem:

$$\begin{aligned} &\min_{x,s} f(x) - \mu \sum_{i \in I} \log(s_i) \\ &\text{subject to } \begin{cases} c_i = 0,\ i \in E \\ c_i - s_i = 0,\ i \in I \end{cases} \end{aligned} \tag{A.17}$$

and to find its approximate solutions for a sequence of positive barrier parameters $\{\mu_k\}$ that converges to zero. The minimization of the barrier term $-\mu \sum_{i \in I} \log(s_i)$ prevents the terms of s from becoming too close to zero. The solution for a given $\{\mu_k\}$ is obtained by applying Newton's method to the nonlinear system that represents the KKT conditions for the barrier problem. The solution is reached once the optimality conditions of A.16 are met.

The Sequential Quadratic Programming (SQP) is an approach in which the Newton's method is used to solve the nonlinear problem raised by the KKT conditions of problem A.6. Basically, it comes down to finding the increment z^* that solves the Quadratic Programming (QP) associated with the problem A.18, at each iterate (x_k, λ_k).

$$\begin{aligned} \min_{z \in \mathbb{R}^n} \quad & \tfrac{1}{2} z^T \nabla^2_{xx} \mathcal{L}(x_k, \lambda_k) z + \nabla f(x_k)^T z \\ \text{subject to} \quad & \nabla c_i(x_k)^T z + c_i(x_k) = 0,\ i \in E \\ & \nabla c_i(x_k)^T z + c_i(x_k) \geq 0,\ i \in I \end{aligned} \tag{A.18}$$

Given z^* the solution of A.18, it can be tempting to directly take the next iterate as $x_{k+1} = x_k + z^*$. But using that method as is can be problematic as the length of the iterate needs not be bounded. Thus it is possible to generate a very large step that satisfies the QP. The QP only approximates the original problem locally. So taking a too big step can get us further from the solution than we previously were. To palliate to that issue, mainly two methods are used:

- The line-search method: The solution z of A.18 is viewed as a direction and we search a parameter $\alpha \in [0;1]$ so that the next iterate $x_k + \alpha z$ is optimal for the original problem.
- The trust-region method adds a set of bound constraints to the QP A.18. So that the length of the step is limited by the trust-region. The size ρ of the trust region is modified along the iterations based on the estimated quality of the QP approximation. In that case, the QP becomes:

$$\begin{array}{ll} \min\limits_{z\in\mathbb{R}^n} & \frac{1}{2} z^T \nabla^2_{xx} \mathcal{L}(x_k, \lambda_k) z + \nabla f(x_k)^T z \\ \text{subject to} & \nabla c_i(x_k)^T z + c_i(x_k) = 0,\ i \in E \\ & \nabla c_i(x_k)^T z + c_i(x_k) \geq 0,\ i \in I \\ & |z| < \rho \end{array} \tag{A.19}$$

A.5 Sequential Quadratic Programming

The Sequential Quadratic Programming (SQP) method is one of the most effective methods to solve small and large scale nonlinear constrained optimization problems. Together with the interior point method, they are currently considered to be the most powerful algorithms for large-scale nonlinear programming. The SQP methods show their strength when solving problems with nonlinearities in the constraints [NW06].

A.5.1 Principle

As we stated before, the main idea behind the SQP approach it to use the Newton's method to solve the nonlinear problem raised by the KKT conditions. For simplicity, we consider a problem without inequality constraints. We denote C the vector of equality constraints. We denote z_x and z_λ the increments on x and λ respectively.

$$\begin{array}{l} x_{k+1} = x_k + z_x \\ \lambda_{k+1} = \lambda_k + z_\lambda \end{array} \tag{A.20}$$

The problem is the following:

$$\begin{array}{l} \min\limits_{x\in\mathbb{R}^n} f(x) \\ \text{subject to } C(x) = 0 \end{array} \tag{A.21}$$

Its KKT conditions are:

$$\begin{cases} \nabla_x \mathcal{L}(x^*, \lambda^*) = 0 \\ C(x^*) = 0 \end{cases} \tag{A.22}$$

We denote with a subscript y_k the value of the quantity y evaluated at point (x_k, λ_k).

The first order linearization of A.22 gives:

$$\begin{cases} \nabla^2_{xx} \mathcal{L}_k z_x + \nabla^2_{x\lambda} \mathcal{L}_k z_\lambda + \nabla_x \mathcal{L}_k = 0 \\ \nabla_x C_k z_x + C_k = 0 \end{cases} \tag{A.23}$$

which is equivalent to:

$$\begin{cases} \nabla^2_{xx} \mathcal{L}_k z_x + \nabla_x C_k (\lambda_k + z_\lambda) = -\nabla_x f_k \\ \nabla_x C_k z_x = -C_k \end{cases} \tag{A.24}$$

Or in matrix form:

$$\begin{pmatrix} \nabla^2_{xx} \mathcal{L}_k & \nabla_x C_k \\ \nabla_x C_k & 0 \end{pmatrix} \begin{pmatrix} z_x \\ \lambda_{k+1} \end{pmatrix} = \begin{pmatrix} -\nabla_x f_k \\ -C_k \end{pmatrix} \tag{A.25}$$

Solving this problem is equivalent to solving a QP problem of the following form:

$$\begin{aligned} \min_{z_x} \quad & f_k + \nabla_x f_k^T z_x + \tfrac{1}{2} z_x^T \nabla^2_{xx} \mathcal{L}_k z_x \\ \text{s.t.} \quad & \nabla_x C_k^T z_x + C_k = 0 \end{aligned} \tag{A.26}$$

The solution of this problem satisfies the following matrix equality:

$$\begin{pmatrix} \nabla^2_{xx} \mathcal{L}_k & -\nabla_x C_k \\ \nabla_x C_k & 0 \end{pmatrix} \begin{pmatrix} z_x \\ l_k \end{pmatrix} = \begin{pmatrix} -\nabla_x f_k \\ -C_k \end{pmatrix} \tag{A.27}$$

Therefore, the problem raised by the linearization of the KKT conditions to the first order can be solved as a QP problem. Denoting the solution of the QP problem (z_x, l_k), the solution to A.23 is given by

$$\begin{pmatrix} z_x \\ \lambda_{k+1} \end{pmatrix} \leftarrow \begin{pmatrix} z_x \\ -l_k \end{pmatrix} \tag{A.28}$$

This development can easily be extended to treat the case of problems constrained with inequality constraints:

$$\begin{aligned} &\min_{x \in \mathbb{R}^n} f(x) \\ &\text{s.t.} \begin{cases} c_i(x) = 0, \ \forall i \in E \\ c_i(x) \geq 0, \ \forall i \in I \end{cases} \end{aligned} \tag{A.29}$$

By solving the following Inequality-constrained Quadratic Problem (IQP) system at each iteration:

$$\begin{aligned} &\min_{z_x} f_k + \nabla_x f_k^T z_x + \frac{1}{2} z_x^T \nabla_{xx}^2 \mathcal{L}_k z_x \\ &\text{s.t.} \begin{cases} \nabla_x c_{ik} z_x + c_{ik} = 0, \ \forall i \in E \\ \nabla_x c_{ik} z_x + c_{ik} \geq 0, \ \forall i \in I \end{cases} \end{aligned} \tag{A.30}$$

The main difficulty in solving that QP A.30 comes from finding the optimal active set A_k. It is a necessary but costly operation, especially when starting from an initial active set A_0. The process is mostly based on a try and guess approach guided by heuristics (that may vary with different resolution algorithms).

To find the optimal active set, the QP starts from an initial active set A_0 ignore the non-active constraints and try to solve the QP with the remaining active constraints. At a step k of the resolution of the IQP, all the constraints in the active set A are treated as equality constraints and the others are simply ignored, the QP to solve becomes:

$$\text{EQP} \equiv \begin{cases} \min\limits_{z_x} & f_k + \nabla_x f_k^T z_x + \frac{1}{2} z_x^T \nabla_{xx}^2 \mathcal{L}_k z_x \\ \text{s.t.} & \nabla_x c_{ik} z_x + c_{ik} = 0, \ \forall i \in A_k \end{cases} \tag{A.31}$$

This EQP is solved, and its solution is checked against the constraints that were previously ignored. If some of them are violated then the solution is not satisfactory and the active set is updated by adding one or several of the violated constraints to it. This operation is repeated until there are no more constraints to be added. Once all the constraints have been added as active constraints to the EQP, the problem may be over-constrained. To detect that, the Lagrange multipliers are computed and if some of them do not agree with the KKT conditions of the IQP A.30, then one of them is removed from the active set and the resulting EQP is solved, and the process continues until a satisfactory solution is found. The heuristics used to choose which constraints to add or remove from the active set and to avoid cycling between active sets are out of the scope of this dissertation.

Once the SQP gets close to the solution, the set of active constraints should not change much from one iterate x_k to the next one. It is often useful to initialize the IQP's active set with the optimal active set of the previous iterate: $A_0(x_{k+1}) \leftarrow A^*(x_k)$. Then the $(k+1)^{th}$ QP starts from an active set close to optimal and can be solved much faster. That method is called the *warm-start*.

The SQP approach to solving the problem A.6 as presented above leans on the assumption that at each step, the QP generated is feasible. If it is not, a restoration phase is entered and finds a feasible point, see Section A.5.3.

Solving the QP problem A.30 gives a solution step to a linearized problem, which in general is not the solution to the original nonlinear problem. Thus, taking that step without further verification can be dangerous and lead to bad iterates. To cope with that issue, several methods exist. In the next few sections, we present the globalization methods that are the Line Search and Trust Region approaches. They are used alongside methods to choose whether to accept or reject a step like the merit function and filters in order to monitor the length and direction of the step with regards to the nonlinear problem. And finally, we discuss some Hessian approximation methods.

A.5.2 Globalization methods

The globalization phase of an SQP algorithm is meant to modify the steps generated by the QP resolution in order to improve the global convergence of the algorithm, either by multiplying them with a coefficient α in the Line-Search approach, or by bounding the norm of the step found in the QP by adding to the QP a boundary constraint on z that reflects our trust in the quality of the approximated problem.

The Line-Search Strategy

Given a step z_k generated by the resolution of the QP, the goal of the line-search is to find a coefficient α (typically in $[0,1]$) such that $x_{k+1} = x_k + \alpha z_k$ makes the violation of the constraints and the value of the cost function decrease sufficiently. To decide whether those decreases are sufficient, a merit function and a criterion are used. A merit function is a function of α that transcribes the value of the objective function penalized by the violation of the constraints. For a problem described by the system A.29, with a penalty parameter μ, a typical expression of the merit function M is:

$$M(\alpha) = f(x_k + \alpha z_k) + \mu \sum_{i \in E} |c_i(x_k + \alpha z_k)| + \mu \sum_{i \in I} \max(0, -c_i(x_k + \alpha z_k)) \tag{A.32}$$

Although finding the optimal value of α to minimize the 1-dimensional function M is possible, it can prove costly, and is not necessary, because the only thing that we want is an optimal solution to the problem A.29, thus the exact value of each iterate does not matter much. Instead, we use a criterion to decide when the value of α gives a satisfactory decrease. Various methods and criterion can be used to find a satisfactory value of α. For example, the Wolfe line-search:

Let μ_1 and μ_2 be 2 real numbers such that: $0 < \mu_1 < \mu_2 < 1$. A sufficient decrease of the merit function can be depicted by the condition $M(\alpha) \leq M(0) + \mu_1 \alpha M'(0)$. A

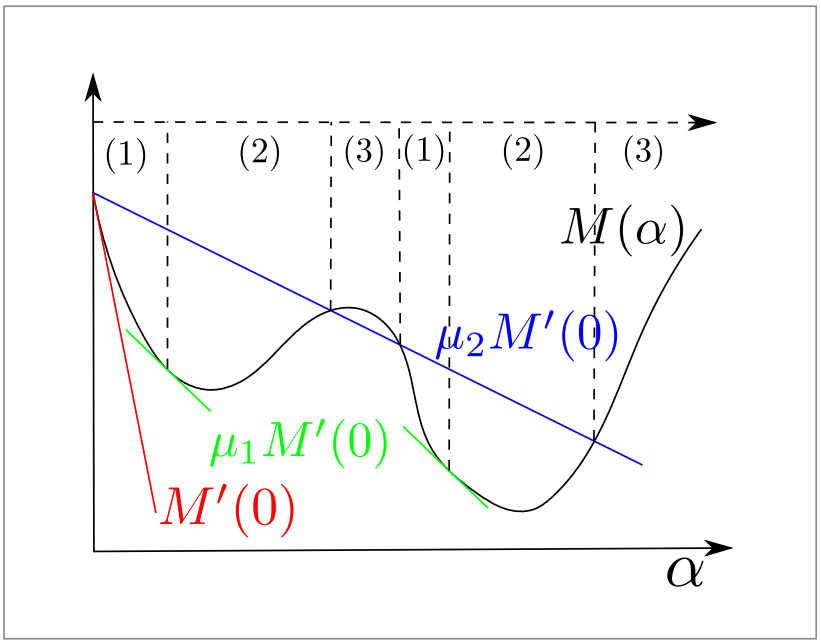


Figure A.1 Illustration of the choice made by the Wolfe line-search for values of α

sufficient increase of its derivative by $M'(\alpha) \geq \mu_2 M'(0)$. The Wolfe Line-Search proceeds by narrowing iteratively an interval $[\alpha_L,\ \alpha_R]$ by trying a value α in this interval against some tests and based on their results, decide to narrow it by the right or the left and repeat that operation with the new interval until a satisfying value is found. Note that the Wolfe line-search is guaranteed to terminate if M is $\mathcal{C}^1$ and bounded below.

The Wolfe Line-Search can be schematically described by algorithm: 9 and illustrated by figure A.1.

Algorithm 9 The Wolfe line-search

Init $[\alpha_L, \alpha_R] \leftarrow [0\ , 1]$
loopChoose $\alpha \in [\alpha_L, \alpha_R]$
Evaluate $M(\alpha)$ and $M'(\alpha)$

(1) $M(\alpha) \leq M(0) + \mu_1 \alpha M'(0)$ and $M'(\alpha) < \mu_2 M'(0)$: α is too small, $\alpha_L \leftarrow a$

(2) $M(\alpha) \leq M(0) + \mu_1 \alpha M'(0)$ and $M'(\alpha) \geq \mu_2 M'(0)$: Terminate, **return** α

(3) $M(\alpha) > M(0) + \mu_1 \alpha M'(0)$: α is too big, $\alpha_R \leftarrow \alpha$

end loop

The Wolfe line-search is popular, but it requires the computation of the derivative of the merit function, which can be prohibitively expensive in some cases, including in posture generation for robotics problems, where the derivatives are usually expensive. Other line-

search methods exist that do not require the derivative of M for $\alpha \neq 0$. For example the Goldstein and Price, or the Armijo methods.

The filter method

The decision to accept on not a step generated by the QP can be made using a Filter approach instead of the Wolfe line search, as introduced by Fletcher in [FL00]. The goal of that method is to minimize the cost function $f(x)$ and the constraint violation $h(x)$ separately. A step is accepted if it improves at least one of the function or the constraint violation.

$$h(x) = \sum_{k \in E} |c_k(x)| + \sum_{k \in I} \max(0, -c_k(x)) \tag{A.33}$$

Denoting $h_i = h(x_i)$ and $f_i = f(x_i)$, we define the notion of domination as follows:

Definition

$$p_i \text{ dominates } p_j \Leftrightarrow \begin{cases} f_i < f_j \\ h_i < h_j \end{cases} \tag{A.34}$$

The filter maintains a list of pairs $p_i = \{f_i, h_i\}$ such that no pair dominates any other pair. When a new point x is submitted to the filter for acceptance, the values $f(x)$ and $h(x)$ are computed and the pair $p = \{f(x), h(x)\}$ is compared to every pair p_i. If there is at least one pair p_i in the filter that dominates p, the point is refused. Otherwise, no pair of the filter dominates p, then p is accepted and added to the filter. Once p is added to the filter, any pair in the filter that is dominated by p is removed from it to ensure to keep a list of pairs non-dominated by each other in the filter. In order to ensure global convergence, this method requires several refinements as explained in [FL00]. First, it needs to avoid accepting pairs that are excessively close to each other. For that, the domination criterion is modified with a sloping envelope, as proposed in Chin [CF03]. The sloping envelope definition takes user defined parameters β and γ in $[0,\ 1]$.

Definition

$$p_i \text{ dominates } p_j \Leftrightarrow \begin{cases} f_i < f_j - \gamma h \\ h_i < \beta h_j \end{cases} \tag{A.35}$$

Another basic necessary improvement is to set an upper bound to the constraint violation, to avoid having the algorithm generate points that minimize the cost function while completely violating the constraints.

A representation of the filter method is given in figure A.2

The filter method is convenient because it does not require the same complicated updates as can be found in penalty based methods where the penalty parameter needs to be updated

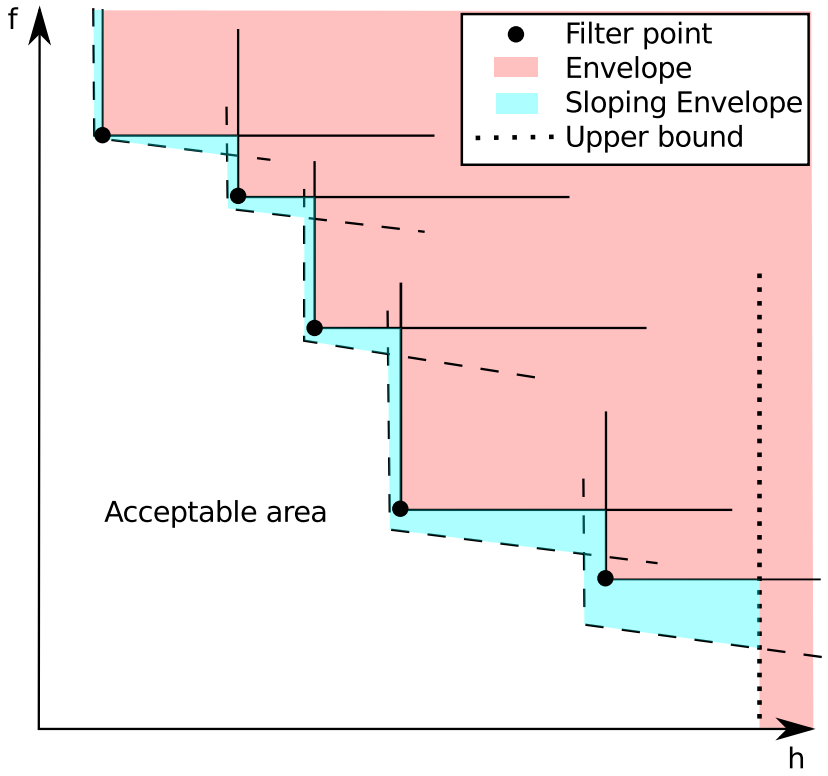


Figure A.2 Representation of a filter with sloping envelope

regularly. Also, with the filter method, no derivative computation is required. That method is generally more permissive than merit-based method in the sense that it refuses fewer points.

The filter method, as well as the merit function based methods, can be used in line-search and trust-region algorithms.

The Trust-Region Strategy

The Trust Region strategy is based on the idea that at any iterate x_k the quadratic model made of the problem and fed to the QP solver can only be trusted to be relevant is a finite region around the iterate. That region is called the trust region and its size is usually governed by a single positive parameter ρ_k that can evolve along the optimization process. This translates in a modified QP to solve at each iteration. A boundary constraint describing the trust region is added to the problem. At each step, the modified problem writes as:

$$\begin{array}{ll} \min\limits_{z\in\mathbb{R}^n} & \frac{1}{2}z^T\nabla^2_{xx}Hz+\sum_{i\in\mathcal{U}}\nabla c_i(x_k)^T z \\ \text{subject to} & \nabla c_i(x_k)^T z+c_i(x_k)=0,\ i\in\mathcal{F} \\ & \nabla c_i(x_k)^T z+c_i(x_k)\geq 0,\ i\in I \\ & |z|<\rho_k \end{array} \tag{A.36}$$

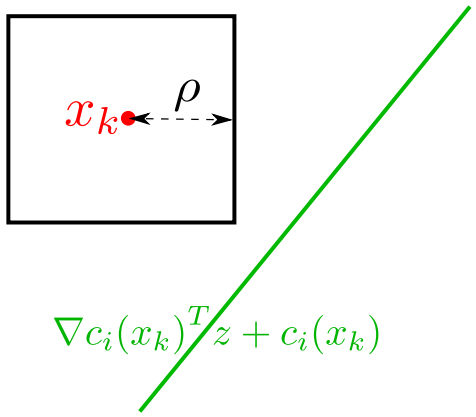


Figure A.3 Constraint incompatibility generated by trust region approach

Once a solution z of A.36 is found, the potential next iterate $x_{k+1} = x_k + z$ is checked against the acceptance criterion (should it be a merit function or a filter or other). If it is accepted, that means that our current model is good, we move to the next step with $x_{k+1} \leftarrow x_k + z$, and the size of the trust region can be augmented based on some heuristics. Otherwise, the iterate is refused, the model is less good than expected, the size of the trust region is reduced based on some heuristics (e.g. $\rho_{k+1} \leftarrow \rho_k/2$) and the problem A.36 is solved again with the new value of ρ_{k+1}. This is repeated until a satisfying point is found, or until an infeasible problem is found.

With this approach, a problem that often arises is the generation of infeasible problems. The reduction of the size of the trust region can lead to an incompatible set of constraints. For example, with a single constraint, as illustrated in figure A.3. The linearisation of the constraint is represented by the green line, and the trust region by the black rectangle. If ρ is too small, there is no intersection between the linearization of the constraint and the trust region. The QP is then infeasible. An obvious solution for that would be to increase the size of the trust region, but that goes against the core idea of the trust region strategy and it would harm the convergence properties of the algorithm. A more appropriate solution is to enter a restoration phase when that event occurs. The purpose of a restoration phase is to reduce the infeasibility of a problem by relaxing infeasible constraints without regards for the cost function. The restoration phase is exited once a feasible point is found, and the course of the SQP is continued from that point.

A.5.3 Restoration phase

As we stated previously, a restoration phase is entered when an infeasible problem is found by the trust region strategy. The goal of the restoration is to find a solution to the following problem:

$$\text{Find } x \in \mathbb{R}^n \text{ such that } \begin{cases} c_i(x) = 0,\ i \in E \\ c_i(x) \geq 0,\ i \in I \\ |z| < \rho_k \end{cases} \tag{A.37}$$

The restoration phase proceeds in a similar way to the original SQP algorithm described earlier. At each step, an approximated QP is solved, then its result is compared to an acceptance criterion, if it is accepted, the trust region can be increased and the algorithm continues. Otherwise, the trust region is reduced and the problem is solved again. The acceptance criterion must be tailored for the restoration problem, meaning that it is based on the values of the cost function of the restoration problem and of its constraints. The big difference comes from the way the quadratic problem is constructed during this phase.

During a restoration phase step, the first action is to estimate the sets $\mathcal{F}$ and $\mathcal{I}$ of feasible and infeasible constraints. This is done by solving the linearization of problem (A.37) as presented in (A.38). Each equality constraint is cut in 2 inequality constraints.

$$\text{Find } z \in \mathbb{R}^n \text{ such that } \begin{cases} \nabla c_i(x_k)^T z + c_i(x_k) \geq 0,\ i \in E \\ -\nabla c_i(x_k)^T z - c_i(x_k) \geq 0,\ i \in E \\ \nabla c_i(x_k)^T z + c_i(x_k) \geq 0,\ i \in I \\ |z| < \rho_k \end{cases} \tag{A.38}$$

The resolution of that FP gives the lists $\mathcal{F}$ and $\mathcal{I}$. If the list of infeasible constraints is empty, the problem is feasible, so the restoration phase is exited to return to the main SQP algorithm. The restoration QP to solve is built based on those lists. The infeasible constraints are removed from the constraint set and the expression of their violation is added to the cost function, while the feasible constraints remain in the constraint list. This results in a problem like A.39 to solve. (Note that the indexes of the constraints are modified to account for the duplication of the equality constraints)

$$\begin{aligned} \min_{z \in \mathbb{R}^n} \quad & \tfrac{1}{2} z^T H z - \sum_{i \in \mathcal{I}} \nabla c_i(x_k)^T z \\ \text{subject to} \quad & \nabla c_i(x_k)^T z + c_i(x_k) \geq 0,\ i \in \mathcal{F} \\ & |z| < \rho_k \end{aligned} \tag{A.39}$$

This problem is solved by a QP solver, its result is checked against an acceptance criterion, based on that, the trust region is enlarged or reduced, and we go back to solving the FP.

In the restoration phase, special care must be taken in the computation of the matrix H, that represents the Hessian of the Lagrangian of a problem that changes at each iteration. One possible approach is to compute it as the sum of the Hessians of all the individual constraints, and those can be computed exactly or with a quasi-newton approximation.

A.5.4 Quasi-Newton Approximation

In the SPQ algorithm, it is necessary to have access to the Hessian of the Lagrangian $\nabla^2_{xx}\mathcal{L}$ to be able to devise the QP subproblem to solve. Sometimes the exact Hessian of the problem is not positive definite. Also, it is often difficult or computationally expensive to compute an exact Hessian of the Lagrangian. Since we are following an iterative process, it is not necessary to have an exact knowledge of the Hessian and using an approximation of it is usually enough. Also, an approximate Hessian is in most cases less expensive to compute than the exact one.

The idea behind computing an approximate Hessian is that, starting from an initial approximate Hessian B_0, we compute at each iteration an update to the approximate Hessian based on the values and first order derivatives of the Lagrangian. This update aims at capturing some curvature information about the Hessian by evaluating the evolution of the gradient along the latest step. At step k, the Hessian update is a function of s_k and y_k:

$$s_k = x_{k+1} - x_k, \quad y_k = \nabla_x \mathcal{L}(x_{k+1}, \lambda_{k+1}) - \nabla_x \mathcal{L}(x_k, \lambda_{k+1}) \tag{A.40}$$

The two most famous Hessian update strategies are called the BFGS (Broyden–Fletcher–Goldfarb–Shanno) and the SR1(Symmetric Rank 1) updates. BFGS is a rank 2 update while SR1 is rank 1.

The most basic formulas for the BFGS update is the following:

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{s_k^T y_k} \tag{A.41}$$

Note that using a BFGS update requires that s_k and y_k satisfy the curvature condition: $s_k^T y_k > 0$. If that condition does not hold, then the value of y_k is modified, which gives rise to the Damped BFGS update, which guarantees to keep B_k definite positive:

$$\theta_k = \begin{cases} 1 & \text{if } s_k^T y_k \geq 0.2 s_k^T B_k s_k \\ \frac{0.8 s_k^T B_k s_k}{s_k^T B_k s_k - s_k^T y_k} & \text{if } s_k^T y_k \geq 0.2 s_k^T B_k s_k \end{cases}$$
$$r_k = \theta_k y_k + (1-\theta_k) B_k s_k \tag{A.42}$$
$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{r_k r_k^T}{s_k^T r_k}$$

The SR1 update is computed with the following formula:

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{(y_k - B_k s_k)^T s_k} \tag{A.43}$$

Both those formulas proved to be efficient in some cases. It is not yet clear in which cases one is better than the other.

A.6 Conclusion

This section gives a general introduction to nonlinear constrained optimization without regards for specificities of robotics problem or formulations on manifolds, which are topics considered in the core of this thesis.

Appendix B

Manifolds Descriptions

This appendix chapter comes as a complement for Chapter 4 in which we explicit the elements necessary for the description of several elementary manifolds.

B.1 The Real Space manifold

The Realspace manifold of dimension n is denoted $\mathbb{R}^n$. Since $\mathbb{R}^n$ is a Euclidean manifold, the operations that we use on it are straightforward.

Table B.1 Description of the $\mathbb{R}^n$ manifold

$\mathcal{M}$	$\mathbb{R}^n$	$\zeta(x,y)$	$\mathbf{y}-\mathbf{x}$
$\mathbb{E}$	$\mathbb{R}^n$	$\frac{\partial \zeta_x}{\partial y}(x)$	$\mathbb{I}_n$
$T_x\mathcal{M}$	$\mathbb{R}^n$	$\mathcal{T}(x,\mathbf{z},\mathbf{v})$	$\mathbf{v}$
$T_x\mathbb{E}$	$\mathbb{R}^n$	$\pi_{\mathcal{M}}(\mathbf{x})$	$\mathbf{x}$
$\phi_x(\mathbf{z})$	$\mathbf{x}+\mathbf{z}$	$\pi_{T_x\mathcal{M}}(\mathbf{z})$	$\mathbf{z}$
$\partial\phi_x(0)$	$\mathbb{I}_n$	lim	$\|\mathbf{v}\| \leq \infty$

B.2 The 3D Rotation manifold: Matrix representation

The 3D rotation manifold is denoted $SO(3)$.

An element x of $SO(3)$ is represented by $\mathbf{x} = \psi(x)$ in $\mathbb{R}^{3\times 3}$ by:

$$x \in SO(3),\ \mathbf{x} = \begin{bmatrix} x_{00} & x_{01} & x_{02} \\ x_{10} & x_{11} & x_{12} \\ x_{20} & x_{21} & x_{22} \end{bmatrix} \tag{B.1}$$

We recall the operators

$$\widehat{\cdot}: \begin{bmatrix} \omega_0 \\ \omega_1 \\ \omega_2 \end{bmatrix} \to \begin{bmatrix} 0 & -\omega_2 & \omega_1 \\ \omega_2 & 0 & -\omega_0 \\ -\omega_1 & \omega_0 & 0 \end{bmatrix} \tag{B.2}$$

And its inverse:

$$\check{\cdot}: \begin{bmatrix} x_{00} & x_{01} & x_{02} \\ x_{10} & x_{11} & x_{12} \\ x_{20} & x_{21} & x_{22} \end{bmatrix} \to \begin{bmatrix} x_{21} \\ x_{02} \\ x_{10} \end{bmatrix} \tag{B.3}$$

The exponential map is known as the Rodrigues formula:

$$\forall \mathbf{v} \in \mathbb{R}^3,\ \exp(\mathbf{v}) = \mathbb{I}_3 + \frac{\sin\|\mathbf{v}\|}{\|\mathbf{v}\|}\hat{\mathbf{v}} + \frac{1-\cos\|\mathbf{v}\|}{\|\mathbf{v}\|^2}\hat{\mathbf{v}}^2 \tag{B.4}$$

Note that when $\|\mathbf{v}\|$ is small, we make the following replacements to avoid numerical instability. It is important to ensure the precision of the retractation near zero because, in an optimization process, many small steps are taken, especially when close to the solution.

$$\frac{\sin\|\mathbf{v}\|}{\|\mathbf{v}\|} = 1 - \frac{\|\mathbf{v}\|}{6} \tag{B.5}$$

$$\frac{1-\cos\|\mathbf{v}\|}{\|\mathbf{v}\|^2} = 0.5 - \frac{\|\mathbf{v}\|}{24} \tag{B.6}$$

And the logarithm is computed as follows (see [Mer09]):

$$\forall R \in \psi(SO(3)),\ f(R) = \begin{cases} 0 & \text{if } Tr(R) = 3 \\ \frac{\arccos\left(\frac{Tr(R)-1}{2}\right)}{2\sin\left(\arccos\left(\frac{Tr(R)-1}{2}\right)\right)}\left(R - R^T\right) & \text{otherwise} \end{cases} \tag{B.7}$$

$$\log(R) = \widetilde{f(R)}$$

We consider a point $x \in \mathcal{M}$ and its representation matrix with the following storing order:

$$\mathbf{x} = \psi(x) = \begin{pmatrix} x_0 & x_3 & x_6 \\ x_1 & x_4 & x_7 \\ x_2 & x_5 & x_8 \end{pmatrix} = \begin{pmatrix} x_{00} & x_{01} & x_{02} \\ x_{10} & x_{11} & x_{12} \\ x_{20} & x_{21} & x_{22} \end{pmatrix} \tag{B.8}$$

The derivative of retractation operation writes as:

$$\frac{\partial \varphi_x}{\partial \mathbf{z}}(0) = \begin{bmatrix} 0 & -x_6 & x_3 \\ 0 & -x_7 & x_4 \\ 0 & -x_8 & x_5 \\ x_6 & 0 & -x_0 \\ x_7 & 0 & -x_1 \\ x_8 & 0 & -x_2 \\ -x_3 & x_0 & 0 \\ -x_4 & x_1 & 0 \\ -x_5 & x_2 & 0 \end{bmatrix} \tag{B.9}$$

And the derivation of the logarithm operation writes as:

$$\mathbf{v} = \begin{pmatrix} \frac{x_{21}-x_{12}}{2} \\ \frac{x_{02}-x_{20}}{2} \\ \frac{x_{10}-x_{01}}{2} \end{pmatrix} \tag{B.10}$$

$$f = \frac{\arccos\left(\frac{Tr(\mathbf{x})-1}{2}\right)}{2\sin\left(\arccos\left(\frac{Tr(\mathbf{x})-1}{2}\right)\right)} \tag{B.11}$$

$$df = \frac{(Tr(\mathbf{x})-1)f-1}{2\left(1-\left(\frac{Tr(\mathbf{x})-1}{2}\right)^2\right)} \tag{B.12}$$

$$\frac{\partial \zeta_x}{\partial y}(x) = \begin{bmatrix} & 0 & 0 & 0 & & f & 0 & -f & \\ df.\mathbf{v} & 0 & -f & 0 & df.\mathbf{v} & 0 & f & 0 & df.\mathbf{v} \\ & f & 0 & -f & & 0 & 0 & 0 & \end{bmatrix} \tag{B.13}$$

Table B.2 Description of the $SO(3)$ manifold with matrix representation

$\mathcal{M}$	$SO(3)$
$\mathbb{E}$	$\mathbb{R}^{3\times 3}$
$T_x\mathcal{M}$	$\mathbb{R}^3$
$T_x\mathbb{E}$	$\mathbb{R}^3$
$\psi(x)=\mathbf{x}$	$\begin{bmatrix} x_{00} & x_{01} & x_{02} \\ x_{10} & x_{11} & x_{12} \\ x_{20} & x_{21} & x_{22} \end{bmatrix}$
$\phi_x(\mathbf{z})$	$\mathbf{x}\exp(\mathbf{z})$

$\partial\phi_x(0)$	see Equation B.9
$\zeta(x,y)$	$\log(\mathbf{x}^T\mathbf{y})$
$\frac{\partial\zeta_x}{\partial y}(x)$	see Equation B.10
$\mathcal{T}(x,\mathbf{z},\mathbf{v})$	$\mathbf{v}$
$\pi_{\mathcal{M}}(\mathbf{x})$	Q from QR decomposition of $\mathbf{x}$
$\pi_{T_x\mathcal{M}}(\mathbf{z})$	$\mathbf{z}$
lim	$\|\mathbf{v}\|\leq\pi$

B.3 The 3D Rotation manifold with quaternion representation

The 3D rotation manifold is denoted $SO(3)$.

An element x of $SO(3)$ is represented by $\mathbf{q}=\psi(x)$ in $\mathbb{R}^4$ by:

$$x\in SO(3),\ \mathbf{q}=\begin{bmatrix} q_w \\ q_x \\ q_y \\ q_z \end{bmatrix}=\begin{bmatrix} q_w \\ \mathbf{q_{vec}} \end{bmatrix} \tag{B.14}$$

The exponential map is:

$$\exp:\left|\begin{array}{ccc} \mathbb{R}^3 & \to & \mathbb{R}^4 \\ \mathbf{z} & \mapsto & \begin{bmatrix} \cos\left(\frac{\|\mathbf{z}\|}{2}\right) \\ \sin\left(\frac{\|\mathbf{z}\|}{2}\right)\frac{\mathbf{z}}{\|\mathbf{z}\|} \end{bmatrix} \end{array}\right.$$

Note that when $\|\mathbf{v}\|$ is small, we make the following replacements to avoid numerical instability.

$$\exp(\mathbf{z})=\begin{bmatrix} 1-\frac{\|\mathbf{z}\|}{8}+\frac{\|\mathbf{z}\|^2}{384} \\ \left(0.5-\frac{\|\mathbf{z}\|}{48}+\frac{\|\mathbf{z}\|^2}{3840}\right)\mathbf{z} \end{bmatrix} \tag{B.15}$$

And the logarithm is:

$$\log : \left| \begin{array}{ccc} \mathbb{R}^4 & \to & \mathbb{R}^3 \\ q & \mapsto & \arctan\left(\frac{2\|\mathbf{q_{vec}}\|q_w}{q_w^2-\|\mathbf{q_{vec}}\|^2}\right)\frac{\mathbf{q_{vec}}}{\|\mathbf{q_{vec}}\|} \end{array} \right. \tag{B.16}$$

We consider a point $q \in \mathcal{M}$ and its representation quaternion with the following storing order:

$$\mathbf{q} = \psi(q) = \begin{pmatrix} q_w \\ q_x \\ q_y \\ q_z \end{pmatrix} \tag{B.17}$$

The derivative of retractation operation writes as:

$$\frac{\partial \varphi_q}{\partial \mathbf{z}}(0) = \frac{1}{2}\begin{bmatrix} -q_x & -q_y & -q_z \\ q_w & -q_z & q_y \\ q_z & q_w & -q_x \\ -q_y & q_x & q_w \end{bmatrix} \tag{B.18}$$

And the derivation of the logarithm operation writes as:

$$f = \frac{\arctan\left(\frac{2\|\mathbf{q_{vec}}\|q_w}{q_w^2-\|\mathbf{q_{vec}}\|^2}\right)}{n} \tag{B.19}$$

$$g = \frac{2q_w - f}{\|\mathbf{q_{vec}}\|^2} \tag{B.20}$$

$$\frac{\partial \zeta_x}{\partial y}(\mathbf{x}) = \begin{bmatrix} \begin{pmatrix} \\ -2\mathbf{q_{vec}} \\ \\ \end{pmatrix} & \begin{pmatrix} \\ f\mathbb{I}_3 + g\mathbf{q_{vec}}\mathbf{q_{vec}}^T \\ \\ \end{pmatrix} \end{bmatrix} \tag{B.21}$$

Table B.3 Description of the $\mathbf{SO(3)}$ manifold with quaternion representation

$\mathcal{M}$	$SO(3)$
$\mathbb{E}$	$\mathbb{R}^4$
$T_x\mathcal{M}$	$\mathbb{R}^3$
$T_x\mathbb{E}$	$\mathbb{R}^3$
$\phi_x(\mathbf{z})$	$\mathbf{x}\exp(\mathbf{z})$
$\partial\phi_x(0)$	see Appendix B.18

$\zeta(x,y)$	$\log(\mathbf{x}^{-1}\mathbf{y})$
$\frac{\partial\zeta_x}{\partial y}(x)$	see Appendix B.19
$\mathcal{T}(x,\mathbf{z},\mathbf{v})$	$\mathbf{v}$
$\pi_{\mathcal{M}}(\mathbf{x})$	$\frac{\mathbf{x}}{\|\mathbf{x}\|}$
$\pi_{T_x\mathcal{M}}(\mathbf{z})$	$\mathbf{z}$
lim	$\|\mathbf{v}\| \leq \pi$

B.4 The Unit Sphere manifold

An element x of $S2$ is represented by $\mathbf{x} = \psi(x)$ in $\mathbb{R}^3$ by:

$$x \in S2,\ \mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} \tag{B.22}$$

With this manifold, the tangent space at x is the tangent plane to the unit-sphere at x. Thus, $T_x\mathcal{M}$ it is a 2-dimensional space, and its representation space is $\mathbb{R}^3$.

For the retractation we simply use a normalized sum:

$$\phi_x(\mathbf{z}) = \frac{\mathbf{x}+\mathbf{z}}{\|\mathbf{x}+\mathbf{z}\|} \tag{B.23}$$

We define a distance operation on $S2$ as:

$$\text{dist}(x,y) = 1 - \mathbf{x}\cdot\mathbf{y} \tag{B.24}$$

And the projection on the tangent space:

$$\pi_{T_x\mathcal{M}}(\mathbf{z}) = \mathbf{z} - (\mathbf{x}\cdot\mathbf{z})\mathbf{x} \tag{B.25}$$

The pseudo-logarithm operation is the following:

$$\zeta_x(y) = \text{dist}(x,y)\frac{\pi_{T_x\mathcal{M}}(\mathbf{z})}{\|\pi_{T_x\mathcal{M}}(\mathbf{z})\|} \tag{B.26}$$

The vector transport operation of vector $\mathbf{v}$ from $T_x\mathcal{M}$ to $T_y\mathcal{M}$ with $y = \phi_x(\mathbf{v})$, corresponds to rotating $\mathbf{v}$ by the rotation that transforms x into y:

$$\mathbf{y} = \phi_x(\mathbf{z}) \tag{B.27}$$

$$R = \mathbb{I}_3 + \widehat{\mathbf{x} \wedge \mathbf{y}} + \frac{\widehat{\mathbf{x} \wedge \mathbf{y}}^2}{1 + \mathbf{x} \cdot \mathbf{y}} \tag{B.28}$$

$$\mathcal{T}(x, \mathbf{z}, \mathbf{v}) = R\mathbf{v} \tag{B.29}$$

Table B.4 Description of the S2 manifold

$\mathcal{M}$	$S2$
$\mathbb{E}$	$\mathbb{R}^3$
$T_x\mathcal{M}$	$\mathbb{R}^2$
$T_x\mathbb{E}$	$\mathbb{R}^3$
$\phi_x(\mathbf{z})$	$\frac{\mathbf{x}+\mathbf{z}}{\|\mathbf{x}+\mathbf{z}\|}$
$\partial\phi_x(0)$	$\mathbb{I}_3 - \mathbf{x} \cdot \mathbf{x}^T$

$\zeta(x, y)$	$\mathbf{y} - (\mathbf{x} \cdot \mathbf{y})\mathbf{x}$
$\frac{\partial \zeta_x(x)}{\partial y}$	$\mathbb{I}_3 - \mathbf{x} \cdot \mathbf{x}^T$
$\mathcal{T}(x, \mathbf{z}, \mathbf{v})$	$\mathbb{I}_3 + \mathbf{x} \wedge \widehat{\phi_x(\mathbf{z})} + \frac{\widehat{\mathbf{x} \wedge \phi_x(\mathbf{z})}^2}{1 + \mathbf{x} \cdot \phi_x(\mathbf{z})}$
$\pi_{\mathcal{M}}(\mathbf{x})$	$\frac{\mathbf{x}}{\|\mathbf{x}\|}$
$\pi_{T_x\mathcal{M}}(\mathbf{z})$	$\mathbf{z} - (\mathbf{x} \cdot \mathbf{z})\mathbf{x}$
lim	$\|\mathbf{v}\| \leq \infty$