On multi-contact dynamic motion using reduced models
Hervé Audren

To cite this version:

HAL Id: tel-01875801
https://tel.archives-ouvertes.fr/tel-01875801
Submitted on 17 Sep 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
THÈSE POUR OBTENIR LE GRADE DE DOCTEUR
DE L’UNIVERSITÉ DE MONTPELLIER

En Robotique
École doctorale I2S
Unité de recherche CNRS-AIST JRL UMI3218

On Multi-Contact Dynamic Motion Using Reduced Models

Présentée par Hervé AUDREN
Le 14 Novembre 2017
Sous la direction de Abderrahmane KHEDDAR

Devant le jury composé de

Philippe BIDAUD, Professeur, Université Paris 6-ONERA
Nicolas MANSARD, Directeur de Recherche, LAAS-CNRS
Yacine CHITOUR, Professeur, Université Paris Sud
Kris HAUSER, Associate Professor, Duke University
Fumio KANEHIRO, Senior Researcher, AIST
Luis SENTIS, Associate Professor, The University of Texas at Austin

Rapporteur
Rapporteur
Président du Jury
Examinateur
Examinateur
Examinateur
Acknowledgements

I would like to thank all members of JRL whose contributions were invaluable in preparing this dissertation: our discussions were instrumental in shaping this thesis. Thank you Adrien, Benjamin, Damien, Hoa, Joris, Pierre, Stanislas, Thomas, for welcoming me at the lab, and making my stay in Tsukuba an unforgettable one.

Special thanks have to go to my advisor, Prof. Kheddar for providing guidance in navigating the world of research, and always encouraging me to go further.

Thank you Mehdi, Rafael, Stéphane for sharing with me your research: it opened my eyes to new areas of research.

I would like to thank my family, my parents and siblings who made me who I am today, and provided unyielding support throughout all those years.

Finally, a special word goes to Amandine, for always being besides me and supporting me, even during the busiest times of this Ph.D.

Tsukuba, February 9, 2018

H. A.
Abstract

In the context of (multi)-legged robotics, equilibrium (static or dynamic) often called *stability* is of critical importance. Indeed, as such robots have a non-actuated floating base they can fall. Notice that this is also the case of robot arms ported on wheeled robots. To avoid falling, we must be able to tell apart stable from non-stable motion. This thesis approaches stability from a reduced model point-of-view: that is to say, our main interest is the Center of Mass, as for now, it is commonly used to compute predictive trajectory for dynamic motions. We show how to compute stability regions for this reduced model, at first based on purely static stability. Although geometrical in nature, we show how they depend on the admissible contact forces.

Then, we show that taking into account robustness, in the sense of acceleration (or contact forces) uncertainties, we can transform the usual two dimensional stability region into a three dimensional one. To compute this shape, we introduce novel recursive algorithms.

We show how we can apply computer graphics techniques for shape morphing in order to continuously deform the aforementioned regions. This allows us to approximate changes in the parameters of those shapes, but also to interpolate between shapes when such stability polyhedron are computed for two distinct contact sequence.

Finally, we exploit the effective decoupling offered by the explicit computation of the stability polyhedron to formulate a linear, minimal jerk model-predictive control problem. We also propose another linear MPC problem that exploits more of the available dynamics, but at an increased computational cost.

We then adopt a hierarchical approach, and use those CoM results as input to our whole-body controller. Results are obtained on real hardware and in simulation.

Key words: Legged robotics, Humanoid robotics, Stability
Résumé

Pour les robots marcheurs, c’est-à-dire bipèdes, quadrupèdes, hexapodes... la notion d’équilibre statique et dynamique (que l’on nommera abusivement stabilité) est primordiale. En effet, ces robots possèdent une base flottante sous-actionnée : il leur faut prendre appui sur l’environnement pour se mouvoir. Toutefois, cette caractéristique les rend vulnérables : ils peuvent tomber. C’est aussi les cas de bras robotique portés par une base mobile à roues. Il est donc indispensable de pouvoir différencier un mouvement stable d’un mouvement non-stable.

Dans cette thèse, la stabilité est considérée du point de vue d’un modèle réduit au Centre de Masse, ou Centre de Gravité, noté CoM, car il est communément utilisé pour calculer des trajectoires d’une commande prédictive. Nous montrons dans un premier temps comment calculer la zone de stabilité de ce modèle dans le cas statique. Bien que cette région soit un objet purement géométrique, nous montrons qu’elle dépend des forces de contact admissibles.

Ensuite, nous montrons qu’introduire la notion de robustesse, c’est-à-dire une marge d’incertitude sur les accélérations (ou les forces de contacts) transforme la forme plane du cas statique en un volume tridimensionnel. Afin de calculer cette forme, nous présentons de nouveaux algorithmes récursifs.

Nous appliquons ensuite des algorithmes provenant de l’infographie qui permettent de déformer continûment ces objets géométriques. Cette transformation nous permet d’avoir une approximation des changements dans les variables influençant ces formes.

Calculer le volume de stabilité explicitement nous permet de découpler les accélérations des positions du CoM, ce qui nous permet de formuler un problème de contrôle prédictif linéaire. Nous proposons aussi une autre formulation linéaire qui, au prix de calculs plus coûteux, permet d’exploiter pleinement la dynamique du robot.

Enfin, nous appliquons ces résultats dans une approche hiérarchique qui nous permet de générer des mouvements du corps complet du robot, aussi bien sur une véritable plateforme humanoïde qu’en simulation.

Mots clefs : Robotique Humanoïde, Stabilité
# Contents

Acknowledgements ........................................... i

Abstract (English/Français) ................................... iii

List of figures ............................................. xi

List of tables ............................................... xv

Introduction .................................................. 1

1 State of the Art ........................................... 5
   1.1 Using a stronger stability criterion ................... 7
   1.2 Going further than locally ............................. 9
   1.3 Handling the non-linearity ........................... 11

2 Preliminaries ............................................. 15
   2.1 Optimization ........................................ 15
   2.2 Posture Generation ................................... 17
   2.3 Task-based inverse dynamics ......................... 18

3 Static equilibrium and interpolation .................... 23
   3.1 Computation of the constrained stability polygon .... 24
      3.1.1 1D CoM - 2 aligned forces ....................... 24
      3.1.2 1D CoM - 2 non-aligned forces .................. 25
      3.1.3 2D CoM - 3 aligned forces ....................... 28
      3.1.4 2D case - n aligned contact points ............... 30
      3.1.5 Direct projection via vertex enumeration ......... 34
      3.1.6 Recursive projection ............................. 36
   3.2 Morphing stability polygons .......................... 39
      3.2.1 What is a morphing? ............................. 39
      3.2.2 Angle-based morphing ........................... 42
      3.2.3 EGI-based morphing ............................. 43
      3.2.4 Optimal matching morphing ....................... 44
# Contents

3.3 Linking force limitation and interpolation ........................................ 45
3.4 Integration ......................................................... 48
3.4.1 Task-based controller ........................................... 48
3.4.2 Limiting the CoM region during motion ............................. 50
3.4.3 Results of stairs climbing ....................................... 50
3.4.4 Combining tasks and constraints for force control ............... 52
3.4.5 Application to a multi-contact setting ........................... 54
3.5 Conclusion .......................................................... 56

4 Robust static equilibrium .................................................... 59
4.1 Robust static stability ............................................... 60
4.1.1 Problem formulation ............................................ 61
4.1.2 Formulating a stricter constraint ................................. 62
4.1.3 Discretizing the hypersphere .................................... 63
4.2 Direct projection ..................................................... 64
4.3 Computing the robust polyhedron ..................................... 65
4.3.1 Recursive projection ............................................. 65
4.3.2 An intersection of prisms ........................................ 70
4.4 Comparative results ................................................ 72
4.5 Incremental projection for testing .................................... 78
4.5.1 Why incremental projection? ................................... 78
4.5.2 Polyhedral case ................................................... 79
4.5.3 Prism case ........................................................ 79
4.5.4 Results ............................................................ 80
4.6 Extensions and discussions ........................................... 82
4.6.1 Morphing and change in robustness ............................. 82
4.6.2 Robust 2D and robust 3D ......................................... 83
4.6.3 Recent work ....................................................... 85
4.7 Integration ............................................................ 86
4.7.1 Case study in multi-contact posture generation ............... 86
4.7.2 Multi-contact Control .......................................... 88
4.8 Conclusion ........................................................... 92

5 CoM Predictive Control ..................................................... 95
5.1 Model Predictive Control ............................................. 96
5.1.1 General principle ............................................... 96
5.1.2 Linear discrete-time MPC ....................................... 97
5.2 Full contact forces .................................................. 99
5.2.1 Previous Formalism ............................................ 99
5.2.2 Arbitrary reference frames ..................................... 102
# List of Figures

3.1 Schema for two non-aligned contacts .............................. 26
3.2 Plot of the inequalities with highlighted intersection points. The left plot was realized with $\alpha_1 = 0.4, \alpha_2 = 0.4$ while the right one uses $\alpha_1 = 0.5, \alpha_2 = 0.4$. Dashed lines represent $\pm mg \cdot \arctan \mu \approx 0.46$ .......................... 27
3.3 Error between the SHrink algorithm and the generic recursive projection algorithm on 1000 random samples. ................................. 34
3.4 Pie chart showing as angular sections the repartition of computation time for 150 iterations: total time 1.87 s ................................. 38
3.5 Example of angle-based morphing between convex polygons. Note how such a morphing generates unnecessary rotations in the bottom-right and bottom-left corners. ................................. 42
3.6 Example of EGI-based morphing between convex polygons. Note how non-monotonous the morphing is. ........................................ 43
3.7 Stability 2D polygon morphing: $P_s$ in blue, $P_d$ in red, the green and black arrows show the motion and the normals directions of the edges respectively. ........................................ 45
3.8 Comparison between direct computation of the constrained static stability polygon and interpolation. ........................................ 46
3.9 Interpolation coefficient as a function of the force constraint coefficient for a 3-contact scenario. Dashed line represents $\kappa = 1 - \gamma$. ........................................ 47
3.10 Climbing the stairs with HRP-2 Kai: with neither the constraint nor gripper torque task activated (left), with only the task (middle) and with both (right) ........................................ 51
3.11 In blue, the CoM trajectory during the experiment. In transparent grey, the successive static stability polygons. The CoM notably interacts with the constraint at around $(-0.25, 0)$ and $(0.5, 0.05)$. ........................................ 52
3.12 Still frames of HRP2 successfully climbing the stairs using the handrail. The CoM motion is limited by the polygonal constraint. .................. 53
List of Figures

3.13 Force (magnitude) applied on left hand during stairs climbing. Data was low-pass filtered by an order 2, Butterworth filter with a critical frequency of 10Hz. For phases, \(a,m,r\) stand for “add”, “move” and “remove” respectively while \(R,L,F,H\) stand for “Right”, “Left”, “Foot”, “Hand”.

3.14 Snapshots of the force control experiment with HRP-2 to smoothly remove the right foot. The CoM is smoothly pushed forward by the shrinking of the stability region, while the foot itself is admittance-controlled. No force distribution between the hand and left hand is given.

3.15 Results of multi-contact experiment and simulation. Top: normal force applied on all contacts during motion. Bottom: interpolation index.

4.1 Illustration of the problem annotated with the variables. We propose algorithms to compute the robust static stability \(P\) for the CoM \(c\), knowing the space \(\mathcal{G}\) of the CoM’s accelerations \(\ddot{c}\). Individual contact forces are not named here. Instead, we only consider the set of all contact forces, denoted \(f\). Contact surfaces are represented by green disks with the contact points and their normals, represented by the blue friction cones. \(p\) is the static stability polygon as computed in [Bretl and Lall, 2008].

4.2 One iteration of the algorithm. (a) Initial state: the outer approximation is a tetrahedron (transparent red). The search direction is perpendicular to the base: the extremal blue point is found, while the intersection by the new plane forms the three green points. (b) The inner approximation is now the blue tetrahedron. (c) Exploded view of the final state. The red tetrahedron is now outside of the outer approximation. The inner tetrahedron’s facets form three prismatoids with intersecting trapezoidal bases (thick orange, thin green, and transparent grey) by cutting the remaining outer approximation.

4.3 The robust static stability polyhedron as an intersection of oblique prisms, for a residual radius \(g_m = 3.55 \text{ m s}^{-2}\) while the acceleration envelope is defined by \(\mathcal{G} = \{\pm g_m, \pm g_m, g\}\).

4.4 Comparison of different \(P\) for different acceleration polytopes \(\mathcal{G}\) generated by \(g_m\). Only the inner approximations \(P_{\text{inner}}\) are rendered (in red), but are almost superimposed with \(P_{\text{outer}}\). The black dots with yellow cones represent contact points with associated friction cones.

4.5 Comparison of different \(P\) for different acceleration spheres of radius \(r\). Only the inner approximations \(P_{\text{inner}}\) are rendered (in red), but are almost superimposed with \(P_{\text{outer}}\). The black dots with yellow cones represent contact points with associated friction cones.
### List of Figures

4.6 Stability polygon in 2D (in red). The black dots with yellow cones represent contact points with associated friction cones. .......................... 74

4.7 Computation times for 3 contacts, polytope with 4 vertices and 3 point contacts. 50 iterations, total time 1.29s (clock time, without interpreter set-up and teardown). Final error: 0.043 m$^3$. .......................... 75

4.8 Comparison between the convex hull of sampled points and the computed $\mathcal{P}$ at various residual radiuses for different number of vertices of $\mathcal{G}$. Sampled points were tested for stability with $|\mathcal{G}| = 18$. .......................... 77

4.9 Precision reached as a function of the number of iterations while approximating a sphere. ................................................................. 78

4.10 Number of iterations required to test random points compared to naive sampling. The blue line represents the average number of iterations, the shaded area corresponds to the minimum-maximum envelope. The red line represents the naive sampling: one iteration per query. The dashed lines represent the number of iterations necessary to reach the indicated precision using algorithm 3. .......................... 81

4.11 Morphing between two different robust static stability polyhedrons. .......................... 82

4.12 Comparison of morphing with direct computation for different keypoints. 84

4.13 Comparison between: the statically stable prism in light blue, the robust static polyhedron with a $\mathcal{G}$ generated using $g_m = 2.6\text{ms}^{-2}$ in red and the prism formed by the robust polygon at the same margin and height 0.1 m in dark blue. Note that the scale is non-uniform across axes. .......................... 85

4.14 Compare 2D robust and 3D robust at different heights (in meters). Dashed lines represent volumes error, while solid lines represent areas error. ................................................................. 86

4.15 Results obtained using posture generation. From left to right, both feet, feet and right forearm, left foot and right forearm, each pair is first non-robust, labelled S, then robust, labelled R. The same robust polyhedron is displayed in both images of each pair. .......................... 87

4.16 Illustration of the inequality morphing from $\mathcal{P}_0$ (dashed) to $\mathcal{P}_1$ (dot-dashed). The intermediate dilatations are in solid black. The interpolate at $\kappa$ is filled in transparent red. .......................... 89

4.17 Climbing the stairs under robust static stability constraints. In this scenario, the weight of the CoM task is set to the lowest possible, 10. The current interpolate of the polyhedron is depicted in transparent green. Note how the robot tilts backwards: the CoM is only pushed forward by the constraint. .......................... 92

5.1 HRP-2 leaning: representation of variables used in modeling. .......................... 99

5.2 Data flow for MPC .......................... 103
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.3</td>
<td>Multi-contact motion planning architecture.</td>
<td>106</td>
</tr>
<tr>
<td>5.4</td>
<td>HRP-2 going over a ridge: contact stances and transitions are illustrated.</td>
<td>111</td>
</tr>
<tr>
<td>5.5</td>
<td>HRP-2 Walking on uneven ground while carrying an object.</td>
<td>112</td>
</tr>
<tr>
<td>5.6</td>
<td>Uncontrolled momentum derivative $\dot{L}_2$ computed on the whole-body</td>
<td>113</td>
</tr>
<tr>
<td></td>
<td>model (thin red) and reduced model (thick grey) with superposition of</td>
<td></td>
</tr>
<tr>
<td></td>
<td>computed trajectories (multicolor transparent).</td>
<td></td>
</tr>
<tr>
<td>5.7</td>
<td>Schema of the polyhedron CoM preview control.</td>
<td>114</td>
</tr>
<tr>
<td></td>
<td>Left shows the spatial trajectory going through three different polygons.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Right shows the evolution of the integer variable with time.</td>
<td></td>
</tr>
<tr>
<td>5.8</td>
<td>Keyframes of HRP2-Kai crossing a ridge: the blue ribbon represents the</td>
<td>118</td>
</tr>
<tr>
<td></td>
<td>current CoM trajectory being tracked.</td>
<td></td>
</tr>
<tr>
<td>5.9</td>
<td>Keyframes of HRP2-Kai climbing the stairs: the blue ribbon represents</td>
<td>119</td>
</tr>
<tr>
<td></td>
<td>the current CoM trajectory.</td>
<td></td>
</tr>
</tbody>
</table>
List of Tables

4.1 Computation time of 50 SOCP problems when considering a spherical \( G \) compared to the linearization. In the latter case, it is a function of the number of vertices of \( G \). .................................................. 76

4.2 Influence of the CoM task weight on success of stairs climbing, with and without the polyhedral constraint activated ........................... 92

5.1 Weights and Stiffness used in CoM Preview ........................................ 110
Introduction

The overarching theme of this thesis is to generate stable motion for legged robots, particularly humanoids. Those robots have the great advantage of being able to perform multi-contact motion: they can use a number of support points on the environment with any potential part of their bodies. They can thus theoretically walk, climb, crawl... while potentially carrying payloads. Yet, they have a fundamental defect compared to fixed platforms: they can loose balance and fall. This problem makes them unreliable, and it is what prevents them from being widely used.

In the industry, robots have been everywhere for decades: most of them are fixed arms, operating at high speeds, and performing high-precision tasks. Those robots must interact with the physical world and manipulate heavy payloads. However they do so without having to consider stability: even when perturbed, there is no way they will not be able to resume operation short of mechanical failure. The other categories of robots that are currently at the center of commercial projects are autonomous cars and drones. The former are inherently stable, the latter can freely move in space without contact.

Nowadays, most homes are equipped with a robotic solution. The most popular home “robotics” applications remain focused on artificial intelligence and are mainly tasked with speech and image recognition. However a close contender are a domotic solutions: a wheeled robot, that performs vacuuming, lawn mowing, pool cleaning, etc. They are naturally stable due to their structure: they all present a large base and are very short. This ensures their stability, especially on flat terrains. Another category of robots is spreading in homes: toys. Whether light remote controlled drones, boats, or spherical robots, they do not need to properly consider stability. Moreover, they present a low number of degrees of freedom.

Finally, service robots are starting to appear and serve as guides, receptionists, and personal assistants. They are often composed of a wheeled base surmounted by an humanoid torso. As they are top-heavy, their stability is not guaranteed but is controlled by properly accelerating the base. Yet, they are not designed to handle
physical contact, and cannot lift heavy charges.

Thus technologies more recently adopted are closer to humanoid robotics, and require more advanced notions of stability. Yet, an important range of applications is not addressed by those platforms:

- Locomotion across rough terrains, including stairs, ladders and narrow catwalks.
- Manipulation of heavy payloads in an unstructured environment.

Those scenarios will typically arise in disaster recovery scenarios, as evidenced by the Darpa Robotics Challenge\(^1\), where a robot will typically have to cross rough terrain, clear heavy debris to access an objective, typically a switch, valve or control panel. However, those use cases are not only exceptional, but an everyday occurrence in manufacturing scenarios. Indeed, on construction sites, in shipyards and in airplane assembly factories, the only way to get around is on foot, as there are no ramps to change floors, and the floor itself is often not complete. Finally, in those places it is very important to be robust to external perturbations as the robot will interact not only with unstructured terrain but humans.

To generate robotic motion, most approaches so far have either considered one of two options:

- Generating stable keyframes and interpolating between them. This interpolation generally does not guarantee stability, which means that the motion is often hand-tuned to obtain a satisfying result.
- Generating a stable trajectory. Note that as the trajectory is often composed of a finite number of waypoints, a similar problem to the above point is raised. However, those waypoints are usually very close to one another, which makes the assumption of stability in between them more reasonable.

Yet, in both cases, no notion of stable affordances was introduced: the generated motion could be only marginally stable. This means that any outside perturbation, or any error in the execution (due to imperfect actuation) could lead to falling.

Hence, in this thesis we will generate motion that is not only stable on a point-to-point basis, but that is constrained to a known region of stability. We will also introduce the notion of affordances that will give us a region of robust stability. As those regions are

projections of high-dimensional convex bodies defined by inequalities, obtaining their explicit representation requires a special set of tools and algorithms to be computed efficiently. Thus, we focus on developing new algorithms to compute those regions and to use them within a multi-contact control framework.

The first chapter of this thesis thus presents the basis of the notion of stability, or equilibrium. Starting from the equations of motion, we show that they are not sufficient to properly discriminate between stable and unstable motion. As this notion is central to the field of legged robotics, we also present the relevant prior work and introduce our ideas.

In the second chapter, we recall some well-known mathematical notions that are central to this thesis.

In the third chapter, we show that we can compute a region of static equilibrium, that is to say a set of acceptable positions supposing that the robot is not moving. This region can be deformed according to the set of acceptable contact forces: this prove to be useful to determine how to move the robot in order to obtain the desired contact force. Moreover, we introduce computer graphics algorithms that continuously deform this region. Finally, this deformable region is integrated in our control framework, and the results evaluated on a real hardware.

In the fourth chapter, we show that introducing the notion of stability affordances results in a substantial increase of complexity: whereas the static equilibrium region is a right prism whose base can be computed in 2 dimensions, the robust static equilibrium region is a general 3-dimensional shape. We show the different ways to compute this shape, and, similarly to the previous chapter, which algorithms can be used to continuously deform it. This can be used for scaling up or down the affordances and for multi-contact control.

Finally, the fifth chapter introduces two different, yet related, methods to perform multi-contact trajectory generation. The first one is a general method to perform multi-contact motion that takes advantage of the whole dynamics of the robot. The second one is based on the notion of robust static equilibrium: it is very efficient in terms of computation time but does not exploit the full extent of the robot dynamics. However, as it includes a notion of affordances, it can be seen as safer in the sense that it is resilient to perturbations.

Finally, we conclude our work and present some ideas for extensions and future works. Enhancements of both a technical and theoretical nature are discussed together with new applications.
Chapter 0. Introduction

In the next chapter, we review the state of the art and contrast it to the works presented in this thesis.
Stability is of prime importance for legged robots, and in particular humanoid robots. As they are under-actuated systems, it is important to retain controlability of the non-actuated floating base. Indeed, a fall is a failure that can cause serious damage to the robot’s structure. Even if some promising approaches are currently researched to limit the damage received during a fall [Samy and Kheddar, 2015] and to autonomously stand back up [Fujiwara et al., 2003], they are not sufficient to make falling safe. Moreover, in industrial scenarios, such as those envisioned in the Darpa Robotics Challenge\(^1\), falling can have catastrophic, non-recoverable consequences, especially if falling from a great height or into sharp debris. Similarly, the European research project COMANOID\(^2\) aims at performing multi-contact motion inside an in-construction airplane. Falling would damage the airplane, and is unacceptable. Thus, the notion of stability has been extensively studied throughout the years, and many different approaches have shed light on how to retain control of a moving legged robot.

To better understand the mathematical challenges underlying stability, let us go back to the rigid-body dynamics that describes the evolution of the robot’s state as it interacts with its environment.

\[
H(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) = J(q)^T f + \tau
\]

This equation shows us that joint accelerations are linked in a non-linear fashion to contact forces, torques and the non-linear Coriolis and gravity terms [Featherstone,

\(^1\)http://archive.darpa.mil/roboticschallenge/
\(^2\)http://comanoid.cnrs.fr/
By adding a Coulomb friction cone constraint, we can construct a full model of the robot’s motion:

\[ \| B f \| \leq u^T f \]  

(1.2)

This form can be considered *locally* linear in \( \dot{q} \) and \( f \), that is to say, if we consider \( q \) and \( \dot{q} \) to be constants as well as discretizing the friction cones. Making this constraint linear makes it easy to use it in a convex optimization program, such as a quadratic program (QP), without any simplification of the robot model. This kind of programs are very fast to solve, in the order of a millisecond for about a hundred variables, making it a great tool to be used in real-time, whole-body, multi-contact applications. This quasi-static interpretation is at the heart of most tasks-based controllers. Some are strictly prioritized task-space controllers [Sentis et al., 2010, Saab et al., 2012]. Others are weighted task-space controllers [Collette et al., 2007, Salini et al., 2010, Ott et al., 2011, Bouyarmane and Kheddar, 2011a, Righetti et al., 2013, Vaillant et al., 2016]. Both approaches are similar, and only slightly differ in capabilities. Although, controller robustness depends heavily on the numerical solver they use (generally off-the-shelf [Gill et al., 1986, Turlach, 1998] or customized [Escande et al., 2014] QP solvers), they were used with great success in a variety of scenarios. Indeed, some of the best performing teams at DRC used such an approach [Feng et al., 2014, Kuindersma et al., 2015, Koolen et al., 2016] as part of their control pipeline in order to perform egress/ingress from a car. Other works [Bouyarmane and Kheddar, 2012, Vaillant et al., 2016] exploited with success such approaches to access confined spaces in more complex scenarios.

However, this approach has many limitations, the main one being that it does not directly prevent the robot from falling. Indeed, setting \( \dot{q}, f \) and \( \tau \) to zero gives us a perfectly acceptable solution where the robot accelerates in the gravity field, i.e. free falls. This problem can become very acute when using the above quasi-static approach: if the robot ventures in an area where the only solution is falling, it is unlikely that it will be able to correct its course. In that sense falling is an *attractor* of the motion, albeit an undesirable one and Equation 1.1 does not describe stability but rather the notion of *compatibility*, that is to say, is an oracle to tell us if some contact forces are coherent with some joint accelerations in a given robot state.

A second issue is that transitions between contacts is a discrete event, that cannot be captured by local regulation schemes. Indeed, transitioning between contacts requires modifying the force distribution, but (1.1) is not sufficient to judge the feasibility of a force distribution.
To tackle these problems, most solutions fall into one of two categories:

- Instead of using general dynamic stability, use a stronger local stability criterion that will reject falling as an unacceptable solution.
- Instead of only considering the local time, look for a solution over a time horizon. This will automatically reject the falling trajectories as they will come with a high cost: it is in general impossible to follow a given objective while falling.

Of course, some solutions may use both. We will see in the following sections that using reduced models leads to great improvements in computation times. This fact can be exploited for real-time operation or for computing more than one solution, sometimes all solutions, in the allotted time.

1.1 Using a stronger stability criterion

In order to smoothly transition between contacts, we want to specify only a desired force profile on a given contact without endangering stability.

Whole body controllers typically solve an optimization problem that aims at finding compatible joint accelerations, \( \ddot{q} \) and contact forces \( \lambda \) by minimizing a set of task errors under constraints. Stability is typically enforced by controlling the position of the Center of Mass, or a derived quantity.

On the one hand, the current frameworks can not handle an unified force-stability objective. Indeed, let us consider a general stability objective that depends on the generalized coordinates and their derivatives, \( q, \dot{q}, \ddot{q} \) and the contact forces \( f \). Then, we will see in chapter 2 that typical frameworks transform this general task error \( X \) into a quadratic objective \( F_X \) in \( \ddot{q} \) and \( f \). However, this is not possible in our case, as one must differentiate the general objective with respect to time. This would entail derivating \( f \), but \( f \) are part of our optimization variables.

On the other hand, separate objectives require carefully setting each objective:

- A task to minimize the 2-norm of \( \lambda \) will not result in modification of angular values: the best local solution is redistributing the internal torques to fall as little as possible.
- Controlling the CoM will require to set gains that will interfere with almost all the other tasks, such as end-effectors trajectories.
Chapter 1. State of the Art

This is why several works [Ott et al., 2011, Lee and Goswami, 2012, Righetti et al., 2013, Herzog et al., 2014] have proposed efficient force regulation control strategies in multi-contact configurations, but in all of those cases, the CoM was regulated independently.

Instead of considering the whole dynamics, we can employ a stronger stability criteria that will lead to simpler, faster, computations.

A very well known reduced-model criterion is the Zero-Moment Point. The original ZMP criterion [Vukobratović and Borovac, 2004] is the extension of the convex hull criterion and has been widely used in biped locomotion on flat grounds assuming high friction, e.g. [Kajita et al., 2003]. Unfortunately, it is only applicable for locomotion on flat ground and does not take friction into account. The ZMP is better defined in [Goswami, 1999], and its similarity to the Center of Pressure (CoP) is highlighted in [Sardain and Bessonnet, 2004].

One can enforce stability in multi-contact by controlling the Center of Pressure (CoP) at each contact to remain within the convex hull of its area [Sentis et al., 2010, Righetti et al., 2013, Herzog et al., 2014, Audren et al., 2014, Feng et al., 2014, Hyon and Cheng, 2007, Wensing et al., 2013, Nori et al., 2015]. This strategy similarly requires controlling the contact forces. In these implementations, the CoM objective was given as a high-level task, and the CoP or contact forces were regulated in the null-space or in conjunction with the CoM task. Yet, the CoP is not defined when the normal force applied on the contact is nil, and is not easily extensible to bilateral contacts.

Thus let us consider an even simpler criterion: static stability [McGhee and Frank, 1968]. We rewrite the whole-body problem (1.1) as:

\[ g(q) = J(q)^T f + \tau \]  

While keeping the Coulomb friction constraint (1.2). This allows us to characterize all postures that are indefinitely (in terms of time) sustainable by the robot being considered. However, this entirely disregards the dynamics of the motion.

Using this formulation, one can formulate a non-linear optimization problem in \( q \) and \( f \) that will aim at finding compatible joint angles (including free-flyer position) and contact forces [Escande et al., 2013, Brossette et al., 2015, Bouyarmane and Kheddar, 2012, Brossette et al., 2016]. Such a problem is often referred to as posture generation.

However, keeping such a non-linear formulations requires an expensive optimization process to be run to find one solution, making it ill-adapted at real-time applications,
but a perfect match for offline planning applications. Indeed, it does not simplify the
kinematics, can be refined to include constraints such as joint limits, torque limits,
and will automatically find the position of the contacts.

Yet, to actually perform the motion, one needs to design a controller that will allow
to move from one planned stance to the next. In general, stability is not guaranteed
in-between the stances, and even basic feasibility is not granted.

Thus, we want to find a region of stability inside which stability is guaranteed. Remaining
in this region would be a hard constraint. We can then add a CoM objective among
the tasks to be achieved at best with a low priority to guide the controller towards a
particular solution.

We have shown in [Audren et al., 2016] and chapter 3 that static stability can be
leveraged, for complex multi-contact motion because it is easy to find every statically
stable position of the CoM given a set of contacts. Indeed, at least three techniques
are available to compute the region of static stability: direct projection based on
double-description [Fukuda and Prodon, 1996a], recursive projection [Bretl and Lall,
2008] and a quasi-analytic formulation [Or and Rimon, 2016]. We will moreover show
that the computation of this set of positions depends on the allowed contact forces,
and can be leveraged to perform multi-contact force regulation. It is then a matter of
designing a controller that can enforce the correct constraints.

1.2 Going further than locally

One usual way to circumvent the local-view issue is to use a preview controller to
enhance the motion with a flavor of dynamics [Audren et al., 2014, Dai et al., 2014],
but slow multi-contact motion can be achieved without preview [Vaillant et al., 2016],
using only a closed-loop local task-based controller.

Trajectory optimization is a very rich field, that provides the ability to look ahead,
and find a solution that is not only locally optimal, but a motion that is optimal over
a time interval. This enforces stability by making sure that we reach a goal: if the
robot fell midway, it will never reach the desired target. However, generic trajectory
optimization remains a slow process. The most general methods do not make any
assumptions on the model nor the motion, but can be very slow to optimize a whole-
body trajectory [Lengagne et al., 2013, Kudruss et al., 2015, Posa et al., 2014], on the
order of several minutes to reach a solution.

Recent recursive approaches can require several seconds to optimize [Hauser, 2013,
Chapter 1. State of the Art

Hauser, 2014] on a single core. Even approaches that leverage the high-parallelism capabilities of Graphic Processing Units (GPU) cannot keep up with real-time [Chrétien et al., 2016].

Although well-suited for planning, those approaches are not reactive. Indeed, the longer the trajectory optimization process takes, the slower the robot will respond to external perturbations. A very popular technique to compensate for perturbations of the current state while taking into account its repercussions into the future is Model Predictive Control (MPC) also known as Model Preview Control or also as Receding Horizon Control.

Appearing about 30 years ago [Clarke et al., 1987] for industrial plant control applications, the technique aims at finding control inputs over a time horizon that minimize an objective function based on the output of the plant model. The model is thus inherently recursive, as successive control inputs will affect the rest of the preview horizon: state feedback is taken into account over the whole horizon. This also gives the opportunity to compensate for modelling errors: if the preview horizon is chosen to be much larger than the computation time, only the beginning of the trajectory will be used. The next optimization will start from the current state, which will differ from the model-based prediction of the plant state.

In robotics, time scales are generally short (from the millisecond to the second) while industrial plant applications operate on a larger scale (from the second to the week). Thus, careful selection of the predictive model is essential to make sure that recomputation of the preview control can happen in time. One of the most well-known application of predictive control is ZMP preview control [Kajita et al., 2003] for humanoid bipedal locomotion. In that case, the reduced model used for predictive modelling is a very simple cart-table model: the robot is assimilated to a single rigid body (the cart) that can only move in a plane of fixed height (the table). This allows for linear formulation of the ZMP dynamics, and thus for very fast resolution, by reducing to an unconstrained Linear Quadratic Regulator problem. A whole-body controller based on Inverse Kinematics (IK) is then applied to track the generated ZMP trajectory. This application set a high performance standard even if it entirely neglects friction constraints, is only applicable to flat ground locomotion, and necessitates as input a target ZMP trajectory.

Since then, the area of predictive control strives to solve more general problems. Using a more complex model, the Spring-Loaded Inverted Pendulum (SLIP), [Mordatch et al., 2010] used this approach to realize more challenging motion and various gaits, from walking to running with full-flight phases. However, this approach was still limited to walking applications and was not real-time, due to the underlying whole-body
1.3 Handling the non-linearity

controller.

To further improve the performance of such a controller, [Ibanez et al., 2014] proposed to extend the state with a set of integer variables that allow for discrete decision making. This controller is able to automatically choose the position of the feet which allows to choose a gait in reaction to changes in desired velocity and external disturbances.

Then, in our work [Audren et al., 2014] that we will elaborate on in chapter 5 we propose a CoM preview-control formulation to extend this formalism to multi-contact locomotion, that takes into account contact friction. It relies on reducing the robot to a freewheel model, that is to say a single body with mass and inertia, located at the center of mass. Then by solving an optimization problem on the CoM acceleration and contact forces, considered as the command inputs, we are able to generate 3-dimensional trajectories. However, this solution is limited by the necessity to fix the acceleration of the CoM in a particular direction and results in uncontrolled momentum over one axis.

Even when the whole-body dynamics is reduced to its center-of-mass [Orin et al., 2013, Audren et al., 2014], the resulting equations exhibit the angular momentum of the body as a cross-product between the contact forces and the CoM position; an operation that is neither linear nor convex:

\[ \ddot{c} = \sum f_i \]  
\[ \dot{L} = \sum f_i \times (c - p_i) \]

To avoid this, two approaches remain: go back to a simpler, but strictly stronger criterion, or try to deal with this non-linearity head-on.

### 1.3 Handling the non-linearity

Going back to preview control, it would be interesting to design three dimensional, unconstrained trajectories. However, one needs to take a particular approach to handle the non-linearity that appears when considering the dynamics of the flywheel model.

Some approaches solve head-on the non-linear problem, for example [Caron and Kheddar, 2017]. Specialized optimization techniques are employed to make the reso-
Chapter 1. State of the Art

...solution very fast [Serra et al., 2016].

To do so, [Herzog et al., 2015] propose to solve the momentum non-linear dynamics head-on by using a dynamic programming scheme but lose the real-time capabilities of the system as they need several passes of quadratic programming and inverse kinematics to reach their goal. In later works [Herzog et al., 2016], the authors modify their approach to better isolate non-linearity of the problem at the cost of increased complexity in the constraints, turning it into a Quadratically Constrained Quadratic Program (QCQP) but in doing so highly reduce the complexity class of their problem, rendering it linear in the number of timesteps. Now, they have developed a convex approximation of the dynamics [Ponton et al., 2016], by approximating the non-convex quadratic constraint part by a difference of convex functions.

Another solution is to decouple the acceleration from the contact forces. By restricting either the center of mass acceleration or position to lie within a convex polytope, one can compute the convex envelope of the other.

Our idea is to compute a hull $\mathcal{P}$ for the CoM, such that $\forall \text{CoM} \in \mathcal{P}$, the stability is guaranteed to be robust w.r.t to a given set of accelerations $\mathcal{G}$. That is to say, for all possible CoM positions that lie within $\mathcal{P}$ and, there exist a set of contact forces that can generate any acceleration in $\mathcal{G}$. In other words, we compute the intersection of $\mathcal{G}$ with the set of all possible motions. We show that this intersection results in a convex volume that we project in the CoM space. Hence, by defining $\mathcal{G}$, the CoM acceleration and its position are decoupled. Other restrictions can be set to obtain such a decoupling. One of the most stringent is to set the CoM acceleration to zero, resulting in the static stability criterion as in the previous section.

Static stability can be expressed as a function of the gravity orientation. Indeed [Mattikalli et al., 1996, Mosemann et al., 1997] present a way to find all the gravity orientations that satisfy static stability of assemblies: it is a convex region defined by inequalities. They apply early vertex enumeration techniques—that we also use, to compute the acceptable region. Yet, the objects were fixed and not actuated.

The next step was to extend the idea of the ZMP and convex hull to multi-contact motion. This gave us a new criteria: the resultant wrench of the contact forces must remain in a polyhedral convex cone. Pioneered by [Saida et al., 2003], it formed the basis of the work in [Hirukawa et al., 2006] ostensibly titled “Adios ZMP”. It is applicable to multi-contact motion while remaining linear and global. This criterion is more clearly and properly established and applied to multi-contact motion in [Caron et al., 2015, Caron et al., 2017]. In the latter works, a new ZMP-like criterion (a pseudo-ZMP) that has to remain in the two-dimensional projection of the convex polyhedral wrench...
cone is proposed. It is applicable to multi-contact motion and takes into account friction. Note that, similarly to the ZMP and the CoP, the pseudo-ZMP support area depends on the instantaneous CoM position.

Constraining the CoM to remain in the static stability polygon $p$ is a global, CoM based, and linear criterion. Unfortunately, static stability (equilibrium) criterion does not imply dynamic stability [Garcia et al., 2002, Audren et al., 2016]. Indeed tracking a statically stable trajectory with changes in acceleration may induce falling. Thus, static stability is marginal, i.e. not robust to changes in the total acceleration.

Instead, our approach, presented in chapter 4 imposes constraints on the resultant acceleration and finds a linear, global corresponding constraint on the CoM. It the natural extension of the previous chapter chapter 3: instead of computing regular static stability regions, we add explicit stability margins.

We propose algorithms to compute a robust stability region $\mathcal{P}$ for a given $G$. This decoupling will be exploited in a MPC formulation in chapter 5.

In what can be seen as the dual of our approach, restricting the CoM position results in the volume of accelerations being an intersection of cones, as presented in [Caron and Kheddar, 2016].

Note that other non-linearities arise when allowing more parameters to vary: the timing of steps, as in [Ibanez et al., 2014], but also the position of the contacts [Deits and Tedrake, 2015, Kuindersma et al., 2015]. In both cases, using integer variables to change between discrete values allows to keep the problem linear. For timings, another approach is to first (linearly) compute an acceptable trajectory over the contacts, and then compute a Time Optimal Path Parametrization [Pham, 2014] to obtain the timings of the change in contacts [Caron and Pham, 2016, Hauser, 2014].
2 Preliminaries

Before delving into the contributions of this thesis, important building blocks have to be presented:

- Optimization, and a few typical programs as a way to find numerical solutions to generic problems.
- Posture generation as a tool to generate key postures and to find contacts.
- Task-based inverse dynamics to perform whole-body control.

2.1 Optimization

Optimization is a field of mathematics that is concerned with minimizing a certain function, called the objective or the cost function. The input is thus a function, \( f \):

\[
\mathcal{F} : E \mapsto \mathbb{R} \quad \quad \quad \quad (2.1)
\]

\[
x \mapsto \mathcal{F}(x)
\]

And the output is a \( x^* \) such that:

\[
\mathcal{F}(x^*) = \min_{x \in E} \mathcal{F}(x) \quad \quad \quad \quad (2.2)
\]

Finding \( x^* \) is the act of solving an optimization program or optimization problem, and
is denoted in short hand as:

\[ \min_x \mathcal{F}(x) \]  

(2.3)

As \( x \) can take any value in \( E \), the above problem is an instance of an \textit{unconstrained} program. In the general case, the acceptable values of \( x \) will be restricted through the use of \textit{constraints}:

\[ \min_x \mathcal{F}(x) \quad \text{s.t.} \quad l \leq \mathcal{C}(x) \leq u \] 

(2.4)

Where \( \mathcal{C} \) is a function:

\[ \mathcal{C} : E \rightarrow \mathbb{R}^n \] 

\[ x \rightarrow \mathcal{C}(x) \]  

(2.5)  

(2.6)

And \( l, u \) are vectors in \( \mathbb{R}^n \) that represent the bounds on \( \mathcal{C} \). If there exists a \( x \in E \) such that \( l \leq \mathcal{C}(x) \leq u \), then the problem is said to be \textit{feasible}.

If \( \mathcal{F} \) and \( \mathcal{C} \) have analytical expressions, it might be possible to analytically derive the optimal value of the optimization problem. However, most often, we can only numerically compute the value of the objective and constraints, as well as their derivatives: we will solve \textit{numerical} optimization problems.

If \( \mathcal{F} \) and \( \mathcal{C} \) have no specific form over \( E = \mathbb{R}^n \), we will refer to the problem they form as a \textit{non-linear} optimization problem. If \( \mathcal{F} \) is a quadratic function i.e. if there exists a positive semi-definite matrix \( Q \) and a vector \( v \) such that:

\[ \forall x \in \mathbb{R}^n, \mathcal{F}(x) = \frac{1}{2} x^T Q x + v^T x \]  

(2.7)

And if \( \mathcal{C} \) is a linear function, i.e. there exists a matrix \( A \) such that:

\[ \forall x \in \mathbb{R}^n, \mathcal{C}(x) = Ax \] 

(2.8)
2.2 Posture Generation

Then the program:

$$\begin{align*}
\min_x & \frac{1}{2} x^T Q x + c^T x \\
\text{s.t.} & \ l \leq A x \leq u
\end{align*}$$

(2.9)

(2.10)

Is said to be a \textit{quadratic} program.

If $\mathcal{F}$ is a linear function, i.e. if there exists a vector $\nu$ in $\mathbb{R}^n$ such that:

$$\forall x \in E, \mathcal{F}(x) = \nu^T x$$

(2.11)

And $\mathcal{C}$ represents $l^2$ cones i.e. there exists a sequence of matrices $A_i$, vectors $b_i$ and $\nu_i$:

$$\mathcal{C}(x) = \begin{bmatrix}
\|A_0 x + b_0\|_2 - \nu_0^T x \\
\vdots \\
\|A_n x + b_k\|_2 - \nu_k^T x
\end{bmatrix}$$

(2.12)

Then this optimization problem is a \textit{second-order cone} program, SOCP in short-hand.

Note that if $\mathcal{F}$ is convex, it has only one global minimum, whereas a general $\mathcal{F}$ can present many local minima. Thus if $\mathcal{F}$ and $\mathcal{C}$ are convex (i.e. $\mathcal{F}$ is a convex function over a convex set), the problem is said to be a \textit{convex} optimization problem. It can generally be solved faster than generic problems, and the solution is guaranteed to be globally optimal. For more details on how to solve such programs, refer to e.g. [Boyd and Vandenberghe, 2004].

2.2 Posture Generation

Posture generation (PG) in the context of legged robotics consists in finding a robot \textit{posture}: it is a \textit{generalized} inverse kinematics problem. Indeed, solving a PG problem not only results in a set of generalized coordinates $q$ but also in a set of contact forces $f$ that satisfies a set of constraints. Those constraints typically comprise:

- Contact constraints: the contact forces $f$ can only be applied at points where the robot’s body touches the environment. Moreover, they must obey Coulomb’s friction law.

- Static equilibrium constraints: the joint torques generated by $f$ must remain...
within their limits, and the resulting momentum at the center of mass must be null.

- Collision constraints: they prevent self-intersection and undesirable contacts with the environment.
- Joint limits: the joint angles \( q \) must remain within their acceptable range.

The objective function can be varied, but will often encode:

- Distance to a usual posture
- Force distribution on contacts
- Specific bodies orientation or position

Finding such a set \( q, f \) can be reduced to solving a non-linear optimization problem, possibly over non-euclidian manifolds [Escande et al., 2013, Brossette et al., 2015, Brossette et al., 2016]. Indeed rotations live in the special orthogonal group and not in an euclidian space.

Due to its non-linear nature, and because a humanoid robot has a high number of degrees of freedom, solving a PG problem cannot be done in real-time: it takes around a second to do so whereas the robot control loop runs at around 500 Hz. Moreover, posture generation is static by nature, and thus cannot be used to generate motion. The next section will present a generic approach to do so.

### 2.3 Task-based inverse dynamics

Task-based inverse dynamics, or tasks-based controller, or also QP controllers aim at finding optimal accelerations to realize certain tasks under constraints. They are most often formalized as a QP over the joint accelerations \( \dot{q} \) and contact forces \( f \). The QP formulation allows to formulate objectives in terms of distance (in the sense of \( L^2 \) norm) but does not compromise performance. Indeed, the programs presented below can be solved in the order of the millisecond by off-the-shelf solvers [Gill et al., 1986, Turlach, 1998].

The first constraint of any motion is of course dynamic compatibility. Rigid body dynamics, as presented in (1.1) can be considered linear in \( \ddot{q}, f, \tau \): this is true if and only if \( \dot{q} \) and \( q \) are considered independent of \( \ddot{q} \). In continuous time, this would
amount to consider small accelerations $\ddot{q}$. In discrete time, it amounts to assuming that $q, \dot{q}$ is constant over one timestep. In both cases, it means that variations of $q, \ddot{q}$ are slow with respect to the time period.

Then, the Coulomb friction constraint (1.2) can also be linearized by considering polyhedral cones instead of $l^2$ cones. To do so, a possible parametrization is to consider a new set of variables, $\lambda$ that represent the force intensity along each generator of the polyhedral cones. Thus,

$$f = N\lambda$$

(2.13)

is guaranteed to be inside the polyhedral cone iff:

$$\lambda \geq 0$$

(2.14)

We also need to consider torque limits:

$$\tau_l \leq \tau \leq \tau_u$$

(2.15)

This can be used to eliminate the $\tau$ variable, and form the motion constraints of our problem:

$$\tau_l \leq H(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) - J(q)^T N\lambda \leq \tau_u$$

$$0 \leq \lambda$$

(2.16)

(2.17)

We now need to form additional constraints and objectives: those will often be specified in the operational space. Thus, the task can be specified by:

- A reference $X_d$, and eventually a speed reference $\dot{X}_d$ and an acceleration reference $\ddot{X}_d$.

- A non-linear function $X(q)$, its jacobian (derivative w.r.t $q$) $J_X$ and $\dot{J}_X$.

The objective is to make $X(q)$ converge to $X_d$. Because $X$ does not depend on $\ddot{q}$ and because it is non-linear, it cannot be used directly in our QP objective. To transform it,
Chapter 2. Preliminaries

we first reveal the dependency to \( \dot{q} \) by deriving twice the error:

\[
\begin{align*}
\epsilon &= X_d - X(q) \\
\dot{\epsilon} &= \dot{X}_d - J_X \dot{q} \\
\ddot{\epsilon} &= \ddot{X}_d - \dddot{X}_d J_q - J_X \ddot{q}
\end{align*}
\] (2.18) (2.19) (2.20)

We then implement a task-space proportional derivative control: finding \( \dot{q} \) that mini-
mizes:

\[
F_X = \| K_p \epsilon - K_d \dot{\epsilon} - \ddot{\epsilon} \|^2
\] (2.21)

Will bring \( X \) to \( X_d \) without violating constraints.

Note that \( F_X \) is a quadratic function in \( \dot{q} \) and is suitable for a QP objective. To combine
different objectives, we use a weighted average: instead of minimizing a single \( F_X \), we
consider a set of \( F_X_i \) and corresponding positive weights \( w_i \):

\[
F = \sum w_i F_X_i
\] (2.22)

The same definition of \( \epsilon \) can be used in inequality constraints: suppose that we want
to enforce \( X(q) \geq X_d \). Then, by ensuring that:

\[
\dot{X}(q) \geq 0
\] (2.23)

we will ensure that \( X \) is increasing, and thus stay away from the bound. As we do
not want to keep this constraint activated at all times, we only activate it under some
interaction threshold \( \epsilon_i \).

We use an integral formulation to ensure that \( \dot{X} \) is increasing at the next step:

\[
J_X \dot{q} + dt (\dddot{X}_d J_q + J_X \ddot{q}) \geq 0
\] (2.24)

To avoid strong discontinuities, we introduce a damping term \( \xi \):

\[
J_X \dot{q} + dt (\dddot{X}_d J_q + J_X \ddot{q}) - \xi(\epsilon) \geq 0
\] (2.25)

The damping is a positive term, that is equal to 0 when \( \epsilon = 0 \) and large when \( \epsilon = \epsilon_i \).

We can then solve the optimization problem composed of a \( F \) defined by (2.22), a
motion constraint of the form Equations (2.16) to (2.17), and possibly more constraints
2.3. Task-based inverse dynamics

of the form (2.25). This yields the optimal $\ddot{q}$ and $\lambda$. If the robot being considered is torque-controlled, it is straightforward to reconstruct $\tau$ from $\ddot{q}$ and $\lambda$ using (1.1). Otherwise, if the robot is position controlled, integrating $\ddot{q}$ twice yields joint angles $q$ that are in turn fed to the joint-level controller of the robot.

The QP can then be solved at each timestep, and produce complete trajectories in real-time. For more details, see [de Lasa et al., 2010, Bouyarmane and Kheddar, 2012, Vaillant et al., 2016].
As presented in chapter 1, we will study how static stability can be used in multi-contact control. This chapter presents three key aspects: computation of the static stability polygon, development of an optimal matching for interpolation and integration with our multi-contact control framework. Finally, experimental results will be presented.

In this chapter, we will limit our developments to multi-contact, quasi-static motion. Hence, constraining the CoM to remain within the frictional static stability polygon is an acceptable criterion. Chapter 4 will extend this work to dynamic motion by taking into account acceleration of the Center of Mass.

Although computation of the static stability polygon is well-known, its dependency on the admissible contact forces has not extensively treated. Thus, we show in section 3.1 how the static stability polygon deforms as a function of the admissible forces, first on simple cases, then in the general case.

As we will integrate this CoM stability region as a hard constraint in our controller, it is necessary to avoid discontinuities. Indeed, if the constraint is not continuous, it might get suddenly violated, thus triggering a controller failure. To solve this problem, we introduce the notion of polygonal morphing in section 3.2. This technique, based on optimal matching, will allow us to approximate changes in the admissible contact forces (c.f. section 3.3).

Finally, section 3.4 will present how the previous points are used in our controller. In subsection 3.4.1 we present how to use the morphing as a constraint. Then we demonstrate how to climb stairs in subsection 3.4.2. We present a way to efficiently combine force control tasks with this constraint in subsection 3.4.4 and present results obtained in another multi-contact scenario in subsection 3.4.5.
Chapter 3. Static equilibrium and interpolation

3.1 Computation of the constrained stability polygon

In the typical case, the static stability region only depends on the geometrical and frictional properties of the contacts. How does this shape evolve as we restrict the admissible contact forces? We will first present simple examples in sections 3.1.1 to 3.1.4 before presenting generic algorithms in sections 3.1.5 and 3.1.6.

In each case, the region is naturally described by a set of inequalities and equalities in \( c \) and \( f \). The goal is to find the extremal values of \( c \). This amounts to a problem of vertex enumeration. Indeed, a set of linear inequalities describes a convex polyhedron, while a set of equalities amounts to projecting on a subspace. The question is thus: what are the vertices of this projection?

3.1.1 1D CoM - 2 aligned forces

We consider a 2D case with a robot whose coordinates are:

\[
c = \begin{bmatrix} c^x \\ c^z \end{bmatrix}
\]  \hspace{1cm} (3.1)

It has two contact points with the environment, with their associated contact forces (expressed in the world frame):

\[
p_1 = \begin{bmatrix} p_1^x \\ 0 \end{bmatrix} \rightarrow f_1 = \begin{bmatrix} f_1^x \\ f_1^z \end{bmatrix}
\]  \hspace{1cm} (3.2)

\[
p_2 = \begin{bmatrix} p_2^x \\ 0 \end{bmatrix} \rightarrow f_2 = \begin{bmatrix} f_2^x \\ f_2^z \end{bmatrix}
\]  \hspace{1cm} (3.3)

The Newton-Euler equations are:

\[
f_1 + f_2 = mg
\]  \hspace{1cm} (3.4)

\[(c - p_1) \times f_1 + (c - p_2) \times f_2 = 0
\]  \hspace{1cm} (3.5)

As the contact points have vertical normals, the friction equations become:

\[|f_1^x| \leq \mu f_1^z \]  \hspace{1cm} (3.6)

\[|f_2^x| \leq \mu f_2^z \]  \hspace{1cm} (3.7)
3.1. Computation of the constrained stability polygon

We then project every equation on every axis:

\[
\begin{align*}
    f_1^x + f_2^x &= 0 & (3.8) \\
    f_1^z + f_2^z &= mg & (3.9) \\
    f_1^x c^z + f_2^x c^z &= 0 & (3.10) \\
    f_1^z (c^x - p_1^x) + f_2^z (c^x - p_2^x) &= 0 & (3.11) \\
    f_1^x &\leq \mu f_1^z & (3.12) \\
    f_2^x &\leq \mu f_2^z & (3.13) \\
    -f_1^x &\leq \mu f_1^z & (3.14) \\
    -f_2^x &\leq \mu f_2^z & (3.15)
\end{align*}
\]

We want to find the boundaries of \(c^x\), i.e. \(\min (c^x)\) and \(\max (c^x)\). Notice that \(f_1^x = f_2^x = 0\) satisfies all the equations in which they appear, and thus any value of \(c^z\) is acceptable.

We reorganize (3.11) to isolate \(c^x\):

\[
c^x = (f_1^z + f_2^z) - p_1^x f_1^z - p_2^x f_2^z = 0
\]

\[
c^x = \frac{p_1^z f_1^z + p_2^z f_2^z}{mg}
\]

Thus \(x\) can only be the barycenter of \(p_1\) and \(p_2\) weighted by \(f_1^z\) and \(f_2^z\). As \(f_1^z\) and \(f_2^z\) are both positive and their sum is non-zero, we are sure that \(c^x \in [p_1^x, p_2^x]\). If we limit the intensity of \(f_1^z\) and \(f_2^z\) to \(\tilde{f}\), we then have:

\[
x \in \left[ \frac{p_1 \tilde{f} + p_2 (mg - \tilde{f})}{mg}, \frac{p_1 (mg - \tilde{f}) + p_2 \tilde{f}}{mg} \right]
\]

Indeed, there are only two ways to saturate the inequality constraints: either saturate \(f_1^z\) first and then \(f_2^z\) or the converse. If \(\tilde{f} < 0.5mg\) then there is no solution.

3.1.2 1D CoM - 2 non-aligned forces

We consider a slightly more general case: we keep the same model as in the previous section, but the contact normals are no longer vertical. We denote them \(n_1\) and \(n_2\), they respectively form an angle \(\alpha_1\) and \(\alpha_2\) with the horizontal. See section 3.1.2 for an illustration.
In that case, the Newton-Euler equations remain the same:

\[ f_1^x + f_2^x = 0 \]  \hspace{1cm} (3.19)  
\[ f_1^z + f_2^z = mg \]  \hspace{1cm} (3.20)  
\[ f_1^x z + f_2^x z = 0 \]  \hspace{1cm} (3.21)  
\[ f_1^z (c^x - p_1) + f_2^z (c^x - p_2) = 0 \]  \hspace{1cm} (3.22)

And we can still deduce that:

\[ c^x = \frac{f_1^z p_1 + f_2^z p_2}{mg} \]  \hspace{1cm} (3.23)

But the Coulomb friction equations are more involved:

\[ |\sin \alpha_1 f_1^z - \cos \alpha_1 f_1^x| \leq \mu (\sin \alpha_1 f_1^x + \cos \alpha_1 f_1^z) \]  \hspace{1cm} (3.24)  
\[ |\sin \alpha_2 f_2^z - \cos \alpha_2 f_2^x| \leq \mu (\sin \alpha_2 f_2^x + \cos \alpha_2 f_2^z) \]  \hspace{1cm} (3.25)

Meaning that indeed, the CoM is still the barycenter of the contact points weighted by the vertical forces, but finding the maximum value for each force is not evident.

We can remove the absolute value by considering both options:

\[ \sin \alpha_1 f_1^z - \cos \alpha_1 f_1^x \leq \mu (\sin \alpha_1 f_1^x + \cos \alpha_1 f_1^z) \]  \hspace{1cm} (3.26)  
\[ - \sin \alpha_1 f_1^z + \cos \alpha_1 f_1^x \leq \mu (\sin \alpha_1 f_1^x + \cos \alpha_1 f_1^z) \]  \hspace{1cm} (3.27)
3.1. Computation of the constrained stability polygon

And similarly for $f_2$. This can be rewritten in shorthand:

$$f_1^x a_1 + f_1^z b_1 \geq 0$$
$$f_1^x c_1 + f_1^z d_1 \geq 0$$  (3.28)

By using the two last lines of Euler-Newton we can reduce it to a set of four inequalities in two variables, $f_1^x, f_1^z$:

$$f_1^x a_1 + f_1^z b_1 \geq 0$$
$$f_1^x c_1 + f_1^z d_1 \geq 0$$
$$-f_1^x a_2 + (mg - f_1^z) b_2 \geq 0$$
$$-f_1^x c_2 + (mg - f_1^z) d_2 \geq 0$$  (3.30) - (3.33)

We can plot this in the $f_1^x, f_1^z$ space, see section 3.1.2. It highlights the fact that the interval we look has basically two forms: if $g$ is in the friction cones, it is $[p_1^x, p_2^x]$. Otherwise, it is a much smaller interval, that is hard to determine analytically. Indeed, the combinatoriality of the above system cannot be reduced: one has to make assumptions to deduce an actual bound. Supposing that all of the inequality coefficients are strictly positive and that $\frac{b_1 a_2}{a_1 b_2} > 1$, $\frac{d_1 a_2}{c_1 b_2} > 1$, $\frac{b_1 c_2}{a_1 d_2} > 1$, $\frac{d_1 c_2}{c_1 d_2} > 1$ the bound on $f_1^z$ is:

$$f_1^z \leq \min \left( \frac{m g a_1}{a_1 b_2 - b_1 a_2}, \frac{m g c_1}{c_1 b_2 - d_1 a_2}, \frac{m g a_1}{a_1 d_2 - b_1 c_2}, \frac{m g c_1}{c_1 d_2 - d_1 c_2} \right)$$  (3.34)

Figure 3.2 Plot of the inequalities with highlighted intersection points. The left plot was realized with $\alpha_1 = 0.4, \alpha_2 = 0.4$ while the right one uses $\alpha_1 = 0.5, \alpha_2 = 0.4$. Dashed lines represent $\pm m g \arctan \mu \approx 0.46$

Then, to find the range of possible $c^x$, one has to find the intersection of the constraint with the aforementioned inequalities. In the case of a normal force limitation $f^l$, it
Chapter 3. Static equilibrium and interpolation

will be a line of equation:

\[ f_1^z = \frac{f^l - \sin \alpha_1}{\cos \alpha_1} f_1^x \]  

(3.35)

In the case of a norm limitation, it will be a circle of radius \( f^l \). This example shows that for a very general case, the static stability region does not have an analytical closed form. The complexity of the enumeration problem is amplified when taking into account contact forces constraints. We will thus focus on keeping the contact normals vertical, and see if we can derive a general result. We will present general numerical methods in later sections.

3.1.3 2D CoM - 3 aligned forces

We consider a 2D case with a robot whose coordinates are:

\[ c = [c^x \ c^y \ c^z] \]  

(3.36)

It has three contact points:

\[ p_1 = [p_1^x \ 0 \ 0] \rightarrow f_1 = [f_1^x \ f_1^y \ f_1^z] \]  

(3.37)

\[ p_2 = [p_2^x \ 0 \ 0] \rightarrow f_2 = [f_2^x \ f_2^y \ f_2^z] \]  

(3.38)

\[ p_3 = [p_3^x \ 0 \ 0] \rightarrow f_3 = [f_3^x \ f_3^y \ f_3^z] \]  

(3.39)

The Newton-Euler equations write:

\[ f_1 + f_2 + f_3 = mg \]  

(3.40)

\[ f_1 \times (c - p_1) + f_2 \times (c - p_2) + f_3 \times (c - p_3) = 0 \]  

(3.41)

The friction equations are:

\[ \| f_1^x + f_1^y \| \leq \mu f_1^z \]  

(3.42)

\[ \| f_2^x + f_2^y \| \leq \mu f_2^z \]  

(3.43)

\[ \| f_3^x + f_3^y \| \leq \mu f_3^z \]  

(3.44)
3.1. Computation of the constrained stability polygon

By projection of newton-euler:

\[
f_1^x + f_2^x + f_3^x = 0 \quad \text{(3.45)}
\]
\[
f_1^y + f_2^y + f_3^y = 0 \quad \text{(3.46)}
\]
\[
f_1^z + f_2^z + f_3^z = mg \quad \text{(3.47)}
\]
\[
f_1^z(c^y - p_1^y) + f_2^z(c^y - p_2^y) + f_3^z(c^y - p_3^y) - (f_1^x c^z + f_2^x c^z + f_3^x c^z) = 0 \quad \text{(3.48)}
\]
\[
f_1^z(c^x - p_1^x) + f_2^z(c^x - p_2^x) + f_3^z(c^x - p_3^x) - (f_1^y c^z + f_2^y c^z + f_3^y c^z) = 0 \quad \text{(3.49)}
\]
\[
f_1^z(c^x - p_1^x) + f_2^z(c^x - p_2^x) + f_3^z(c^x - p_3^x) - (f_1^y c^z + f_2^y c^z + f_3^y c^z) = 0 \quad \text{(3.50)}
\]

We can see that \(c^z\) has no influence, as we can simplify (3.48), (3.49) using (3.45), (3.46) to equations that do not require it.

We want to find the boundaries of \(c^x\) and \(c^y\), i.e. enumerate vertices of the region.

Reorganize (3.48), (3.49) to isolate \(c^x\) and \(c^y\):

\[
c^x = \frac{f_1^z p_1^y + f_2^z p_2^y + f_3^z p_3^y}{f_1^z + f_2^z + f_3^z} = \frac{f_1^z p_1^y + f_2^z p_2^y + f_3^z p_3^y}{mg} \quad \text{(3.51)}
\]
\[
c^y = \frac{f_1^z p_1^y + f_2^z p_2^y + f_3^z p_3^y}{f_1^z + f_2^z + f_3^z} = \frac{f_1^z p_1^y + f_2^z p_2^y + f_3^z p_3^y}{mg} \quad \text{(3.52)}
\]

Thus the 2D projection of \(c\) can only be the barycenter of \(p_1, p_2, p_3\) weighted by \(f_1^z, f_2^z, f_3^z\). As \(f_1^z, f_2^z, f_3^z\) are both positive and their sum is non-zero, we are sure that \((c^x, c^y) \in \text{CONV}(p_1, p_2, p_3)\). As for \(f_1^x, f_2^x, f_3^x, f_1^y, f_2^y, f_3^y\), they need to satisfy the friction constraints and (3.50).

By simplifying (3.50):

\[
f_1^x p_1^y + f_2^x p_2^y + f_3^x p_3^y - f_1^y p_1^x - f_2^y p_2^x - f_3^y p_3^x = 0 \quad \text{(3.53)}
\]

It is obvious that setting to 0 all tangential components is an acceptable solution.

Then, if we add a new unilateral constraint (On \(f_1^z\), but without loss of generality):

\[
f_1^z \leq \bar{f} \quad \text{(3.54)}
\]
Chapter 3. Static equilibrium and interpolation

As long as \( \bar{f} \geq mg \), this constraint has no effect. If \( \bar{f} \leq 0 \), then this problem has no solution as the force limitation constraint would be incompatible with the friction constraints.

In the case where \( 0 \leq \bar{f} \leq mg \), we can use the friction constraints, (3.54) and (3.47) to deduce that:

\[
\begin{align*}
0 &\leq f^z_1 \leq \bar{f} \\
0 &\leq f^z_2 \leq mg \\
0 &\leq f^z_3 \leq mg
\end{align*}
\] (3.55) (3.56) (3.57)

Thus, the vertices of the polygon can be taken by trying to saturate all possible combinations of constraints while respecting (3.47). We obtain that the convex polygon is the convex hull of:

\[
\left\{ p_2, p_3, \frac{\bar{f}}{mg} p_1 + \left(1 - \frac{\bar{f}}{mg}\right) p_2, \frac{\bar{f}}{mg} p_1 + \left(1 - \frac{\bar{f}}{mg}\right) p_3 \right\}
\] (3.58)

Note that this results in the three original points when \( \bar{f} = mg \), and in \( p_2 \) and \( p_3 \) when \( \bar{f} = 0 \). This also does not depend on the actual value of \( \bar{f} \), but only on the ratio \( \alpha = \frac{\bar{f}}{mg} \).

3.1.4 2D case - \( n \) aligned contact points

Let’s consider a convex set of \( n \) ordered points, \( p_0, \cdots, p_{n-1} \). We can write one more time the Newton-Euler and friction equations:

\[
\begin{align*}
\sum_{i=0}^{n} f_i &= mg \\
\sum_{i=0}^{n} (p_i - c) \times f_i &= 0 \\
\forall i &\in [0..n-1] \left\| \begin{bmatrix} f^x_i \\ f^y_i \end{bmatrix} \right\| \leq \mu f^z_i
\end{align*}
\] (3.59) (3.60) (3.61)

By projecting the momentum equation on the z direction, we find again that:

\[
c^{xy} = \sum_{i=0}^{n} \frac{p_i f^z_i}{mg}
\] (3.62)
3.1. Computation of the constrained stability polygon

Thus, if we limit the force on the \(i\)th point, we obtain the same result as in the previous section. However, the difference is that we need \textit{a priori} to add \(n-1\) points, the \(q_k\), that are obtained by saturating the \(i\)th force constraint and the \(k\)th:

\[
\forall k \in [0..n-1], k \neq i, q_k = \alpha p_i + (1 - \alpha) p_k
\]  

\[\text{(3.63)}\]

\textbf{Theorem 1.} All \(q_k\) are inside the convex hull formed by the \(p_k\) and \(q_{i-1}, q_{i+1}\).

\textbf{Proof.} We suppose that there exist a \(q_j\) that is inside the triangle \(\{q_{i-1}, p_i, q_{i+1}\}\) with \(j \notin \{i-1, i+1\}\) and perform a \textit{reductio ad absurdum}. Under that hypothesis, there exists three positive real numbers \(\beta_{i-1}, \beta_i, \beta_{i+1}\) such that:

\[
\beta_{i-1} + \beta_i + \beta_{i+1} = 1 \\
q_j = \beta_{i-1} q_{i-1} + \beta_i p_i + \beta_{i+1} q_{i+1}
\]

Developping:

\[
\alpha p_i + (1 - \alpha) p_k = \beta_{i-1} (\alpha p_i + (1 - \alpha) p_{i-1}) + \beta_i p_i + \beta_{i+1} (\alpha p_i + (1 - \alpha) p_{i+1}) \\
(1 - \alpha) p_k = (\alpha \beta_{i-1} + \beta_i + \alpha \beta_{i+1} - \alpha) p_i + (1 - \alpha) \beta_{i-1} p_{i-1} + (1 - \alpha) \beta_{i+1} p_{i+1} \\
(1 - \alpha) p_k = (\alpha (1 - \beta_i) + \beta_i - \alpha) p_i + (1 - \alpha) \beta_{i-1} p_{i-1} + (1 - \alpha) \beta_{i+1} p_{i+1} \\
(1 - \alpha) p_k = (1 - \alpha) \beta_i p_i + (1 - \alpha) \beta_{i-1} p_{i-1} + (1 - \alpha) \beta_{i+1} p_{i+1} \\
p_k = \beta_i p_i + \beta_{i-1} p_{i-1} + \beta_{i+1} p_{i+1}
\]

Thus \(p_k\) is inside the triangle \(\{p_{i-1}, p_i, p_{i+1}\}\) which directly contradicts the convexity.

\[\square\]

Hence we see that limiting the force on a single point is equivalent to cutting it by a line parallel to the line passing by its two neighbours, at a distance \(\alpha\).

Now, what happens if we to limit the force on neighbouring vertices? Let us suppose that we apply a limitation \(\alpha_i\) on each vertex. As long as the neighbouring limits do not interact, we can apply the above rule independently:

\textbf{Theorem 2.} \textit{If the two following conditions are realized:}

\[
\forall i \quad \begin{cases} 
\alpha_i + \alpha_{i+1} \geq 1 \\
\alpha_i + \alpha_{i-1} \geq 1
\end{cases}
\]  

\[\text{(3.64)}\]
Chapter 3. Static equilibrium and interpolation

Then the reduced polygon is obtained by replacing every vertex \( p_i \) by the two vertices, \( q_{i-1} \) and \( q_{i+1} \).

**Proof.** This proof will be similar to the previous proof. We choose a vertex \( p_i \): any vertex that saturates \( \alpha_i \) has one of two forms:

1. A \( q_k \), that is thus inside \( \{ q_{i-1}, q_{i+1}, p_j \neq i \} \)

2. For some point \( p_k \), \( 1 - \alpha_i > \alpha_k \): in that case, we have to consider additional points for each \( p_k \). By the hypotheses of the theorem, this is only possible if \( k \notin \{ i, i + 1, i - 1 \} \)

The latter are the \( o_j^{i,k} \):

\[
o_j^{i,k} = \alpha_i p_i + \alpha_k p_k + (1 - \alpha_i - \alpha_k) p_j
\]

There may be multiple \( o_j^{i,k} \) spanning different combinations of \( p_k \). Let’s suppose that there exists one such point \( o_j^{i,k} \) that is inside the triangle \( \{ q_{i-1}, p_i, q_{i+1} \} \) with \( j \notin \{ i, k \} \) and perform a reductio ad absurdum.

Under those hypotheses there exists three reals \( \beta \) such that:

\[
\beta_{i-1} \beta_i \beta_{i+1} = 1
\]

\[
o_j^{i,k} = \beta_{i-1} q_{i-1} + \beta_i p_i + \beta_{i+1} q_{i+1}
\]

Developing:

\[
\alpha_i p_i + \alpha_k p_k + (1 - \alpha_i - \alpha_k) p_j = \beta_{i-1} (\alpha_i p_i + (1 - \alpha_i) p_{i-1}) + \beta_i p_i + \beta_{i+1} (\alpha_i p_i + (1 - \alpha_i) p_{i+1})
\]

\[
\alpha_i p_k + (1 - \alpha_i - \alpha_k) p_j = \alpha_i (1 - \beta_i) + \beta_i - \alpha_i)
\]

\[
\beta_{i-1} p_{i-1} + (1 - \alpha_i) \beta_{i-1} p_{i-1} + (1 - \alpha_i) \beta_{i+1} p_{i+1}
\]

\[
\alpha_i p_k + (1 - \alpha_i - \alpha_k) p_j = (1 - \alpha_i) \beta_i p_i + (1 - \alpha_i) \beta_{i-1} p_{i-1} + (1 - \alpha_i) \beta_{i+1} p_{i+1}
\]

\[
\alpha_k p_k + (1 - \alpha_i - \alpha_k) p_j = \beta_i p_i + \beta_{i-1} p_{i-1} + \beta_{i+1} p_{i+1}
\]
3.1. Computation of the constrained stability polygon

Thus a point on the line \((p_k, p_j)\) is inside the triangle \(\{p_{i-1}, p_i, p_{i+1}\}\) while \(k \neq i\) and \(j \notin \{i, k\}\). This contradicts the convexity except if \(j = i \pm 1\) and \(\alpha_k = 0\), but in this case, \(o_j^{i,k} = q_{i \pm 1}\).

Else we can use this stronger expression,

**Conjecture 1.** The reduced polygon is obtained by replacing every vertex \(p_i\) by the two vertices, \(r_i^0\) and \(r_i^1\). To obtain \(r_i^0\) and \(r_i^1\), apply Algorithm 1.

**Algorithm 1** Walking the convex hull to produce intermediate vertices

```plaintext
procedure SHRINK(i, \{p_0, \cdots, p_{n-1}\}, \{\alpha_0, \cdots, \alpha_{n-1}\})
  if \(\sum \alpha_i < 1\) then
    return Error
  end if
  \(r_i^0, r_i^1 \leftarrow \alpha_i p_i\)
  \(\alpha_i^0, \alpha_i^1 \leftarrow \alpha_i\)
  \(k_0, k_1 \leftarrow i\)
  while \(k_0 \neq \text{done}, k_1 \neq \text{done}\) do
    \(k_0 \leftarrow \text{left}(i, k_0)\)
    \(k_1 \leftarrow \text{right}(i, k_1)\)
    for \(t \in \{0, 1\}\) do
      if \(k_t \neq \text{done}\) then
        if \(\alpha_i^t + \alpha_k \geq 1\) then
          \(r_i^t \leftarrow r_i^t + (1 - \alpha_i^t) p_k\)
          \(k_t \leftarrow \text{done}\)
        else
          \(r_i^t \leftarrow r_i^t + \alpha_k p_k\)
          \(\alpha_i^t \leftarrow \alpha_i^t + \alpha_k\)
        end if
      end if
    end for
  end while
  return \(r_i^0, r_i^1\)
end procedure
```

Where \(\text{left}\) and \(\text{right}\) are procedures that, when called \(n\) times, return \(i + u_n\) and \(i - u_n\) respectively, where:

\[
u_n = 0, 1, -1, 2, -2, \cdots \quad (3.65)
\]

We were unfortunately unable to prove this expression. However, it has been proven [Bern et al., 1995] that this hull has at most \(2n\) points: we also generate \(2n\) points. Thus, if
those points are indeed the vertices of the projection we are close to optimality. To confirm that conjecture 1 is plausible, we generated random arrangements of points and force limitations. We compare the output of algorithm 1 to the output of the more general algorithm that will be presented in section 3.1.6. To do so, we compare the area of both polygons: our metric is the area of their symmetric difference normalized by the sum of their areas. The results are shown in fig. 3.3, and show that the output of algorithm 1 is very close to the output of our other algorithm, which means that there is a great probability that our conjecture is true. Indeed, 99% of the sampled arrangements result in a normalized error of less than $10^{-7}$, and all of them in an error smaller than $3.5 \cdot 10^{-7}$.

We did not pursue further this proof because a more efficient algorithm, in $O(n \log n)$ already exists [Bern et al., 1995] while algorithm 1 runs (in the worst-case) in $O(n^2)$. However, we feel that this more efficient algorithm is less legible than algorithm 1, and is much harder to implement as a custom data structure is necessary.

![Figure 3.3](image.png)

**Figure 3.3** Error between the **SHRINK** algorithm and the generic recursive projection algorithm on 1000 random samples.

As presented in the previous section 3.1.2, it is hard to construct analytical methods for this vertex enumeration problem. Hence, the two following sections will present general-purpose numerical methods to construct such projections.

### 3.1.5 Direct projection via vertex enumeration

This method uses the *double description*, therefore we briefly recall its principles.
3.1. Computation of the constrained stability polygon

Double description

The Weyl-Minkowski theorem [Weyl, 1934, Minkowski, 1897] states that any convex region defined by a finite number of inequalities is a convex polyhedron with a finite number of vertices. Hence, any convex region can be equivalently represented in two ways:

- **hyperplanes representation** (H-rep): a matrix $H$ composed of its normals and a vector $b$ of associated offsets. A point $a$ is interior to this region iff $Ha \leq b$;

- **vertices representation** (V-rep): a set of vertices $V$. A point $a$ is interior to this region iff there exists a vector $\bar{a}$ having the size of $V$ and with positive coefficients, such that $a = V^T \bar{a}$ with $\sum_i \bar{a}_i = 1$.

A number of algorithms and implementations are available to perform the H-rep to V-rep mapping and vice-versa. They are referred to as the double description algorithms, and vary in their capabilities and performance. Yet, none of them is inherently superior to the others. Notable examples are Qhull [Barber et al., 1996] that computes convex hull and half-space representations from vertices using the Beneath-Beyond algorithm; the Cddlib [Fukuda and Prodon, 1996b] and the Parma Polyhedra Library (PPL) [Bagnara et al., 2008] that both use the dual of the Beneath-Beyond algorithm. All of these were preceded by the Fourier-Motzkin elimination algorithm [Fourier, 1826, Motzkin, 1936] that has much worse performance but allows to solve more generic problems.

Unfortunately, all double description algorithms have super-exponential complexity in the worst case [Bremner et al., 1999, Joswig and Takayama, 2003].

Direct projection

Using the double description, we can now describe how to perform the direct projection of a set defined by convex constraints. It is a four-step process:

1. Reduce the problem to a finite set of linear inequalities and equalities;
2.Enumerate the vertices bounding that set (i.e. find the V-rep from an the H-rep);
3. Project the V-rep in the given space, resulting in another set of vertices.
4. Compute the convex hull of the previously obtained set of vertices.
In general only the second step is costly, but it can be prohibitively so. Indeed, the original inequalities may describe a set in a large-dimensional space, and double description methods behave badly in such spaces. Thus, this method is only practical when the original number of variables is small.

3.1.6 Recursive projection

Principle

This approach consists in solving a series of convex optimization problems that allows for iterative refinement of the convex projection without computing the high-dimensional convex polytope.

Let us consider two variables $x$ and $y$, spanning respectively the sets $E$ and $F$. Our objective is to find the projection of the region described by the convex constraints $\eta_l \leq \mathcal{C}(x, y) \leq \eta_u$ onto $E$, denoted $P(\mathcal{C}, E)$.

To do so, we first select a vector $d \in E$, the direction. Solving the following optimization problem:

$$\max_{x, y} \quad d^T x$$
$$\text{s.t.} \quad \eta_l \leq \mathcal{C}(x, y) \leq \eta_u$$

Will yield a point $x^*$ of $E$ that is extremal in that direction: any point further along $d$ will be out of $P(\mathcal{C}, E)$. Because $P(\mathcal{C}, E)$ is convex, we know that:

- The convex hull of all $x^*$, $\text{CONV}(x^*)$ is included in $P(\mathcal{C}, E)$.
- $P(\mathcal{C}, E)$ is included in the intersection of $\{ x \in E | d^T x \leq x^* \}$.

Thus, by iteratively solving (3.66) for various directions $d$ we can build an inner and an outer approximation of $P(\mathcal{C}, E)$. At the limit, when all directions have been enumerated, the inner and outer approximations will exactly match $P(\mathcal{C}, E)$. A rigorous proof of this result in the context of our algorithm will be given in section 4.3.1.

The main difficulty of this class of algorithms is the choice of the search direction $d$: appropriately choosing $d$ is paramount to obtain fast convergence. Another important point is to choose a stopping criterion. Indeed, to perfectly determine $P(\mathcal{C}, E)$ it is a priori necessary to solve (3.66) for every direction $d \in E$. If $E$ is two-dimensional or greater, we would have to solve an infinity of optimization problems. Thus, we will...
3.1. Computation of the constrained stability polygon

stop the algorithm after a finite number of steps, when a certain measure of precision has been reached.

Application to stability polygon computation

The algorithm in [Bretl and Lall, 2008] consists in building an inner and outer approximation of the true polygon, \( p \), by solving a sequence of second-order cone programs:

\[
\max_{c,f} \quad d^T c \\
\text{s.t.} \quad A_1 f + A_2(g)c = T(g) \\
\|B f\| \leq u(\mu)^T f
\]  

where (refer to appendix C for more details),

- \( f \) is the set of contact forces;
- \( c \) is the CoM position;
- \( d \) is the given search direction;
- \( A_1 \) is the contact matrix that sums all forces and moments;
- \( A_2 \) represents the gravity wrench at position \( c \);
- \( B \) is the friction cone matrix;
- \( u \) is the matrix representing the contact normals and the friction coefficient \( \mu \);
- \( T \) is the gravity wrench in function of the gravity \( g \) and the robot mass \( m \);
- \( \|\| \) is the euclidian norm.

The solution \( c^* \) is an extremal CoM position in the direction \( d \), and is thus added to the inner approximation. Conversely, the half-plane defined by \( \{ c \in \mathbb{R}^2 | d^T c < d^T c^* \} \) is a facet of the outer approximation. All points within the inner approximation are stable, those outside of the outer approximation are unstable, whereas the stability is undetermined for the points in-between the inner and outer approximations.

In statics, we consider that \( c \) lies in \( \mathbb{R}^2 \), thus the undetermined region between the inner and outer approximation is entirely composed of disjoint triangles. The next search direction is chosen to be perpendicular to the edge of the inner approximation that belongs to the triangle of maximum area and pointing outwards.
Chapter 3. Static equilibrium and interpolation

The algorithm stops when the difference between the outer and inner approximations area is less than a given precision $\sigma$. This area is exactly the sum of the triangles’ area. The authors prove that there is a strict upper bound on the number of iterations needed to reach $\sigma$.

In its original expression, this method is not applicable to the case of mixed unilateral-and-bilateral contacts, that we are using in some of our experiments [Vaillant et al., 2016]. The support region may be unlimited in some directions. Thus, we need additional constraints limiting the search region. As our model of bilateral contacts is simply a set of points with non-aligned friction cones, we still can use Equation 3.67, providing additional constraints on the acceptable CoM positions. Accordingly, we can either use a constraint $|c| < r_{\text{max}}$ that will constrain the CoM to a rectangular (polygonal) region or a conic constraint of the form $||c|| < r_{\text{max}}$ that will constraint the CoM to a circle. A circular constraint is difficult to approximate by a polygon. However, it may be closer to the real geometric constraint: using a maximum limb length per contact, we can write a series of constraints $||c - \iota|| < r_{\text{max}}$ that ensure that the CoM never goes further than $d$ from the contact at position $\iota$.

The stability region is unlimited in some directions because we can apply arbitrary forces on opposite contact points, allowing us to compensate for any momentum. In fact, the robot cannot apply infinite torques. Translating the torque constraint into a precise force limit of the form $|f| < f_{\text{m}}$ requires setting $f_{\text{m}}$ from the robot posture, whereas we reason on the CoM model to compute the stability polygon. Instead, we use a reduced torque constraint, $|[\iota] \times f| < \tau_{\text{m}}$ where $|[\iota] \times$ represents the cross product between the contact points and the contacting link. In this case, $\tau_{\text{m}}$ can be obtained from the characteristics of the link that is in contact (e.g. gripper actuators or feet/ankle actuators). This constraint is linear because we consider non-sliding contacts.

![Pie chart showing as angular sections the repartition of computation time for 150 iterations: total time 1.87 s](image)

This method is quite fast but not enough for real-time. It takes few seconds for over
3.2. Morphing stability polygons

a hundred iterations. However, it has two interesting properties compared with the direct projection method:

- It scales well with the number of contacts and the number of constraints.
- It is an *anytime* algorithm: even if stopped before completion, it will return a valid, albeit low-quality solution.

To solve the second-order cone programs, Equation 3.67, we use the convex optimisation package *CVXOPT* [Andersen et al., ] that natively supports conic constraints. Our implementation is sub-optimal as we recompute the area spanned by every edge at every iteration but the cost for solving each problem is by far the most consuming task. This means that even if we curb the time spent finding the next best edge, the overall runtime of the algorithm will not be significantly improved. Although customizing the solver [Boyd and Wegbreit, 2007] allows for faster resolution, our naive Python implementation (see appendix A) did not bring any substantial speedup.

### 3.2 Morphing stability polygons

Now, we are able to compute the stability region for each contact configuration and force limitation distribution. We want to be able to morph, or reshape this stability polygon to approximate two changes without recomputing the whole shape:

- Changes in the force distribution.
- Contact changes.

Indeed, if we do not enforce the CoM to be maintained within a moving/morphing polygon, the controller might start in an infeasible state. It is thus necessary to smoothly morph our stability polygons. Obtaining a morphing will pave the way for quick recomputation of the stability polygon during motion.

#### 3.2.1 What is a morphing?

We define a morphing as a *higher-order function*, i.e. a function returning a function. This function takes as input two polygons, and returns a function that takes a real
Chapter 3. Static equilibrium and interpolation

number in \([0, 1]\) and returns the interpolated polygon at that value.

\[
\mathcal{M} : \mathbb{P} \times \mathbb{P} \rightarrow [0, 1]^\mathbb{P}
\]
\[
p_a, p_b \mapsto \mathcal{M}(p_a, p_b)
\]
\[
\mathcal{M}(p_a, p_b) : [0, 1] \rightarrow \mathbb{P}
\]
\[
\kappa \mapsto p_\kappa
\]

The basic property of a morphing is that it must match \(p_a\) at the start, and \(p_b\) at the end i.e.:

\[
\mathcal{M}(p_a, p_b)(0) = p_a
\]
\[
\mathcal{M}(p_a, p_b)(1) = p_b
\]

For ease of notation, we introduce \(\mathcal{bM}_a\) as a lighter alternative to \(\mathcal{M}(p_a, p_b)\). The above morphing is too general to be of practical interest: we will introduce basic desirable properties of morphings.

**Definition 1.** A function \(F\) of \([0, 1]^\mathbb{P}\) is continuous at a point \(c\) if and only if:

\[
\forall \varepsilon > 0, \exists \delta > 0, |x - c| < \delta \rightarrow \|F(x) - F(c)\| < \varepsilon
\]

The norm on the polygon space can be chosen freely. A choice is the sum of distances between points and normals. Another choice is the (unsigned) area of the difference between polygons.

**Definition 2.** A morphing \(\mathcal{M}\) is continuous if and only if for every \(p_a, p_b\) their morphing \(\mathcal{bM}_a\) is a continuous function over \(\mathbb{R}\).

**Definition 3.** A morphing \(\mathcal{M}\) is convex if and only if for every convex \(p_a\), convex \(p_b\) and for every \(\kappa \in [0, 1]\), \(\mathcal{bM}_a(\kappa)\) is convex.

**Definition 4.** A morphing \(\mathcal{M}\) is right-bounded if and only if for every \(p_a \subseteq p_b\) and for every \(\kappa \in [0, 1]\), \(\mathcal{bM}_a(\kappa) \subseteq p_b\).

**Definition 5.** A morphing \(\mathcal{M}\) is monotonously increasing if and only if for every \(p_a \subseteq p_b\) and for every \(\kappa_1, \kappa_2 \in [0, 1]^2\), \(\mathcal{bM}_a(\kappa_1) \subseteq \mathcal{bM}_a(\kappa_2)\).

The above definitions extend naturally to:

- Smoothness by taking into account derivatives of the morphing.
3.2. Morphing stability polygons

- Left-boundedness.
- Generic monotonocity.
- Morphings in generic metric spaces.

For our application, continuity is the most important property. When changing contacts, one polygon is often included in the other: left and right boundedness will be required. For interpolating force limitations, monotonocity is needed.

Those three properties are difficult to enforce \textit{a posteriori} as they would require powerful regularization techniques. Only convexity can be approximated by taking the convex hull of the interpolated polygon at each interpolation step. Yet, this is not desirable as computing a convex hull is potentially costly. We will compute the morphing $b\mathcal{M}_a$ itself once, but we will evaluate $b\mathcal{M}_a(\epsilon)$ numerous times. Thus, costly operations should, as much as possible, be done in the construction of the morphing, not in its evaluation.

Morphing polygons is typically done in three steps:

1. Adding points to the shapes;
2. Finding a correspondence between the points from the start and target shape;
3. Generating intermediate shapes by interpolating each of these couples.

The main difficulty when morphing polygons is that if corresponding points are badly chosen, the resulting polygon will self-intersect during interpolation.

It is not necessary to use state-of-the-art 2D or 3D shape morphing found in the computer graphics and animation community, see e.g. [Sederberg and Greenwood, 1992, Alexa et al., 2000] because in this study we only morph 2D convex polygons. Furthermore, as we are interested in adding and removing contacts one at a time, successive stability polygons have a non-empty intersection. In particular, when adding a contact, the previous stability polygon is entirely included in the new one. Similarly, for small changes in the force limitations, the polygons will have a non-empty intersection. Thus, our successive polygons will be both convex and have non-empty intersections, which allows us to use a simpler, faster, morphing algorithm. In the following subsections, we first present two previous works that yield inappropriate morphings before our contribution in section 3.2.4.

We are mostly interested in the correspondence problem, and will use linear interpolation between couples of points: the smoothness of the morphing is guaranteed.
3.2.2 Angle-based morphing

This is possibly the simplest possible morphing. It relies on the fact that vertices of a convex polygon can be ordered in i.e. clockwise order.

It is a three steps process:

1. Move polygons to barycentric coordinates. Project the vertices on the unit circle.

2. Find on which (circular) edge of the other polygon lies every vertex of the one polygon. Compute the (circular) distance between this vertex and each extremity of the edge.

3. Reproject those vertices onto the original polygons using barycentric coordinates of the projected vertices.

This morphing transforms the original polygons which have respectively $n$ and $m$ vertices into polygons with $n + m$ vertices. Then, the interpolated polygon is composed of the interpolated matching vertices translated by the interpolation of the barycenters. Although its simplicity is appealing, this morphing is not convex. It also generates high local rotations. However, those problems disappear when one is included in the other.
3.2. Morphing stability polygons

3.2.3 EGI-based morphing

This morphing [Kamvysselis, 1997] relies on computing the Extended Gaussian Image [Horn, 1984] of the source and target polygons. The EGI represents a polygon as masses at the surface of the unit circle: the masses represent the length of an edge, and are positioned such that the normal to the circle at that point is the normal to the original edge. By doing so, the EGI transformation discards the translation information.

The algorithm itself is a three step process:

- Compute the EGI of the source and destination.
- For each source normal, compute a cost to transform it into each destination normal $c_{ij}$. This cost is the weighted sum of the difference of masses and the difference of angles.
- The source and destination scaling $s_{ij}$ and $s_{ji}$ are computed by $\frac{e^{-c_{ij}}}{\sum_i e^{-c_{ij}}}$ and $\frac{e^{-c_{ij}}}{\sum_j e^{-c_{ij}}}$ respectively.

The interpolated EGI is computed by linearly interpolating the scaled masses and scaled normal angular position. Finally, the polygon is obtained by taking the inverse EGI and translating it to the interpolated barycenter. Thus, the interpolations generates $nm$ vertices.
By definition, this morphing is convex as the interpolation takes place in the EGI space. However, it is non right-bounded, and thus non monotonous.

### 3.2.4 Optimal matching morphing

In this section we present Algorithm 2 to smoothly morph the stability polygons between stances.

**Algorithm 2** Morphing for interpolation of two polygons : \( f(P_s, P_d) \)

1. **Input:** \( P_s, P_d \) start and destination polygons, \( \kappa \) percentage
2. \( a_{cur}, done \leftarrow \{\}, \{\} \)
3. **while** \( |P_s| < |P_d| \) **do**
   1. \( P_s \leftarrow P_s \cup \text{midpoints}(P_s) \)
4. **end while**
5. \( M \leftarrow \text{zeros}(|P_s|, |P_d|) \)
6. **for** \( a_{tar} \in P_d \) **do**
   1. **for** \( b \in P_s \) **do**
   2. \( M_{ij} = \|a_{tar} - b\|_2 \)
   **end for**
7. **end for**
8. \( a_{cur}, done \leftarrow \text{Munkres}(M) \)
9. **for** \( b \in P_s - done \) **do**
   1. \( a_{tar} \leftarrow P^\perp(b, P_d) \)
10. **end for**
11. **return** \( (a_{cur}, a_{tar}) \)

To solve the problem of self-intersection during morphing, we use the Munkres (Hungarian) algorithm [Munkres, 1957]. This algorithm solves the assignment problem, i.e. finds a minimum weight matching in a weighted bipartite graph. In our case, we want to match every point from the target polygon to one from the current polygon while minimizing the sum of distances between each couple of matched points. In order to warrant that the current polygon has more points than the target polygon, we first extend the current polygon with the midpoint of each edge until it contains more points than the target. The result of the Munkres algorithm is the minimal, in terms of distance travelled, set of points \( a_{cur} \) from the current polygon matching all points \( a_{tar} \) from the target polygon. The remaining \( b \) points of the current polygon are mapped to their orthogonal projections onto the target polygon. The intermediate shape is generated by interpolating each couple of points by a percentage \( \kappa \). Figure 3.7 presents the results obtained on simple shapes.

The Munkres algorithm is in \( \mathcal{O}(n^3) \), yet pre-computations are possible. The start and
target polygons only depend on the geometry and friction properties of each stance. The algorithm returns the set of correspondences $a^{cur} \rightarrow a^{far}$. Then, to compute the interpolate, linear interpolation between each couple of points for a given morphing percentage $\kappa$ is used. Moreover, as we usually use a few tens of points, running the Munkres algorithm is not very costly. To map a 25-sided polygon onto a 50-sided one, it takes about 0.67 s. Solving the assignment problem takes about 95% of the time.

![Figure 3.7 Stability 2D polygon morphing: $P_s$ in blue, $P_d$ in red, the green and black arrows show the motion and the normals directions of the edges respectively.](image)

3.3 Linking force limitation and interpolation

From intuition, maintaining the CoM in a shape that is similar to the static stability polygon of a restricted set of contacts should reduce the force applied on the others. We can compute the real static stability polygon with an additional force constraint: for a set of real numbers $\gamma \in [0, 1]$ we compute the static stability polygon as described in section 3.1 with an additional constraint on the forces applied on each contact point of the grippers:

$$||f_i|| \leq \gamma mg$$

Then we compute two stability polygons at each step of the climbing, i.e. every time the active set of contacts is changed: one using all contacts, the other using all contacts but those between grippers and rail, and their interpolation. We then find by dichotomy which $\kappa$ yields the the interpolation of the “full” and “restricted” polygon with the closest area to the real constrained static stability polygon. Captions of this
process are shown in Figure 3.8 while a numerical comparison of values of $\gamma$ and $\kappa$ are shown in Figure 3.9.

Figure 3.8 Comparison between direct computation of the constrained static stability polygon and interpolation.

This shows us that interpolation is a good approximation of the force constraint with restrictions:

- Because we limit the search region, the interpolation retains the shape of this limit contrarily to the constrained static stability polygon.

- $\kappa$ is a non-linear function of $\gamma$. Indeed, as a consequence of the above point, there is a “dead-zone” when the force constraint results in a shape modification.
3.3. Linking force limitation and interpolation

Figure 3.9 Interpolation coefficient as a function of the force constraint coefficient for a 3-contact scenario. Dashed line represents $\kappa = 1 - \gamma$.

smaller than the distance constraint. However, this function does not have a general expression. Indeed, even when linearizing the friction constraints, the two-dimensional stability polygon is the result of a series of non-linear transformations. As our problem is defined as a series of inequalities describing contact geometry and friction and equalities describing stability conditions, the first step is to enumerate the vertices of the convex polyhedron hence described in the manifold of forces and CoM positions. Then, adding a linear force limitation constraint corresponds to capping this n-dimensional polyhedron, which may modify any number of edges, and will present discontinuities depending on which edges intersect with the constraint. Note that the two above operations can be computed in a single pass of double description, but capping would allow for efficient testing of various force limitations. Finally, the polygon is the convex hull of the projections of those vertices in the plane. Another non-linear, complex operation. It is thus impossible to derive a general relationship between $\gamma$ and $\kappa$ that is independent of the contact configuration. Moreover, as double description is an NP-hard problem [Khachiyan et al., 2008], enumerating all vertices in a high-dimensional space may be much more computationally costly than repeatedly computing only the projection, as the worst-case complexity is doubly exponential. Similarly, capping in high-dimensional spaces is much more expensive than in two or three dimensional space [Günther and Wong, 1991].

• Interpolation is more accurate for extreme values of $\gamma$ which is a direct consequence of using interpolation.
Thus, maintaining the CoM inside such a polygonal region corresponds to limiting the force applied on the gripper. This polygonal region is smoothly interpolated from the current static stability polygon, to an intermediate one. We can still maintain a task of the CoM in the cost function with low gains and limit the CoM motion in the constraint part of the QP. Note that we can use a static stability polygon because the accelerations and speeds generated by our controller are small and because the safety distance $\delta_s$ acts as a stability margin.

### 3.4 Integration

#### 3.4.1 Task-based controller

Now we discuss how this morphing is integrated in the controller. We control our humanoid robot using the multi-contact, QP task controller described in [Vaillant et al., 2016] and section 2.3, which compact form is:

\[
\begin{align*}
\min_{\ddot{q}, \lambda} & \quad \frac{1}{2} \begin{bmatrix} \ddot{q} \\ \lambda \end{bmatrix}^T Q \begin{bmatrix} \ddot{q} \\ \lambda \end{bmatrix} + \rho^T \begin{bmatrix} \ddot{q} \\ \lambda \end{bmatrix} = \sum w_i F_{X_i} \\
\text{s.t.} & \quad \eta_l \leq \mathcal{C}(\ddot{q}, f) \leq \eta_u
\end{align*}
\]

the decision variables are the joint accelerations $\ddot{q}$, and the intensity of the forces $\lambda$ applied at each contact point. The problem is written as a sum of quadratic or linear objectives weighted by $w_i$ (task weights). $\mathcal{C}$ linearly encodes joint position, velocity and torque limits, non-sliding contacts, and non-desired collisions avoidance. Typical tasks are written either directly or using the set-point objective formulation and comprise a wide variety of objectives [Vaillant et al., 2016]. Here, we focus on the CoM task.

To subscribe the CoM within a moving polygon, we use a collision constraint as in [Kanehiro et al., 2010, Vaillant et al., 2016]. However, in our case the CoM interacts with a changing shape, which means that we have to take into account the speed and acceleration of the edges w.r.t the CoM, that is:

\[
\begin{align*}
\dot{\delta} &= n^T \cdot (J \ddot{q} - \dot{p}) \\
\ddot{\delta} &= \ddot{n}^T \cdot (J \ddot{q} - \dot{p}) + n^T \cdot (J \dddot{q} + J \ddot{q} - \ddot{p})
\end{align*}
\]

where $\dot{p}$ is the position of the projection of the CoM on one such edge, $n$ the normal to that edge, $\delta$ the resulting distance, and $J$ the CoM Jacobian. We formulate a constraint:
3.4. Integration

\[
\begin{align*}
  u &= J \dot{q} - \dot{p} \quad e = J \dot{q} - \ddot{p} \\
  -d t n^T J \dot{q} &\leq \xi \frac{\delta - \delta_i}{\delta_i - \delta_s} + \delta + d t [n^T u + n^T e]
\end{align*}
\]

(3.76)

with \( \xi \) a damping coefficient, \( \delta_i \) and \( \delta_s \) the interaction and safety distances respectively. Activating the above constraint whenever \( \delta < \delta_i \) will ensure that \( \delta \) is never smaller than \( \delta_s \).

In a number of cases, the quantities \( \dot{p}, \ddot{p}, n, \dot{n} \) can be explicitly computed from the mapping obtained by applying Algorithm 2. Indeed for every ordered vertex \( v_k \) of our current (convex) polygon that is the interpolate between \( a^\text{cur}_k \) and \( a^\text{tar}_k \) at \( \kappa(t) \), \( t \) being the time:

\[
v_k(\kappa) = a^\text{cur}_k (1 - \kappa) + a^\text{tar}_k \kappa
\]

\[
\frac{\partial^n v_k}{\partial t^n} = (a^\text{tar}_k - a^\text{cur}_k) \frac{\partial^n \kappa}{\partial t^n} \quad \forall n \geq 1
\]

Thus, for every point \( p_k \) of the segment \([v_k, v_{k+1}]\) at a distance \( \alpha_k \) from the \( v_k \) and the associated normal \( n_k \):

\[
\frac{\partial^n p_k}{\partial t^n} = \sum_{i=0}^{n} \binom{n}{i} \frac{\partial^i \alpha_k}{\partial t^i} \frac{\partial^{n-i} v_k}{\partial t^{n-i}} + \frac{\partial^i (1 - \alpha_k)}{\partial t^i} \frac{\partial^{n-i} v_{k+1}}{\partial t^{n-i}}
\]

(3.77)

\[
n_k = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} (v_{k+1} - v_k) \overset{\text{def}}{=} (v_{k+1} - v_k)^\perp
\]

(3.78)

\[
= \left[ a^\text{cur}_{k+1} (1 - \kappa) + a^\text{tar}_{k+1} \kappa - (a^\text{cur}_k (1 - \kappa) + a^\text{tar}_k \kappa) \right] \perp
\]

\[
= (1 - \kappa)(a^\text{cur}_{k+1} - a^\text{cur}_k) \perp + \kappa(a^\text{tar}_{k+1} - a^\text{tar}_k) \perp
\]

(3.79)

\[
\frac{\partial^n n_k}{\partial t^n} = (n_{a^\text{cur}_k} - n_{a^\text{tar}_k}) \frac{\partial^n \kappa}{\partial t^n}
\]

(3.80)

Whenever two points of the current polygon map to the same point of the destination polygon, i.e. \( n_{a^\text{tar}_k} = \begin{bmatrix} 0 & 0 \end{bmatrix} \), the direction of \( n_k \) is constant but its norm is strictly decreasing. To avoid numerical issues whenever \( ||n_k|| \) is small, we consider \( n_k \) constant with respect to \( \kappa \) in this case.

Thus, knowing the \( n^{\text{th}} \) time derivative of \( \kappa \) and \( \alpha \), one can compute the \( n^{\text{th}} \) time derivative of the segments’ position and their normals. However, when \( \kappa \) or \( \alpha \) depend implicitly on the time, these quantities have to be evaluated through numerical differentiation. We can make the following approximations to neglect high-order
terms:

- As the CoM only interacts with edges closer than $\delta_i$, it usually interacts with few edges at a time. Because all points are interpolated uniformly and because the constraint pushes the CoM perpendicularly to the edge, variations of $\alpha_k$ are small during interaction: we consider that $\frac{\partial^n \alpha_k}{\partial t^n} = 0 \forall n > 0$.

- In most of our use-cases, we interact when the interpolation is almost done (the CoM is generally nearby the barycenter of the polygons). At this moment, $\ddot{p}$ is either zero (when we interpolate at constant speed) or negative (using quadratic interpolation, the polygon is contracting and slowing down): we set $\ddot{p} = 0$. This puts a harder bound on the constraint.

Overall rotation speeds are usually small compared to the homothetic part of the transformation, but we can not neglect $\dot{n}$. Depending on the shape of the support polygons, local rotations can be important, such as in the top corner of the polygon in Figure 3.7.

### 3.4.2 Limiting the CoM region during motion

For the sake of clarity and without loss of generality, we illustrate our approach using the stair climbing with handrail task by our humanoid robot (HRP-2Kai). In our experiment trials the robot applied too much forces on the handrail. Although part of the excess of force is due to positioning error and would be solved by applying lower-level compliance, part of it comes from the fact that our QP controller does not explicitly reduce the force applied on the gripper. Adding a task of the form $\min ||S_{gripper}\lambda||^2$ did minimize the force and torque computed, but kept the posture unchanged meaning that the force applied by the real, position-controlled, robot would remain high (this was also observed for a torque controlled robot in [Righetti et al., 2013], p. 288, last sentence). Thus, we want to modify the trajectory such that the force applied on the gripper diminishes without endangering stability.

### 3.4.3 Results of stairs climbing

To limit the stability region, we use the constraints of section 3.1 in the stability polygon computation. We strictly limit the torque applied on the gripper, and loosely the forces and CoM position using the following values:

$$|f_i| = |S_i x| \leq 5mg; |\tau_i| = |T_i S_i x| \leq 5N m; ||c|| \leq 1m$$
3.4. Integration

With $S_i$ a selection matrix of forces at the $i^{th}$ contact. The interpolation between the current and target polygon takes place over the course of one second. To avoid strong discontinuities when the constraint gets activated, we use quadratic interpolation, i.e. a trapezoidal speed profile. We also use relatively high damping, $\xi = 0.1$, to further avoid discontinuity when activating the constraint. To make sure the QP does not fail, we allow a small violation of the constraint. The intermediate polygon is the interpolate at $90\%$: we need to choose a high $\kappa$ because we want to strongly limit the forces applied on the gripper. This $\kappa$ corresponds to an upper force of about $34$ N, which means that in the worst case the torque applied on the gripper is lower than our torque limit.

\[ \delta w = w_{mes} - w_{obj} \]  
\[ X(t + dt) = X(t) + K_p \delta w + K_d \delta \dot{w} \]

With $K_p$, $K_d$ positive diagonal gain matrices, $w_{mes}$ the measured 6 dimensional wrench, $w_{obj}$ an objective wrench. This task allows us to align the contacting body.
with the contacting surface by specifying a pure normal force objective.

The resulting trajectory is shown in Figure 3.11: while the constraint was correctly added to our QP controller, it is rarely activated. This confirms that the manual tuning concords with the stability information. Note that during this testing phase, being sure that the CoM would never exit the polygon was very useful as it allowed us to tinker without worrying about changes in other tasks disturbing the CoM. In that sense, this technique is a complement to posture generation as it adds explicit boundaries in between the known to be static stances.

However, we can see in Figure 3.13 that the effective force applied during the motion is often superior to the designated force. Although part of this error stems from the fact that the polygons are not recomputed according to the actual contacts taken and more generally modeling errors, most of it is due to the fact that a low-force solution only exists, but it is not the one we get when doing raw position control.

### 3.4.4 Combining tasks and constraints for force control

In this section we present how to combine the CoM constraint and an impedance force control task to drive smooth removal of contacts. This is in fact a particular case of potentially more complex robot force control tasks. We also show that it is possible to recompute on the fly, but not in real-time the important polygons.

In a \( n \) contact scenario, we want to force control the \( i^{\text{th}} \) contact, \( \text{contact}_i \). We can...
3.4. Integration

Figure 3.12 Still frames of HRP2 successfully climbing the stairs using the handrail. The CoM motion is limited by the polygonal constraint.

then compute two stability polygons:

\[ P_s = \bigcup \{ \text{contact}_k \}_{k \in [0 \cdots n]} \]  \hfill (3.83)
\[ P_d = \bigcup \{ \text{contact}_k \}_{k \in [0 \cdots n]} \ k \neq i \]  \hfill (3.84)

We can then interpolate between \( P_s \) and \( P_d \) at a rate \( \kappa(t) \). We choose \( \kappa(t) \) to have a trapezoidal speed profile of total duration 30 s. Denoting \( f_k \) the force at the \( k^{th} \) contact, we can derive an objective force at that contact, \( \tilde{f}_k \) that is a function of \( \kappa \). Knowing that we only control one contact, we choose:

\[ \tilde{f}_k = mg(1 - \kappa) \]  \hfill (3.85)

We can then use an admittance wrench control task such as the one presented in subsection 3.4.2 to regulate the actual \( f_k \) around \( \tilde{f}_k \) by setting \( f_{obj} \) to \( \tilde{f}_k \) and zero wrench at each timestep. In reality, we only activate the admittance task objective whenever \( f_k > \tilde{f}_k \) to simply maintain \( f_k \) under the maximum desired force.
Chapter 3. Static equilibrium and interpolation

Figure 3.13 Force (magnitude) applied on left hand during stairs climbing. Data was low-pass filtered by an order 2, Butterworth filter with a critical frequency of 10Hz. For phases, \(a,m,r\) stand for “add”, “move” and “remove” respectively while \(R,L,F,H\) stand for “Right”, “Left”, “Foot”, “Hand”.

This linear objective in \(\kappa\) is in general sub-optimal as lower forces could be attained, but without an explicit relationship between \(\kappa\) and \(\gamma\), we have to remain conservative.

3.4.5 Application to a multi-contact setting

In this experiment, the controller establishes three contacts with the environment. Each of the two contacts that are added use the force control task to ensure parallism of the contacting body and surface. When done, the robot has established three unilateral contacts with its environment: one between the right foot and the rear flat platform, one with its left foot and an inclined ramp and the last one is with its hand and an elevated flat block.

After establishing the contacts, we reposition the CoM before starting the experiment. During those few seconds, we asynchronously compute the corresponding polygons: \(P_s\) and \(P_d\) for every contact as well as every interpolator.

While maintaining those contacts, we interpolate \(P_s\) towards \(P_d\) for every contact: first for removing the right foot, then for removing the left foot and finally for removing the right hand. Another shorter experiment presented in Figure 3.14 consists in doing only one interpolation, but then entirely removing the right foot.

In order to check that this technique would be applicable on the robot, we first tested...
3.4. Integration

it in the Choreonoid dynamics simulator [Nakaoka, 2012], which is very reliable (it embeds flexibilities at the ankle and noise) to ensure that the trajectory is indeed feasible and that the robot does avoid both environment and self-collisions.

Figure 3.14 Snapshots of the force control experiment with HRP-2 to smoothly remove the right foot. The CoM is smoothly pushed forward by the shrinking of the stability region, while the foot itself is admittance-controlled. No force distribution between the hand and left hand is given.

In this experiment, we do not use a CoM task, only the constraint with wide safety margin. This ensures smoothness and stability. We also add a very low weight posture task to avoid having uncontrolled joints. The last task is the admittance control task that is only inserted when interpolating for the first contact. This will allow us to confirm its importance and role. Note that without an explicit force regulation task, as the Hessian of the problem is positive definite, the controller will try to minimize $||\lambda||^2$ i.e. an equal distribution that does not reflect the real forces being applied.

Whenever we finish the interpolation (i.e. $\kappa = 1$), the CoM is inside the reduced support polygon of the supporting contacts: there exists a zero-force solution–at the contact to be removed–to support the robot in this configuration. Of course, we do not know a priori if this solution fulfills the torque limits of the robot, but as we started from a feasible configuration and smoothly drove the robot to the desired one while continuously reducing the force applied on the contact, it is to be expected.

We show in Figure 3.15 how the force did decrease continuously, both in simulation and in reality albeit with slightly different profiles. We reach a zero-force configuration whenever the interpolation ends for the first contact. Note that for successive contacts, this is no longer true. When we do not explicitly aim for the desired force distribution, the robot establishes itself in a configuration that balances the contact forces with the joint torques generated by the lower level joint position controller. Indeed, the worst case is shown in the third part, when interpolating to remove the hand: as the CoM is already almost inside the reduced polygon, the force applied on the right hand is only slightly reduced. However, it could safely be controlled to smoothly decrease towards
zero as the interpolation index decreases as was done for the foot.

Figure 3.15 Results of multi-contact experiment and simulation. Top: normal force applied on all contacts during motion. Bottom: interpolation index.

### 3.5 Conclusion

In this chapter, we show that it is possible to enforce stability as a constraint in low dynamics multi-contact transitions. We devised a continuous morphing of the stability polygon as part of the constraint in QP controllers. We use this technique to limit the force applied on a given contact, whether by directly restraining the accessible region of the CoM throughout the movement or by specifying the force to be applied
on a contact and displacing the CoM accordingly. Once the CoM is correctly placed, we can be sure that admittance or torque control is usable to select the appropriate force distribution. Although we still use postures generated from planning, it is an interesting addition to it, as it allows us to modify the generated motion according to our needs without any expensive re-computation. This is even more interesting as it allows us to specify that the CoM has to remain in a deformable stability region instead of explicitly forcing the CoM to remain at the planned position.

However, to ensure the feasibility of the motion we have to use hand-tuned stability margins in the form of safety distances from the boundary. Hence, in the next chapter we will explore how we can incorporate motion, i.e. acceleration, in this framework.
As presented in chapter 1, we are here interested in finding a region of stability $\mathcal{P}$ such that for every CoM position $c$ in $\mathcal{P}$, and for every CoM acceleration $\ddot{c}$ in a given acceleration polytope $\mathcal{G}$ there exists compatible contact forces.

Our approach has commonalities with [Barthélemy and Bidaud, 2008], which devised a method to check the robustness of a known trajectory and used linearized friction cones. [Caron et al., 2015] noted that such a region exists, but only sampled it. In [Or and Rimon, 2006] a vertical cross-section of this region was computed using a line-sweep algorithm, not unlike that of [Bern et al., 1995] but of course not directly transposable to 3D.

We will thus show the following:

- $\mathcal{P}$ is a three dimensional convex shape (section 4.1);
- It can be efficiently approximated by convex polyhedrons (section 4.3). Also, if $\mathcal{G}$ is limited to a convex polytope, it is the intersection of oblique prisms;
- Our construction algorithm can be modified to quickly test the equilibrium of many points (section 4.5);
- Changes in acceleration can be well approximated by morphing polyhedrons (section 4.6);
- We exemplify our stability criteria with a robust posture generation problem (subsection 4.7.1) and multi-contact motion (section 4.7.2).

This chapter is a generalization of the recursive projection algorithm [Bretl and Lall, 2008] that computes the static stability polygon $p$. The main difference is that we are constructing a 3D volume, as represented in Fig. 4.1.
Chapter 4. Robust static equilibrium

4.1 Robust static stability

The previously generated shape \( p \) gathers all the statically stable CoM positions. For a given set of contacts, we would like to know where the CoM can be, under a given set of contacts.
4.1. Robust static stability

of feasible accelerations\(^1\), so as to keep the forces within their friction cones. This is what we call the robust stability region, illustrated by \( \mathcal{P} \) in the Fig. 4.1.

### 4.1.1 Problem formulation

**Definition 6 (Robust static equilibrium).** A CoM position \( c \) is in robust static equilibrium with respect to a given residual radius \( r \) iff:

\[
\forall \tilde{g}_i \text{ such that } \|	ilde{g}_i\| < r, \exists f_i \text{ such that } \\
A_1 f_i + A_2 (g + \tilde{g}_i) c = T (g + \tilde{g}_i) \\
\|B f_i\| \leq u^T f_i
\]

(4.1) (4.2)

Where all notations are the same as Equation 3.67. Recall that \( A_2 \) represents the cross product with the gravity. In the non-robust case, its last line is nil as \( g \) is aligned with the vertical axis. Hence, the vertical component of \( c \) does not contribute anything, and (3.67) defines a two-dimensional shape. In the robust case, \( g + \tilde{g}_i \) will almost always not be aligned with the vertical axis and (4.1) defines a three dimensional shape. Moreover, this three-dimensional volume is an infinite intersection of convex shapes—one for each couple \((f_i, g_i)\), therefore it is a convex shape.

In [Barthélemy and Bidaud, 2008], \( \mathcal{P} \) is a given point, of a predefined CoM trajectory, at which they compute \( \mathcal{G} \). The latter is a sphere, of unknown radius \( r \), centered on the acceleration that is obtained for given contact forces. They provide an algorithm to compute \( r \) based on gravito-inertial wrench projections and use it as a robustness measure for the given trajectory.

In [Caron et al., 2015] robust static stability was envisioned as the set of CoM positions \( \mathcal{P} \), where a set of lateral accelerations \( \mathcal{G} \), centered around \( g \), could be generated by contact forces. There, \( \mathcal{P} \) is only sampled, whereas our method systematically computes \( \mathcal{P} \) and allows us to fast test robust equilibrium of any points (see section 4.5).

Another approach in [Prete et al., 2016] proposes a set of algorithms, including a faster version of [Bretl and Lall, 2008], to deal with postural robustness. However, the authors considered robustness to contact force errors. That is to say, for a given uncertainty of the force, they seek if it can be balanced by the others. They thus missed a point of the utmost importance: in the robust case, the region describing all possible CoM positions is no longer a 2-dimensional polygon, but a 3-dimensional polyhedron as

\(^1\)The said CoM accelerations can result from a perturbation, i.e. applied external forces, or from the control, i.e. from the actuators.
Chapter 4. Robust static equilibrium

described previously.

In Definition (6), we state the problem \( \forall \tilde{g}_i \). In order to compute \( \mathcal{P} \), it is necessary to reduce it to a tractable form. We do so using two different approaches. The first approach writes a more conservative constraint on (4.1) and (4.2). The second approach discretizes \( \mathcal{G} \).

The goal of both approaches is to obtain a finite set of convex constraints. Those constraints can then be used in a convex optimization problem. Combining this problem with the recursive projection algorithm [Kelley, 1960, Bretl and Lall, 2008] allows to compute the corresponding convex shape. Furthermore, if those constraints are linear they can also be used in a direct projection algorithm as in [Caron et al., 2015].

4.1.2 Formulating a stricter constraint

In this section, we derive a new inequality (depending on \( r \) but not individual \( \tilde{g}_i \) ), that will induce (4.2). This new constraint has the same form as (3.67). Let us consider the solution for the static problem:

\[
\begin{align*}
A_1 f + A_2(g) c &= T(g) \\
\|Bf\| &\leq u^T f
\end{align*}
\]

(4.3)

(4.4)

To do so, we introduce two bound vectors, \( l \) and \( s \). Please refer to appendix E for details and proofs of the following statements.

**Theorem 3.** A suitable lower bound \( l \) is given by:

\[
\begin{align*}
u^T f_i - u^T f &\geq l = -\mu r m\tilde{\sigma}(A_1^\dagger)(1 + \|c\|)
\end{align*}
\]

(4.5)

With \( \dagger \) the Moore-Penrose pseudo inverse and \( \tilde{\sigma}(A) \) the largest singular value of \( A \).

**Theorem 4.** A suitable upper bound \( s \) is given by:

\[
\begin{align*}
\|Bf_i\| - \|Bf\| &\leq s = r m\tilde{\sigma}(BA_1^\dagger)(1 + \|c\|)
\end{align*}
\]

(4.6)

**Corollary 1.** For a given CoM position \( c \), if there exist contact forces \( f \) that satisfy:

\[
\begin{align*}
A_1 f + A_2(g) c &= T(g) \\
\|Bf\| &\leq u^T f - r m(\tilde{\sigma}(BA_1^\dagger) + \mu\tilde{\sigma}(A_1^\dagger))(1 + \|c\|)
\end{align*}
\]

(4.7)
4.1. Robust static stability

Then \( c \) is in robust static equilibrium.

Proof. The proof is straightforwardly obtained by combining Theorem 4 and Theorem 3.

Unfortunately, (4.7) is in general not tight, and the converse of Corollary 1 is not true. Indeed, let us examine under which conditions we achieve tightness (refer to appendix E for more details):

1. \( \tilde{g}_i \) is collinear to the maximum singular vector of \( A_1 \)
2. \( \tilde{g}_i \) is perpendicular to the maximum singular vector of \( A_1 \)
3. \( \tilde{g}_i \) is perpendicular to \( c \)

The two first items are evidently incompatible. Moreover, \( c \) will be different at each iteration, but \( A_1 \) is a constant which means that no single \( \tilde{g}_i \) can even saturate two of the above conditions.

Even if this constraint is the tightest conic constraint (in terms of \( l_2 \)-norm) we found, it is still conservative. However, it allows to reduce the dimensionality of the problem substantially as we only need to find a single set of forces associated to a CoM position.

4.1.3 Discretizing the hypersphere

Instead of the conservative bound of Definition 6 described previously, we approximate the sphere \( \| \tilde{g}_i \| < r \) by a polytope whose \( k \) vertices are selected among \( \tilde{g}_i \), that is:

\[
g_i = g + \tilde{g}_i \quad i \in [0 \cdots k]
\]

(4.8)

The closer this polytope is to a sphere of radius \( r \), the closer the computed polyhedron will be to the real \( \mathcal{P} \).

The projection of the acceleration on the horizontal directions is not null, thus angular momentum is generated along every axis. Hence, we still cannot use the method in [Bretl and Lall, 2008]. Instead, we need to find 3D CoM positions that realize all \( k \) accelerations. That is, can we find \( k \) sets of forces \( F = [f_0^T \cdots f_k^T]^T \) that produce \( G = [g_0^T \cdots g_k^T] \) at a given CoM position \( c \)?
Proposition 1. If a CoM position \( c \) is in robust static equilibrium (\( c \in \mathcal{P} \)), there exists \( F \) that verifies:

\[
\Omega F + \Theta c = \Gamma \\
\|\Lambda F\| \leq \Upsilon F
\]  

(4.9)

With:

\[
\Omega = \text{diag}(A_1 \cdots A_1) \quad (4.10)
\]

\[
\Theta = \left[ A_2(g_0)^T \cdots A_2(g_k)^T \right]^T \quad (4.11)
\]

\[
\Gamma = \left[ T(g_0)^T \cdots T(g_k)^T \right]^T \quad (4.12)
\]

\[
\Lambda = \text{diag}(B \cdots B) \quad (4.13)
\]

\[
\Upsilon = \left[ u \cdots u \right]^T \quad (4.14)
\]

Proof. We stack \( k \) problems given by Definition 6, one for each \( g_i \). As a robust CoM position verifies the constraints of Definition 6 for any \( \tilde{g}_i \), it does also for any \( k \) of them. □

With this approach, the robustness of the static equilibrium \( \mathcal{P} \) is obtained w.r.t. the polytope \( \mathcal{G} \) defined by the \( k \tilde{g}_i \).

Remark. Contrarily to the bounds-based approach, this does not rely on the fact that \( \mathcal{G} \) is spherical. Indeed, robustness is obtained w.r.t. any convex polytope.

As we are looking for an explicit representation of \( \mathcal{P} \) we need to project the presented systems of equalities and inequalities into the CoM space, \( \mathbb{R}^3 \). In the next section, we show how the direct projection method is not suited to our particular problem. Efficient algorithms will be introduced in section 4.3.

4.2 Direct projection

\( \mathcal{P} \) is a 3D shape, and as such the 2D recursive projection algorithm presented in section 3.1.6 does not apply. A simpler method, that is applicable in any dimension is direct projection (see section 3.1.5), but it does not apply either in our case.

Indeed, direct projection means computing the boundary of a set, then project said boundary, yielding the boundary of the projection. In our case, this means computing
4.3 Computing the robust polyhedron

the boundary of the set defined by either (4.7) or (4.9). Such a problem is known as a vertex enumeration problem and is only applicable to sets defined by a finite number of linear constraints.

On the one hand, we can linearize (4.9) by using a polyhedral approximation of the friction cones. Unfortunately, a linear \( G \) with \( k \) vertices, leads to searching for \( k \) sets of forces. This is not an option since the double description has a worst case complexity that is exponential in the number of dimensions [Henk et al., 1997]. Plus, all iterative conversion algorithms are super-polynomial in the combined size of input and output [Bremner et al., 1999, Joswig and Takayama, 2003].

On the other hand (4.7) is of low dimension, but has non-linear terms on both sides; its linearization is far from being trivial. We thus investigate two alternatives: the recursive projection and the prism intersection.

4.3 Computing the robust polyhedron

4.3.1 Recursive projection

Principle

As presented in section 3.1.6, the recursive projection technique consists in approximating the projection of a convex set by linear boundaries [Kelley, 1960]. We seek for the approximation of \( P \) as a projection in the 3D space, of a higher dimensional space.

The core process of the recursive projection is to iteratively select directions \( d \), and solve optimization problems that yield extremal CoM positions \( c^* \) along that direction \( d \).

In the following section, we present two ways to formulate an optimization problem that yields the appropriate \( c^* \): (i) using bounds in subsection 4.1.2 and (ii) linearization in subsection 4.1.3.

We will afterwards present the whole 3D recursive projection algorithm including a stopping criterion and the choice of suitable search directions.
Chapter 4. Robust static equilibrium

Using bounds

Unfortunately, the formulation ((4.7)) obtained in subsection 4.1.2 is not suitable to an optimization problem. To turn it into a second-order cone program (SOCP), we need to transform (4.7) into two single conic inequalities by introducing an additional slack variable $\xi$:

\[
\begin{align*}
\|c\| &\leq \xi \\
\|Bf\| &\leq u^T f - rm(1 + \xi)(\bar{\sigma}(BA_1^\dagger) + \mu\bar{\sigma}(A_1^\dagger))
\end{align*}
\] (4.15) (4.16)

Then, solving the following optimization problem gives us an extremal point:

\[
\begin{align*}
\max_{c, f, \xi} & \quad d^T c \\
\text{s.t.} & \quad A_1 f + A_2(g)c = T(g) \\
& \quad \|c\| \leq \xi \\
& \quad \|Bf\| \leq u^T f - rm(1 + \xi)(\bar{\sigma}(BA_1^\dagger) + \mu\bar{\sigma}(A_1^\dagger))
\end{align*}
\] (4.17)

Using linearization

It is much simpler as the constraints presented in subsection 4.1.3 are already in a form that can be combined with a linear cost function to form a SOCP:

\[
\begin{align*}
\max_{c, F} & \quad d^T c \\
\text{s.t.} & \quad \Omega F + \Theta c = \Gamma \\
& \quad \|\Lambda F\| \leq \Upsilon F
\end{align*}
\] (4.18)

The projection algorithm

Given that we are dealing with convex 3-dimensional sets, a natural measure of size is the volume. Thus we use as a stopping criterion the volume difference between the inner and outer approximations.

To maximize the error reduction at each step, we took inspiration from [Bretl and Lall, 2008] and choose the search direction to be perpendicular to the facet of the inner approximation. We associate to each facet of the inner approximation a convex polyhedron called an uncertainty volume ($\nu_\sigma$ for short): it is the set of points that are outside that facet but inside the outer approximation. This $\nu_\sigma$ can be seen as a
4.3. Computing the robust polyhedron

cut of the outer approximation by the facet. A facet is defined by its normal \( n \) and its offset \( o \) as \( \{ c \in \mathbb{R}^3 | n^T c \leq o \} \). Thus, the \( \nu_\sigma \) is easily computed if an H-rep of \( \mathcal{P}_{outer} \), \((H_{outer}, b_{outer})\) is available:

\[
\nu_\sigma(facet) = \{ c \in \mathbb{R}^3 | c \in \mathcal{P}_{outer}, n^T c > o \} = \{ c \in \mathbb{R}^3 | H_{outer} c \leq b_{outer}, n^T c > o \} = \text{CUT}(\mathcal{P}_{outer}, \text{facet})
\] (4.19)

In [Bretl and Lall, 2008], the difference between the inner and outer approximation was a disjoint union of triangles. Therefore, computing their areas and adding and removing points and hyperplanes was easy. Indeed, a simple list of ordered vertices is sufficient and all operations can be performed without a dedicated algorithm.

Figure 4.2 One iteration of the algorithm. (a) Initial state: the outer approximation is a tetrahedron (transparent red). The search direction is perpendicular to the base: the extremal blue point is found, while the intersection by the new plane forms the three green points. (b) The inner approximation is now the blue tetrahedron. (c) Exploded view of the final state. The red tetrahedron is now outside of the outer approximation. The inner tetrahedron's facets form three prismatoids with intersecting trapezoidal bases (thick orange, thin green, and transparent grey) by cutting the remaining outer approximation.

In our case, we need to use the double description. Indeed, one iteration starting from the simplest volume, a tetrahedron, does not generate a union of disjoint tetrahedrons, but an union of intersecting prismatoids, see Figure 4.2.

For a given facet of the outer approximation, if we cut its \( \nu_\sigma \) with a parallel (to the facet) hyperplane, there is no guarantee that this hyperplane does not intersect another uncertainty volume.

Instead, we propose the following Algorithm 3:

- The input is the 6D contact positions and their friction coefficients.
Chapter 4. Robust static equilibrium

• We initialize $P_{\text{inner}}, P_{\text{outer}}$ and the list $\nu$ of uncertainty volumes, by solving (4.18) or (4.17). $\nu$ associates to each facet its uncertainty volume.

• For all facets, we check if they are invalid. This can happen in two cases: (i) the facet $k$ was added at the last iteration and has no $\nu_k$ or (ii) its $\nu_k$ was intersected by the plane added at the previous iteration. In both cases $\nu_k$ needs to be computed.

• We then select the direction $d$ to be the normal to the facet having the biggest uncertainty volume.

• Solve the SOCP (4.18) or (4.17).

• Resulting $c^*$ is added to the $P_{\text{inner}}$, and the plane normal to this direction passing by $c^*$ is added to $P_{\text{outer}}$.

• Invalidate the uncertainty volumes $\nu_i$ that are cut by the plane we just added.

Remark. Computing the volume of a set of points is equivalently difficult to finding their convex hull.

Indeed, to compute the volume of a random set of points, it is necessary to decompose it into a set of elementary volumes, the simplest being tetrahedrons. This is exactly what a convex hull of a set of points does. Similarly, finding the volume of a convex set defined by inequalities is equivalently difficult to enumerating its vertices. The convex hull operation is very well implemented in Qhull [Barber et al., 1996] when taking vertices as input. But, we found that its performance when taking inequalities as input is poor, so we used CDD [Fukuda and Prodon, 1996b]. However, Qhull was found to be slightly more numerically stable, and can be used in incremental mode, which makes it an attractive fallback. Still, numerical issues are inherent to the double description computations, and in pathological cases we switch to the PPL [Bagnara et al., 2008] that uses exact integer arithmetic and is much faster than CDD in this setting.

Proof of convergence

The following result holds.

Theorem 5. The sequences $(P_{\text{outer}})^n$ and $(P_{\text{inner}})^n$ converge towards $\mathcal{P}$.

Although it can seem obvious, we need to make sure that both the inner and outer approximations converge, that they have the same limit and that this limit is equal to $\mathcal{P}$. We thus introduce the following intermediate results.
Algorithm 3 Robust polyhedron computation by recursive projection

```
contacts ← the set of contacts
e ← the approximation difference precision

procedure ROBUST(contacts, e)
    \( P_{inner}, P_{outer}, v \leftarrow \text{INITIALIZE}(contacts) \)
    while \( V(P_{outer}) - V(P_{inner}) > e \) do
        for facet \( \in P_{inner} \) do
            \( UVf \leftarrow v[\text{facet}] \)
            if \( \text{INVALID}(UVf) \) then
                if \( UVf \) exists then
                    \( v[\text{facet}] \leftarrow \text{CUT}(UVf, \text{plane}) \)
                else
                    \( v[\text{facet}] \leftarrow \text{CUT}(P_{outer}, \text{facet}) \)
                end if
            end if
            \( \text{volumes}[\text{facet}] \leftarrow V(v[\text{facet}]) \)
        end for
        \( d \leftarrow \text{normal}(\arg\max(\text{volumes})) \)
        \( c^* \leftarrow \text{OPTIM}(d) \)
        plane \( \leftarrow \text{PLANE}(d, c^*) \)
        \( P_{inner} \leftarrow P_{inner} \cup c^* \)
        \( P_{outer} \leftarrow P_{outer} \cap \text{plane} \)
        \( \text{INVALIDATE}(v, \text{plane}) \)
    end while
    return \( P_{inner}, P_{outer} \)
end procedure
```

Lemma 1. \( (P_{outer})^n \) is monotonous non-increasing.

Proof. \( P_{outer}^{n+1} = \text{CUT}(P_{outer}^n, \text{plane}) \) thus, \( \forall a \in P_{outer}^{n+1}, a \in P_{outer}^n \) and \( P_{outer}^{n+1} \subseteq P_{outer}^n \).

Lemma 2. \( (P_{inner})^n \) is monotonous non-decreasing.

Proof. \( P_{inner}^{n+1} = \text{CONV}(P_{inner}^n, c^*) \) thus, \( \forall a \in P_{inner}^{n+1}, a \in P_{inner}^n \) and \( P_{inner}^{n+1} \subseteq P_{inner}^n \).

Lemma 3. \( \forall n \in \mathbb{N}, P_{inner}^n \subseteq P \subseteq P_{outer}^n \)

Proof. \( \forall n \in \mathbb{N}, c^* \in \mathcal{P} \). We denote \( \pi_n = \{c | d_i^T c \leq c_n^+ \} \). Thus, \( P_{inner}^n = \text{CONV}(c_i^* | i \in \{0 \cdots n\}) \subseteq \mathcal{P} \). Denoting \( \overline{c} \) the complement operator, we have \( \forall n \in \mathbb{N}, \pi_{c}^n \subseteq \mathcal{P}^c \). Thus \( P_{outer}^n = \bigcap_{0 \cdots n} \pi_n \supseteq \mathcal{P} \).

69
Chapter 4. Robust static equilibrium

We can now prove our main result:

Proof of Theorem 5. By the Lemmas 1 to 3, the sequences are monotonous and bounded: they converge. Thus, we can consider their limits, \( P_{\text{outer}}^{\infty} \) and \( P_{\text{inner}}^{\infty} \). At this limit, we have:

- \( c^* \in P_{\text{inner}}^{\infty} \)
- \( \pi \cap P_{\text{outer}}^{\infty} = P_{\text{outer}}^{\infty} \)

Thus, in that direction \( d \), \( V(\nu_d) = 0 \). As we choose the maximum volume for the stepping direction, \( \max_{\text{facet}} V(\nu) = 0 \). Then, \( \forall \text{ facet}, V(\nu_k) = 0 \). As the difference between \( P_{\text{outer}} \) and \( P_{\text{inner}} \) is exactly \( V(\bigcup \nu_k) \), \( V(P_{\text{outer}}^{\infty} \setminus P_{\text{inner}}^{\infty}) = 0 \). As both \( P_{\text{inner}}^{\infty} \) and \( P_{\text{outer}}^{\infty} \) are non-empty convex sets \( P_{\text{outer}}^{\infty} = \bigcup_{i \in \mathbb{N}} \pi^c_n \), thus \( P_{\text{outer}}^{\infty} \) is closed, and we have \( P_{\text{inner}}^{\infty} = P_{\text{outer}}^{\infty} \). As the sequence of \( c^* \) is included in \( P \), it is bounded, and thus compact. Then, as \( P_{\text{inner}}^{\infty} = \text{CONV}(c^*_n | n \in \mathbb{N}) \), \( P_{\text{inner}}^{\infty} \) is also compact, and we get \( P_{\text{inner}}^{\infty} = P_{\text{inner}}^{\infty} = P_{\text{outer}}^{\infty} \), which entails \( P_{\text{inner}}^{\infty} = P_{\text{outer}}^{\infty} = P \) by Lemma 3.

4.3.2 An intersection of prisms

The 3-dimensional shape described in Equation 4.17 is the intersection of \( k \) shapes. We show that each of these shapes is a prism, whose base can be computed as in Equation 3.1.6.

Let us consider a single \( \tilde{g}_i \) and the 3D vector, \( c_0 \) that describes the 2D CoM position in the plane \( c_z = 0 \):

\[
c_0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} c
\]

(4.20)

Let’s find which shape is described by the constraints:

\[
A_1 f + A_2 (g + \tilde{g}_i) c = T(g + \tilde{g}_i) \\
||B f|| \leq u^T f
\]

(4.21)

Note that \( A_2 \) is a function of \( g_i = g + \tilde{g}_i \), and it is the only part of the equation that
4.3. Computing the robust polyhedron

affects $c$. We know that:

$$A_2 c = \begin{bmatrix} 0 \\ -T(mg)c \end{bmatrix} = -m \begin{bmatrix} 0 \\ g \times c \end{bmatrix}$$  \hspace{1cm} (4.22)

Thus, for any $c = \begin{bmatrix} c_x & c_y & c_z \end{bmatrix}^T$:

$$T(mg)c = m \begin{bmatrix} -g_i^zc_y + g_i^yc_z \\ g_i^xc_z - g_i^zc_y \\ -g_i^yc_x + g_i^xc_y \end{bmatrix} = m \begin{bmatrix} g_i^yc_z \\ g_i^xc_z \\ -g_i^yc_x + g_i^xc_y \end{bmatrix} + m \begin{bmatrix} -g_i^zc_x \\ g_i^xc_z \\ 0 \end{bmatrix}$$  \hspace{1cm} (4.23)

$$= T(mg)c_0 + m \begin{bmatrix} g_i^yc_z \\ g_i^xc_z \\ 0 \end{bmatrix}$$  \hspace{1cm} (4.24)

Thus we can show that the solution at any altitude is the translated of the solution at $c_z = 0$. Indeed,

$$T(mg) \left( c_0 - \begin{bmatrix} g_i^yc_z \\ g_i^xc_z \\ 0 \end{bmatrix} \right) = T(mg)c_0 + m \begin{bmatrix} g_i^yc_z \\ g_i^xc_z \\ 0 \end{bmatrix} = T(mg)c$$  \hspace{1cm} (4.26)

Thus, for any $\hat{g}_i$, the shape described by Equation 4.21 is an infinite prism, whose base can be computed by finding the 2D polyhedron associated to a variation of the problem defined by (3.67):

$$\max_{c_2, f} d^T c_2$$

subject to

$$A_1 f + A_2 \phi c_2 = T(g + \hat{g}_i)$$

$$||Bf|| \leq u^T f$$  \hspace{1cm} (4.27)

Where $c_2$ is the 2-dimensional CoM position in the plane $c_z = 0$ and $\phi$ is a projection.
Chapter 4. Robust static equilibrium

matrix:

\[
\phi = \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0
\end{bmatrix}^T
\]  

(4.28)

The axis of this prism is thus given by the \( \rho_i \), collinear to the \( g_i \):

\[
\rho_i = \left[ \frac{g_i^x}{g_i^z} \frac{g_i^y}{g_i^z} 1 \right]^T = \frac{1}{g_i^z} g_i
\]  

(4.29)

Then, the resulting intersection of prisms can be easily computed as follows:

1. Compute an approximation of each base using (4.27) and the recursive projection in Equation 3.1.6.
2. Compute the convex hull of each prism using a fixed height (possibly large) for the prism.
3. Compute the intersection of the prisms. It is the V-rep of the stacked H-rep of each prism.

4.4 Comparative results

We tested the computation on a simple 3-contact scenario, using an acceleration polytope \( \mathcal{G} \) with 4 vertices. It is a “lozenge” whose vertices are generated from adding and removing an acceleration \( g_m \) to each lateral component of the gravity, that is \( g_i = [\pm g_m, \pm g_m, g] \). We use Algorithm 3 with the linearization (subsection 4.1.3): at each step, the problem contains 39 variables; 36 linear constraints that limit the maximum amplitude of each force component; 13 conic constraints that enforce friction and limit the CoM to a sphere of unit radius.

The resulting \( \mathcal{P} \)s are presented in Figure 4.4. The obtained \( \mathcal{P} \)s are not right prisms as those of the non-robust case (displayed for reference in Figure 4.6); they have a diamond-like shape instead.

When using the acceleration sphere developed in subsection 4.1.2, the polyhedron shrinks faster, see Figure 4.5. Indeed, the constraint (4.7) is isotropic: only the norm
Figure 4.3 The robust static stability polyhedron as an intersection of oblique prisms, for a residual radius $g_m = 3.55 \text{ms}^{-2}$ while the acceleration envelope is defined by $\mathcal{G} = \{\pm g_m, \pm g_m, g\}$. 
Chapter 4. Robust static equilibrium

Figure 4.4 Comparison of different $\mathcal{P}$ for different acceleration polytopes $\mathcal{G}$ generated by $g_m$. Only the inner approximations $\mathcal{P}_{\text{inner}}$ are rendered (in red), but are almost superimposed with $\mathcal{P}_{\text{outer}}$. The black dots with yellow cones represent contact points with associated friction cones.

Figure 4.5 Comparison of different $\mathcal{P}$ for different acceleration spheres of radius $r$. Only the inner approximations $\mathcal{P}_{\text{inner}}$ are rendered (in red), but are almost superimposed with $\mathcal{P}_{\text{outer}}$. The black dots with yellow cones represent contact points with associated friction cones.

Figure 4.6 Stability polygon in 2D (in red). The black dots with yellow cones represent contact points with associated friction cones.

of $c$ intervenes. Hence, as the residual radius increases, the polyhedron shrinks in all directions at the same rate. Because (4.7) uses maximum singular values, this shrinking rate is high. This shows that (4.7) is far from being tight, and should only be used when fast computation is paramount.
4.4. Comparative results

Indeed Algorithm 3 converges in around 1 s to an acceptable precision of 0.043 m$^3$ (1.02%), see Figure 4.7. The computation time can be split in two halves: (i) solving the optimization problems, and (ii) other operations.

The problem (4.17) has a constant number of variables, whereas the problem (4.18)’s size depends on the number of vertices of $\mathcal{G}$. As the other parts of algorithm 3 do not depend on the method of discretization, we compare the computation times of the optimization problems in table 4.1. It shows that reducing the number of variables makes a great difference in favor of (4.17).

This also proves that it is very important to limit the number of the double-description operations, that we do in the invalidation function in Algorithm 3.

What is not apparent in Figure 4.7 is that the computation time increases with the number of iterations. Indeed, as the algorithm progresses, more volumes have to be compared and potentially recomputed to find the best direction.

Figure 4.7 Computation times for 3 contacts, polytope with 4 vertices and 3 point contacts. 50 iterations, total time 1.29s (clock time, without interpreter set-up and teardown). Final error: 0.043 m$^3$.

To verify that our construction is correct, we compare those polyhedrons to the naive sampling approach. On the one hand, we compute the polyhedrons for 20 residual radiuses linearly spaced in the interval [0.55, 3.55] and for different linearizations of a spherical $\mathcal{G}$, from 4 to 18 vertices. On the other hand, we randomly select 100000 CoM positions $c_t$ in the bounding box of the polyhedron, and test them for robust stability by solving (4.18) at constant $c = c_t$ for a 18-vertices approximation of a spherical $\mathcal{G}$.

To compare those two objects, we introduce the symmetric difference metric:

$$
\delta(P_1, P_2) = \frac{V(P_1) + V(P_2) - 2V(P_1 \cap P_2)}{V(P_1) + V(P_2)}
$$

(4.30)
Table 4.1 Computation time of 50 SOCP problems when considering a spherical $G$ compared to the linearization. In the latter case, it is a function of the number of vertices of $G$.

<table>
<thead>
<tr>
<th>Number of vertices</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spherical (1)</td>
<td>0.372</td>
</tr>
<tr>
<td>4</td>
<td>0.628</td>
</tr>
<tr>
<td>6</td>
<td>1.23</td>
</tr>
<tr>
<td>8</td>
<td>1.70</td>
</tr>
<tr>
<td>10</td>
<td>2.58</td>
</tr>
<tr>
<td>14</td>
<td>5.01</td>
</tr>
<tr>
<td>18</td>
<td>8.22</td>
</tr>
</tbody>
</table>

This metric is the volume of the symmetric difference $P_1 \ominus P_2$ normalized by the sum of volumes. Thus, we compare the convex hull of the stable $c_t$ with the computed polyhedrons using the symmetric difference metric in fig. 4.8. It shows that at high number of vertices, and at any residual radius, the difference between the two objects is very small. This relative difference increases as the residual radius increases: as the polyhedron gets smaller, the bounding box shrinks, and thus the sampling density increases. Thus the sampling becomes more precise but not our approximation. Moreover, as we normalize by smaller quantities, this effect is amplified. Still, some error remains: because we only compute $P$ to a certain accuracy, some points that fall in the undetermined region are outside our approximation $P$ but still stable. This is also the reason why using 10 vertices seems slightly better than 18: the 10-vertex polytope corresponds to a slightly bigger $P$ that compensates for the width of the undetermined region.

To assess the convergence characteristics of our algorithm, we devised a worst-case scenario: $P$ would be a sphere (we took 6 contacts, one per each axis plus a limitation on the CoM or the contact forces). The results in Figure 4.9 show that the convergence is almost linear in the number of iterations. This does not compare to the nice quadratic result of [Bretl and Lall, 2008], which is expected because our problem is 3D and not 2D. Indeed, it has been proven [Gruber, 1997, Böröczky, 2000] that the error $\delta$ between a smooth $\zeta$-dimensional convex body $P$ and its best $n$-vertices polyhedral approximation asymptotically ($n \to \infty$) behaves as:

$$\delta \sim \gamma(\zeta, P) \frac{1}{n^{\frac{2}{\zeta+1}}}$$  \hspace{1cm} (4.31)\n
With $\gamma$ a constant that only depends on the dimension $\zeta$ and the shape of the approx-
4.4. Comparative results

Figure 4.8 Comparison between the convex hull of sampled points and the computed \( \mathcal{P} \) at various residual radiiuses for different number of vertices of \( \mathcal{G} \). Sampled points were tested for stability with \( |\mathcal{G}| = 18 \)

It is faster to compute \( \mathcal{P} \) using the prism intersection method (subsection 4.3.2). First, at each iteration we solve \( k \) SOCP, each of which is approximately \( \mathcal{O}(n^3) \) in the number of variables; whereas in Algorithm 3, we solve at each iteration a single problem whose size grows with \( k \) and has a complexity of about \( \mathcal{O}((nk)^3) \). Second, we need \( \mathcal{O}(\frac{1}{\epsilon}) \) iterations to reach the desired precision \( \epsilon \) in Algorithm 3, while only \( \mathcal{O}(\frac{1}{\sqrt{\epsilon}}) \) is needed using the prism intersection method. Last, with the prism intersection method, we do not need to keep track of the uncertainty volumes \( \nu \). In practice, it takes about 0.83 s to compute the same example as Figure 4.7 with a similar precision using the prism intersection method; in this case, most of the time is spent solving optimization problems. The intersection computation is not significant. Another advantage in using prisms is its robustness to numerical errors. On the other hand, it is not possible to target exactly a desired precision \( \epsilon \). Indeed, we do not know how large the intersection of prisms will be before computing it.
## 4.5 Incremental projection for testing

### 4.5.1 Why incremental projection?

Testing if a particular point $c_t$ is statically stable, can be done by solving the SOCP derived from Equation 3.67 as follows:

\[
\begin{align*}
\max_f \quad & 1 \\
\text{s.t.} \quad & A_1 f = T(g) - A_2 c_t \\
& \|B f\| \leq u^T f
\end{align*}
\]  

(4.32)

Hence, we would need to solve as many SOCP as there are CoM to be tested. To circumvent this, one can leverage the fact that a CoM position is in (resp. robust) static equilibrium iff it is inside the (resp. robust) static polygon (resp. polyhedron). To do so, a TEST-SAMPLE routine was introduced in [Bretl and Lall, 2008] for the static case. Their idea is to test if points are inside/outside of the static stability polygon, or fall in-between the inner and outer approximation. In that latter case, it is necessary to refine the approximations until the position of $c_t$ is determined.

The difference between TEST-SAMPLE and the polygon computation in Equation 3.1.6 is only in the choice of the search direction: the point being tested is outside one (and only one) of the facets of the inner approximation. Hence, the search direction is
4.5. Incremental projection for testing

chosen perpendicular to that facet.

By doing so, all the points are tested by two matrix multiplications. The result of the latter operation can tell us if points are in or out the stability polygon, otherwise they reside in-between and need refinement. This leads to an interesting observation: we can start with a very rough approximation of the stability polygon and refine it with the tests as needed.

We present how to adapt this methodology to $c_t \in \mathcal{P}$.

4.5.2 Polyhedral case

We need to update Algorithm 3 for testing. In 3D, testing $c_t \in \mathcal{P}$ can also be done by two matrix multiplications. However, when we need to update the approximations, $c_t$ may lie outside of multiple facets of the $\mathcal{P}_{\text{inner}}$. Randomly selecting a facet gives us a direction. Hence, we get rid of the bookkeepings related to maintaining a map of the facets to their corresponding uncertainty volumes $\nu$.

Alternatively, the direction can be selected to be perpendicular to the facet of $\mathcal{P}_{\text{inner}}$ forming the greatest uncertainty volume $\nu$ containing $c_t$. This however does not present any significant reduction in the number of iterations (i.e. the number of optimization problems being solved), but adds overheads due to the volume computations.

We thus choose a random selection for its efficiency.

4.5.3 Prism case

To test a point while using the prisms intersection method in subsection 4.3.2, we apply Algorithm 4, derived from Algorithm 3. It takes as input a set of polygons, $\{p_i\}$ that contain an inner and outer approximation. Each of these polygons uses a different $\tilde{g}_i$, and is computed using Equation 4.27.

For every operation, we do not compute the intersection of prisms; we only store the polygons that form their bases.

To test if a point is inside or outside one of the prisms, we compute its oblic projection...
along the prism axis, $\rho_i$, on the plane $c_z = 0$. We denote $\psi$ the projection operator:

$$c_2 = \phi(c_t - \rho_i c_t^z) = \phi c_t - \frac{c_t^z}{g_i} \begin{bmatrix} \tilde{g}_i^x \\ \tilde{g}_i^y \\ \tilde{g}_i^z \end{bmatrix} \triangleq \psi(\tilde{g}_i) c_t$$ (4.33)

$\phi$ is defined in Equation 4.28.

Then we check if $c_2$ is inside every polygon, or outside any polygon. If it is neither inside nor outside, we refine the appropriate polygons until $c_2$ is determined. If it can not, within a predefined iterations $\maxIter$, we reject it. In fact $\maxIter$ is reached only when a point is exactly on the boundary. Alternatively, the area of the current triangle can be used as a stopping criteria whenever it becomes very small.

**Algorithm 4**
Algorithm to test a sample while iteratively improving the underlying polygons

```plaintext
procedure TEST-PRISMS($\{p_i\}, \{\tilde{g}_i\}, c_t, \maxIter$)

nrIter ← 0
while nrIter ≤ maxIter do
    if $c_t \in \cap \{p_i^{\text{inner}}\}$ then
        return true, $\{p_i\}$
    else if $c_t \notin \cup \{p_i^{\text{outer}}\}$ then
        return false, $\{p_i\}$
    else
        for $p_i, \tilde{g}_i \in (\{p_i\}, \{\tilde{g}_i\})$ do
            $c_2 ← \psi(\tilde{g}_i) c_t$
            if $c_2 \in p_i^{\text{outer}}$ and $c_2 \notin p_i^{\text{inner}}$ then
                $p_i ←$ TEST-SAMPLE($p_i, c_2$)
            end if
        end for
    end if
end while
nrIter + +
return false, $\{p_i\}$
end procedure
```

**4.5.4 Results**

We use the examples defined in section 4.4 for testing one million uniformly sampled random points in a box of dimensions $1 \text{ m} \times 1 \text{ m} \times 3 \text{ m}$. We used the incremental version of Algorithm 3 presented in subsection 4.5.2. The results are illustrated in Figure 4.10. Note that as the number of query points increases, the number of the iterations –to determinate whether they are robustly stable or not– increases but at a
4.5. Incremental projection for testing

Figure 4.10 Number of iterations required to test random points compared to naive sampling. The blue line represents the average number of iterations, the shaded area corresponds to the minimum-maximum envelope. The red line represents the naive sampling: one iteration per query. The dashed lines represent the number of iterations necessary to reach the indicated precision using algorithm 3.

For testing, it is faster to use the polyhedral version rather than the prism intersection one. Indeed, when testing many samples, the performance bottleneck is no longer the number of iterations, but rather the time necessary to test a sample. In the polyhedral case, two matrix multiplications at most are necessary to test a point, while in the prism intersection, at most $2k$ multiplications are necessary. Moreover, to test a point in the prism intersection, it is necessary to refine all polygons in which the projected point lands between the inner and outer approximation until it is outside of at least one of them or all of them contain it.

On a mono-core, Python implementation, it takes about 40s to test one million points, but this could of course be greatly improved using parallelization. Parallel computing is impractical to build $\mathcal{P}$ in Algorithm 3 because each iteration will modify the approximations, but in the testing case, most samples can be tested independently.
Chapter 4. Robust static equilibrium

4.6 Extensions and discussions

4.6.1 Morphing and change in robustness

We know how to compute \( P \) corresponding to a given \( G \), which allows us to know which CoM positions are robust w.r.t this acceleration polytope.

![Figure 4.11 Morphing between two different robust static stability polyhedrons.](image)

Now, having \((G_0, P_0)\) and \((G_1, P_1)\), can we compute \( P_\kappa \) from \( G_\kappa = \kappa G_0 + (1 - \kappa)G_1 \), \( \kappa \in [0, 1] \)?

Both \( P_0 \), \( P_1 \) are the projection of a high-dimensional convex shapes that lives in the manifold of the contact forces and the CoM position. Thus, deforming \( G \) induces non-linear changes that are projected in the 3-dimensional space.

We propose to use interpolation (linear morphing) between convex polyhedrons to approximate changes in the acceleration polytope \( G_\kappa \). Hence, we consider scaling of the acceleration polyhedron, i.e. changes in the residual radius \( r \) but not in the linearization. To compute \( P_\kappa \), we use the morphing algorithm presented in [Kent et al., 1992]. Although the latter presents strategies to morph non-convex polyhedrons, at its core the algorithm is a 5-step process to find a common topology of two convex polyhedrons:

1. Get two polyhedrons (vertices and associated topology) and translate them to barycentric coordinates. Project them on the unit sphere.
2. Using neighbouring information to limit the search space, find every intersection of all spherical edges of the projections.
3. Order the intersections according to topological information and find in which facet of one polyhedron lie the vertices of the other.
4. For each vertex on the unit sphere, use barycentric coordinates to find its position on the original polyhedron.

5. Output the combined topology.

Then, we obtain intermediate polyhedrons by linearly interpolating between matching topologies. The Figure 4.11 shows an example of the results obtained by this algorithm.

We also compare the volumes obtained in Figure 4.4 with those resulting from the morphing. To compare the volumes, we select an interval \([r_l, r_u]\) and a number of percentages \(\kappa\). For each point we compute the exact polyhedron \(\mathcal{P}_e\) corresponding to the discretization of the robust sphere of radius \(r = (1 - \kappa)r_l + \kappa r_u\) and the interpolated polyhedron at \(\kappa\), \(\mathcal{P}_\kappa\).

We show in Figure 4.12 how \(\delta(\mathcal{P}_e, \mathcal{P}_\kappa)\) evolves as a function of \(r\), given different interpolation keypoints. The key main result is that morphing is a good fit (less than 12%) error as long as the distance \(r_u - r_l\) between keypoints is less than 2 m s\(^{-2}\). In this case, the problem becomes infeasible above \(r = 4.0\) m s\(^{-2}\) meaning that computing 3 keypoints is enough to cover the whole acceptable range of residual radiuses. However, a drawback of this morphing is that the obtained polyhedron is bigger than the actual one, which may lead to false positives when testing for robust stability.

### 4.6.2 Robust 2D and robust 3D

We present a serious limitation of the 2-dimensional robust approaches [Or and Rimon, 2006, Prete et al., 2016]: the robust static region is a 3-dimensional polyhedron, but they consider a 2-dimensional region, and then extend it to a right prism. We show that this approximation is a poor fit of the real volume, that only gets worse as the residual radius increases.

In order to compare our method to 2-dimensional approaches, we compare the volumetric computation to right (vertical) prisms whose bases are polygons. The right prism will have the same height as the diameter of the limiting sphere we use to bound the CoM accessible region, \(c_{\text{max}}\).

We first ensure that the volume that we compute is always included in the prism formed by the non-robust static stability polygon. We then compute a “robust static
Chapter 4. Robust static equilibrium

Figure 4.12 Comparison of morphing with direct computation for different keypoints.

stability polygon” at height \( h \), i.e. the recursive projection defined by:

\[
\Omega F + \Theta \left( \phi c_2 + \begin{bmatrix} 0 & 0 & h \end{bmatrix}^T \right) = \Gamma
\]

\[
||\Lambda F|| \leq Y F
\]

where \( c_2 \) is a 2D vector, \( \phi \) is defined in Equation 4.28.

We use our previous examples, i.e. the same contacts and \( \mathcal{G} \) than in section 4.4, also for 20 residual radiiues linearly spaced in the interval [0.55, 3.55], and for five different heights we compute both forms (right prism and \( \mathcal{P} \)). We then compute the error metric \( \delta \) between the right prism and the 3D volume at each residual radius. We ensure that both computations lead to the same result by comparing the intersection
4.6. Extensions and discussions

of $\mathcal{P}$ and the plane $c_z = h$ with the robust static stability polygon (2D). We use the same error metric in (4.30), replacing the volume with the area. An example of the objects being compared is in Figure 4.13 while results are shown in Figure 4.14. The area error remains low (less than 3%) at any margin while the volume error shoots towards 100%.

This confirms that using a 2D approach is equivalent to approximating the 3D polyhedron by a prism formed by its section at some constant height $c_z = h$. This leads to a poor approximation of the real volume because it is not similar to a prism, and this similarity only diminishes with the increase in the stability margin. Note that at any margin the robust 3D polyhedron is not included in the prism formed by the “robust polygon” so the “robust prism” is neither a conservative nor optimistic approximation.

![Figure 4.13 Comparison between: the statically stable prism in light blue, the robust static polyhedron with a $g$ generated using $g_m = 2.6 \text{ m s}^{-2}$ in red and the prism formed by the robust polygon at the same margin and height 0.1 m in dark blue. Note that the scale is non-uniform across axes.](image)

4.6.3 Recent work

Our approach is in a way the dual of the that shown in [Caron and Kheddar, 2016]: instead of limiting the CoM positions to a known convex polytope and finding the envelope of acceptable CoM accelerations, we propose to limit the CoM acceleration and find all acceptable CoM positions.

It is also more general than what is presented in [Prete et al., 2016]: the authors only consider errors on the contact forces without considering that the resulting acceleration could change.
Very recent works, developed in parallel to our approach have pointed at ways to compute the proposed region: [Nozawa et al., 2016] showed that when the CoM acceleration is not null, the CoM accessible region, nicknamed CFR was an oblique prism, while [Dai and Tedrake, 2016] has shown how to use a polyhedral or ellipsoidal region, but this time to limit the CoM position.

### 4.7 Integration

#### 4.7.1 Case study in multi-contact posture generation

**Posture generation**

In this section, we explore how the idea of robust static stability can be applied to posture generation. Indeed, a typical posture generation problem aims at finding a statically stable robot stance. We show how we can modify such problems to find robust statically stable stances and how this affects the resulting stances.

We use the posture generation framework (PG) developed in [Brossette et al., 2015, Brossette et al., 2016] and add an extra constraint to maintain the CoM in the robust static stability polyhedron. The core of the PG is to solve a non-linear optimization problem in the generalized robot coordinates $q$, and the contact forces $f$. We can thus
add our constraint by specifying:

\[ Hc(q) \leq b \]  \hspace{1cm} (4.36)

Where \( H \) and \( b \) represent the H-rep of the robust static polyhedron. Other constraints encode:

- Contacts fixed to pre-defined locations;
- Forces in friction cones;
- Both torque and position limits for every joint;
- Self-collision constraints.

The objective function is the distance to a usual posture.

It is also possible to extend the state of the posture generator to \((q,F)\) and directly
encode the robust stability by specifying that for every $f_i \in F$:

$$A_1(q)f_i + A_2(\tilde{g}_i)c(q) = T(\tilde{g}_i) \tag{4.37}$$

In this form, the contact positions are functions of $q$ and thus $A_1$ is also a function of $q$. Thus, contact positions would be determined by the optimization process instead of being fixed. However, dimensionality remains a problem in non-linear programming, and extending the state increases drastically the computation times. Therefore, we only consider the effects of adding a constraint of the form (4.36).

**Results**

We generated three random contact positions: one for each foot and one for the right arm. Then we computed three pairs of postures using:

- Both feet;
- Both feet and the arm;
- The left foot and the arm.

In each pair, the first posture is generated using the static stability constraint while the second one is generated using the same problem plus a constraint of the form (4.36). The results are shown in Figure 4.15. They qualitatively highlight a problem with the regular static stability: the solver has a tendency to stop on the edge of the constraint which is an unstable equilibrium position. The postures generated with the robust static stability look more resilient to external perturbations.

Note that in those simple examples, using (4.37) led to solver failure in two out of three cases, even though the contact positions were fixed.

### 4.7.2 Multi-contact Control

The previous subsection used the robust stability region as a constraint in single posture generation. Similarly to section 3.4.2, we will again climb the stairs using the handrail, but this time while maintaining the CoM in the robust statically stable region.

As in section 3.4.2, we use a continuous morphing between successive polyhedrons and a collision constraint between the CoM and the interpolated planes to achieve the
motion. However, the morphing presented in the previous section 4.6 is not suitable for our purposes:

- The morphing generates a lot of small facets, whose normals are not always well-defined.
- The morphing does not guarantee that facets will not flip during morphing.
- Computing the matching is rather slow.

Thus, we will use a simpler morphing.

**Inequality morphing**

![Figure 4.16 Illustration of the inequality morphing from $P_0$ (dashed) to $P_1$ (dot-dashed). The intermediate dilatations are in solid black. The interpolate at $\kappa$ is filled in transparent red.](image)

Knowing that in most cases we will interpolate between polyhedrons that are included in one another, or that have at least a non-empty intersection, we no longer look for a morphing that is globally monotonous. Instead, we create a morphing between $P_0$ and $P_1$ that monotonously decreases (shrinks) towards $P_0 \cap P_1$ and then monotonously increases (dilates) towards $P_1$. This morphing is illustrated in fig. 4.16, using simple 2D ellipses for $P_0$ and $P_1$.

We introduce $H_0, b_0$ the H-rep of $P_0$ (resp. 1) and $V_0$ the V-rep of $P_0$ (resp. 1). Our
morphing is:

\[
\mathcal{I} : \mathbb{P} \times \mathbb{P} \rightarrow [-\kappa, \kappa]^P
\]

\[
(\mathcal{P}_0, \mathcal{P}_1) \mapsto \mathcal{I}(\mathcal{P}_0, \mathcal{P}_1)
\]

With:

\[
\forall \kappa \in [-\kappa_0, \kappa_1], \ \mathcal{I}(\mathcal{P}_0, \mathcal{P}_1)(\kappa) = \begin{cases} 
\kappa \leq 0 & : \left[\begin{array}{l}
H_0 \\
H_1
\end{array}\right], \left[\begin{array}{l}
b_0 \\
b_1 - \kappa
\end{array}\right] \\
\kappa > 0 & : \left[\begin{array}{l}
H_0 \\
H_1
\end{array}\right], \left[\begin{array}{l}
b_0 + \kappa \\
b_1
\end{array}\right]
\end{cases}
\]

(4.39)

This is equivalent to returning the intersection of the source polyhedron and a dilatation of the target polyhedron when \( \kappa \) is negative, and returning the intersection of the target polyhedron and a dilatation of the source polyhedron when \( \kappa \) is positive. When \( \kappa = 0 \), the interpolate is exactly \( \mathcal{P}_0 \cap \mathcal{P}_1 \). Thus, \( \mathcal{I} \) defines a morphing iff \( \mathcal{I}(\mathcal{P}_0, \mathcal{P}_1)(-\kappa_0) = \mathcal{P}_0 \) and \( \mathcal{I}(\mathcal{P}_0, \mathcal{P}_1)(\kappa_1) = \mathcal{P}_1 \).

We thus define:

\[
\kappa_0 = \max (H_1 V_0 - b_1)
\]

(4.40)

\[
\kappa_1 = \max (H_0 V_1 - b_0)
\]

(4.41)

This ensures that each vertex of \( \mathcal{P}_0 \) is inside \( (H_1, b_1 + \kappa_0) \) and vice-versa. Thus the intersection of \( \mathcal{P}_0 \) and \( (H_1, b_1 + \kappa_0) \) is \( \mathcal{P}_0 \) and vice-versa.

Finally, we can obtain a proper morphing \( \mathcal{I}' \) by linearly mapping \([0, 1]\) onto \([-\kappa_0, \kappa_1]\).

This morphing allows for very fast computation: only two dot products are necessary to compute the necessary \( \kappa_{0,1} \). It also generates no superfluous inequalities. Finally the derivatives of the planes w.r.t to time are extremely simple:

\[
\frac{\partial p^0_i}{\partial t} = \begin{cases} 
\kappa \leq 0 & : 0 \\
\kappa > 0 & : \frac{\partial \kappa}{\partial t} h^0_i
\end{cases}
\]

(4.42)

\[
\frac{\partial p^1_i}{\partial t} = \begin{cases} 
\kappa \leq 0 & : -\frac{\partial \kappa}{\partial t} h^1_i \\
\kappa > 0 & : 0
\end{cases}
\]

(4.43)
4.7. Integration

i.e. the derivative of each plane is simply the derivative of \( \kappa \) along its normal. Thus, every plane acceleration is zero, as well as normals speed.

A disadvantage of this method, other than the non-monotony, is that we return an H-rep, thus the behaviour of the vertices is largely unpredictable. For example, we will need to compute a V-rep of each interpolate for visualization.

**Results**

As in section 3.4.1, we constrain the CoM to remain within the deformable stability polyhedron by adding a collision constraint in our QP controller. To reduce the length of the interpolation interval, \([-\kappa_0, \kappa_1]\) we will compute \( \kappa_0 \) based on the current CoM position when switching stances:

\[
\kappa_0 = \min (\kappa_0, H_1 c - b_1 + \delta_i) \tag{4.44}
\]

That is to say, we set the initial dilatation of the target polyhedron to at least contain the CoM with a margin \( \delta_i \) that is the interaction distance of our collision constraint. This makes sure that we interpolate over the smallest distance (in terms of \( \kappa \)) while not triggering brutal constraint changes.

We also need to ensure that the CoM acceleration, \( \ddot{c} \) remains in the given acceleration polytope \( \mathcal{G} \). Given a H-rep \( (H_\mathcal{G}, b_\mathcal{G}) \) of \( \mathcal{G} \), we add the following linear constraint to our quadratic program:

\[
H_\mathcal{G} \left( J \ddot{q} + \dot{J} \dot{q} \right) \leq b_\mathcal{G} \tag{4.45}
\]

\[
\iff H_\mathcal{G} J \dot{q} \leq b_\mathcal{G} - H_\mathcal{G} \dot{J} \dot{q} \tag{4.46}
\]

Using those two constraints, we perform stairs climbing as in section 3.4.2. A few frames are presented in fig. 4.17: in each one of them the constraint is activated and “pushes” the CoM in order to avoid falling backwards. This allows us to check that the motion remains feasible, i.e. the constraints we added are compatible with the whole-body motion. Moreover, it also serves as a check that compatible contact forces indeed exist when \( c \in \mathcal{P} \) and \( \ddot{c} \in \mathcal{G} \). However, enforcing (4.46) strictly leads to infeasibility when changing constraints. Yet, only a very small proportion of points (less than 0.5%) do not satisfy (4.46). Those points usually correspond to the activation of the polyhedral constraint: as there is no anticipation, acceleration impulses are necessary to satisfy (4.46). We will show in the next chapter that it is possible to use preview control to mitigate this problem.
Chapter 4. Robust static equilibrium

Figure 4.17 Climbing the stairs under robust static stability constraints. In this scenario, the weight of the CoM task is set to the lowest possible, 10. The current interpolate of the polyhedron is depicted in transparent green. Note how the robot tilts backwards: the CoM is only pushed forward by the constraint.

Table 4.2 Influence of the CoM task weight on success of stairs climbing, with and without the polyhedral constraint activated

<table>
<thead>
<tr>
<th>CoM Task Weight</th>
<th>CoM Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Activated</td>
</tr>
<tr>
<td>1.0</td>
<td>✗</td>
</tr>
<tr>
<td>10.0</td>
<td>✓</td>
</tr>
<tr>
<td>50.0</td>
<td>✓</td>
</tr>
<tr>
<td>100.0</td>
<td>✓</td>
</tr>
<tr>
<td>500.0</td>
<td>✓</td>
</tr>
</tbody>
</table>

When performing the motion, we typically have a CoM task that servo's the CoM around the output of the multi-contact planner. Adding our constraints allow us to reduce this weight, and still be able to generate a feasible motion. Thus, the CoM task will less the conflict with other tasks. The results presented in table 4.2 indeed show that when the CoM weight is lowered below its nominal value of 500, the motion fails. Indeed, when the weight is lowered, the controller allows the CoM to move away from the desired position, which leads to falling. However, with the addition of our constraint, the motion remains feasible until the CoM task weight is inferior to 10. Under that value, the controller drives the robot to regions where the torque limits conflict with the polyhedron constraint: the controller can no longer recover.

4.8 Conclusion

In this chapter, we have thoroughly explored the notion of robust static stability polyhedron: its nature and properties, how to compute it and how to use it for testing
robust static equilibrium. We have also shown how to approximate variations in the residual radius with polyhedral morphing.

An interesting property is that it is no longer a right prism: approximating this shape by simply shrinking the static stability polygon is not correct as the height of the CoM now plays a role.

We have shown that this robust static stability polyhedron can be efficiently used for planning and control. However, we could not implement the acceleration constraint in the control framework because of a lack of anticipation. To solve this problem, the next chapter will introduce a model preview control formulation based on the robust static stability polyhedron.
As presented in chapter 1, in order to perform multi-contact dynamic motion, a possibility is to look ahead. More precisely, being able to forecast at least one or two contact sequences ahead is essential to properly exploit robot dynamics through transitions. This is why we present a model-preview approach, that allows to look ahead, across contact changes. Whole-body model preview control (MPC) is still too computationally intensive to be used in a closed-loop scheme. Thus, most MPC approaches employ a two-level scheme:

1. Compute quickly the dynamic motion over a time horizon by means of a simplified dynamic model that captures the essence of the whole body dynamics.

2. Provide the CoM computed trajectory to be tracked at best by whatever chosen local low-level whole body motion planner/controller

CoM-based models appeared, until recently, to be difficult to extend beyond walking. In this chapter we will present two MPC approaches that are applicable in multi-contact.

The first one is based on the work of Nagasaka et al. [Nagasaka et al., 2012] and our previous work [Audren et al., 2014]. It presented an elegant formulation of MPC using the CoM for multi-contact. To linearize the dynamics, we assume that the acceleration along a particular axis is known. This allows us to realize fully dynamic motions but comes at a cost: as the problem is large, we have to rely on a spatial heuristic to determine contact transition timings.

The second one is based on chapter 4: by restricting the possible CoM accelerations, we can linearize the CoM dynamics. This allows us to easily generate a CoM trajectory
as we do not have to check for the existence of contact forces. Moreover, we can exploit this reduced computational cost to look for automatic timings.

5.1 Model Predictive Control

5.1.1 General principle

Model Predictive Control (MPC) was designed for industrial plant control applications, mostly chemical. See for example [García et al., 1989] for a survey. Those applications were motivated by large delays and/or high order dynamics present in the process. In robotics, a bad movement can lead to falling a few seconds later, compared to typical control rates of 500 Hz. In that sense, we have significant delays in the plant dynamics: we need to account for the future consequences of our current control.

To properly introduce the notion of MPC, let’s consider a generic system: its state is \( x \) and its command is \( u \). The state evolution differential equation is then:

\[
\dot{x} = f(x, u) \tag{5.1}
\]

Then, the state \( x \) at a time \( t \) is:

\[
x(t) = \int_0^t f(x(s), u(s)) \, ds \tag{5.2}
\]

As we only have an approximate model \( \hat{f} \) of the system dynamics, we can derive an expected state \( \hat{x} \):

\[
\hat{x}(t) = \int_0^t \hat{f}(\hat{x}(s), u(s)) \, ds \tag{5.3}
\]

Meaning that it depends on all values of \( u \) over \([0, t]\) and \( \hat{x}(0) \). Note that modelling errors between \( \hat{f} \) and \( f \) are accumulated by the integration. We can then find a optimal command \( u \) over \([0, t]\) that minimizes some cost \( \mathcal{F} \) while respecting some constraints \( \mathcal{C} \):

\[
\min_{u(s), s \in [0,t]} \mathcal{F}(x, u) \tag{5.4}
\]

s.t. \( c_l \leq \mathcal{C}(x, u) \leq c_u \)

The insight of model-predictive control compared to optimal control is that if \( u^* \) can be computed “fast enough”, only the beginning of the control sequence \( u \) is used.
While this part of the trajectory is executed, another round of optimization can be launched from the current state $x_0$. This allows the MPC to compensate for errors in the modelling, as it is always restarted from a current state. For this scheme to be applicable, the resolution time of (5.4) must be small compared to the horizon length $t$ and not too important compared to the controller period. Indeed, if one of those conditions is not realized, the real output $x$ will drift too much compared to the model output $\hat{x}$. To ensure fast computation, two main considerations are in order:

- The cost $\mathcal{F}$ and by extension $f$ should be as simple as possible (ideally linear, or at least convex) to allow for fast optimization. Similarly, the constraints $\mathcal{C}$ should be as simple as possible.

- The number of variables, that depends on the size of the state $x$ and the duration of the trajectory $t$ should be kept small.

To that end, we will focus on discrete-time linear systems, as they allow to formulate fast quadratic programs.

### 5.1.2 Linear discrete-time MPC

Let us consider a discrete-time, linear system:

$$\dot{x}_{k+1} = A_k \dot{x}_k + B_k u_k$$

Where $\dot{x}_k$ is the system state at instant $k$, while $u_k$ is the command at instant $k$. $A_k$ represents the integration of the state while $B_k$ is a mapping from the command space to the state space.

We can then apply (5.5) repeatedly to obtain every state in a finite horizon, composed of $K$ steps. Let $X = \begin{bmatrix} \dot{x}_1^T & \cdots & \dot{x}_K^T \end{bmatrix}^T$ and $U = \begin{bmatrix} u_0^T & \cdots & u_{K-1}^T \end{bmatrix}^T$, we obtain:

$$X = \Phi \dot{x}_0 + \Psi U$$

Where $\dot{x}_0$ is the given initial state.
Chapter 5. CoM Predictive Control

With:

\[
\Psi = \begin{bmatrix}
B_0 & 0 & 0 & 0 & \ldots \\
A_1 B_0 & B_1 & 0 & 0 & \ldots \\
A_2 A_1 B_0 & A_2 B_1 & B_2 & 0 & \ldots \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
\prod_{i=1}^{K-1} A_i B_0 & \prod_{i=2}^{K-1} A_i B_1 & \ldots & A_{K-1} B_{K-2} & B_{K-1}
\end{bmatrix} \tag{5.7}
\]

\[
\Phi = \begin{bmatrix}
A_0 \\
A_1 A_0 \\
\vdots \\
\prod_{i=0}^{K-1} A_i
\end{bmatrix} \tag{5.8}
\]

We can then define a quadratic cost in commands \( U \) to be minimized while tracking a given reference state trajectory \( \bar{X} \):

\[
\mathcal{F} = (X - \bar{X})^T W_x (X - \bar{X}) + U^T W_u U \\
= U^T Q U + 2 U^T v + \gamma \quad \text{(by using (5.6))} \tag{5.9}
\]

\[
Q = \Psi^T W_x \Psi + W_u \\
v = \Psi^T W_x \Phi \hat{x}_0 - \Psi^T W_x \bar{X}
\]

\( W_x \) and \( W_u \) are diagonal, positive-definite cost matrices.

We can then form a quadratic program, by adding (or not) linear constraints \( \mathcal{C} \):

\[
\min_U \quad \mathcal{F}(U) = U^T Q U + 2 U^T v \\
\text{s.t.} \quad \mathcal{C}(U) \tag{5.10}
\]

Solving this program will yield an optimal command trajectory, \( U^* \). The optimal state trajectory \( X^* \) can be retrieved by applying (5.6).

We will see in the following sections how using simplified dynamics and contact model allows us to linearize the dynamics of our system, as in (5.5). To that end, we will present two different approaches in sections 5.2 and 5.3:

- The first one goes back to the full dynamics: by specifying the motion along a privileged axis, and specifying the transition timings, we obtain a linear formulation of the dynamics.
5.2. Full contact forces

5.2.1 Previous Formalism

Figure 5.1 HRP-2 leaning: representation of variables used in modeling.

In [Nagasaka et al., 2012], the robot is assimilated to a single rigid body, as in (1.5). Moreover, only the horizontal momentum is considered: (1.5) is projected on the $x$ and $y$ axes. As only unilateral, polygonal contacts are considered, instead of associating to each contact point a 3D force vector as in (1.5), to each surface is associated only one 6-dimensional wrench. The problem variables are represented in fig. 5.1. We can thus write the state evolution equation:

$$m \ddot{c} = \sum_{i=1}^{n} f^i - mg$$

$$\dot{L} = \sum_{i=1}^{n} [f^i \times (c - p^i) + l^i]$$

with $f^i$ and $l^i$ the force and momentum applied at $i$-th (contact) surface centroid $p^i$, 

- The second one derives from the previous sections: as long as $c \in \mathcal{D}$ and $\dot{c} \in \mathcal{G}$, we know that there exists compatible contact forces. Thus, by choosing $\mathcal{G}$ we greatly reduce the problem dimensionality. It also allows us to automatically determine the contact transitions timings.
Chapter 5. CoM Predictive Control

$m$ the total mass of the robot, $c$ the position of the CoM and $L$ the angular momentum at the CoM. The time-variant number of contacts is denoted by $n$.

We will now use a discrete-time formulation along with a discrete integration scheme to derive a linear formulation of (5.12) (as (5.11) is already linear). Recall that $L_z$ is ignored (we only consider horizontal projections of the momentum). Then, if the sum of external forces on the $z$-axis is constant and known at each discrete time interval $T_k$, the projection of (5.12) on the $(x, y)$ plane will become linear.

That is, we provide $F_{z_k}$ and impose

$$F_{z_k} = \sum_{i=0}^{n_k} f_{z_k}^i$$  \hfill (5.13)

$n_k$ being the number of contacts at instant $k$. (5.12) then becomes (omitting the time instant $k$ for brevity):

$$\dot{L}_x = \sum_{i=1}^{n_k} \left[ f_y^i (c_z - p_z^i) - f_z^i (c_y - p_y^i) + l_x^i \right]$$  \hfill (5.14)

$$= -c_y F_z + c_z \sum_{i=1}^{n_k} f_y^i + \sum_{i=1}^{n_k} \left( p_y^i f_x^i - f_y^i p_z^i + l_x^i \right)$$  \hfill (5.15)

$$\dot{L}_y = \sum_{i=1}^{n_k} \left[ f_z^i (c_x - p_x^i) - f_x^i (c_z - p_z^i) + l_y^i \right]$$  \hfill (5.16)

$$= c_x F_z - c_z \sum_{i=1}^{n_k} f_x^i + \sum_{i=1}^{n_k} \left( p_z^i f_y^i - p_x^i f_z^i + l_y^i \right)$$  \hfill (5.17)

As $F_z$ is known, $c_z$ is known, given the initial conditions. Thus the above equations are linear in the CoM and contact forces and momentum.

We now build a state vector $\hat{x}$ composed of: the linear momentum, their integral and the angular momentum all of which are along the $x$- and $y$-axes. In [Nagasaka et al., 2012], the $z$-axis is aligned with the gravity $\vec{g}$. Because $\dot{L}_z$ is non-linear, we drop it and thus have uncontrolled angular momentum around the $z$-axis. The command vector is composed of the forces and momenta applied at each centroid of the contact polygons. Under the constraint (5.13) and the assumption that the centroid positions of the contact-polygons are known, we can formulate a linear system, as in (5.5):

$$\dot{x}_k = \begin{bmatrix} m c_x & m \ddot{c}_x & m c_y & m \ddot{c}_y & L_x & L_y \end{bmatrix}^T$$  \hfill (5.18)

$$u_k = \begin{bmatrix} f_x^1 & f_y^1 & f_z^1 & l_x^1 & l_y^1 & l_z^1 & \ldots & l_z^{n_k} \end{bmatrix}^T$$  \hfill (5.19)
5.2. Full contact forces

Where

\[ A_k = \begin{bmatrix}
1 & T_k & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & T_k & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & -\frac{T_k F_{zk}}{m} & 0 & 1 & 0 \\
-\frac{m}{m} & 0 & 0 & 0 & 0 & 1
\end{bmatrix} \]  \hspace{1cm} (5.20)

\[ B_k = \begin{bmatrix}
B_{1k}^1 & B_{2k}^2 & \ldots & B_{nk}^n
\end{bmatrix} \]  \hspace{1cm} (5.21)

\[ B_{ik}^j = \begin{bmatrix}
\frac{T^2_k}{2} & 0 & 0 & 0 & 0 \\
T_k & 0 & 0 & 0 & 0 \\
0 & \frac{T^2_k}{2} & 0 & 0 & 0 \\
0 & -\beta^i_k & T_k p_{yk}^i & T_k & 0 \\
0 & \beta^i_k & 0 & -T_k p_{zk}^i & 0 & T_k
\end{bmatrix} \]  \hspace{1cm} (5.22)

\[ \beta_{ik}^j = T_k (p_{zk}^i - c_{zk}) \]  \hspace{1cm} (5.23)

Additionally, we impose constraints on forces to sustain contacts (that are unilateral) and non-sliding using the associated (4-sided) linearized friction cones written in each contact reference frame (here, $\tilde{z}$ is the contact normal direction). Assuming that the local-to-world frame transforms are known, these constraints write:

\[ \tilde{f}_{ikz}^i \geq 0 \ \forall \ i \in [1, n_k] \ \forall \ k \in [0, K - 1] \]  \hspace{1cm} (5.24)

\[ |\tilde{f}_{ikz}^i| \leq \mu^i \tilde{f}_{ikz}^i \ \forall \ i \in [1, n_k] \ \forall \ k \in [0, K - 1] \]  \hspace{1cm} (5.25)

We also apply a center of pressure (CoP) condition on each line $(a, b, c)$ of each contact polygon’s edge at each time $k$:

\[ -a^i_k \tilde{l}_{kj}^i + b^i_k \tilde{n}_{kj}^i + c^i_k \tilde{f}_{kj}^i \geq 0 \]  \hspace{1cm} (5.26)

\[ \forall \ k \in [0, K - 1] \ \forall \ i \in [1, n_k] \ \forall \ j \in [1, m_k] \]

$m_k$ is the total number of all contact’s polygon edges. Minimizing (5.9) under constraints (5.13) and (5.24)–(5.26) is a QP. Its solution is the optimal set of contact forces (in their local frames). Substituting them in (5.6) yields a feasible $X$, to be tracked at best by the lower level controller. $W_x$ and $W_u$ are QP diagonal tuning gains.
5.2.2 Arbitrary reference frames

In [Nagasaka et al., 2012]'s formulation the $z$-axis is aligned with $-\vec{g}$. This is limiting in cases where general direction of motion is along the gravity field (e.g. climbing a ladder or spider-walking): setting the $z$ force trajectory is rather limiting and can even be not feasible. One would instead set the transversal swaying to zero rather than the 'climbing' trajectory which can preferably be left to the planning process.

Now, we allow choosing an arbitrary direction along which one component of the force trajectory is set. Therefore, we will account for the gravity components and project eqs. (5.11) and (5.12) onto an arbitrary reference frame $W\{\vec{w}_1, \vec{w}_2, \vec{w}_3\}$. We can then rewrite the state vector and the state equation:

$$\hat{x}_{k+1} = A_k \hat{x}_k + B_k u_k + C_k$$

$$\hat{x}_k = \begin{bmatrix} m r_1 & m \dot{r}_1 & m r_2 & m \dot{r}_2 & L_1 & L_2 \end{bmatrix}$$

where $C_k$ represents the contribution of the gravity in $W$:

$$C_k = \begin{bmatrix} -\frac{T_k}{T_k} R_{k,1} \vec{g} \\ -\frac{T_k}{T_k} R_{k,1} \vec{g} \\ -\frac{T_k}{T_k} R_{k,2} \vec{g} \\ -\frac{T_k}{T_k} R_{k,2} \vec{g} \\ 0 \\ 0 \end{bmatrix}$$

This leads us to a new formulation of eq. (5.6):

$$X = \Phi \hat{x}_0 + \Psi U + \begin{bmatrix} C_0 \\ A_1 C_0 + C_1 \\ A_2 A_1 C_0 + A_2 C_1 + C_2 \\ \vdots \\ \sum_{i=0}^{K} \left( \prod_{j=i+1}^{K} A_j \right) C_i \end{bmatrix}$$

As the command vector $U$ is written in local reference frames, we do not need to change neither the constraints nor the basic expression of our matrix $\Psi$, which now depends on the transform matrix from local frames to $W$. Note that in this formulation, $W$ must remain fixed throughout the preview window.
5.2.3 Dealing with other external known forces

For the time being $C_k$ is used to represent the effect of gravity in $W$. Yet, this term can actually encompass any other external known force. For example, we can account for sustained forces due to holding an object during motion. But, for the preview to compensate accurately for this additional force, it is necessary to know at what time the force will apply and how much force and momentum is applied at the CoM (hence, where it will be applied). This is not possible in general because the force will be applied at a point that is moving and depend on the actual position of the robot which we do not know a priori.

5.2.4 Additional inputs

Presentation

To compute one force and CoM trajectory it is necessary to provide the inputs presented in fig. 5.2. First and foremost, the current state $\hat{x}_0$ is obviously necessary. Then, the geometry and friction of the problem, defined by the $p_i$, $\mu_i$ and contact polygon geometry are considered given. The other inputs are specific to this approach:

- For the cost function, it is necessary to provide a reference trajectory $\bar{X}$ and gain matrices $W_x$ and $W_u$.

- A time parametrization: discrete time intervals defined by the $T_k$ and the number of contacts at each of those instants, $n_k$.

- The given force trajectory $F_{zk}$

![Figure 5.2 Data flow for MPC](image-url)
Chapter 5. CoM Predictive Control

Defining the reference $\dot{X}$ and tuning the MPC gains

The desired angular moments are the easiest to define: as we do not want the robot to sway, we will select a zero reference, $\dot{L}_x = 0, \dot{L}_y = 0$ in $\dot{X}$. We set the corresponding weights in $W_x$ to large values.

It is slightly harder to define the CoM reference, $m\dot{r}_x, m\dot{r}_y$, and their derivatives. Indeed there is no relationship between the contact positions and the CoM position in our reduced model. To define the reference, we will suppose that associated to each set of contact points we know a reference CoM position. This CoM position is usually the result of a planning algorithm, and can be computed using a full-model [Escande et al., 2013] or simplified kinematics [Tonneau et al., 2016]. The reference is thus given as a constant speed interpolated between the first and the last stance. Associated weights in $W_x$ are put higher on the speed relatively to position, but smaller than the weights corresponding to angular momentum. This simple setting allows fast, on-the-fly, computations. Moreover, this speed reference is continuous, whereas piecewise interpolation between stances gave worse results because of abrupt changes in acceleration. Spline interpolation between key CoM positions was also not a great fit, as it took longer to compute but did not give much better results, and steered the trajectory towards conservative paths.

Yet, this linear interpolation would not be applicable in strongly curved paths where the robot has to change motion direction. Subsequently, we approximate the path by a sequence of gross segments to better match it.

Concerning $W_u$ tuning, we want the robot to follow the trajectory first, and then minimize the contact forces, we will put much lower weights w.r.t. $W_x$. Indeed, setting $W_u$ to a small values encourages regularization of the trajectory, but too high gains prioritize not moving and falling as a solution. To summarize the weights tuning, we want:

$$(W_x)_L > (W_x)_f > (W_x)_r > W_u$$ (5.31)

Privileged axis

We need to define a force trajectory $F_{3_k}$ along the privileged axis ($\vec{w}_3$). This trajectory will in turn define the CoM position $c_3$. As very little information is available from our reduced model, we chose it simply as a trajectory at constant velocity. Thus, we set $F_{3_k}$ to always be equal to the projection of $g$ along $\vec{w}_3$, $g_3$. $\dot{c}_3$ is also constant: we choose it from the ratio of the distance separating final from current CoMs along $\vec{w}_3$ divided by the difference between final and current times. We present in the next section how
5.2. Full contact forces

to obtain those instants.

Note that making this choice prevents us from having any phase during which the projection of the gravity along $\tilde{w}_3$ is not feasible. This notably excludes free-flight phases along this axis (if the projection of the gravity is not null), and situations where the projection of the gravity along $\tilde{w}_3$ is not contained in any friction cone. However, in most situations, $\tilde{w}_3$ can be appropriately chosen, typically in those scenarios where the vertical motion is crucial, one may select at will any transversal axis.

**Time parametrization of the stances**

As in the previous section, the problem is linear if and only if the stance transition timings are given. Indeed, the size of $U$ is $6 \sum n_k$. We tried setting intuitive ad-hoc values; but it quickly turned out to be a very critical issue as many simulated scenarios failed if the robot was given too much or not enough time to complete desired multi-contact motions. To alleviate this problem, we decide to heuristically select an acceptable set of timings. Our intuition is that the timings are linked to the geometry of the problem: we should allocate more time to move across greater distances and conversely.

Thus, we categorized the transitions between stances into **support**, when the CoM moves with sustained contacts, and **move**, when the number of contacts changes. For the **support** transitions, it appears that the distance between the CoM of the stances is a good metric for the time needed to accomplish the transition. For the **move** stances, we need the distance travelled by the link currently changing its contact state. To compute this distance, we have to choose a trajectory linking the start-point $\tilde{p}_0$ and the end-point $\tilde{p}_1$. We use a cubic polynomial, starting and ending with zero velocity, passing through a waypoint defined as follows:

$$\tilde{p}_{wp} = \tilde{p}_0 + d_\parallel (\tilde{p}_1 - \tilde{p}_0) + d_\perp (\tilde{p}_1 - \tilde{p}_0)\perp$$

Typical values we use are: $d_\parallel = 0.1$, $d_\perp = 0.15$. We define $\tilde{v}^\perp = \frac{\tilde{n}}{||\tilde{n}||} |\tilde{n}\in \text{plane } (\tilde{v}, \tilde{z})$ and $\tilde{n} \cdot \tilde{z} \geq 0$ and $\tilde{n} \cdot \tilde{v} = 0$.

Those trajectories are limited to simple scenarios where the environment is not cluttered. In more complex scenarios, the trajectory would have to be generated through more advanced but not too costly techniques. Indeed, these trajectories need to be recomputed online as the starting point depends on the actual position of the robot. Thus, techniques such as parallel parameter space exploration [Pan and Manocha, 2012], CHOMP [Schulman et al., 2013] or recursive Hermite projection [Hauser, 2014] are good candidates.
We propose the following heuristic for computing the timings:

\[ t_{v+1} = t_v + \frac{[d,l]}{v_0} \left( 1 - \frac{\min(t_m,\tau)}{\alpha} \right)^\kappa \]  

(5.33)

meaning that for transiting from stance \( v \) to \( v + 1 \), the CoM motion starts with a desired given speed \( v_0 \) and accelerates in a degree \( \kappa \) during a time \( \tau \). The time scale \( \alpha \) regulates the final speed for each stance. In addition, we impose \( \alpha > \tau \) to keep positive timings. \( d \) or \( l \) are respectively the distance between CoM_{\nu+1} and CoM_{\nu}, and the length of contacting point computed trajectory. This heuristic describes a strategy similar to a trapezoidal speed command but with no deceleration. Simulations show that adding deceleration did not improve much the behavior. After careful tuning of the heuristic for our robot, we set: \( v_0,d = 0.75, v_0,l = 0.5, \kappa = 2, \tau = 4 \) and \( \alpha^2 = 50 \).

Fully nonlinear optimization techniques (resolving for the timing) are still computationally costly and not very robust [Lengagne et al., 2013], even when leveraging the newfound power of GPUs [Chrétien et al., 2016]. Later approaches have proposed other sequential approaches: first select the trajectory, then find the corresponding timing. They are based on Time Optimal Path Parametrization [Hauser, 2014, Caron and Pham, 2016] or on hierarchical optimization [Homsi et al., 2016].

### 5.2.5 Integration

![Multi-contact motion planning architecture](image-url)

Figure 5.3 Multi-contact motion planning architecture.
5.2. Full contact forces

We present in fig. 5.3 how we integrated the model-predictive controller within our planning and control architecture. To provide the necessary stances, i.e. robot configuration and contacts, we used the multi-contact planner described in [Bouyarmane and Kheddar, 2012, Vaillant et al., 2016]. This multi-contact planner finds statically-stable postures (including the free-flyer) and contacts, by solving multiple non-linear posture generation problems (see section 2.2).

They are in turn fed into the preview controller, which, will derive timings from the stances information. Then, given a reference, gains and based on the current state of the robot, will output a CoM trajectory.

Finally, this trajectory is given to a QP task-based controller [Bouyarmane and Kheddar, 2011b, Bouyarmane et al., 2012, Vaillant et al., 2014, Vaillant et al., 2016], similar to the one described in section 2.3. This controller can minimize a variety of objectives under constraints, and is equipped with a finite-state machine (FSM). The FSM monitors task execution and performs task and constraint switching to achieve multi-contact stance-to-stance transitions. The output of the QP controller is a joint position command that is then sent to the robot’s PD control loop, whether real or simulated.

Note that none of those components execute at the same rate:

- The multi-contact planning is done offline, as it takes several seconds to compute one plan.

- Our proposed preview controller runs at about 20 Hz. We present in appendix B implementation choices that strongly influenced the performance.

- The QP controller typically runs at 200 Hz.

- The robot control loop executes at 1 kHz.

Without going into too much detail here, the fact that the components do not run at the same rate, and the fact that there are transport delays leads to synchronization issues. See [Koenemann et al., 2015] for an in-depth analysis of those issues.

We will present two application scenarios of the above control architecture, but first we introduce necessary changes to our QP controller to account for interaction with rigid, mobile objects.
Chapter 5. CoM Predictive Control

Lifting an object

Our MPC accounts for a given external force. A good example to assess the latter claim is to lift an object and walk with it. We discuss three approaches we considered:

On way to deal with an additional held object is to integrate contacts’ locations and forces between the robot and the objet as part of the MPC with additional constraints on grasp stability. This however increases the complexity of the problem (that will end as a non-linear formulation); therefore, we do not use it.

Apart from using robot/object contact forces (left for the QP task controller), the second way could be to exploit the external forces term $C_k$ in subsection 5.2.2. Indeed, $C_k$ can also embed the weight of the object: we simply need to compute the weight and moment produced by the object w.r.t the CoM. By doing so, we are able to account, in the MPC, for an “instant” pick-up when the robot lifts the object. Unfortunately, it will be hard to predict exactly the object’s trajectory ahead of time because this is left to –and results from– the task controller. It is then difficult to predict the moment applied by the object w.r.t the CoM along the preview window.

Third, we consider our reduced point-mass system to change from the robot system to the robot + object system. This turns the constant mass $m$ into two values of the masses: $m_r$ or $m_{robot + object}$ at each sample $k$. By doing so, the CoM position and velocity will be discontinuous at the pick-up phase. This discontinuity is ‘filtered’ by the lower-level task-based controller when possible. We use this approach and assess it with simulations.

Change in the QP task controller

We modify the task-based QP controller presented in section 2.3 to include the dynamics of the manipulated object. This work predates, and was a starting point for the extensions presented in [Bouyarmane et al., 2017]. The main idea is to consider both the robot state and the object’s state in the same controller. This is the best way to have the robot use the object to regulate its own dynamics, yielding a much plausible plan that eventually allows performing real experiments. However, this solution requires identifying the inertia parameters (we may not know the precise weight of the manipulated body). Yet, we can consider a new state for our controller:

$$\begin{bmatrix} \ddot{q}_r & \ddot{q}_o & \lambda_{e\rightarrow r} & \lambda_{o\rightarrow r} \end{bmatrix}$$

subscript $e$ stands for environment, $o$ for manipulated object and $r$ for robot. The arrow denotes a contact between two entities, applied on the latter. Note that the
5.2. Full contact forces

A reciprocal pair of contacts between the robot and the manipulated body use the same forces intensity, \( \lambda_{o\rightarrow r} \) on the reversed friction cone generators, so that the forces are effectively reciprocal. By considering those new variables, we can rewrite the dynamics constraints:

\[
\begin{bmatrix}
\tau_l \\
0
\end{bmatrix} \leq
\begin{bmatrix}
H_r & 0 \\
0 & H_o
\end{bmatrix}
\begin{bmatrix}
\ddot{q}_r \\
\ddot{q}_o
\end{bmatrix} +
\begin{bmatrix}
G_r(q_r, \dot{q}_r) \\
G_o(q_o, \dot{q}_o)
\end{bmatrix} +
\begin{bmatrix}
J_r N_{e\rightarrow r} & J_r N_{o\rightarrow r} \\
0 & -J_o N_{o\rightarrow r}
\end{bmatrix}
\begin{bmatrix}
\lambda_{e\rightarrow r} \\
\lambda_{o\rightarrow r}
\end{bmatrix} \leq
\begin{bmatrix}
\tau_u \\
0
\end{bmatrix}
\] (5.35)

As for the friction constraints, they simply become:

\[
0 \leq \begin{bmatrix}
\lambda_{e\rightarrow r} \\
\lambda_{o\rightarrow r}
\end{bmatrix}
\] (5.36)

where \( H \) is the inertia matrix, \( J \) is the Jacobian, \( G \) represents the nonlinear and gravitational effects, subscripts \( o, r \) and \( e \) are defined as previously. We also add a manipulation acceleration constraint that enforces the non slipping condition. For every contact between the robot and the object, at each contact point:

\[
J_r \ddot{q}_r + J_o \ddot{q}_o = J_o \ddot{q}_o + J_o \ddot{q}_o
\] (5.37)

Finally the remaining changes are as follows:

First, we enhanced the initial Finite State Machine of [Bouyarmane et al., 2012] to account for additional steps that are: (i) reach a target posture before lifting an object, (ii) trigger on/off the necessary modifications to bilaterally switch between robot and robot+object (iii) bridge on/off to the robot multi-contact FSM.

For CoM and angular momentum tracking tasks, we compute these quantities by assuming the ensemble robot + object attached – recall that the MPC outputs results considering the total mass and the resultant CoM for the reduced model. This assumption is plausible since enforced by non-slip constraints that we impose on the contact points between the end effectors and the object.

To sum-up, our controller computes torques and joint accelerations for the robot and manipulated body (see [Bouyarmane et al., 2012]):

- The equality and inequality constraints are:
  - Dynamics of the object and robot (5.35);
  - Non-sliding contacts between the robot and the environment and the object (5.37);
Chapter 5. CoM Predictive Control

- Torque and joints limits;
- Collision avoidance with itself and the environment.

• The objective function composed of:

  - CoM objective: position, velocity and angular momentum track the output of the preview control;
  - Posture objective: match at best the stances generated by the static multi-contact planner;
  - Contact objectives: activate on/off target contact orientation and position tasks.

5.2.6 Results

Walking The first trial for this control method is done with walking on flat grounds. This scenario was very helpful for tuning some parameters of our simulation, including those presented in Equation 5.2.4. In the simulation the robot walked on both short and long distances, provided we use the following gains for our various tasks, (empty cells stand for “all axes”):

<table>
<thead>
<tr>
<th>Task</th>
<th>Type</th>
<th>Axis</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoM</td>
<td>Position, Velocity</td>
<td>x,y</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td></td>
<td>z</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Angular Momentum</td>
<td>x,y</td>
<td>1</td>
</tr>
<tr>
<td>Posture</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Target</td>
<td>Position</td>
<td></td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Orientation</td>
<td></td>
<td>150</td>
</tr>
</tbody>
</table>

Table 5.1 Weights and Stiffness used in CoM Preview

We use a lower weight for the CoM tracking task on the z-axis otherwise the geometrical constraints of the robot (e.g. joint limits) conflict with tracking the CoM along that direction. It could seem strange that we do not impose a high weight on this particular task, as it determines a good part of our model, but the error (being low) does not disturb the closed-loop control.

This first simulation revealed that we were indeed capable of computing our lower-level control in less than 5ms, and our CoM trajectory in about 20ms, when using our default setting of a 3-second window sampled into 20 points. Experiments show that in our hardware, the number of points should not exceed 40 to keep the computation
5.2. Full contact forces

fast, and that the maximum interval length should be less than 250ms to keep a good sampling of our trajectory. We also found out that the minimum window length is around 3s to preview at least through the next stance.

Figure 5.4 HRP-2 going over a ridge: contact stances and transitions are illustrated.

**Ridge**  This scenario was designed to demonstrate multi-contact capabilities of our planner/controller. The environment consists of a flat ground that is reduced to a narrow ridge, wide of about 15cm, shown on Figure 5.4. To make crossing it feasible, we added a support plank on the left side. We disabled the momentum tracking task because the preview control was able to generate a very low-momentum trajectory while the robot has to bend forward to take the contact with his left hand. Thus, the conflicting tasks resulted in failures to properly position the hand in the given ad-hoc computed time.

**Walking with an object on uneven terrain**  As we use both hands to carry a 2kg object, shown in red on Figure 5.5, it is difficult to present a scenario showing at the same time the multi-contact capabilities and object manipulation. The setting is made for walking on a succession of non-coplanar surfaces while wielding the object. This trial consists of a first stair step, followed by a slope and ending on a flat floor. This trial assessed our approach to modify the mass used by the preview while walking but showed that we had to perform one modification to our general approach.
While holding the object, it is necessary to slightly penalize its use to regulate the CoM position. Otherwise, the robot would use this easy-to-move weight to follow the CoM, hence being late on the desired position of the robot. As this delay accumulates, the robot ends up in a position where it needs to exert tremendous torques to reach the CoM target in time, resulting in instability and toppling.

In terms of results, the robot executes a correct motion, although not fully ‘human-like’, mainly because the robot crouches down more than we would do, as shown by the dip at the end of the trajectory. Although HRP-2 is quite strong in the arms, lifting heavy weights is usually done using handles rather than two small unilateral contacts, as in this paper. On top of this the loaded robot’s CoM is higher than unloaded but we still target the stable end CoM computed by the planner, resulting in an increased tendency to crouch.

In this experiment, we also plotted on Figure 5.6 the uncontrolled momentum derivative, $\dot{L}_z$ computed by two methods over the two first seconds. The first one is the momentum of the full robot, computed by the controller. We plot the finite difference derivative of this quantity. The second one is computed by the preview control on the reduced, single-body, model. Using the fact that $L_z$ depends (non-linearly) on the state $X$ and control vector $U$, we can compute it *a posteriori*. This figure shows that the momentum derivative computed by the preview is almost always lower than the one of the real robot, and does not present the same variations.
5.3 Polyhedron optimization

Although the previous method gave satisfactory results, we will now use the results of chapter 4 to formulate another MPC controller. This will allow us to greatly simplify the problem while alleviating two drawbacks: the choice of motion along an axis and the choice of timings. To do so, we will derive a simpler stability condition that will allow us to greatly reduce the state size.

From chapter 4 we know that there exists a couple of convex polytopes $\mathcal{P}, \mathcal{G}$ such that: if $c \in \mathcal{P}$ and $\ddot{c} \in \mathcal{G}$, there exists contact forces that realize that acceleration.

We can thus say that a motion is robustly stable iff:

$$\forall t, \ c(t) \in \mathcal{P}(t) \text{ and } \ddot{c}(t) \in \mathcal{G}(t)$$  \hspace{1cm} (5.38)

This condition is sufficient to guarantee that the motion is feasible, in the sense that there exists contact forces for every time $t$. Given that $\mathcal{P}$ and $\mathcal{G}$ are entirely determined by the contact geometry and friction, we can denote $\text{contacts}(t)$ the active contacts.
at the time $t$. The above condition is thus:

$$\forall t \quad c(t) \in \mathcal{P}(\text{contacts}(t)) \quad \text{and} \quad \ddot{c} \in \mathcal{G}(\text{contacts}(t)) \quad (5.39)$$

Thus, to generate a trajectory we need not only optimize the CoM trajectory itself ($c$ and its derivatives) but the instants at which the contacts will change ($\text{contacts}$).

We will consider a number of stances, i.e. consecutive sets of contacts, $\text{contacts}_i$. We discretize the time into $K$ time intervals. We can thus write that:

$$\forall k, \quad c_k \in \mathcal{P}(\text{contacts}_{i(k)}) \quad \text{and} \quad \ddot{c} \in \mathcal{G}(\text{contacts}_{i(k)}) \quad (5.40)$$

Where $i(k)$ represents the active contact set index at instant $k$. As changing contacts is a discrete event, it is a function over finite integer sets:

$$i : [0..K] \rightarrow [0..N] \quad (5.41)$$

$$k \rightarrow i(k) \quad (5.42)$$

In the following we will slightly simplify the notation by dropping the $\text{contacts}$ and simply denoting the polyhedron and polytope $\mathcal{P}_{i(k)}$ and $\mathcal{G}_{i(k)}$ respectively.

Our problem is formulated as a minimal jerk problem. First we form the evolution

Figure 5.7 Schema of the polyhedron CoM preview control. Left shows the spatial trajectory going through three different polygons. Right shows the evolution of the integer variable with time.
5.3. Polyhedron optimization

equation:

\[
\begin{bmatrix}
    c_{k+1}
    \\
    \dot{c}_{k+1}
    \\
    \ddot{c}_{k+1}
\end{bmatrix}
= \begin{bmatrix}
    1 & dt & \frac{dt^2}{2} \\
    0 & 1 & dt \\
    0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    c_k \\
    \dot{c}_k \\
    \ddot{c}_k
\end{bmatrix}
+ \begin{bmatrix}
    \frac{dt^3}{6} \\
    \frac{dt^2}{2} \\
    dt
\end{bmatrix}
\vec{c}_k
\]

(5.43)

Which, in the formulation of section 5.1 is a linear control problem with:

\[
\dot{x}_k = \begin{bmatrix}
    c_k^T \\
    \dot{c}_k^T \\
    \ddot{c}_k^T
\end{bmatrix}
\]

(5.45)

\[u_k = \begin{bmatrix}
    \vec{c}_k
\end{bmatrix}
\]

(5.46)

As \(c_k\) and \(\ddot{c}_k\), are part of our state (5.40) can be written as linear constraints in \(U\):

\[
H_{\mathcal{P}_i(k)} c_k \leq b_{\mathcal{P}_i(k)}
\]

\[
\Rightarrow H_{\mathcal{P}_i(k)} S_{c_k} (\Phi x_0 + \Psi U) \leq b_{\mathcal{P}_i(k)}
\]

\[
\Rightarrow H_{\mathcal{P}_i(k)} S_{\ddot{c}_k} (\Phi x_0 + \Psi U) \leq b_{\mathcal{P}_i(k)}
\]

\[
\Rightarrow H_{\mathcal{P}_i(k)} S_{\ddot{c}_k} \Psi U \leq b_{\mathcal{P}_i(k)} - \Phi x_0
\]

(5.47)

With \(S_{c_k}\) and \(S_{\ddot{c}_k}\) selection matrices that extract \(c_k\) and \(\ddot{c}_k\) from \(X\) respectively.

However, we still need to find the best function \(i(k)\). There are a priori a very large number of them, \(N^K\). However, we are only interested in a restricted subset of them: strictly increasing by one functions. That is to say, a stair-shaped function. It is defined by the \(\tau_j\) that indicate at which time we will switch from \(\mathcal{P}_j\) to \(\mathcal{P}_{j+1}\), i.e:

\[
\forall k \in [\tau_j, \tau_{j+1}) \quad i(k) = j
\]

(5.50)

Thus, instead of looking for a function among \(N^K\) possibilities, we have to find the value of \(N\) strictly increasing variables in \([0..K]\). See fig. 5.7 for a simple illustration of such a function. The actual number of possible functions is given by:

\[
\sum_{i=1}^{N} p_i(K) i!
\]

(5.51)

Where \(p\) stands for the partition function (see e.g. [Hardy and Wright, 1979]). Indeed the partition function tells us how many tuples of integers of a certain size sum to \(K\). We then account for the fact that we might not switch, and thus sum over all partitions of size less or equal than \(N\). Finally, we consider that there are \(i!\) possible orderings of a partition of size \(i\).
Chapter 5. CoM Predictive Control

A numerical application for a typical scenario where $K = 15$ and $N = 3$ illustrates the huge reduction in the size of the search space:

- For the complete space search: $N^K = 14348907$.
- For the function search (5.51): 129.

Once the transition times $\tau_i$ are given, the problem defined by (5.10) is a QP in few variables (typically a few dozens) and a lot of constraints (a few thousands). This can be solved in milliseconds (using the QL solver [Schittkowski, 2011]), and thus exploring all the possible $\tau_i$ for a small number of transitions, i.e. $N = 2$, can be done in about 500 ms.

To further reduce the search space, for each choice of $\tau_0$, we test if $\tau_1 = N$ is feasible. If it is, we explore all possible $\tau_1$ starting from $N$ and working backwards towards $\tau_0$ until the problem is no longer feasible. This amounts to discarding all solutions where it is not possible to stop in the intermediate polyhedron. We then select the solution $(X, \tau_0, \tau_1)$ that yields the smallest cost. This algorithm is resumed in pseudocode in algorithm 5. It is easily extended to more $\tau_j$ but would require a lot of computational power, as the size of the search space is exponential in $N$.

### Algorithm 5

Algorithm MINTIM to determine the optimal timings

```plaintext
for $\tau_0 \in [0..K]$ do
  for $\tau_1 \in [K..\tau_0 + 1]$ do
    res ← solveQP($\tau_0, \tau_1$)
    if res then
      $X, cost ← res$
    else
      break
    end if
  end for
end for
return argmin$_{\tau_0, \tau_1, X}(cost)$
```

#### 5.3.1 Results

The following sections present two illustrations of the framework presented above: crossing a ridge and climbing stairs. In both cases, HRP2-Kai is able to perform the motion if we do not enforce real-torque limitations. This means that indeed, there exists contact forces that allow the controller to track the MPC CoM trajectory.
However, those contact forces are not always realizable by the whole-body system. Indeed, during trajectory generation, we only relied on a reduced model, that does not take into account joint torques as those depend non-linearly on the joint angles.

The controller is also unable to perform the end-effector trajectory in time, but this is not problematic: the CoM remains in the previous polyhedron as long as the new contact has not been established. The MPC then computes new timings based on the current state.

In both scenarios, we used a single $G$ for all stances, that is the convex hull of lateral accelerations with a magnitude of $r = 0.5 \text{ m s}^{-2}$:

$$G = \text{CONV}([r, 0, 0], [-r, 0, 0], [0, r, 0], [0, -r, 0])$$

We also set the duration of each time-step to $dt = 0.15 \text{ s}$, which yields a total duration of 1.5 s for the trajectories.

### 5.3.2 Ridge crossing

In this simulation, we want to cross a narrow ridge, while pushing on a flat, non-horizontal, support for added stability. This is the same environment as in the previous section, but the multi-contact plan we used is newly generated. A few keyframes are available in fig. 5.8. At most, we have three unilateral contacts with the environment.

The CoM trajectory that we obtain is very conservative. This is most evident in the first stance: HRP2-Kai leans quite heavily backwards with its CoM above the right foot. That stance is composed of a single unilateral contact, thus the region of stability is very small at that time. Indeed, as $G$ contains the gravity, we are rather statically stable at all times. For this particular stance, it means that the CoM has to remain above the right foot, at a low altitude.

Compared to section 5.2.6, we obtain a less dynamic motion: all phases are now statically stable. However, we no longer need to use a carefully tuned timing heuristic and the timings automatically adjust based on the current state. We also do not specify the $z$ motion, and the choice of $\bar{X}$ is more straightforward.

### 5.3.3 Stairs climbing

In this scenario, HRP2-Kai is tasked with climbing a flight of stairs, using the handrail. Halfway up, it grasp again the handrail while using its other hand as unilateral support.
on the top platform. Thus, some stances have three unilateral contacts and a bilateral one. The resulting motion is illustrated in fig. 5.9.

We first remark that this motion is much more plausible compared to the previous scenario. The CoM trajectory is less constrained because the robot establishes more than one contact with the environment at all times: we do not produce the ‘non-expected’ poses of the previous scenario. However, as research on how humans decide to make or break contacts is scarce, it is difficult to quantify that feeling. It would seem coherent that humans use more contacts when confronted with a difficult and/or uncertain terrain. When walking under such conditions, humans indeed adopt stiffer, more static gaits [Cappellini et al., 2010].

Still, the motion is executed much faster than in the quasi-static case: we climb the whole flight in about 30 s whereas it took about 4 min in the work presented in chapter 3. Indeed, we gain substantial time by not waiting for the CoM to reach a predetermined position. Yet, as this motion generates high-torques, we would probably have to slow it down to execute it on a real hardware, especially when approaching contact surfaces so as not to damage the robot in case of unpredicted contact.

5.4 Conclusion and future work

We presented two approaches to perform multi-contact dynamic motions, eventually holding another rigid body. In one case, we computed fully dynamic motion, but at the cost of using heuristics to determine timings and having to specify the motion along one axis. In the other one, we automatically derived the stances transition timings, at the cost of restricting the possible motions.

Hence, ideally we would bridge the gap between the two approaches to combine
5.4. Conclusion and future work

Figure 5.9 Keyframes of HRP2-Kai climbing the stairs: the blue ribbon represents the current CoM trajectory.

their advantages. For example, we do not modify $G$ during optimization. As we can approximate changes in $G$ by interpolation, it would be interesting to adopt an approach similar to [Coevoet et al., 2017] and deform $G$ according to the saturated constraints.

Another avenue of improvement is to forgo the reduced models and go back to full models, only using the reduced model information as a hint or starting point for faster trajectory generation. Indeed, those models are not yet fast enough for on-line optimization, but starting from an educated guess can drastically reduce their computation time.
Conclusion

The central question explored in this thesis is the notion of stability for legged robots in multi-contact. Although central to the field, we found that a number of concepts had not been fully developed and explored. Those concepts allowed us to propose new control strategies for legged robots.

Among our contributions, we have shown that:

- The region of static stability depends on the admissible contact forces, and using shape transformation algorithms can be used both for force control and stable multi-contact control.

- The region of static stability under perturbations is a 3D convex shape that can be efficiently computed by recursive algorithms. This region can then be used for multi-contact planning and control using similar shape transformations.

- Stability of a humanoid robot can also be seen as the tracking of a stable trajectory: we proposed two new ways to generate CoM trajectories in real-time and in multi-contact.

Starting from the usual notions of stability we have shown that static stability is a powerful tool that can be extended to motion. Indeed, by restricting the possible motions in terms of acceleration, we are able to explicitly compute acceptable regions in terms of position. This yields a completely geometric criteria of stability, that decouples position from acceleration and can be leveraged to compute motion inside those regions.

Based around those contributions, we can produce a full pipeline that first generates a multi-contact plan, then compute stability regions and finally generate dynamic motion inside said regions of stability. Although we focused on humanoid robotics applications, our approach can be applied to manipulation and legged robotics applications.
Chapter 5. Conclusion

However, a lot of possible improvements are open: both on the theoretical and on the technical sides.

Theoretical improvements

The most important question that remained unanswered in this work is the analytical formulation of the regions of stability. Although [Or and Rimon, 2016] have proposed an almost analytical formulation of the region of static equilibrium, it would be of great interest to obtain an expression of the derivative of said region w.r.t. the various geometric and frictional parameters. Furthermore, in the robust case obtaining the derivative w.r.t. the acceleration polytope is a priority. Indeed, by using this information we could formulate fast problems that would optimize both the shape and the motion. So far, we can only approximate those derivatives by using interpolation methods.

On the topic of interpolation, all the methods we have used do not provide strong guarantees. Finding an interpolator that provably satisfies all the properties outlined in section 3.2 would make our methods more robust.

Finally, we have mostly worked with fixed acceleration envelopes, but this restricts the possible motions, and the CoM trajectories we generated in section 5.3 were too conservative. To alleviate this problem, we would probably have to optimize the acceleration envelope as well as the motion. Although a variational formulation of our region would be best, we could employ an alternate optimization scheme.

Technical improvements

The first missing point of this thesis is implementation on real hardware: although our algorithms are in no way limited to simulation, further tuning is necessary. Moreover, we only used simple Finite-State-Machines to control our robot, complex scenarios require more complete ones [Vaillant et al., 2016].

While we strived to present efficient algorithms, that would scale well with the number of variables, some areas would benefit from additional improvements.

One limiting factor is the time necessary to solve optimization problems, mainly QP. We only considered off-the-shelf solvers in this work and their performance is largely out of our hands. Even if writing an efficient QP solver is not an easy task, in some cases a specialized solver might present better performance.
In most of our work, we have kept to single-threaded algorithms: especially when computing the timings in section 5.3 we could parallelize the search and exploit multiple cores on a machine. This would require a rewrite of our MPC from Python to a language that better supports light concurrency primitives instead of processes. However, the biggest gain is certainly to be had by developing another method than exhaustive search for the timings.

Finally, we have extensively used a weighted formulation for both our MPC formulations and whole-body controller. This means that it is impossible to strictly prioritize tasks compared to a hierarchical formulation [Escande et al., 2014]. Moreover, using a hierarchical formulation could lower the number of parameters that we had to tune in our various algorithms.
A Fast SOCP resolution

In the static case, we want to solve the SOCP:

\[
\begin{align*}
\text{max}_{c,f} & \quad d^T c \\
\text{s.t.} & \quad A_1 f + A_2 c = T \\
& \quad f_i \in K_i
\end{align*}
\]  

(A.1)  
(A.2)

At each step the problem being solved by cvxopt [Andersen et al., ] is (KKT with scaling for primal-dual Newton step):

\[
\begin{bmatrix}
0 & 0 & A_1^T & G^T W^{-1} & 0 & 0 & d_f \\
0 & 0 & A_2^T & 0 & 0 & 0 & d_c \\
A_1 & A_2 & 0 & 0 & -I & 0 & d_y \\
W^{-T} G & 0 & 0 & -I & 0 & 0 & d_z
\end{bmatrix}
= \begin{bmatrix}
rf \\
rf \\
rf \\
rf \\
rf \\
rf
\end{bmatrix}
\]  

(A.3)

With:

- \(f\) contact forces (size 3n)
- \(c\) com position (size 2)
- \(y\) Lagrange multipliers of the equality constraint (size 6)
- \(z\) Slack variables for cones (size 4n)
- \(A_1\) grasp matrix
- \(A_2\) cross-product with gravity
Appendix A. Fast SOCP resolution

- \( G \) Friction cone matrices
- \( W \) Hyperbolic householder transformations for scaling

\[
W = \begin{bmatrix}
W_0 & \cdots & W_n \\
\vdots & \ddots & \vdots \\
W_n & & W_n
\end{bmatrix} \quad \quad G = \begin{bmatrix}
g_0 & \cdots & g_n \\
\vdots & \ddots & \vdots \\
g_n & & g_n
\end{bmatrix} \quad \quad (A.4)
\]

Note that each \( W \) is square (4x4) and invertible, with both \( W \) and \( W^{-1} \) computable by a closed formula. Each \( g_i \) is a (4x3) matrix representing an \( L_2 \) cone.

We thus have:

\[
A_1^T d_y + \begin{bmatrix}
g_0^T W_0^{-1} & \cdots & g_n^T W_n^{-1}
\end{bmatrix} d_z = r_f \quad \quad (A.5)
\]

\[
A_2^T d_c = r_c \quad \quad (A.6)
\]

\[
A_1 d_f + A_2 d_c = r_y \quad \quad (A.7)
\]

\[
\begin{bmatrix}
W^{-T} g_0 & \cdots & W^{-T} g_n
\end{bmatrix} d_f - d_z = r_z \quad \quad (A.8)
\]

Now, denoting \( \Omega_i = W_i^{-T} g_i \) and \( \Omega = \begin{bmatrix}
\Omega_0 & \cdots & \Omega_n
\end{bmatrix} \), we will first eliminate \( d_z \):

\[
d_z = \Omega d_f - r_z \quad \quad (A.9)
\]

Then \( d_f \):

\[
A_1^T d_y + \Omega^T (\Omega d_f - r_z) = r_f \quad \quad (A.10)
\]

\[
d_f = (\Omega^T \Omega)^{-1} (r_f - A_1^T d_y + \Omega r_z) \quad \quad (A.11)
\]
Finally, denoting $\lambda_i = \Omega_i^T \Omega_i$ and $\Lambda = \begin{bmatrix} \lambda_0 & \cdots & \lambda_n \end{bmatrix}$, we have:

\[
\begin{bmatrix}
0 & A_2^T \\
A_2 & -A_1 \Lambda^{-1} A_1^T
\end{bmatrix} \begin{bmatrix} dc \\ dy \end{bmatrix} = \begin{bmatrix} r_c \\ e \end{bmatrix} \tag{A.12}
\]

With:

\[
e = r_y - A_1 \Lambda^{-1}(\Omega^T r_z + r_f) \tag{A.13}
\]

This is a (8x8) system, much smaller than the original (A.3).

To solve this, we do not want to compute the explicit inverses of $\lambda_i$. Instead we can base the resolution entirely around solving linear equations, as summarized in algorithm 6:

- Compute the QR decompositions of $\Omega_i$, the Cholesky of $\lambda_i$ is $R_i^T$.
- Compute $e, -A_1 \Lambda^{-1} A_1^T$ by solving for $R_i^T$.
- Solve the system (A.12), and obtain $d_c, d_y$.
- From eqs. (A.9) and (A.11) compute $d_f$ and then $d_z$.

**Algorithm 6** Procedures that allow for faster resolution of a SOCP. OUTER will be called at each outer step, NEWTON will be called at each inner step to solve the KKT equations.

```plaintext
procedure OUTER($W_i$)
    $\Omega_i \leftarrow$ HOUSEHOLDER($W_i$)
    $R_i \leftarrow$ QR($\Omega_i$)$^T$
    $M \leftarrow -A_1 \cdot BLOCKSOLVE(R_i, A_1^T)$
    $\mathcal{A} \leftarrow \begin{bmatrix} 0 & A_2^T \\ A_2 & M \end{bmatrix}$
end procedure

procedure NEWTON($r_f, r_c, r_y, r_z$)
    $e \leftarrow r_y - A_1 \cdot BLOCKSOLVE(R_i, r_f + \Omega^T r_z)$
    $d_c, d_y \leftarrow SOLVE(\mathcal{A}, r_c, e)$
    $d_f \leftarrow BLOCKSOLVE(R_i, r_f - A_1^T d_y + \Omega r_z)$
    $d_z \leftarrow \Omega d_f - r_z$
end procedure
```
Appendix A. Fast SOCP resolution

Unfortunately implementing algorithm 6 in Python did not yield any significant improvements over the C implementation in *cvxopt*.
This annex presents a few ideas that allow for the fast computation of the Model-Preview Control presented in section 5.2. To compute a new preview control solution $U$, the two main computational efforts are:

- Building the problem, i.e. computing $Q$, $v$ and all the constraints.
- Solving the problem.

The first point is influenced by our implementation choices, and while we cannot tune the second, it is important to appropriately choose a solver.

To build the problem, computing $\Psi$ is one of the big costs. As $\Psi$ is almost diagonal inferior, using sparse matrices yields small improvements. However, bigger gains can be realized by using the following the following formula to directly compute the product of $A_i$:

$$\prod_{k=i}^{j} A_k = \begin{bmatrix}
1 & \sum_i^j T_k & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & \sum_i^j T_k & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & -P_{i,j} & -S_{i,j} & 1 & 0 \\
P_{i,j} & S_{i,j} & 0 & 0 & 0 & 1
\end{bmatrix} \quad (B.1)$$

$$P_{i,j} = \sum_{k=i}^{j} \frac{T_k F_{zk}}{m} \quad (B.2)$$

$$S_{i,j} = \sum_{k=i+1}^{j} \frac{T_k F_{zk}}{m} \left( \sum_{l=0}^{k-1} T_l \right) \quad (B.3)$$
Appendix B. Fast MPC computation

Another way to compute $\Psi$ that yields the same gains is to use a recursive formula. Indeed, one can write $\Psi$ as:

$$
\Psi = \begin{bmatrix}
B_0 \\
\Psi_1 & B_1 \\
\Psi_2 & B_2 \\
\vdots & \ddots \\
\Psi_{N-1} & B_{N-1}
\end{bmatrix}
$$

(B.4)

$$
\Psi_i = A_i \begin{bmatrix} \Psi_{i-1} & B_{i-1} \end{bmatrix}
$$

(B.5)

Unfortunately, we do not have a real way to improve the computing time for $Q$, because there is no obvious analytic formula. However, as it is a symmetric matrix computed as a product by its transpose, we can use the Eigen matrix library rank update functionality to compute only one half very efficiently.

1  Q.setZero();
2  Q.selfadjointView<Eigen::Upper>().rankUpdate(psi.transpose()*sqrtWx.asDiagonal());
3  Q.triangularView<Eigen::StrictlyLower>() = Q.transpose();
4  Q.diagonal() += Wu;

For the solver, we decided to use quadprog [Turlach, 1998]. Indeed, quadprog accepts a dense objective matrix and a sparse constraint matrix. In our case, $Q$ is not sparse enough to justify a full sparse solver, but using a sparse constraint matrix led to huge improvements. Moreover, quadprog supports warm-start: we can pass in the previous solution as an initial guess, which also helped reduce the computation times.

All things considered, we reached a point where:

- The total time to build and solve is of the order of 50 ms for a small number of instants $K \approx 30$. It increases to about 2 s for $K \approx 100$.

- Of this total time, only 5% is spent building the problem. This means that to go even faster, only using a better solver would help.

Given that the period of our controller is 5 ms, we are able to recompute a trajectory of several seconds in about 10 iterations of our controller, which is exactly in the bounds we described in section 5.1.
C Matrix definitions

We recall in this appendix the notations of [Bretl and Lall, 2008] that we use throughout this paper. All of those notations are introduced to obtain a more compact representation of the Euler-Newton equations. We thus start from them, in the static equilibrium setting:

\[
\sum f_i = -mg \\
\sum f_i \times (c - r_i) = 0 \\
\forall i \quad \|f_i^t\| \leq \mu f_i^n
\]

(C.1) \hspace{1cm} (C.2) \hspace{1cm} (C.3)

Where \(f_i^t\) represent the tangential forces and \(f_i^n\) the normal forces. The \(r_i\) are the contact point positions in the world frame.

We can then slightly reorganize them to isolate the parts that depend on \(c\):

\[
\sum f_i = -mg \\
\sum r_i \times f_i - mg \times c = 0 \\
\forall i \quad \|f_i^t\| \leq \mu f_i^n
\]

(C.4)

We then define the following quantities:

\[
T(g) = \begin{bmatrix} -mg \\ 0 \end{bmatrix}
\]

(C.5)

\[
A_1 = \begin{bmatrix} I_3 & I_3 & I_3 & \cdots & I_3 \\ [r_0]_x & [r_1]_x & [r_2]_x & \cdots & [r_n]_x \end{bmatrix}
\]

(C.6)
Appendix C. Matrix definitions

\[ A_2(g) = \begin{bmatrix} 0 \\ -m[g]_x \end{bmatrix} \] (C.7)

\[ B_i = I_3 - n_i n_i^T \quad u_i = \mu_i n_i \] (C.8)

Where \( n_i \) is the normal to the \( i \)th contact, \([.]_x\) represents the screw symmetric matrix such that \( \forall u, v \in \mathbb{R}^3, a \times b = [a]_x b \). They are such that the system (C.4) is equivalent to:

\[ A_1 f + A_2(g)c = T(g) \]
\[ \forall i \quad \|B_i f_i\| \leq u_i^T f_i \] (C.9)

And thus we can define:

\[ B = \begin{bmatrix} B_0 \\ B_1 \\ \vdots \\ B_n \end{bmatrix} \quad u = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_n \end{bmatrix} \] (C.10)

Which means that (C.4) is equivalent to:

\[ A_1 f + A_2(g)c = T(g) \]
\[ \|B f\| \leq u^T f \] (C.11)
As presented in section 4.4, algorithm 3 has a linear or quasi-linear rate of convergence. This is a let-down compared to the quadratic result of [Bretl and Lall, 2008] but was expected. Indeed, from intuition alone, approximating a circle by a square only requires 4 vertices, but approximating a sphere by a cube requires 8 vertices. As both algorithm 3 and [Bretl and Lall, 2008] generate vertices one by one, we expected that the augmentation of the dimension would slow down the rate of convergence.

This is reinforced by the stochastic geometry literature: it has been proven in [Gruner, 1997, Boroczky, 2000] that the rate of convergence depends on the dimension as follows. Consider \( \mathcal{P} \) the shape to be approximated, \( \mathcal{P}_{\text{outer}}^n \) and \( \mathcal{P}_{\text{inner}}^n \) the best circumscribing and inscribed polyhedral approximations of \( \mathcal{P} \) respectively. \( \mathcal{P}_{\text{outer}}^n \) has \( n \) facets tangent to \( \mathcal{P} \) while \( \mathcal{P}_{\text{inner}}^n \) has \( n \) vertices located on the surface \( \partial \mathcal{P} \) of \( \mathcal{P} \).

If \( \mathcal{P} \) is convex (gaussian curvature positive or null everywhere) and \( C^2 \)-smooth, then for the symmetric difference metric \( \delta \) the best approximations verify:

\[
\delta(\mathcal{P}_{\text{outer}}^n, \mathcal{P}) \sim \frac{1}{2} \text{del}_{\zeta-1} \mathcal{A}(\partial \mathcal{P}) \left( \frac{1}{n^{\zeta-1}} \right) \]

\[
\delta(\mathcal{P}, \mathcal{P}_{\text{inner}}^n) \sim \frac{1}{2} \text{div}_{\zeta-1} \mathcal{A}(\partial \mathcal{P}) \left( \frac{1}{n^{\zeta-1}} \right) \]

With \( \text{del} \) and \( \text{div} \) being constants that only depend on the dimension \( \zeta \). Note that \( \mathcal{A} \) in this context denotes the affine surface area rather the regular surface area. The result can be extended to the best approximating sequences of \( \mathcal{P} \).

Other similar results hold for different metrics, the most interesting being the Schneider metric. The Schneider metric \( \delta^{\text{SCH}}(\mathcal{P}_1, \mathcal{P}_2) \) only applies if \( \mathcal{P}_2 \subset \mathcal{P}_1 \). It is the maximum volume of caps of \( \mathcal{P}_1 \) cut by the supporting hyperplanes of \( \mathcal{P}_2 \). In algorithm 3 \( \delta^{\text{SCH}} \) is implicitly used: we choose the search direction perpendicular to the
Appendix D. Convergence

facet that forms the biggest uncertainty volume i.e. maximizes $\delta_{SC\,H}^{C}(P^{n}_{outer}, P^{n}_{inner})$.

Our measure of convergence is defined by:

$$\delta(P^{n}_{outer}, P^{n}_{inner}) = \delta(P^{n}_{outer}, P) + \delta(P, P^{n}_{inner})$$  \hspace{1cm} (D.3)

We can thus use the previous results:

- Our algorithm, if it is indeed linear, is optimal in terms of rate of convergence.
- Approximating the sphere in section 4.4 is the worst-case scenario as it maximizes $A(\frac{\partial P}{\partial i})$ for a given volume.

However, we cannot directly use this result to prove our rate of convergence:

- It only applies to $C^2$ bodies, while the volume we are approximating is $C^2$ almost-everywhere only.
- We are computing two sequences of approximations of $P$, that minimize $\delta_{SC\,H}$ at each step but they probably do not minimize $\delta$.
- This result is only asymptotic: as we will stop the algorithm after a finite (potentially small) number of iterations, we need to prove a result that holds even for small $n$. 
Proposition 2. If $\exists (l, s) \in \mathbb{R}^2$ such that $\forall f_i$:

$$u^T f_i \geq u^T f + l \quad (E.1)$$

$$\|B f_i\| \leq \|B f\| + s \quad (E.2)$$

we have the following implication:

$$\|B f\| \leq u^T f + l - s \Rightarrow \|B f_i\| \leq u^T f_i \quad (E.3)$$

Proof. $\|B f\| \leq u^T f + l - s$ writes $\|B f\| + s \leq u^T f + l$ then $\|B f_i\| \leq \|B f\| + s \leq u^T f + l \leq u^T f_i$ from (E.1) and (E.2).

In what follows, we show that bounds $l$ and $s$ exist and subsequently (E.3) holds.

We need intermediary results that will be used in expressing both $l$ and $s$.

Lemma 4. The only possible $f_i$ are given by:

$$f_i = f + A_1^T \left(T(\tilde{g}_i) - A_2(\tilde{g}_i)c + (I - A_1^T A_1)w\right) \quad (E.4)$$

with $^\dagger$ the Moore-Penrose pseudo inverse, and $w$ a vector having the size of contact forces.

Proof. We first exploit the linearity of $A_2$ (screw operator) and $T$ (stacking acceleration with zero angular momentum):

$$(4.1) \text{ and } (4.3) \Leftrightarrow A_1 \cdot (f_i - f) + A_2(\tilde{g}_i)c = T(\tilde{g}_i) \quad (E.5)$$
Appendix E. Bounds proofs

Since $A_1$ is full row-rank\(^1\), the only possible $f_i$ is given by:

$$f_i = f + A_1^\dagger (T(\tilde{g}_i) - A_2(\tilde{g}_i)c) + (I - A_1^\dagger A_1) w$$  \hspace{1cm} (E.6)

Proof of theorem 3. We develop $u^T f_i$ using Lemma 4, considering the minimal norm solution: $w = 0$.

$$u^T f_i = u^T f + u^T A_1^\dagger (T(\tilde{g}_i) - A_2(\tilde{g}_i)c)$$  \hspace{1cm} (E.7)

$$= u^T f + u^T A_1^\dagger T(\tilde{g}_i) - u^T A_1^\dagger A_2(\tilde{g}_i)c$$  \hspace{1cm} (E.8)

Now, we will use the following properties:

- For any vectors $a$ and $b$, $|a^T b| \leq \|a\| \|b\|$ (Cauchy-Schwarz inequality);
- $\|u\| = \mu$ (by definition);
- $\|Ax\| \leq \bar{\sigma}(A) \|x\|$ with $\bar{\sigma}(A)$ the largest singular value of $A$.

Firstly,

$$u^T A_1^\dagger T(\tilde{g}_i) \geq -\|u\| \|A_1^\dagger T(\tilde{g}_i)\| \geq -\mu \bar{\sigma}(A_1^\dagger)mr$$  \hspace{1cm} (E.9)

Secondly,

$$u^T A_1^\dagger A_2(\tilde{g}_i)c \leq \|u\| \|A_1^\dagger A_2(\tilde{g}_i)c\| \leq \mu \bar{\sigma}(A_1^\dagger)mr \|c\|$$  \hspace{1cm} (E.11)

$$-u^T A_1^\dagger A_2(\tilde{g}_i)c \geq -\mu \bar{\sigma}(A_1^\dagger)mr \|c\|$$  \hspace{1cm} (E.12)

Thus, from (E.10) and (E.13), we have

$$u^T f_i \geq u^T f - \mu mr \bar{\sigma}(A_1^\dagger)(1 + \|c\|)$$  \hspace{1cm} (E.14)

\(^1\)Except the degenerate unlikely case where all the contact points are aligned.
Proof of theorem 4. We develop $|Bf|$ using Lemma 4, considering the minimal norm solution: $w = 0$.

$$ |Bf| = \|Bf + BA_1^T \tilde{g}_i + BA_1^T A_2(\tilde{g}_i)c\| $$ (E.15)

We first apply the triangular inequality:

$$ |Bf| \leq \|Bf\| + \|BA_1^T \tilde{g}_i\| + \|BA_1^T A_2(\tilde{g}_i)c\| $$ (E.16)

Firstly:

$$ \|BA_1^T \tilde{g}_i\| \leq rm\tilde{\sigma}(BA_1^T) $$ (E.17)

Secondly:

$$ \|BA_1^T A_2(\tilde{g}_i)c\| \leq rm\tilde{\sigma}(BA_1^T) \|c\| $$ (E.18)

We thus have:

$$ |Bf| - \|Bf\| \leq rm\tilde{\sigma}(BA_1^T)(1 + \|c\|) $$ (E.19)

We thus see that tightness can only be achieved if the following conditions are achieved at the same time:

- $\|Aa\| \leq \tilde{\sigma} \|a\|$ is tight whenever $a$ is aligned with the singular vector associated with the maximum singular value, say $\tilde{\sigma}$. This gives us:
  - $\tilde{g}_i = r \tilde{\nu}$ thus $\tilde{g}_i$ is parallel to $\tilde{\nu}$.
  - $A_2(\tilde{g}_i)c = k \tilde{\nu}$ i.e. $\tilde{g}_i \times c = ar \|c\| \tilde{\nu}$ that is to say a vector perpendicular to $\tilde{g}_i$ is parallel to $\tilde{\nu}$.

- $\|A_2(\tilde{g}_i)c\| \leq mr \|c\|$ is equal iff $c \perp \tilde{g}_i$

Which are equivalent to those presented in section 4.1.2.
References


References


References


References


References


References


References


References


