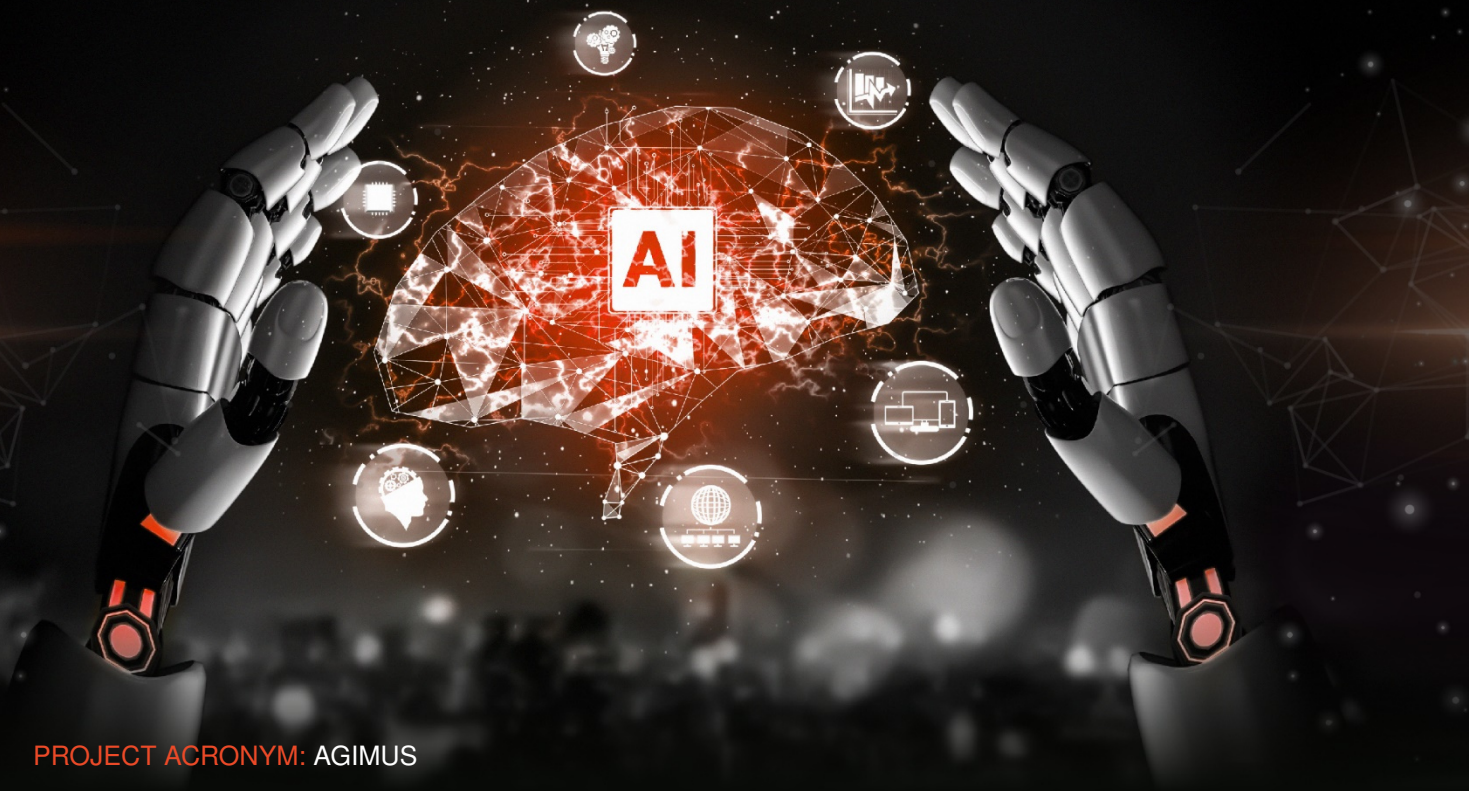




## D2.1 – Efficient and versatile differentiable simulator



**PROJECT ACRONYM:** AGIMUS

**PROGRAMME:** Horizon Europe

**GRANT AGREEMENT:** No 101070165

**TYPE OF ACTION:** Horizon Research & Innovation Actions

**START DATE:** 1 October 2022

**DURATION:** 48 months



Funded by  
the European Union

## Document information

Issued by:	Inria
Issue date:	01/09/2023
Due date:	30/09/2023
Work package leader:	Inria
Start date:	1 October 2022
Dissemination level:	PUB (Public)

## Document History

Version	Date	Modifications made by
1	14/09/2023	Justin Carpentier and Louis Montaut

## Authors

First name	Last name	Beneficiary
Justin	Carpentier	Inria
Louis	Montaut	Inria & CTU

*In case you want any additional information, or you want to consult with the authors of this document, please send your inquiries to: [justin.carpentier@inria.fr](mailto:justin.carpentier@inria.fr) (change the e-mail to the one of the person in charge of this Deliverable)*

## Quality reviewers

First name	Last name	Beneficiary
Vladimir	Petrik	CTU
Nicolas	Mansard	CNRS

### **Disclaimer**

*Funded by the European Union under GA no. 101070165. Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union. The granting authority cannot be held responsible for them.*

**© AGIMUS Consortium, 2022**

*Reproduction is authorised provided the source is acknowledged.*

### Executive summary

This report was developed in the context of the AGIMUS project, funded by the Horizon Europe Framework Program for Research and Innovation of the European Union for 2021-2027. Its objective is to present the progress mainly towards the specific objective **SpO1.2: Develop advanced perception combining data-driven training with physics-based models** that is part of the **Objective 1: Significantly accelerate the introduction and deployment of a robotic system to a new agile production environment through advanced perception and understanding of human-centric feedback.**



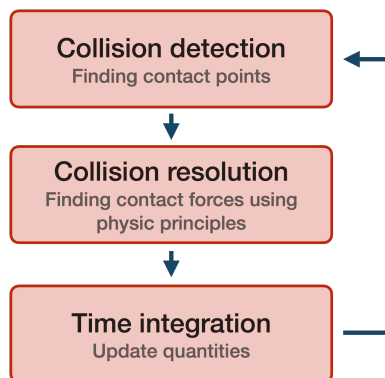
## Table of Contents

1 Introduction and motivation .....	6
2 Contribution #1 – Acceleration of collision detection .....	7
3 Contribution #2 – Differentiable collision detection .....	8
4 Contribution #3 – Review of contact models in robotics.....	9
5 Contribution #4 – The differentiable simulator, a first version.....	10
Reference List .....	12

# 1 Introduction and motivation

Simulation is a core component of modern robotics applications. Robot design, trajectory optimization, policy learning, and estimation all deeply rely on physical simulation intrinsically. Over the past two decades, various physical simulators have become established in robotics: Bullet, MuJoCo, DART, ODE, PhysX, and RaiSim, to name the most in usage.

Simulation pipelines are essentially composed of two main stages: (i) the collision detection phase and (ii) the contact restitution phase, as illustrated in Fig. 1. The collision detection phase enumerates the collision points and surfaces between the scene's various geometries. The contact restitution phase computes the contact forces originating from the collision according to the phenomenological law of mechanics (Coulomb friction law, Maximum Dissipation principle, Signorini conditions, etc.). From an algorithmic and computational perspective, these two phases can be cast as optimization problems that might be hard or complex to solve.



**Figure 1: Illustration of a standard simulation pipeline composed of two main phases: the collision detection phase and the collision resolution phase.**

In AGIMUS, we argue that simulation, despite its inherent complexity, is a key enabler for providing robots with increased autonomy to apprehend richer contact interactions with their environment and, thus, extend their capacities to solve complex tasks. Yet, modern robotics simulators remain imperfect: the resulting simulations come with several physical artifacts (inaccuracy, unphysical behaviors, violations of physical laws, etc.) that might impact the applications. Furthermore, simulation comes with an important computational cost, which should be lowered to facilitate its usage within robots equipped with limited computational resources. Additionally, modern simulators remain largely unable to provide gradient information, which is crucial and required by many applications: trajectory optimization, policy learning, etc.

We have been addressing these limitations during this first year of the AGIMUS project. We notably contributed to accelerating collision detection, providing an extended comparison of contact models, and developing methods and algorithms to differentiate these two stages rigorously from a mathematical and automatic-control perspective. These contributions correspond to the three following papers and submitted articles, published at the IEEE International Conference on Robotics and Automation 2023 or submitted to the journal IEEE Transactions on Robotics:

- Louis Montaut et al. “Differentiable collision detection: a randomized smoothing approach”. In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 3240–3246
- Louis Montaut et al. “GJK++: Leveraging Acceleration Methods for Faster Collision Detection”. working paper or preprint. Apr. 2023. URL: <https://hal.science/hal-04070039>
- Quentin Le Lidec et al. “Contact Models in Robotics: a Comparative Analysis”. working paper or preprint. Apr. 2023. URL: <https://hal.science/hal-04067291>

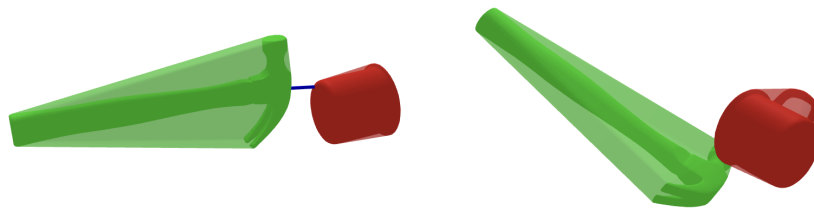
They are summarized in the following sections, and the associated preprints are reported at the end of this document.

## 2 Contribution #1 – Acceleration of collision detection

In our first contribution, we propose revisiting collision detection through the lens of convex optimization and accelerating it to reduce its overall computational burden in the simulation pipeline. This contribution has been submitted to IEEE Transactions on Robotics: Louis Montaut et al. “GJK++: Leveraging Acceleration Methods for Faster Collision Detection”. working paper or preprint. Apr. 2023. URL: <https://hal.science/hal-04070039>.

As shown in Fig. 2, collision detection implies determining which objects in the environment are in contact and where the contacts occur; this is the collision detection stage. In robotics applications involving complex scenes, a collision detection check must be made for each pair of simulated objects and each time step of the simulation. Thus, solving a task may require millions or even billions of collision checks to be made. It is, therefore, crucial for the collision detection pipeline to be as fast as possible. While introduced in 1988, the GJK algorithm [7] remains the most effective solution for collision detection in modern physics simulators. Over the years, it was shown to be efficient, scalable, and generic, operating on a broad class of convex shapes, ranging from simple primitives (sphere, ellipsoid, box, cone, capsule, etc.) to complex meshes involving thousands of vertices. However, the GJK algorithm has often been presented in the literature as a computational geometry algorithm; we argue that this view has been detrimental to understanding and enhancing collision detection.

Therefore, in our first contribution, we propose revisiting and accelerating collision detection by considering it as fundamentally a convex optimization problem. First, we reframe the GJK algorithm as a sub-case of the well-known Frank-Wolfe algorithm, one of the first convex optimization algorithms designed in 1956. This reframing allows us for a simpler and deeper understanding of collision detection. Additionally, it allows us to use recent results linking the Frank-Wolfe algorithm and the Polyak and Nesterov accelerations of gradient descent, two very common acceleration techniques used in unconstrained convex optimization. As our main contribution in this work, we therefore propose two new algorithms: the Polyak-accelerated and Nesterov-accelerated GJK algorithms, both of which accelerate the standard GJK algorithm, leading to an overall faster collision detection pipeline for physics



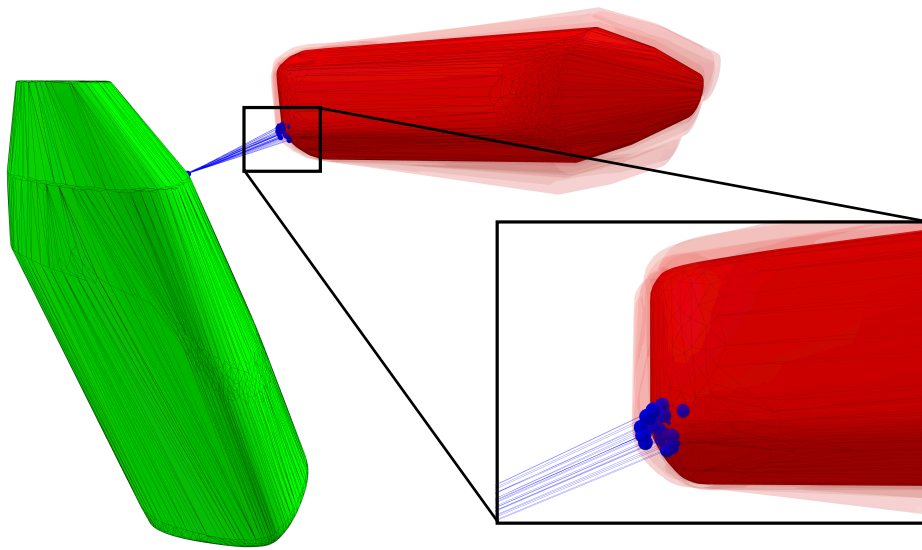
**Figure 2: Two distinct collision problems using shapes typically involved in robotics manipulation tasks: on the left, the shapes are not in collision, whereas on the right, the shapes are in collision. In robotics tasks involving complex environments, such collision problems can occur millions or billions of times.**

simulation. We benchmark these algorithms against the vanilla GJK algorithm and show that Nesterov and Polyak-accelerated GJK are up to two times faster than the original GJK algorithm. These two algorithms have been implemented and integrated into the open-source HPP-FCL C++ library [18], notably used by Pinocchio [3], and which counts an increasing number of users, reaching nowadays more than 50k downloads per month.

### 3 Contribution #2 – Differentiable collision detection

Although collision detection has been extensively studied over the years, differentiating through collision detection has only recently emerged as a central issue, motivated by various efforts from the scientific community on the topic of differentiable simulation.

So far, very few solutions have been suggested, only with a strong assumption on the nature of the shapes involved. In this second contribution, Louis Montaut et al. “Differentiable collision detection: a randomized smoothing approach”. In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 3240–3246, we introduce a generic and efficient approach to computing the derivatives of collision detection for any pair of convex shapes by notably leveraging randomized smoothing techniques, which have shown to be particularly adapted to capture the derivatives of non-smooth problems. Because shapes used in physics simulation and in real are often non-strictly convex, collision detection problems fall into that category of non-smooth problems, making them particularly difficult to differentiate. Consequently, simply using finite differences or automatic differentiation is insufficient to capture meaningful collision detection gradients. In contrast, randomized smoothing techniques [2, 6], which rely on smartly sampling a function to differentiate, have recently been reintroduced in machine learning to differentiate through non-smooth problems. We therefore propose two gradient estimators, one of which is illustrated in Fig. 3. Both of these gradient estimators use randomized smoothing at their core. We experimentally show that both methods give better gradients than standard methods like finite differences and are orders of magnitude faster to compute.

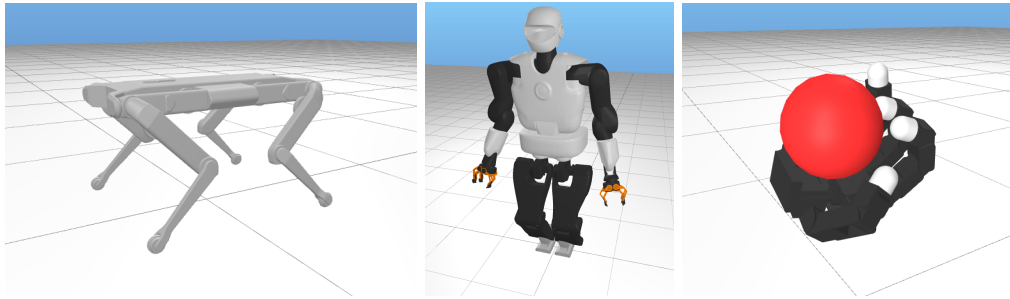


**Figure 3: Illustration of randomized smoothing approximation on two meshes typically used in robotics manipulation tasks. Randomized smoothing allows for estimating the curvature of object shapes.**

## 4 Contribution #3 – Review of contact models in robotics

In this third contribution, we provide an extended comparative analysis of contact models and computational methods implied in the collision restitution phase. It has been submitted to IEEE Transactions on Robotics: Quentin Le Lidec et al. “Contact Models in Robotics: a Comparative Analysis”. working paper or preprint. Apr. 2023. URL: <https://hal.science/hal-04067291>.

This comparative analysis complements the collision detection aspects covered in [17] by focusing on the resolution aspects of physical simulation. More precisely, it is motivated by the observation that, among all the simulators commonly used in robotics, there is a large variability in the choice of contact modeling (Signorini conditions, maximum dissipation principle, etc.) and the optimization approach for solving them. Because there is not yet a consensus on the most appropriate way of evaluating the contact restitution phase (performances, computational burden, accuracy, realism, etc.), we have proposed a detailed and extensive comparative analysis of the different models and algorithmic variations commonly used in robotics, reported in Table 1. We have notably compared three formulations: (i) the Linear Complementarity Problem (LCP), (ii) the Convex Complementarity Problem (CCP), and (iii) the Nonlinear Complementarity Problem (NCP), and their algorithmic variants, which come with different guarantees (model shift, fulfillment of the maximum dissipation principle, robustness to ill-conditioning, etc.). We compare them on different robotics scenarios ranging from sliding cube scenarios to more complex ones involving dexterous object manipulations and humanoid simulation, as depicted in Fig. 4. The proposed comparison goes beyond similar works on the topic [8] by investigating a richer class of contact problems and a larger number of algorithmic solutions for contact simulation. We will also release as open-source a complete software to properly and extensively compare contact formulations in robotics,



**Figure 4: Robotics systems used for the experiments. The Solo-12 quadruped (Left), the Talos humanoid (Center) and the Allegro hand (Right) allow to respectively exhibit locomotion, high-dimensional, and manipulation contact scenarios.**

**Table 1: Characteristics of various contact models.**

	Signorini	Coulomb	MDP	No shift	No internal forces	Robust	Convergence guarantees
<b>LCP</b>							
PGS [5, 19, 15, 14]	✓			✓			✓
Staggered projections [10]	✓			✓	✓	✓	
<b>CCP</b>							
PGS [1]		✓	✓	✓			✓
MuJoCo [20]		✓	✓		✓	✓	✓
ADMM		✓	✓	✓	✓	✓	✓
RaiSim [9]	✓	✓		✓			
<b>NCP</b>							
PGS	✓	✓	✓	✓			✓
Staggered projections [12]	✓	✓	✓	✓	✓	✓	

leveraging the Pinocchio toolbox [4].

The main conclusion of this analysis is that there is not yet a proper formulation and algorithmic solution to solve contact problems in general. Some further research needs to be done, notably at the interface of optimization and rigid-body dynamics algorithms to enhance physics simulation in robotics. This is precisely the topic of our next milestones within the frame of the AGIMUS project.

## 5 Contribution #4 – The differentiable simulator, a first version

The overall goal of WP2 is to build an efficient and accurate differentiable simulator for learning and control in robotics. During this first year, we built this differentiable simulator's first software version. This version leverages Pinocchio [4, 3] for efficient evaluation of kinematic and dynamic quantities involved in contact equations, and HPP-FCL [18] for efficient collision detection evaluation. The differentiation of the collision detection phase is restricted to basic geometries (e.g., sphere) where analytical formulation can be easily obtained. The



differentiation of the collision restitution phase leverages our recent work [12] on the differentiation of a cascade of Constrained Quadratic Programming problems, also known as the staggered projection algorithm [10] in the computer graphics community. Notably, as illustrated in our recent survey on contact simulation in robotics [11], the staggered projection contact algorithm provides accurate simulation results while being numerically robust. This first version of the differentiable simulator has been successfully applied in the context of optimal control of nonsmooth dynamical systems [13]. Yet, the current proposed approach remains relatively slow due to the cascade of QCQPs to differentiate.

During the second year of AGIMUS, we will extend and enhance the differentiable simulator. From a collision detection perspective, we will extend this software to account for larger sets of collision primitives, including convex and nonconvex meshes, by leveraging our recent work [16]. From a contact simulation perspective, we are currently developing new optimization techniques and methods to accurately and efficiently solve the complementarity problems at the core of contact dynamics with friction and dissipation. In parallel, we are also developing new rigid-body dynamics algorithms to drastically reduce the computational complexity of rigid-body simulation in robotics. To support all these developments, one senior software engineer and one post-doc will join Inria in October 2023.

## Reference List

- [1] Mihai Anitescu and Alessandro Tasora. “An iterative approach for cone complementarity problems for nonsmooth dynamics”. In: *Computational Optimization and Applications* 47.2 (2010), pp. 207–235.
- [2] Quentin Berthet et al. “Learning with Differentiable Perturbed Optimizers”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 9508–9519. URL: <https://proceedings.neurips.cc/paper/2020/file/6bb56208f672af0dd65451f869fedfd9-Paper.pdf>.
- [3] Justin Carpentier, Florian Valenza, Nicolas Mansard, et al. *Pinocchio: fast forward and inverse dynamics for poly-articulated systems*. <https://stack-of-tasks.github.io/pinocchio>. 2015–2021.
- [4] Justin Carpentier et al. “The Pinocchio C++ library: A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives”. In: *2019 IEEE/SICE International Symposium on System Integration (SII)*. IEEE. 2019, pp. 614–619.
- [5] Erwin Coumans and Yunfei Bai. *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. <http://pybullet.org>. 2016–2021.
- [6] John C Duchi, Peter L Bartlett, and Martin J Wainwright. “Randomized smoothing for stochastic optimization”. In: *SIAM Journal on Optimization* 22.2 (2012), pp. 674–701.
- [7] Elmer G Gilbert, Daniel W Johnson, and S Sathiya Keerthi. “A fast procedure for computing the distance between complex objects in three-dimensional space”. In: *IEEE Journal on Robotics and Automation* 4.2 (1988), pp. 193–203.
- [8] Peter Horak and Jeffrey C. Trinkle. “On the Similarities and Differences Among Contact Models in Robot Simulation”. In: *IEEE Robotics and Automation Letters* 4 (2019), pp. 493–499.
- [9] Jemin Hwangbo, Joonho Lee, and Marco Hutter. “Per-contact iteration method for solving contact dynamics”. In: *IEEE Robotics and Automation Letters* 3.2 (2018), pp. 895–902. URL: [www.raisim.com](http://www.raisim.com).
- [10] Danny M Kaufman et al. “Staggered projections for frictional contact in multibody systems”. In: (2008), pp. 1–11.
- [11] Quentin Le Lidec et al. “Contact Models in Robotics: a Comparative Analysis”. working paper or preprint. Apr. 2023. URL: <https://hal.science/hal-04067291>.
- [12] Quentin Le Lidec et al. “Differentiable simulation for physical system identification”. In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 3413–3420.
- [13] Quentin Le Lidec et al. “Leveraging Randomized Smoothing for Optimal Control of Nonsmooth Dynamical Systems”. working paper or preprint. Mar. 2022. DOI: [10.48550/arXiv.2203.03986](https://doi.org/10.48550/arXiv.2203.03986). URL: <https://hal.science/hal-03480419>.
- [14] Jeongseok Lee et al. “DART: Dynamic Animation and Robotics Toolkit”. In: *The Journal of Open Source Software* 3.22 (Feb. 2018), p. 500. DOI: [10.21105/joss.00500](https://doi.org/10.21105/joss.00500). URL: <https://doi.org/10.21105/joss.00500>.

- [15] Miles Macklin et al. “Small Steps in Physics Simulation”. In: *Proceedings of the 18th Annual ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA ’19. Los Angeles, California: Association for Computing Machinery, 2019. ISBN: 9781450366779. DOI: [10.1145/3309486.3340247](https://doi.org/10.1145/3309486.3340247). URL: <https://doi.org/10.1145/3309486.3340247>.
- [16] Louis Montaut et al. “Differentiable collision detection: a randomized smoothing approach”. In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 3240–3246.
- [17] Louis Montaut et al. “GJK++: Leveraging Acceleration Methods for Faster Collision Detection”. working paper or preprint. Apr. 2023. URL: <https://hal.science/hal-04070039>.
- [18] Jia Pan et al. *HPP-FCL: an extension of the Flexible Collision Library*. <https://github.com/humanoid-path-planner/hpp-fcl>. 2015–2022.
- [19] Russell Smith. *Open Dynamics Engine*. <http://www.ode.org/>. 2008. URL: <http://www.ode.org/>.
- [20] Emanuel Todorov, Tom Erez, and Yuval Tassa. “Mujoco: A physics engine for model-based control”. In: *2012 IEEE/RSJ international conference on intelligent robots and systems*. IEEE. 2012, pp. 5026–5033.

# GJK++: Leveraging Acceleration Methods for Faster Collision Detection

Louis Montaut, Quentin Le Lidec, Vladimir Petrik, Josef Sivic and Justin Carpentier

**Abstract**—Collision detection is a fundamental computational problem in various domains, such as robotics, computational physics, and computer graphics. In general, collision detection is tackled as a computational geometry problem, with the so-called Gilbert, Johnson, and Keerthi (GJK) algorithm being the most adopted solution nowadays. While introduced in 1988, GJK remains the most effective solution to compute the distance or the collision between two 3D convex geometries. Over the years, it was shown to be efficient, scalable, and generic, operating on a broad class of convex shapes, ranging from simple primitives (sphere, ellipsoid, box, cone, capsule, etc.) to complex meshes involving thousands of vertices. In this article, we introduce several contributions to accelerate collision detection and distance computation between convex geometries by leveraging the fact that these two problems are fundamentally optimization problems. Notably, we establish that the GJK algorithm is a specific sub-case of the well-established Frank-Wolfe (FW) algorithm in convex optimization. By adapting recent works linking Polyak and Nesterov accelerations to Frank-Wolfe methods, we also propose two accelerated extensions of the classic GJK algorithm. Through an extensive benchmark over millions of collision pairs involving objects of daily life, we show that these two accelerated GJK extensions significantly reduce the overall computational burden of collision detection, leading to up to two times faster computation timings. Finally, we hope this work will significantly reduce the computational cost of modern robotic simulators, allowing the speed-up of modern robotic applications that heavily rely on simulation, such as reinforcement learning or trajectory optimization.

**Index Terms**—Convex Optimization, Collision Detection, Computational Geometry, Computer Graphics, Simulation, Trajectory Optimization, Motion Planning

## I. INTRODUCTION

**P**HYSICS engines designed to simulate rigid bodies are an essential tool used in a wide variety of applications, notably in robotics, video games, and computer graphics [1]–[3]. Collision detection, a crucial feature of any physics engine or robot motion planner [4]–[6], consists of finding which objects are colliding or not, *i.e.* are sharing at least one common point or if there exists a separating hyper-plane between both. As simulation often needs to deal with multiple objects and run in real-time (*i.e.*, in video games) or at very high frequencies (*i.e.*,

Louis Montaut is with Inria, Département d’Informatique de l’École Normale Supérieure, PSL Research University in Paris, France and also with the Czech Institute of Informatics, Robotics and Cybernetics in Prague, Czech Republic.

Vladimir Petrik and Josef Sivic are with the Czech Institute of Informatics, Robotics and Cybernetics, Czech Technical University in Prague.

Quentin Le Lidec and Justin Carpentier are with Inria and Département d’Informatique de l’École Normale Supérieure, PSL Research University in Paris, France.

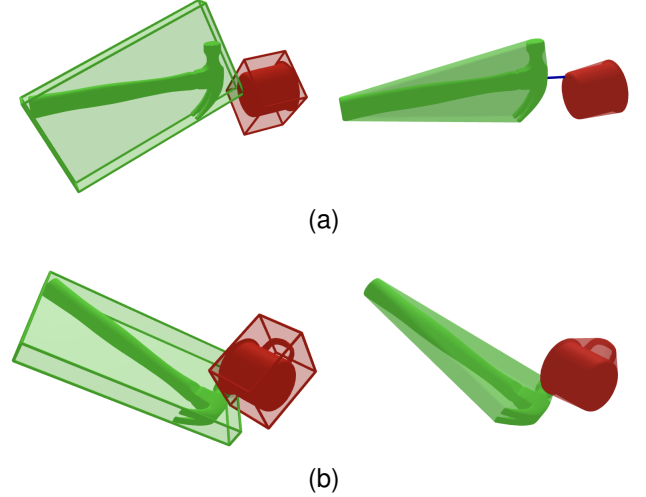


Fig. 1. Two distinct collision problems using shapes from the YCB dataset: in (a) the shapes  $\mathcal{A}_1$  and  $\mathcal{A}_2$  are not in collision ( $\text{dist}(\mathcal{A}_1, \mathcal{A}_2) > 0$ ) whereas in (b) the shapes are in collision ( $\text{dist}(\mathcal{A}_1, \mathcal{A}_2) = 0$ ). In the left column, the oriented bounding boxes (OBB) of the objects are represented in light colors. In the right column, the light colors represent the convex hull of each object. In both collision problems, (a) and (b), the broad phase finds a collision between the object’s OBBs; the narrow phase must thus be called to confirm or infirm the collision. The right column corresponds to the narrow phase in which the GJK algorithm is called on the objects’ convex hulls. In this paper, we propose the Polyak-accelerated GJK and Nesterov-accelerated GJK algorithms in order to accelerate collision detection.

in robotics), collision detection must be carried out as fast as possible. To reduce computational times, collision detection is usually decomposed into two phases thoroughly covered in [7]. The first phase is the so-called *broad phase* which consists in identifying which pair of simulated objects are potentially colliding. The broad phase relies on the simulated objects’ bounding volumes, as shown in Fig. 1, allowing to quickly assess if the objects are *not* in collision. The second phase is the so-called *narrow phase* in which each pair identified in the broad phase is tested to check whether a collision is truly occurring. Collision detection during the narrow phase is the focus of this paper.

**Problem formulation.** We consider two convex shapes  $\mathcal{A}_1$  and  $\mathcal{A}_2$  in  $\mathbb{R}^n$  (with  $n = 2$  or  $3$  in common applications). If the shapes are not convex, we use their respective convex hulls or decompose them into a collection of convex sub-shapes [8]. The *separation distance* between  $\mathcal{A}_1$  and  $\mathcal{A}_2$ , denoted by  $\text{dist}(\mathcal{A}_1, \mathcal{A}_2) \in \mathbb{R}_+$ , can be formulated as a

minimization problem of the form:

$$\begin{aligned} d_{1,2} &= \min_{\mathbf{x}_1 \in \mathcal{A}_1, \mathbf{x}_2 \in \mathcal{A}_2} \|\mathbf{x}_1 - \mathbf{x}_2\|^2 \\ \text{and } \text{dist}(\mathcal{A}_1, \mathcal{A}_2) &= \sqrt{d_{1,2}}, \end{aligned} \quad (1)$$

where  $\mathbf{x}_1 \in \mathcal{A}_1$  and  $\mathbf{x}_2 \in \mathcal{A}_2$  are both vectors in  $\mathbb{R}^n$ ,  $d_{1,2}$  is the optimal value of (1) and  $\|\cdot\|$  is the Euclidian norm of  $\mathbb{R}^n$ . If  $\mathcal{A}_1$  and  $\mathcal{A}_2$  intersect (*i.e.*, they are in collision), we necessarily have  $\text{dist}(\mathcal{A}_1, \mathcal{A}_2) = 0$ . If the two shapes do not intersect, we have  $\text{dist}(\mathcal{A}_1, \mathcal{A}_2) > 0$ . These two cases are illustrated in Fig. 1.

Problem (1) allows us to consider both the *distance computation* problem and the computationally cheaper *Boolean collision check* as one single convex optimization problem. In the distance computation problem, we aim at computing the separation distance between  $\mathcal{A}_1$  and  $\mathcal{A}_2$ , denoted  $\text{dist}(\mathcal{A}_1, \mathcal{A}_2)$ , *i.e.* the distance between their closest points. This distance is helpful in some applications such as collision-free path planning [9], [10], especially for pairs of objects entering the narrow phase. If the broad phase has not selected a pair of objects, a cheap estimate of  $\text{dist}(\mathcal{A}_1, \mathcal{A}_2)$  is usually enough [7]. In the Boolean collision check, we only aim at determining if  $\mathcal{A}_1$  and  $\mathcal{A}_2$  intersect, and computing  $\text{dist}(\mathcal{A}_1, \mathcal{A}_2)$  is unnecessary. However, we will later see that the Boolean collision check is a sub-problem of the distance computation problem: solving (1) can be early-stopped once a separating plane between  $\mathcal{A}_1$  and  $\mathcal{A}_2$  has been found. In the rest of this paper, we will use the generic term “collision detection” to refer to distance computation and Boolean collision checking altogether. We will specify when the distinction is needed.

**Related work.** The so-called Gilbert-Johnson-Keerthi algorithm (GJK) [11] is the most well-known algorithm for collision detection between two convex shapes. It can handle the distance computation and the Boolean collision check [12]. The expanding polytope algorithm (EPA) [13], an extension to GJK, can compute the penetration depth *i.e.* the norm of the separation vector, when shapes are in collision. The separation vector is the vector of smallest norm needed to translate one of the two shapes such that the two shapes do not intersect. The EPA solves a non-convex and more complex problem than (1), which is not the focus of this paper.

Most alternatives to GJK in the literature focus on computing collisions between convex polyhedra, such as the Lin-Canny algorithm [14] or the V-Clip [15] algorithm. Although GJK is equivalent in performance to these algorithms [16], it is not restricted to convex polyhedra. The strength of GJK is formulating the collision detection problem on the Minkowski difference. The properties of the Minkowski difference are used to cleverly compute support vectors on the Minkowski difference (these notions are introduced and detailed in Sec. II). GJK can thus handle collision detection, and distance computation for many different shapes such as convex polyhedra and basic primitives (*i.e.*, spheres, ellipsoids, cylinders, capsules etc.) [7], [12], [17]. The generality of GJK, efficiency, good precision, and ease of implementation make it the state-of-the-art algorithm for collision detection between

two convex shapes.

Traditionally, collision detection is considered a computational geometry problem. Over the years, this computational geometric perspective allowed enhancing the computational efficiency of GJK, thanks to improvements to its internal sub-routines [12], [18]. However, we argue that this view has also limited collision detection improvement. Instead, we propose to tackle collision from the perspective of convex optimization. This correlates with some observations raised in the original GJK papers. Indeed, as briefly mentioned already in their 1988 paper [11] and brought up again by [19], [20], the ideas developed by Gilbert, Johnson, and Keerthi are rooted in convex optimization, notably in the works of [21] and [22] for solving Minimum-Norm Point (MNP) problems. This article proposes exploiting the Frank-Wolfe convex optimization setting to tackle collision detection. In particular, by leveraging recent progresses in acceleration methods in convex optimization [23], we show how to accelerate collision detection by directly lowering the number of iterations needed to solve a collision problem instance compared to the vanilla GJK algorithm.

The Frank-Wolfe algorithm (FW) dates back to 1956 and is one of the first convex optimization algorithms. It has been heavily studied over the years by the optimization community. This algorithm iterates over the computation of *support points* to approach the optimal solution. The undesired zig-zagging behavior of FW, already identified by its authors, has been addressed by introducing corrections to the original FW method [21], [22], [24]–[28]. In [26] and [28], widely used corrections of the FW algorithm are analyzed, and their convergence properties. In this work, we notably show in Sec. II that the GJK algorithm is an instance of the *fully-corrective* Frank-Wolfe algorithm, covered in [28], applied to solving a MNP problem. Finally, recent works have also tried accelerating the FW algorithm by applying the so-called Nesterov acceleration [29], a classic acceleration technique in unconstrained optimization. Nesterov momentum has been successfully added by [30] to accelerate FW. In [20], Qin and An take a different approach as they are interested in the general problem of projecting a point onto a Minkowski difference in *any* dimension. To accelerate the theoretical convergence of the 1966 Gilbert algorithm, the authors devise the NESMINO algorithm, which exploits the classic Nesterov acceleration. However, by introducing a smoothing term, the minimization problem (1) is modified. By doing so, the authors rely on successive projections on the shapes instead of computing support points. This makes the NESMINO algorithm similar to the projected-gradient descent method. Furthermore, although the NESMINO algorithm uses the Nesterov acceleration, as pointed out by the authors, it does not accelerate over the original 1966 Gilbert algorithm. In Sec. IV, we experimentally confirm that the NESMINO algorithm is slower when compared to GJK and our accelerated versions.

**Contributions.** Our work builds on the seminal works by [31] and [11] as well as on the work of [30], [32] to globally accelerate distance computation and collision checking algo-

rithms between convex shapes. We make these three main contributions:

- We recast the collision detection problem as a convex optimization problem that the FW algorithm can solve. Using the ideas developed by Gilbert, Johnson, and Keerthi, we show that GJK is, in fact, a sub-case of the fully-corrective FW algorithm;
- We adapt recent works on Polyak and Nesterov-accelerated FW to accelerate both the distance computation and the Boolean collision check problems;
- We empirically analyze the convergence of our proposed approach on two large shape benchmarks. Results show a faster convergence of our approach leading to a computational time up to two times faster than the state-of-the-art GJK algorithm on both distance computation and Boolean collision checking.
- We empirically show that GJK-like algorithms, which our proposed methods belong to, are superior by orders of magnitude to generic quadratic programming solvers on collision detection problems;
- Finally, we show that our methods can be used in any physics simulator by benchmarking them on trajectories generated by the Bullet simulator. Like GJK, our methods can benefit from being warm-started using the previous simulation time steps, enabling temporal coherence for our proposed accelerated collision detection algorithms.

This article is an extended version of a previously published paper [33] which presented the Nesterov-accelerated GJK algorithm. In the present article, we notably introduce the Polyak-accelerated GJK algorithm and show that this acceleration is faster than the vanilla GJK algorithm while being more robust than the Nesterov acceleration of GJK when shapes involved in a collision problem are distant or have a large overlap. We benchmark this novel Polyak-accelerated GJK algorithm as well as Nesterov-accelerated GJK against vanilla GJK on a dataset of objects used in robotics manipulation, the YCB dataset [34]. This dataset contains 3D scans of real-life objects and yields more challenging collision problems than those constructed with the ShapeNet dataset used in [33]. In addition to these extensive benchmarks, we complement the experimental approach in [33] by comparing vanilla GJK and our methods to state-of-the-art generic quadratic programming solvers. Finally, we extend on [33] by experimentally demonstrating that our proposed methods can be used in the context of physics simulation. Like the vanilla GJK algorithm, we show that both Polyak and Nesterov-accelerated GJK benefit from being warm-started using previous simulation time steps.

**Paper outline.** The paper is organized as follows. In Sec. II, we recast the distance computation problem as a Frank-Wolfe instance. We introduce the duality gap of the FW method, allowing us to bound the distance to the optimal solution of the distance computation problem. We also present the fully-corrective version of FW and show the link between GJK and FW. In Sec. III, we introduce recent work on Polyak and Nesterov-accelerated FW and show how to adapt them for both distance computation and Boolean collision checking. For

distance computation, we adapt the convergence criterion of FW when using Polyak and Nesterov accelerations in order to retain the bound on the distance to the optimal solution. We also propose to adapting the Nesterov and Polyak acceleration schemes for non-strictly convex shapes. Finally, in Sec. IV, we evaluate our approach against the state-of-the-art GJK algorithm on two benchmarks containing both strictly convex shapes and non-strictly convex shapes.

## II. COLLISION DETECTION FROM A FRANK-WOLFE PERSPECTIVE

In this section, we highlight the natural connection between computing the distance between convex shapes and convex optimization, particularly within the frame of the Frank-Wolfe setting. We notably show that the GJK algorithm can be seen as a variant of the Frank-Wolfe algorithm that leverages properties of convex 3D shapes to lower the computational complexity drastically.

**Distance computation and Boolean collision checking.** As recalled in Sec. I, collision detection is a sub-case of distance computation:  $\text{dist}(\mathcal{A}_1, \mathcal{A}_2) > 0$  means that the two shapes do not overlap while  $\text{dist}(\mathcal{A}_1, \mathcal{A}_2) = 0$  means that the shapes are in collision. In the case of  $\text{dist}(\mathcal{A}_1, \mathcal{A}_2) > 0$ , finding a strictly positive lower bound on  $d_{1,2}$  to solve the collision problem is sufficient. In the context of convex shapes, this is often simpler than computing the distance between the two shapes [10] and can be done by finding a plane separating  $\mathcal{A}_1$  from  $\mathcal{A}_2$ . In the rest of the paper, we focus on the generic problem of computing the distance between  $\mathcal{A}_1$  and  $\mathcal{A}_2$ , as it encapsulates the more straightforward Boolean collision check covered later in this section. Results for the particular Boolean collision checking case are analyzed in the experimental section IV.

**Collision detection from the perspective of quadratic programming.** From the perspective of numerical optimization, the first idea is to look at problem (1) through the lens of quadratic programming. In the case of meshes, which are shapes represented by soups of 3D points and which faces represented as triangles, we can use the implicit description of a convex mesh as a linear inequality of the form  $Ax \leq b$ . The collision detection problem between two meshes can thus be cast as a quadratic programming (QP) problem:

$$\begin{aligned} d_{1,2} = \min_{\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^3} \quad & \|\mathbf{x}_1 - \mathbf{x}_2\|^2 \\ \text{s.t.} \quad & A_1 \mathbf{x}_1 \leq \mathbf{b}_1 \\ & A_2 \mathbf{x}_2 \leq \mathbf{b}_2. \end{aligned} \quad (2)$$

While many off-the-shelf solvers exist to solve QP problems, their performances scale poorly with respect to the number of constraints [35]. This is especially true in the presence of complex meshes composed of hundreds or thousands of vertices, for which QP solvers can take a few milliseconds to assess a collision, as we experimentally highlight in Sec. IV-C. Instead, we turn our attention to dedicated solutions such as GJK, which has been shown to operate on a large class



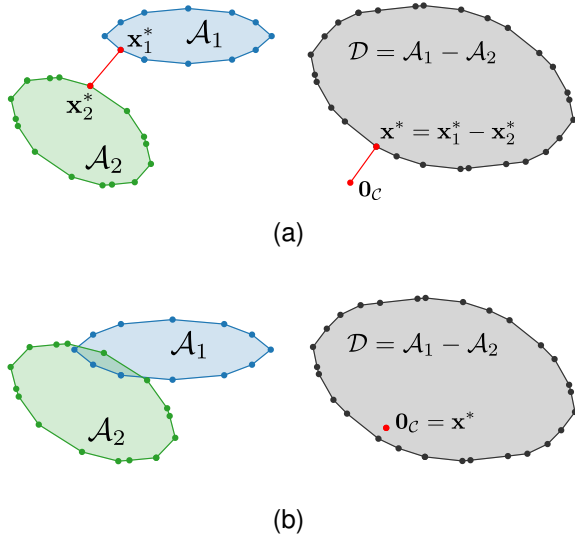


Fig. 2. (a) Distant vs. (b) overlapping pairs of shapes and their respective Minkowski difference. Left column: two convex shapes in 2D. Right column: the Minkowski difference  $\mathcal{D}$  of  $\mathcal{A}_1$  and  $\mathcal{A}_2$ . Since  $\mathcal{A}_1$  and  $\mathcal{A}_2$  are convex,  $\mathcal{D}$  is also convex. In (a), the shapes are not in collision hence the origin of the configuration space  $\mathcal{C}$ ,  $\mathbf{0}_C$  (in red) lies outside the Minkowski difference,  $\mathbf{0}_C \notin \mathcal{D}$ . The vector  $\mathbf{x}^* = \mathbf{x}_1^* - \mathbf{x}_2^*$  separates  $\mathcal{A}_1$  from  $\mathcal{A}_2$ . It is also equal to the projection of  $\mathbf{0}_C$  onto the Minkowski difference,  $\mathbf{x}^* = \text{proj}_{\mathcal{D}}(\mathbf{0}_C)$ . In (b), the shapes overlap, thus  $\mathbf{0}_C \in \mathcal{D}$ . In this case, we have  $\mathbf{x}^* = \text{proj}_{\mathcal{D}}(\mathbf{0}_C) = \mathbf{0}_C$ .

of shapes, ranging from simple primitives to very complex meshes.

**Recasting the distance computation problem onto the Minkowski difference.** The first important idea of 1988's paper by Gilbert, Johnson, and Keerthi [11] is to recast the distance computation problem onto the Minkowski difference  $\mathcal{D}$  of the shapes and defined as follows:

$$\mathcal{D} = \mathcal{A}_1 - \mathcal{A}_2 = \{\mathbf{x} = \mathbf{x}_1 - \mathbf{x}_2 \mid \mathbf{x}_1 \in \mathcal{A}_1, \mathbf{x}_2 \in \mathcal{A}_2\} \subset \mathcal{C}, \quad (3)$$

where  $\mathcal{C} = \mathbb{R}^n$  is the so-called *collision space*. The shapes  $\mathcal{A}_1$  and  $\mathcal{A}_2$  lie in the shape space and the Minkowski difference  $\mathcal{D}$  lies in the collision space. Although both the shape space and the collision space are in  $\mathbb{R}^n$ , we distinguish between the two to highlight the change in perspective. In Fig. 2, we illustrate the link between a pair of two convex shapes and their corresponding Minkowski difference. We stress that the Minkowski difference  $\mathcal{D}$  is *specific* to shapes  $\mathcal{A}_1$  and  $\mathcal{A}_2$ . If the relative position or relative orientation between  $\mathcal{A}_1$  and  $\mathcal{A}_2$  changes, their Minkowski difference changes accordingly.

The following properties, illustrated in Fig. 2, hold for the Minkowski difference  $\mathcal{D}$ :

- 1) Since  $\mathcal{A}_1$  and  $\mathcal{A}_2$  are convex sets,  $\mathcal{D}$  is also convex.
- 2) If  $\mathcal{A}_1$  and  $\mathcal{A}_2$  are intersecting, the origin of  $\mathcal{C}$ , denoted as  $\mathbf{0}_C$ , lies inside the Minkowski difference  $\mathcal{D}$ , *i.e.*  $\mathbf{0}_C = \mathbf{x}_1 - \mathbf{x}_2$  for some  $\mathbf{x}_1 \in \mathcal{A}_1$  and  $\mathbf{x}_2 \in \mathcal{A}_2$ .
- 3) If  $\mathcal{A}_1$  and  $\mathcal{A}_2$  are not intersecting, the projection of  $\mathbf{0}_C$  onto  $\mathcal{D}$ ,  $\mathbf{x}^* = \text{proj}_{\mathcal{D}}(\mathbf{0}_C)$ , corresponds to two vectors  $\mathbf{x}_1^* \in \mathcal{A}_1$  and  $\mathbf{x}_2^* \in \mathcal{A}_2$ , also called witness vectors in the computational geometry literature [7]. Contrary to  $\mathbf{x}^*$ , these vectors  $\mathbf{x}_1^*$  and  $\mathbf{x}_2^*$  are not necessarily unique, as

---

**Algorithm 1** Frank-Wolfe algorithm with linesearch [26]

---

Let  $\mathbf{x}_0 \in \mathcal{D}$ ,  $\epsilon > 0$

For  $k=0, 1, \dots$  do

- 1:  $\mathbf{d}_k = \nabla f(\mathbf{x}_k)$  ▷ Direction of support
  - 2:  $\mathbf{s}_k \in \arg \min_{\mathbf{s} \in \mathcal{D}} \langle \mathbf{d}_k, \mathbf{s} \rangle (= S_{\mathcal{D}}(\mathbf{d}_k))$  ▷ Support (8)
  - 3: If  $g_{FW}(\mathbf{x}_k) \leq \epsilon$ , return  $f(\mathbf{x}_k)$  ▷ Duality gap (16)
  - 4:  $\gamma_k = \arg \min_{\gamma \in [0,1]} f(\gamma \mathbf{x}_k + (1-\gamma)\mathbf{s}_k)$  ▷ Linesearch
  - 5:  $\mathbf{x}_{k+1} = \gamma_k \mathbf{x}_k + (1-\gamma_k)\mathbf{s}_k$  ▷ Update iterate  
*In the case of the distance computation problem (4), where  $f(\mathbf{x}) = \|\mathbf{x}\|^2$ , line 4-5 correspond to projecting  $\mathbf{0}_C$  on the segment  $[\mathbf{x}_k, \mathbf{s}_k]$ :*
  - 6:  $\mathbf{x}_{k+1} = \text{proj}_{[\mathbf{x}_k, \mathbf{s}_k]}(\mathbf{0}_C)$  ▷ Project  $\mathbf{0}_C$  on  $[\mathbf{x}_k, \mathbf{s}_k]$
- 

is the case for non-strictly convex shapes such as two parallel boxes.

- 4) Finally, we always have  $\|\mathbf{x}^*\| = \text{dist}(\mathcal{A}_1, \mathcal{A}_2)$ .

This final remark allows us to recast the distance computation problem (1) onto the Minkowski difference as follows:

$$d_{1,2} = \min_{\mathbf{x} \in \mathcal{D}} \|\mathbf{x} - \mathbf{0}_C\|^2 = \min_{\mathbf{x} \in \mathcal{D}} \|\mathbf{x}\|^2. \quad (4)$$

The convex optimization problem (4) is equivalent to (1) and is known as a Minimum-Norm Point problem in the optimization literature [22], [28], [36]. In our case,  $\mathbf{0}_C \in \mathcal{C} = \mathbb{R}^n$  is the null vector *i.e.* the origin of the collision space. We thus aim at finding the point in  $\mathcal{D}$  with the lowest norm. This vector  $\mathbf{x}^*$  is the *optimal solution* to (4), given by  $d_{1,2} = \|\mathbf{x}^*\|^2 = \text{dist}(\mathcal{A}_1, \mathcal{A}_2)^2$ .

Directly computing the Minkowski difference  $\mathcal{D}$  is neither analytically tractable nor computationally efficient. Most of the first and second-order methods for constrained convex optimization problems, such as projected gradient descent or interior point methods [37], are thus sub-optimal choices. However, computing support vectors of the Minkowski difference  $\mathcal{D}$ , a notion defined hereinafter in this section, is relatively simple and largely demonstrated by [11]. As we discuss next, solving convex optimization problems by computing support vectors is the strength of the Frank-Wolfe algorithm and its variants [26].

**Distance computation using the Frank-Wolfe algorithm.**

The Frank-Wolfe algorithm (FW) [31] is one of the oldest convex optimization methods and solves the following constrained optimization problem:

$$f(\mathbf{x}^*) = \min_{\mathbf{x} \in \mathcal{D}} f(\mathbf{x}), \quad (5)$$

where  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  is a convex and differentiable function and  $\mathcal{D}$  is a compact convex set. For our distance computation problem (4), we use  $f(\mathbf{x}) = \|\mathbf{x}\|^2$  and the Minkowski difference  $\mathcal{D}$  as convex constraint set. As a side note, the following discussed algorithms all require an initial starting point  $\mathbf{x}_0 \in \mathcal{D}$ . Shapes used in physics engines are usually attached to a frame to keep track of their position and orientation in space. We denote  $\mathbf{c}^1 \in \mathcal{A}_1$  and  $\mathbf{c}^2 \in \mathcal{A}_2$  the origins of the frames attached to  $\mathcal{A}_1$  and  $\mathcal{A}_2$ , respectively. In the rest of this paper, we take  $\mathbf{x}_0 = \mathbf{c}^1 - \mathbf{c}^2$ .

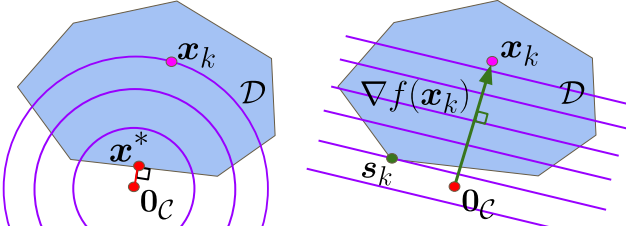


Fig. 3. Computing a support vector  $s_k$  in direction  $\nabla f(x_k)$  on convex set  $\mathcal{D}$ . We illustrate with the example of distance computation. On the left, we draw the Minkowski difference  $\mathcal{D}$  of which point of minimum norm (MNP) is  $x^*$  i.e.  $x^*$  is the projection of  $0_C$  onto  $\mathcal{D}$ ,  $x^* = \text{proj}_{\mathcal{D}}(0_C)$ . The iterate at iteration  $k$  of the FW algorithm is  $x_k$ . In purple we draw the level sets of the function  $f(x) = \|x\|^2$ . On the right, we draw in purple the level sets of the linearization of  $f$  at iterate  $x_k$ ,  $h_k$ . The first step of the FW algorithm is to compute support vector  $s_k$  in the direction of  $\nabla f(x_k)$  (green arrow),  $s_k \in S_{\mathcal{D}}(\nabla f(x_k))$ . In the second step of the FW algorithm, we compute  $x_{k+1}$  as a convex combination of  $x_k$  and  $s_k$  i.e.  $x_{k+1}$  is a point on the segment  $[x_k, s_k]$ .

The FW algorithm, summarized in Alg. 1, is a gradient-descent method. It consists in iteratively applying two steps in order to converge towards the optimal solution  $x^*$  of (5). If we denote by  $x_k$  the estimate of  $x^*$  at iteration  $k$ , these two steps correspond to:

- 1) First, we compute a support vector  $s_k$  in the direction of  $\nabla f(x_k)$ , by solving a linear optimization problem on  $\mathcal{D}$ .
- 2) Second, we update our current iterate  $x_k$  to obtain  $x_{k+1}$ , by taking a convex combination of the current iterate  $x_k$  and the computed support vector  $s_k$ .

In the following, we detail these steps in the context of distance computation. At iteration  $k$ , the current iterate  $x_k$  is the estimate of the optimal solution  $x^*$  and  $f(x_k)$  is the estimate of the optimal value of (5),  $f(x^*)$ , at iteration  $k$ . We write the linearization of the function  $f$  at  $x_k$  and denote it as  $h_k$ :

$$h_k(s) = f(x_k) + \langle \nabla f(x_k), s - x_k \rangle \quad (6)$$

where  $s$  is a vector of  $\mathbb{R}^n$ ,  $\nabla f(x_k)$  is the gradient of  $f$  at  $x_k$  and  $\langle \cdot, \cdot \rangle$  denotes the dot product between two vectors of  $\mathbb{R}^n$ .

→ **Step 1.** The first step of the FW algorithm at iteration  $k$  consists of finding a minimizer  $s_k \in \mathcal{D}$  of  $h_k$  on the convex set  $\mathcal{D}$  (line 2 in Alg. 1). Such a vector  $s_k$  is called a *support vector* of  $\mathcal{D}$  or simply a *support* and is defined as follows:

$$s_k \in \arg \min_{s \in \mathcal{D}} h_k(s) = \arg \min_{s \in \mathcal{D}} \langle \nabla f(x_k), s \rangle. \quad (7)$$

Fig. 3 gives a graphical understanding of support  $s_k$ . The vector  $s_k$  belongs to  $\mathcal{D}$  and is in the *most opposite* direction w.r.t.  $\nabla f(x_k)$ . In order to highlight the importance of the direction in which a support  $s_k$  is computed, we now introduce the notion of *support direction* and *support function*. Given a support direction  $d \in \mathbb{R}^n$ , the support function  $S_{\mathcal{D}}$  returns a set of  $\mathcal{D}$  and is defined as:

$$S_{\mathcal{D}}(d) = \arg \min_{s \in \mathcal{D}} \langle d, s \rangle \subset \mathcal{D}. \quad (8)$$

The support function  $S_{\mathcal{D}}$  may return a set with more than one vector. We only need to use one vector of this set. Thinking in terms of the direction of support allows us to understand

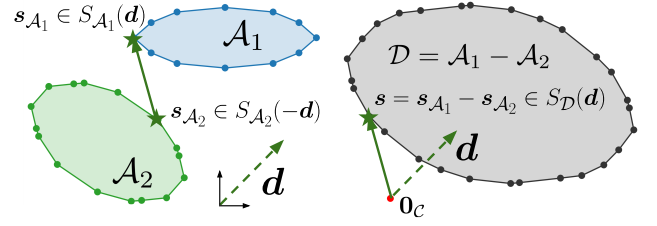


Fig. 4. Computing a support vector on the Minkowski difference using support vectors (represented by star shapes in the drawing) on the individual shapes. The vector  $s_{A_1}$  is a support vector of shape  $A_1$  in direction  $d$ . The vector  $s_{A_2}$  is a support vector of shape  $A_2$  in direction  $-d$ . The constructed vector  $s = s_{A_1} - s_{A_2}$  is a support vector of the Minkowski difference  $\mathcal{D}$  in the direction  $d$ .

that this direction can be rescaled while preserving the output of the support function:

$$\forall d \in \mathbb{R}^n, \forall \alpha > 0, S_{\mathcal{D}}(\alpha d) = S_{\mathcal{D}}(d). \quad (9)$$

A support  $s_k \in \mathcal{D}$  at iteration  $k$  is thus computed in the direction  $d_k = \nabla f(x_k)$  and belongs to  $S_{\mathcal{D}}(\nabla f(x_k))$ ,  $s_k \in S_{\mathcal{D}}(\nabla f(x_k))$ .

We now explain how to compute the support vector  $s_k$  in the case of the distance computation problem (4) where we minimize  $f(x) = \|x\|^2$  on the Minkowski difference  $\mathcal{D}$  of  $A_1$  and  $A_2$ . First, we have  $\nabla f(x) = 2x$ . Therefore, in the case of problem (4), it follows that:

$$s_k \in S_{\mathcal{D}}(x_k) = \arg \min_{s \in \mathcal{D}} \langle x_k, s \rangle. \quad (10)$$

As demonstrated by [11], any vector  $s \in S_{\mathcal{D}}(d)$  related to the Minkowski difference can be decomposed as the difference between two support vectors  $s_{A_1} \in S_{A_1}(d)$  and  $s_{A_2} \in S_{A_2}(-d)$  over the two individual shapes, leading to the following relation:

$$s = s_{A_1} - s_{A_2} \in S_{\mathcal{D}}(d). \quad (11)$$

Equation (11) shows that we can construct a support of the Minkowski difference from the supports of the original shapes. This property highlights the powerful change of perspective of working on the Minkowski difference. Indeed, there exists a large number of shapes for which computing supports is simple: spheres, ellipsoids, cylinders, capsules, polytopes etc. [7], [12], [17]. Fig. 4 illustrates the construction of a support of the Minkowski difference  $\mathcal{D}$  using the supports of the original shapes  $A_1$  and  $A_2$ .

→ **Step 2.** Once a support vector  $s_k \in S_{\mathcal{D}}(x_k)$  has been computed, we update the iterate  $x_k$  to obtain  $x_{k+1}$  by taking a convex combination between  $s_k$  and  $x_k$ . The original FW algorithm uses a parameter-free update:

$$x_{k+1} = \gamma_k x_k + (1 - \gamma_k) s_k, \quad (12)$$

where  $\gamma_k = \frac{k+1}{k+2} \in [0, 1]$  controls the step size. Alternatively, a line search can be carried out to find a better iterate  $x_{k+1}$  (line 4 in Alg. 1):

$$\begin{aligned} \gamma_k &= \arg \min_{\gamma \in [0, 1]} f(\gamma x_k + (1 - \gamma) s_k) \\ x_{k+1} &= \gamma_k x_k + (1 - \gamma_k) s_k. \end{aligned} \quad (13)$$

In the distance computation case where  $f(x) = \|x\|^2$ , this linesearch (13) is equivalent to projecting  $\mathbf{0}_C$  onto the segment  $[\mathbf{x}_k, \mathbf{s}_k]$ ,  $\mathbf{x}_k = \text{proj}_{[\mathbf{x}_k, \mathbf{s}_k]}(\mathbf{0}_C)$  (line 4 in Alg. 1). Since  $\mathcal{D}$  is convex, both (12) and (13) updates are guaranteed to remain in  $\mathcal{D}$ .

**Stopping criteria.** As Frank-Wolfe deals with convex problems, the *duality gap* associated with problem (5) can be used as a stopping criterion. Due to its convexity, the function  $f$  is always above its linearization. Otherwise said, for any  $\mathbf{x} \in \mathbb{R}^n$  and any  $\mathbf{s} \in \mathbb{R}^n$ :

$$f(\mathbf{s}) \geq f(\mathbf{x}) + \langle \nabla f(\mathbf{x}), \mathbf{s} - \mathbf{x} \rangle. \quad (14)$$

Reworking this inequality and applying the min operator enables us to compute the Frank-Wolfe duality gap  $g_{\text{FW}}(\mathbf{x}) \in \mathbb{R}_+$  which gives an upper-bound on the difference  $f(\mathbf{x}) - f(\mathbf{x}^*)$ :

$$f(\mathbf{x}) - f(\mathbf{x}^*) \leq -\min_{\mathbf{s} \in \mathcal{D}} \langle \nabla f(\mathbf{x}), \mathbf{s} - \mathbf{x} \rangle = g_{\text{FW}}(\mathbf{x}). \quad (15)$$

In particular, at iteration  $k$  of the FW algorithm, we have:

$$f(\mathbf{x}_k) - f(\mathbf{x}^*) \leq g_{\text{FW}}(\mathbf{x}_k) = \langle \nabla f(\mathbf{x}_k), \mathbf{x}_k - \mathbf{s}_k \rangle, \quad (16)$$

where  $\mathbf{s}_k \in S_{\mathcal{D}}(\nabla f(\mathbf{x}_k))$  is the support vector computed at iteration  $k$  in the direction of  $\nabla f(\mathbf{x}_k)$ . The duality-gap  $g_{\text{FW}}(\mathbf{x}_k)$  serves as a convergence criterion for the Frank-Wolfe method and is cheap to compute. Applied to the distance computation problem (4), the duality gap at iteration  $k$ ,  $g_{\text{FW}}(\mathbf{x}_k)$ , guarantees that:

$$\|\mathbf{x}_k\|^2 - \|\mathbf{x}^*\|^2 \leq g_{\text{FW}}(\mathbf{x}_k) = 2\langle \mathbf{x}_k, \mathbf{x}_k - \mathbf{s}_k \rangle. \quad (17)$$

Using the triangular inequality of the Euclidian norm and the convexity of the Minkowski difference  $\mathcal{D}$ , we can show that:

$$\|\mathbf{x}_k - \mathbf{x}^*\|^2 \leq \|\mathbf{x}_k\|^2 - \|\mathbf{x}^*\|^2 \leq g_{\text{FW}}(\mathbf{x}_k). \quad (18)$$

Inequality (18) is useful in practice as it allows the fine control of the desired tolerance on the distance to the optimal solution  $\mathbf{x}^*$  (line 3 in Alg. 1). Indeed, if ones wants to compute an estimate  $\mathbf{x}$  of the optimal solution  $\mathbf{x}^*$  at precision  $\epsilon$ , meaning that  $\|\mathbf{x} - \mathbf{x}^*\| \leq \sqrt{\epsilon}$ , it is sufficient to check that  $g_{\text{FW}}(\mathbf{x}) \leq \epsilon$ .

**Boolean collision checking.** As mentioned earlier, the problem of distance computation encompasses the problem of collision checking. Indeed, in collision checking, we are only interested in finding a separating plane between  $\mathcal{A}_1$  and  $\mathcal{A}_2$ , if it exists. This is equivalent to finding a separating plane between  $\mathcal{D}$  and  $\mathbf{0}_C$ . For *any* support direction  $\mathbf{d}$ , if we have:

$$\langle \mathbf{d}, \mathbf{s} \rangle > 0, \mathbf{s} \in S_{\mathcal{D}}(\mathbf{d}), \quad (19)$$

then the plane supported by the vector  $\mathbf{d}$  separates  $\mathcal{D}$  from  $\mathbf{0}_C$  [12]. This also means that, in the case where the two shapes intersect, collision checking has the same computational complexity as distance computation. As shown in Alg. 2, we add this separating plane condition before line 2 in Alg. 1.

**Computing support vector on meshes.** While the support vector of basic primitives (sphere, ellipsoid, box, etc.) presents closed-form solutions, this is not the case for meshes. In the

**Algorithm 2** Boolean collision checking: separating plane condition

*Insert after line 2 in Alg. 1:*

1: **If**  $\langle \mathbf{d}_k, \mathbf{s}_k \rangle > 0$ , **return False**  
**If after termination**  $d_{1,2} = 0$ , **return True**

case of convex meshes, an efficient approach for computing the support direction of meshes is the *hill-climbing* algorithm [17], which allows retrieving the supporting vertex or face of the meshes thanks to a simple neighbor-descent procedure. Yet, this procedure is sensitive to the initial-guess solution. By leveraging Nesterov and Polyak acceleration schemes introduced in Sec. III, which both tend to reduce the oscillations hindered by gradient-descent type algorithms, we show in Sec. IV that this helps the hill-climbing algorithm to perform less iterations in practice, leading to faster computation times.

**The Frank-Wolfe active-set.** As with many gradient-descent algorithms, the FW method tends to zig-zag towards the optimal solution [28], slowing down the convergence to the optimum. This behavior is undesired and amplified if the optimal solution  $\mathbf{x}^*$  lies close to the boundary of the constraint set  $\mathcal{D}$ . In collision detection, this corresponds to the case where the two shapes are not intersecting. This zig-zagging behavior is due to the way that Frank-Wolfe approaches the set of active constraints [28], also called *active-set* in the optimization literature [37]. In the FW setting, the active set at iteration  $k$ , denoted  $W_k = \{\mathbf{s}^0, \dots, \mathbf{s}^r\} \subset \mathcal{D}$ , is the set of vectors in  $\mathcal{D}$  used by the algorithm to maintain a convex combination of the iterate  $\mathbf{x}_k$ :

$$\mathbf{x}_k = \sum_{i=0}^r \lambda^i \mathbf{s}^i, \sum_{i=0}^r \lambda^i = 1 \text{ with } \mathbf{s}^i \in W_k \subset \mathcal{D} \text{ and } \lambda^i > 0. \quad (20)$$

In Alg. 3, we rewrite the FW algorithm with line search (Alg. 1) in order to highlight the notion of active set:

- A iteration  $k$ , the active-set is only composed of  $\mathbf{x}_k$ ,  $W_k = \{\mathbf{x}_k\}$ .
- The active-set  $W_k$  is then augmented by computing a support  $\mathbf{s}_k$  (line 2 in Alg. 3) to obtain  $\widetilde{W}_{k+1} = \{\mathbf{x}_k, \mathbf{s}_k\}$  (line 4 in Alg. 3).
- We then minimize function  $f$  on the convex-hull of  $\widetilde{W}_{k+1}$ ,  $\text{conv}(\widetilde{W}_{k+1})$ , which is simply the segment  $[\mathbf{x}_k, \mathbf{s}_k]$ . For the distance computation problem (4), this linesearch operation is equivalent to projecting  $\mathbf{0}_C$  onto the segment  $[\mathbf{x}_k, \mathbf{s}_k]$  (line 5 in Alg. 3).
- Finally, the active-set is updated  $W_{k+1} = \{\mathbf{x}_{k+1}\}$  (line 6 in Alg. 3).

In practice, discarding previously computed supports when updating the active set is inefficient and causes the zig-zagging phenomenon observed in the FW algorithm [28]. In the optimization literature, a rich and wide variety of variants of the FW algorithm have been introduced to efficiently cope with the active set in order to improve the convergence rate of the FW method [22], [24], [25], [38], [39]. However, these variants remain too generic and are not suited for the specific problem of collision detection. In the following, we

propose instead incorporating the active-set strategy used in GJK within the Frank-Wolfe setting.

**Connection between GJK and Frank-Wolfe.** In the case of collision detection, [11] developed an efficient strategy to handle the active set at a minimal cost. To represent the current estimate  $\mathbf{x}_k$  and the optimal solution  $\mathbf{x}^*$ , GJK exploits the concept of *simplexes* in  $\mathbb{R}^3$ . A simplex in  $\mathbb{R}^n$  corresponds to a set containing *at most*  $n + 1$  vectors of  $\mathbb{R}^n$  and the *rank*  $r$  of a simplex is the number of vectors it contains ( $0 < r \leq n + 1$ ). For 3-dimensional spaces, a simplex corresponds either to a point ( $r = 1$ ), a segment ( $r = 2$ ), a triangle ( $r = 3$ ) or a tetrahedron ( $r = 4$ ). Similarly to the simplex methods for Linear Programming [40], the Carathéodory theorem [41] motivates the use of simplexes. Let  $\mathcal{Y}$  be a set of  $N \geq n$  vectors in  $\mathbb{R}^n$ ,  $\mathcal{Y} = \{\mathbf{y}^i \in \mathbb{R}^n\}_{0 \leq i \leq N}$ . The Carathéodory theorem states that any vector  $\mathbf{x} \in \text{conv}(\mathcal{Y})$  can be expressed as the convex combination of at most  $n + 1$  vectors of  $\mathcal{Y}$ :

$$\mathbf{x} = \sum_{j=0}^r \lambda^j \mathbf{y}^j, \text{ with } \mathbf{y}^j \in \mathcal{Y}, \lambda^j > 0, \sum_{i=0}^r \lambda^i = 1. \quad (21)$$

Hence, any vector in  $\mathcal{D}$ , and particularly the optimal solution  $\mathbf{x}^* \in \mathcal{D} = \text{conv}(\mathcal{D})$  of the distance computation problem (4), can be identified as a convex combination of the vectors composing a simplex  $W$ . Relying on simplexes is attractive as there is no need to run any algorithm to compute the convex hull of a simplex as they are convex by construction. Frank-Wolfe algorithms may operate on more complex active sets, which might become hard to tackle from a computational point of view [26], [28]. In other words, the problem of finding the optimal solution  $\mathbf{x}^*$  can be reformulated as the problem of identifying the optimal simplex  $W^*$  on which  $\mathbf{x}^*$  can be decomposed into a convex combination. This is precisely the approach followed by GJK that we now detail as well as illustrate in Fig. 5.

At iteration  $k$  of GJK, the current iterate  $\mathbf{x}_k$  is a convex combination of the vectors composing the simplex  $W_k$  of rank  $r_k \leq n$ . This corresponds to Fig. 5a. To update  $\mathbf{x}_k$  and  $W_k$ , the following procedure is applied:

- After computing support vector  $\mathbf{s}_k$  (line 2 in Alg. 3, illustrated in Fig. 5b), we add  $\mathbf{s}_k$  to  $W_k$  to obtain  $\widetilde{W}_{k+1} = W_k \cup \{\mathbf{s}_k\}$  (line 4 in Alg. 3). The set  $\widetilde{W}_{k+1}$  is now a simplex of rank  $\widetilde{r}_{k+1} \leq n + 1$ , as shown in Fig. 5c.
- We then minimize function  $f(\mathbf{x}) = \|\mathbf{x}\|^2$  on  $\widetilde{W}_{k+1}$  to obtain  $\mathbf{x}_{k+1}$ , corresponding to projecting  $\mathbf{0}_C$  onto  $\widetilde{W}_{k+1}$ :  $\mathbf{x}_{k+1} = \text{proj}_{\text{conv}(\widetilde{W}_{k+1})}(\mathbf{0}_C)$ <sup>1</sup> (line 5 in Alg. 3). This projection is illustrated in figures 5c and 5d.
- We then have two cases, summarized in Alg 4:
  - 1) If  $\mathbf{x}_{k+1} = \mathbf{0}_C$ , the algorithm is stopped. Thus, we have  $\mathbf{x}^* = \mathbf{0}_C$  and  $d_{1,2} = 0$  in (4) (line 1 in Alg. 4).
  - 2) Otherwise, we construct  $W_{k+1}$  from  $\widetilde{W}_{k+1}$ . To do so, we retain only the minimal number of vectors in  $\widetilde{W}_{k+1}$  needed to express  $\mathbf{x}_{k+1}$  as a convex combination (line 2 in Alg. 4). Indeed, as  $\mathbf{0}_C \notin \widetilde{W}_{k+1}$ ,

<sup>1</sup>The efficient projection onto simplexes in  $\mathbb{R}^3$ , named the *distance sub-algorithm* by [11], is thoroughly covered in [7], [12] and its robustness is improved in [18].

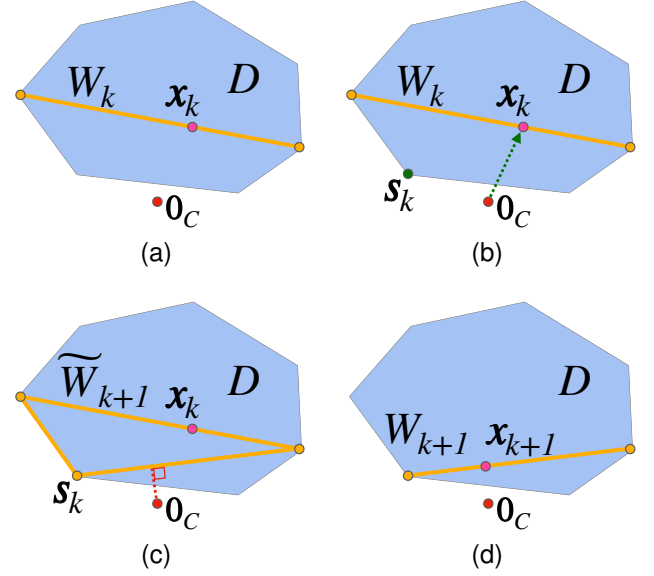


Fig. 5. Illustration of the GJK simplex strategy in 2D: (a) beginning of the  $k^{\text{th}}$  iteration, (b) support point computation, (c) simplex augmentation, (d) simplex update.

**Algorithm 3** Frank-Wolfe algorithm with line-search (see Alg. 1) rewritten with active-sets and applied to the distance computation problem (4)

**Let**  $\mathbf{x}_0 \in \mathcal{D}$ ,  $W_0 = \{\mathbf{x}_0\}$ ,  $\epsilon > 0$

**For**  $k=0, 1, \dots$  **do**

- 1:  $\mathbf{d}_k = \mathbf{x}_k$  ▷ Direction of support
- 2:  $\mathbf{s}_k \in S_{\mathcal{D}}(\mathbf{d}_k)$  ▷ Support (8)
- 3: **If**  $g_{FW}(\mathbf{x}_k) \leq \epsilon$ , **return**  $f(\mathbf{x}_k)$  ▷ Duality gap (16)
- 4:  $\widetilde{W}_{k+1} = W_k \cup \{\mathbf{s}_k\}$  ▷ Augment active-set
- 5:  $\mathbf{x}_{k+1} = \text{proj}_{\text{conv}(\widetilde{W}_{k+1})}(\mathbf{0}_C)$  ▷ Project  $\mathbf{0}_C$  on  $\text{conv}(\widetilde{W}_{k+1})$
- 6:  $W_{k+1} = \{\mathbf{x}_{k+1}\}$  ▷ Update active-set

the projection  $\mathbf{x}_{k+1}$  of  $\mathbf{0}_C$  on  $\widetilde{W}_{k+1}$  necessarily lies on a face of  $\widetilde{W}_{k+1}$ , and can be expressed as a convex combination of the vectors composing this face. This ensures that  $W_{k+1}$  is necessarily of rank  $r_{k+1} < \widetilde{r}_{k+1} \leq n + 1$ . As an example, Fig. 5d shows the result of the simplex update obtained in Fig. 5c.

Through this discussion, it is clear that GJK is a particular case of Frank-Wolfe. More specifically, it is a sub-case of the fully-corrective Frank-Wolfe algorithm analyzed by [28]. The strategy used by GJK to handle the active set has proved to be very efficient in practice and renders the GJK algorithm state-of-the-art for collision detection. In the next section, we propose to leverage the formulation of collision detection as a Frank-Wolfe sub-case to accelerate its convergence following the well-established Polyak and Nesterov acceleration paradigm [29].

### III. ACCELERATING COLLISION DETECTION

Gradient descent (GD) is the backbone of many convex optimization methods and relies solely on the gradient of

**Algorithm 4** Fully-corrective FW using simplexes, applied to the distance computation problem (4). This algorithm is identical to GJK [11]

In Alg. 3, let  $W_0 = \emptyset$  and replace line 6 by:

- 1: **If**  $\mathbf{x}_{k+1} = \mathbf{0}_C$ , **return** 0  
*If the algorithm has not terminated, update  $\widetilde{W}_{k+1}$  to retain only the smallest number of vectors needed to express  $\mathbf{x}_{k+1}$ .*
- 2:  $W_{k+1} = \{\mathbf{s}^1, \dots, \mathbf{s}^r\}$  where  $\mathbf{s}^1, \dots, \mathbf{s}^r$  are the smallest number of vectors in  $\widetilde{W}_{k+1}$  such that  $\mathbf{x}_{k+1}$  is a convex combination of  $\mathbf{s}^1, \dots, \mathbf{s}^r$ .

the objective function. Second-order methods [37], such as Newton methods, have faster convergence rates than GD at the price of requiring the computation and the inversion of Hessian quantities. Momentum methods have thus been introduced in the optimization literature to provide gradient-based methods with improved convergence rates without requiring costly Hessian evaluation. In this section, we use recent work linking the *Polyak* and *Nesterov* accelerations of GD to the FW algorithm [30], [32] to globally accelerate collision detection. These global accelerations of collision detection are experimentally evaluated in Sec. IV on several benchmarks.

#### A. Background on acceleration methods in convex optimization

**Polyak acceleration for unconstrained optimization.** We initially consider the following *unconstrained* minimization problem:

$$f(\mathbf{x}^*) = \min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}), \quad (22)$$

where  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  is a convex and differentiable function. The vanilla gradient-descent algorithm follows the slope of  $f$  given by its gradient  $\nabla f$ . The following scheme is applied iteratively until a given convergence criterion is met (e.g.,  $\|\nabla f(\mathbf{x}_k)\| < \epsilon$ , with  $\epsilon$  being the desired precision):

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \nabla f(\mathbf{x}_k), \quad (23)$$

where  $\mathbf{x}_k \in \mathbb{R}^n$  is the current iterate and  $\alpha_k \in \mathbb{R}$  is the gradient step. This standard setting leads to a simple implementation with linear convergence rate ( $O(1/k)$ ).

To go beyond this linear convergence regime, acceleration techniques have been devised in the optimization community to provide quadratic convergence rate ( $O(1/k^2)$ ) or more [23], by relying on relatively cheap gradient evaluations. Among these gradient-descent acceleration techniques, the Polyak (or Heavy-Ball) [42] and Nesterov acceleration [29] are two of the better-studied and most popular in practice [23]. These techniques are based on accumulating previously computed gradients in a *momentum* term  $\mathbf{d}_k$  and using this momentum  $\mathbf{d}_k$  to update the current iterate  $\mathbf{x}_k$ . The Polyak update scheme for unconstrained gradient descent is illustrated in Fig. 6a and goes as follows:

$$\mathbf{d}_k = \delta_k \mathbf{d}_{k-1} + \alpha_k \nabla f(\mathbf{x}_k) \quad (24a)$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{d}_k, \quad (24b)$$

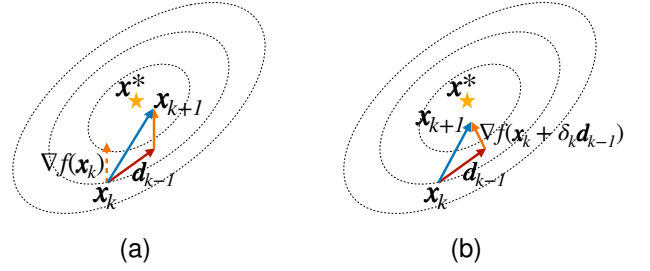


Fig. 6. (a) Polyak and (b) Nesterov acceleration schemes for unconstrained gradient descent. The gradient descent algorithm aims at finding the optimum  $\mathbf{x}^*$  by following the slope given by the gradient of function  $f$ ,  $\nabla f$ . The vector  $\mathbf{d}_{k-1}$  is the momentum accumulated over the optimization trajectory. The two schemes differ in where the gradient is computed at iteration  $k$ ; the Nesterov scheme introduces an intermediary point  $\mathbf{y}_k = \mathbf{x}_k + \delta_k \mathbf{d}_{k-1}$  to compute the gradient.

where schemes  $\delta_k \in \mathbb{R}$  is the momentum parameter. The role of momentum  $\mathbf{d}_k$  is to smooth the trajectory of iterates converging towards the optimum by geometrically averaging previously computed gradients. The  $\delta_k$  momentum parameter is selected to prevent damping or overshooting of the iterate trajectory when going towards the optimal solution  $\mathbf{x}^*$ .

**Nesterov acceleration for unconstrained optimization.** The Nesterov update scheme is the second most well-known method for accelerating unconstrained gradient descent and it is only a slight modification on top of the Polyak scheme. Contrary to the Polyak case, in the Nesterov acceleration scheme the current iterate  $\mathbf{x}_k$  is extrapolated using the momentum term  $\mathbf{d}_k$  to compute the intermediate vector  $\mathbf{y}_k = \mathbf{x}_k + \delta_k \mathbf{d}_k$ . The gradient is then computed at the vector  $\mathbf{y}_k$ . The Nesterov update scheme for unconstrained gradient descent is illustrated in Fig. 6b and goes as:

$$\mathbf{y}_k = \mathbf{x}_k + \delta_k \mathbf{d}_{k-1} \quad (25a)$$

$$\mathbf{d}_k = \delta_k \mathbf{d}_{k-1} + \alpha_k \nabla f(\mathbf{y}_k) \quad (25b)$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{d}_k \quad (25c)$$

where  $\delta_k$  is the momentum parameter as in the Polyak scheme, and  $\mathbf{y}_k \in \mathbb{R}^n$  is an intermediate quantity. Computing the term  $\mathbf{y}_k$  leads to an anticipatory behavior in similar spirit to extra-gradient methods [23].

**Accelerating the Frank-Wolfe algorithm with Polyak and Nesterov.** Recent works of [30], [32] have proposed to adapt the Polyak and Nesterov accelerations to the FW setting. We propose to leverage and adapt this FW acceleration scheme to the context of collision detection, by notably extending the FW formulation of collision detection previously developed in Sec. II.

In the original FW algorithm, the support vector at iteration  $k$ ,  $\mathbf{s}_k$ , is computed in the direction of the gradient  $\nabla f(\mathbf{x}_k)$  (line 1 in Alg. 1). In the Polyak acceleration of FW proposed by [32], the direction of support for computing  $\mathbf{s}_k$  is instead defined by:

$$\mathbf{d}_k = \delta_k \mathbf{d}_{k-1} + (1 - \delta_k) \nabla f(\mathbf{x}_k) \quad (26a)$$

$$\mathbf{s}_k = S_D(\mathbf{d}_k), \quad (26b)$$



**Algorithm 5** Polyak-accelerated and Nesterov-accelerated Frank-Wolfe [30], [32]

In Alg. 1 and Alg. 3, let  $\mathbf{d}_{-1} = \mathbf{s}_{-1} = \mathbf{x}_0$ ,  $\delta_k = \frac{k+1}{k+3}$  and replace line 1 by:

$$\begin{aligned} 1: \mathbf{y}_k &= \begin{cases} \mathbf{x}_k & \text{Polyak} \\ \delta_k \mathbf{x}_k + (1 - \delta_k) \mathbf{s}_{k-1} & \text{Nesterov} \end{cases} \\ 2: \mathbf{d}_k &= \delta_k \mathbf{d}_{k-1} + (1 - \delta_k) \nabla f(\mathbf{y}_k) \end{aligned}$$

where  $\delta_k = \frac{k+1}{k+3} \in [0, 1]$  is the momentum parameter and  $S_{\mathcal{D}}$  is the support function as defined in (8). In the Nesterov acceleration of FW proposed by [30], the direction of support for computing  $\mathbf{s}_k$  is slightly different from the Polyak scheme (26) as it introduces  $\mathbf{y}_k$ , an intermediary vector as in the GD Nesterov scheme (25) in order to evaluate the gradient  $\nabla f(\mathbf{y}_k)$ :

$$\mathbf{y}_k = \delta_k \mathbf{x}_k + (1 - \delta_k) \mathbf{s}_{k-1} \quad (27a)$$

$$\mathbf{d}_k = \delta_k \mathbf{d}_{k-1} + (1 - \delta_k) \nabla f(\mathbf{y}_k) \quad (27b)$$

$$\mathbf{s}_k = S_{\mathcal{D}}(\mathbf{d}_k), \quad (27c)$$

where  $\mathbf{s}_{k-1}$  is the support vector computed at the previous iteration. To ensure  $\mathbf{y}_k$  stays in  $\mathcal{D}$ , it is a convex combination of  $\mathbf{x}_k$  and  $\mathbf{s}_{k-1}$ , both vectors of  $\mathcal{D}$ . The direction of support is then obtained by taking a convex combination of the previous support direction  $\mathbf{d}_{k-1}$  and the gradient  $\nabla f(\mathbf{y}_k)$ . Both the Polyak and Nesterov accelerations of Frank-Wolfe are summed up in Alg. 5.

The works [30], [32] have experimentally shown that these accelerations strategies lead to a better convergence rate of the FW algorithm when compared to the original FW algorithm. In the following, we explain how to adapt the Polyak and Nesterov accelerations of FW to collision detection.

**B. Acceleration of collision detection and distance computation**

**Adapting Nesterov and Polyak fully-corrective Frank-Wolfe to distance computation.** Preserving GJK's simplex strategy is crucial for collision detection as it greatly speeds up the vanilla FW algorithm. Therefore, we adapt (26) and (27) accordingly as:

$$\mathbf{y}_k = \begin{cases} \mathbf{x}_k & \text{if Polyak} \\ \delta_k \mathbf{x}_k + (1 - \delta_k) \mathbf{s}_{k-1} & \text{if Nesterov} \end{cases} \quad (28a)$$

$$\mathbf{d}_k = \delta_k \mathbf{d}_{k-1} + (1 - \delta_k) \nabla f(\mathbf{y}_k) \quad (28b)$$

$$\mathbf{s}_k = S_{\mathcal{D}}(\mathbf{d}_k), \quad (28c)$$

$$\widetilde{W}_{k+1} = W_k \cup \{\mathbf{s}_k\}, \quad (28d)$$

$$\mathbf{x}_{k+1} = \text{proj}_{\text{conv}(\widetilde{W}_{k+1})}(\mathbf{0}_C). \quad (28e)$$

These steps are also summarized in Alg. 6. The update of simplex  $W_{k+1}$  from  $\widetilde{W}_{k+1}$  is then identical to the one described in Alg. 4. The original duality gap defined in Sec. II (Eq. 16) can no longer be used as a convergence criterion. Indeed, the following inequality:

$$\|\mathbf{x}_k - \mathbf{x}^*\|^2 \leq g_{\text{FW}}(\mathbf{x}_k) = 2\langle \mathbf{x}_k, \mathbf{x}_k - \mathbf{s}_k \rangle, \mathbf{s}_k \in S_{\mathcal{D}}(\mathbf{x}_k),$$

**Algorithm 6** Polyak and Nesterov-accelerated GJK

Let  $\mathbf{x}_0 \in \mathcal{D}$ ,  $W_0 = \emptyset$ ,  $\mathbf{d}_{-1} = \mathbf{s}_{-1} = \mathbf{x}_0$ ,  $\epsilon > 0$

**For**  $k=0, 1, \dots$  **do**

$$1: \delta_k = \frac{k+1}{k+3} \quad \triangleright \text{Momentum parameter value}$$

$$2: \mathbf{y}_k = \begin{cases} \mathbf{x}_k & \text{Polyak} \\ \delta_k \mathbf{x}_k + (1 - \delta_k) \mathbf{s}_{k-1} & \text{Nesterov} \end{cases} \quad \triangleright \text{Intermediary point (28a)}$$

$$3: \mathbf{d}_k = \delta_k \mathbf{d}_{k-1} + (1 - \delta_k) \nabla f(\mathbf{y}_k) \quad \triangleright \text{Support dir. (28b)}$$

$$4: \mathbf{s}_k \in S_{\mathcal{D}}(\mathbf{d}_k) \quad \triangleright \text{Support (8)}$$

$$5: \text{if } g(\mathbf{x}_k) \leq \epsilon \text{ then} \quad \triangleright \text{Fixed-point condition (32)}$$

$$6: \quad \text{If } \mathbf{d}_k = \mathbf{x}_k, \text{ return } f(\mathbf{x}_k) \quad \triangleright \text{Algorithm terminates}$$

$$7: \quad \mathbf{s}_k \in S_{\mathcal{D}}(\nabla f(\mathbf{x}_k)) \quad \triangleright \text{Compute } \mathbf{s}_k \text{ in dir. } \nabla f(\mathbf{x}_k)$$

**Replace line 3 by:  $\mathbf{d}_k = \mathbf{x}_k$  until termination.**

$$8: \widetilde{W}_{k+1} = W_k \cup \{\mathbf{s}_k\} \quad \triangleright \text{Augment active-set}$$

$$9: \mathbf{x}_{k+1} = \text{proj}_{\text{conv}(\widetilde{W}_{k+1})}(\mathbf{0}_C) \triangleright \text{Project } \mathbf{0}_C \text{ on conv}(\widetilde{W}_{k+1})$$

$$10: \text{If } \mathbf{x}_{k+1} = \mathbf{0}_C, \text{ return } 0$$

$$11: W_{k+1} = \{\mathbf{s}^1, \dots, \mathbf{s}^r\} \text{ where } \mathbf{s}^1, \dots, \mathbf{s}^r \text{ are the smallest number of vectors in } \widetilde{W}_{k+1} \text{ such that } \mathbf{x}_{k+1} \text{ is a convex combination of } \mathbf{s}^1, \dots, \mathbf{s}^r.$$

is no longer valid because the support vector  $\mathbf{s}_k$  is no longer computed in the direction of the gradient  $\nabla f(\mathbf{x}_k) = 2\mathbf{x}_k$ . Next we will show that the original stopping criterion devised in Sec. II cannot be used and we need to derive a new one.

**Stopping criterion.** As the number of iteration  $k$  increases,  $\delta_k \rightarrow 1$  in (28). Therefore,  $\mathbf{d}_k$  tends to be equal to  $\mathbf{d}_{k-1}$  (28b) and thus  $\mathbf{s}_k = \mathbf{s}_{k-1}$  (28c). As a consequence, augmenting  $W_k$  with  $\mathbf{s}_k$  to construct  $\widetilde{W}_{k+1}$  (see (28d)) and then projecting  $\mathbf{0}_C$  onto  $\widetilde{W}_{k+1}$  (28e) will not result in any progress. Therefore,  $\mathbf{x}_{k+1} = \mathbf{x}_k$ : the algorithm reaches a fixed point and is stuck on constant support direction  $\mathbf{d}$ .

In order to cope with this issue, we use the following strategy. Suppose  $\mathbf{x}_k \neq \mathbf{0}_C$ . Since  $\mathbf{x}_k = \text{proj}_{\text{conv}(W_k)}(\mathbf{0}_C)$  we have:

$$\forall \mathbf{s}^i \in W_k, \langle \mathbf{x}_k, \mathbf{x}_k - \mathbf{s}^i \rangle = 0. \quad (29)$$

After computing  $\mathbf{s}_k \in S_{\mathcal{D}}(\mathbf{d}_k)$ , if we have:

$$\langle \mathbf{x}_k, \mathbf{x}_k - \mathbf{s}_k \rangle \neq 0, \quad (30)$$

then  $\mathbf{s}_k$  is not a linear combination of vectors in  $W_k$ . Therefore, augmenting  $W_k$  with  $\mathbf{s}_k$  to obtain  $\widetilde{W}_{k+1}$  and projecting  $\mathbf{0}_C$  onto  $\text{conv}(\widetilde{W}_{k+1})$  to obtain  $\mathbf{x}_{k+1}$  will result in the algorithm progressing toward the optimum  $\mathbf{x}^*$ . Suppose on the contrary that:

$$\langle \mathbf{x}_k, \mathbf{x}_k - \mathbf{s}_k \rangle = 0, \quad (31)$$

then  $\mathbf{s}_k$  is a linear combination of vectors in  $W_k$ . Adding  $\mathbf{s}_k$  to  $W_k$  will thus not result in any progress towards the optimum. As a consequence, Eq. (31) encompasses two cases:

- If the support direction  $\mathbf{d}_k$  is aligned with  $\nabla f(\mathbf{x}_k)$ , Eq. (31), corresponding to  $g_{\text{FW}}(\mathbf{x}_k) = 0$ , matches the termination criterion of the distance computation problem and therefore we have reached the optimum i.e  $\mathbf{x}_k = \mathbf{x}^*$ .
- Otherwise, if  $\mathbf{d}_k$  is not aligned with  $\nabla f(\mathbf{x}_k)$ , the algorithm cannot stop as a null duality gap is not met.



The algorithm thus enters a cycle where it iterates until Eq. (31) does not hold. To cope with this undesired behavior we simply stop the Polyak or the Nesterov acceleration as soon as Eq. (31) is met and switch back to the non-accelerated version Alg. 4.

We thus define the function  $g$  such that for any  $s_k \in \mathcal{D}$ :

$$g(x_k) = 2\langle x_k, x_k - s_k \rangle, \quad (32)$$

$g$  is used in Alg. 6 as an optimality criterion ( $g \leq \epsilon$ ) either for stopping the Polyak and Nesterov accelerations in order to continue with the vanilla GJK, or as stopping criteria qualifying an optimal solution, in which case  $g = g_{FW}$  and (18) holds. The entire algorithm is summarized in Alg. 6.

**Nesterov acceleration for non-strictly convex shapes.** Let us explain the effect of the Nesterov acceleration on the support direction update (28b) and distinguish between strictly convex and non-strictly convex  $\mathcal{D}$ :

- If  $\mathcal{D}$  is strictly convex, any vector  $s$  belonging to the surface of  $\mathcal{D}$  has a unique corresponding direction  $d$  such that  $s = S_{\mathcal{D}}(d)$ . Here, we stress the fact that the support function  $S_{\mathcal{D}}$  returns only *one* vector. Consequently, we have  $d_k \neq d_{k-1}$  and therefore  $s_k \neq s_{k-1}$ . The fixed point condition (31) is thus not met unless  $\delta_k = 1$  and Nesterov acceleration continues to be applied in Alg. 6. In practice, the algorithm runs until  $\delta_k$  gets close to 1 or  $x_k$  gets close to  $0_C$ . The condition (31) is then satisfied as the algorithm starts to cycle. The Nesterov acceleration is thus removed and the algorithm runs until the convergence criteria is satisfied, guaranteed by the Frank-Wolfe algorithm.
- Otherwise, if  $\mathcal{D}$  is non-strictly convex, multiple support directions  $\{d^1, \dots, d^m, \dots\}$  can yield the same support vector  $s \in S_{\mathcal{D}}(d^1) = \dots = S_{\mathcal{D}}(d^m) = \dots$  etc. Consequently, it is possible to have  $d_{k-1} \neq d_k$  and  $s_k = s_{k-1}$ . Therefore, even though  $\delta_k$  is not close to 1, the fixed point condition (31) can be verified. The Nesterov acceleration is stopped, possibly prematurely.

The latter case is especially problematic when shapes  $\mathcal{A}_1$  and  $\mathcal{A}_2$  are in close-proximity, which is ultimately the type of collision problems commonly encountered in simulation or motion planning with contacts. In (28b), this is due to the norm of  $\nabla f(y_k)$  being predominant over the norm of  $d_{k-1}$  as  $k$  increases,  $\|d_{k-1}\| \ll \|\nabla f(y_k)\|$ . As a consequence, the Nesterov acceleration enters a cycle: the support direction  $d_k$  does not change enough compared to  $d_{k-1}$ , hence the support point  $s_k$  is identical to  $s_{k-1}$  and therefore the intermediary point  $y_k$  does not change and the cycle repeats. As a consequence, the criterion (31) is met and the Nesterov acceleration is stopped to escape the cycle, possibly prematurely. To prevent this phenomenon observed on non-strictly convex  $\mathcal{D}$ , we propose to replace (28b) by a simple heuristic which normalizes the gradient and momentum directions as follows:

$$d_k = \delta_k \frac{d_{k-1}}{\|d_{k-1}\|} + (1 - \delta_k) \frac{\nabla f(y_k)}{\|\nabla f(y_k)\|}, \quad (33)$$

summarized in Alg. 7. In Sec. IV, we experimentally prove this heuristic to significantly reduce the number of steps for

**Algorithm 7** Normalize direction for non-strictly convex shapes in Nesterov-accelerated GJK

Replace line 3 in Alg. 6 by:

$$1: d_k = \delta_k \frac{d_{k-1}}{\|d_{k-1}\|} + (1 - \delta_k) \frac{\nabla f(y_k)}{\|\nabla f(y_k)\|}$$

distance computations for non-strictly convex shapes. We also show that this heuristic does not need to be applied to the Polyak acceleration, as, contrary to the Nesterov acceleration, the Polyak acceleration does not compute an intermediary point  $y_k$ .

#### IV. EXPERIMENTS

In this section, we study the performance of both Polyak and Nesterov-accelerated GJK (Alg. 6) against the vanilla GJK (Alg. 4) algorithm.

In sections IV-A and IV-B, we benchmark our proposed Polyak-accelerated and Nesterov-accelerated GJK algorithms against the vanilla GJK algorithm on these two distinct benchmarks. The benchmark made of strictly-convex shapes represents a worst-case scenario regarding the number of iterations for all variants of GJK. The benchmark of non-strictly convex shapes represents shapes typically used in robotic or computer graphics applications. Then, in Sec. IV-C, we benchmark GJK and our proposed accelerated gradients against the state-of-the-art quadratic programming solver ProxQP [43]. We show that GJK and our proposed accelerated variants vastly outperform generic quadratic programming (QP) solvers, making these QP solvers prohibitive for collision detection. In Sec. IV-D, we benchmark vanilla, Polyak-accelerated, and Nesterov-accelerated GJK on a dataset of trajectories obtained using a physics simulator. We show that, similarly to vanilla GJK, our accelerated GJK algorithms can benefit from being warm-started with previous simulation time steps, outperforming the vanilla GJK in physics simulation scenarios. Finally, in Sec. IV-E, we empirically show that the simplex strategy used by GJK and our methods (discussed in Sec. II) is crucial for efficient collision detection. We show that GJK and our methods significantly outperform the original FW algorithm and the recent NESMINO [20] algorithm. Although it differs from FW algorithms, we include the NESMINO algorithm in this analysis as its projected-gradient descent procedure is accelerated using the classic Nesterov acceleration scheme [29].

**Implementation.** We leverage the HPP-FCL C++ library [44], [45], an extension of the original FCL library [44]. Unlike FCL, HPP-FCL provides its own implementation of GJK, which we have extended by implementing the Polyak and Nesterov-accelerated GJK algorithms (Alg. 6). The open-source code of the HPP-FCL library is publicly available at <https://github.com/humanoid-path-planner/hpp-fcl> under the BSD-3 license. The benchmark code is publicly available at <https://github.com/lmontaut/colbench> under the GNU AGP License.

**Shapes datasets.** To distinguish between pairs of strictly convex and non-strictly convex shapes, we build a first benchmark

only composed of pairs of ellipsoids (strictly convex shapes) and a second benchmark using pairs of standard meshes (represented by their convex hulls) which are taken from the commonly-used YCB dataset [34].

**Ellipsoids.** In the ellipsoids benchmark, the ellipsoids are randomly generated by sampling positive-definite matrices. In total, we generate 1000 random pairs of ellipsoids. Given a pair of ellipsoids, we randomly sample relative poses between the shapes, using a uniform distribution for the relative rotation between the shapes. Regarding the translation part of the random poses, the directions are selected at random, but the norms are chosen so that we control the distance  $\text{dist}(\mathcal{A}_1, \mathcal{A}_2)$  between the objects. This enables us to measure the influence of the separation distance on the performance of the studied algorithms. The values used for  $\text{dist}(\mathcal{A}_1, \mathcal{A}_2)$  range from  $-0.1$  m to  $1$  m. Negative values correspond to scenarios where the shapes intersect, with  $\text{dist}(\mathcal{A}_1, \mathcal{A}_2)$  corresponding to the separating vector's norm. The separating vector is the vector of the smallest norm needed to translate one of the two shapes such that the two shapes do not intersect. Therefore, for each pair of ellipsoids, 100 random relative poses are sampled, so the shapes do not intersect. We translate the shapes along the axis given by their closest points for each relative pose to study the impact of  $\text{dist}(\mathcal{A}_1, \mathcal{A}_2)$ . We then set  $\text{dist}(\mathcal{A}_1, \mathcal{A}_2)$  to fixed values between  $-0.1$  m to  $1$  m.

**YCB meshes.** On the other hand, the YCB mesh dataset contains about 60 shapes commonly used for robotic manipulation tasks (kitchen appliances, tools, toys, etc.). Each object has three different resolution levels corresponding to the number of points representing the mesh. For each object, we take the lowest resolution, i.e., the *google-16k* versions of the meshes, as it is resolute enough for any robotic task. As GJK-like algorithms work on convex shapes, we pre-compute the convex hulls of each object in the YCB dataset. This procedure needs only to be done once; if more precision is required for a certain robotic task, it is common to decompose a non-convex object into a set of convex sub-objects. For the sake of simplicity, we will not decompose YCB objects into sub-objects, as the results presented in this section would essentially be the same. In the rest of this section, when we mention a shape, we refer to its convex hull unless explicitly stated otherwise. The resulting meshes extracted from the YCB dataset contain between 100 and 8000 vertices. About 50% of meshes contain between 100 and 1000 vertices. As in the ellipsoids benchmark, 100 random relative poses are sampled for each pair such that the shapes do not intersect and then set  $\text{dist}(\mathcal{A}_1, \mathcal{A}_2)$  to fixed values between  $-0.1$  m and  $1$  m.

In both benchmarks (ellipsoids and YCB meshes), the characteristic sizes of the shapes range from a few centimeters up to a meter. Finally, for the distance computation problem, we select a convergence tolerance of  $\epsilon = 10^{-8}$ .

**Initialization strategy.** Apart from Sec. IV-D, the GJK algorithm and our proposed accelerated GJK algorithms are initialized with the centers of the shapes' bounding boxes. The bounding box of a shape fully encapsulates it, as is shown in Fig. 1. Hence, if we denote  $c_1$  and  $c_2$  the geometric centers of the bounding boxes of  $\mathcal{A}_1$  and  $\mathcal{A}_2$ , then we initialize vanilla

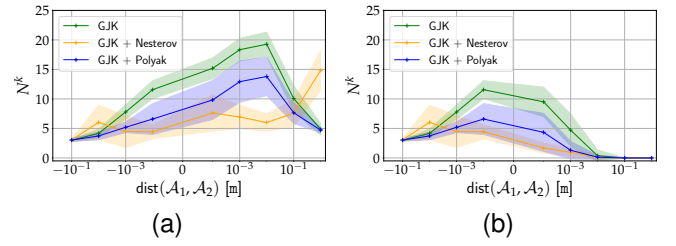


Fig. 7. Comparison of Polyak-accelerated GJK, Nesterov-accelerated GJK, and vanilla GJK on the ellipsoid benchmark for (a) distance computation and (b) Boolean collision checking. The graphs show the number of iterations  $N^k$  (y-axis) vs. the signed distance between the two shapes (x-axis). The curve shows the mean value over 100,000 random trials. The shaded region corresponds to the standard deviation. The Nesterov-accelerated GJK algorithm requires fewer iterations when the shapes are in close proximity. The Polyak-accelerated GJK algorithm is more robust when shapes are strongly overlapping or distant.

GJK, Polyak-accelerated GJK and Nesterov-accelerated GJK to  $x_0 = c_1 - c_2$ .

**Metrics.** To measure the performances of Polyak-accelerated GJK, Nesterov-accelerated GJK, and the vanilla GJK algorithms, we measure the number of iterations  $N^k$  to solve a given collision problem. For the mesh benchmark, we also measure the execution time  $T^\mu$  of both methods. We solve each generated collision problem 100 times to cope with CPU throttling. We then report the average of the 90% lowest computation times. All the benchmarks in this paper were run on an Apple M1 Max CPU.

#### A. Worst case scenario: strictly convex shapes - ellipsoids

We first focus on the ellipsoid benchmark to get a statistical understanding of the performance of Polyak and Nesterov-accelerated GJK against vanilla GJK. In the following, we explain why these shapes are interesting to study experimentally, as they represent the worst-case scenario that GJK-like algorithms can be confronted with. First, as previously explained, GJK-like algorithms look for the optimal active set of the solution  $x^*$ . Otherwise said, GJK-like methods find a set of support points  $W^* = \{s^1, s^2, \dots\}$  such that the optimal solution  $x^*$  is a convex combination of the points of  $W^*$ , where  $s^i$  are support points computed while running GJK or our proposed accelerations. Then, contrary to non strictly-convex shapes, strictly-convex shapes have an infinite amount of support points. As explained at the end of Sec. III, each normalized support direction  $d$  corresponds to a unique support point  $S_D(d)$ . Therefore, it is fundamentally harder to identify the optimal active set when considering strictly-convex shapes, as there is an infinite amount of potential support points to consider. In contrast, there is a finite amount of support points to consider when using non strictly-convex shapes.

In Fig. 7, we show the performance of the vanilla, the Polyak-accelerated, and the Nesterov-accelerated GJK algorithms on the ellipsoids benchmark. Fig. 7a and Fig. 7b show the mean and standard deviation of the number of iterations  $N^k$  of each method for the distance computation and the Boolean collision checking problems, respectively. When

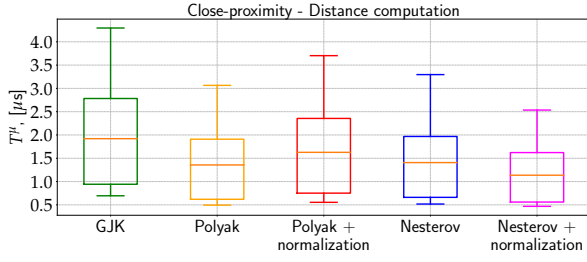


Fig. 8. Impact of support direction normalization in Polyak and Nesterov-accelerated GJK on the YCB benchmark. The graph shows the computation time  $T^\mu$  (lower is better) for vanilla GJK, Polyak-accelerated GJK, and Nesterov-accelerated GJK with and without support direction normalization. Here, the two shapes are in close-proximity:  $0 \text{ m} < \text{dist}(\mathcal{A}_1, \mathcal{A}_2) \leq 0.1 \text{ m}$ . Normalizing the support direction benefits Nesterov-accelerated GJK, reducing the overall number of iterations compared to GJK and non-normalized Nesterov-accelerated GJK.

the shapes are shallowly intersecting, Polyak and Nesterov-accelerated GJK converge with the same or even fewer number of iterations than vanilla GJK. However, the shallower the penetration, the more Polyak and Nesterov accelerate over vanilla GJK, with Nesterov providing the most acceleration. The irregularity in standard deviation at  $-0.01 \text{ m}$  is a critical zone for the Nesterov momentum where the variance increases. When shapes are in close proximity, the Nesterov acceleration of GJK significantly reduces the number of iterations compared to vanilla GJK and Polyak-accelerated GJK. Finally, when shapes are distant,  $1 \text{ m} \leq \text{dist}(\mathcal{A}_1, \mathcal{A}_2)$ , the Nesterov acceleration is detrimental to convergence on the distance computation problem while Polyak-accelerated GJK remains competitive against vanilla GJK. This indicates that the Polyak acceleration is generally more robust than the Nesterov acceleration. However, it offers less acceleration over vanilla GJK when the shapes are in close-proximity or shallowly overlapping. A similar pattern of speed-ups of Polyak and Nesterov-accelerated GJK over vanilla GJK is shown for the collision detection problem in Fig. 7b.

### B. Non-strictly convex shapes: meshes

**Effect of support direction normalization.** For meshes, the importance of normalizing the support direction (see Eq. (33)) in the Nesterov-accelerated GJK is highlighted in Fig. 8. For both the distance computation and Boolean collision checking problems, the normalization heuristic prevents the Nesterov acceleration from reaching a fixed point too early, and consequently, it reduces the overall amount of iterations needed to converge. This is, however, not the case for the Polyak-accelerated GJK algorithm, which does not benefit from support normalization. As explained at the end of Sec. III, the Polyak acceleration does not compute an intermediary point, unlike the Nesterov acceleration scheme. In the following, we thus focus only on Polyak-accelerated GJK *without* support normalization and Nesterov-accelerated GJK *with* support normalization. We compare the performances of these two algorithms against the vanilla GJK algorithm.

**Statistical validation over the YCB dataset.** In Fig. 9 and Fig. 10, we report the number of iterations  $N^k$  and execution

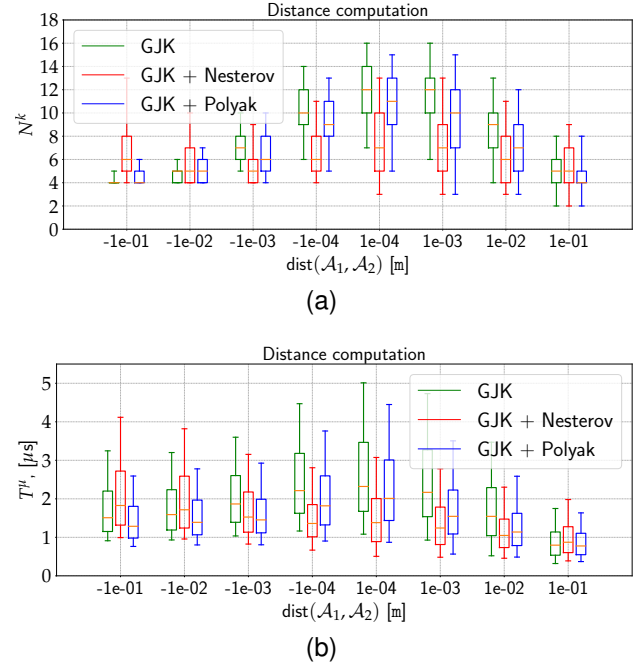


Fig. 9. Distance computation on the YCB benchmark. The graphs show the number of iterations  $N^k$  (a) and the execution time  $T^\mu$  (b) for Polyak-accelerated GJK, Nesterov-accelerated GJK (with normalization) and vanilla GJK for a range of distances (x-axis) between the shapes. For both metrics, lower is better.

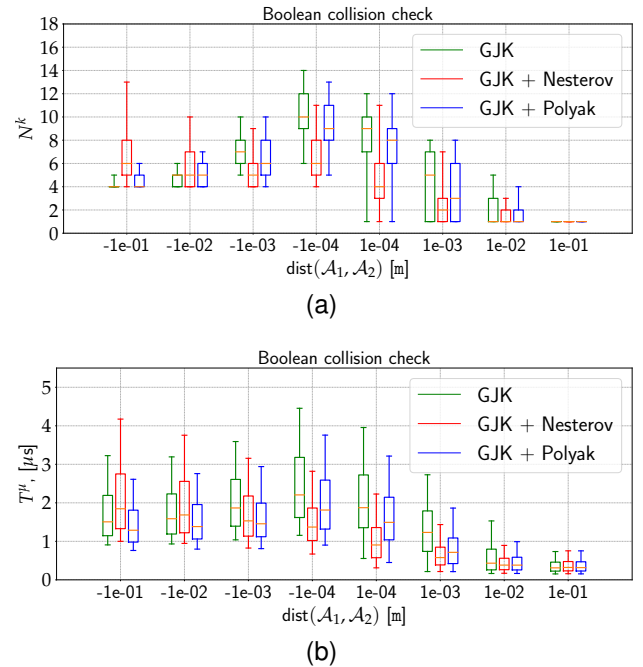
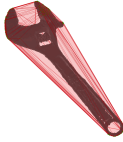

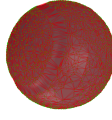





Fig. 10. Boolean collision check on the YCB benchmark. The graphs show the number of iterations  $N^k$  (a) and the execution time  $T^\mu$  (b) for the Polyak-accelerated GJK, Nesterov-accelerated GJK (with normalization), and vanilla GJK algorithms for a range of distances (x-axis) between the shapes. For both metrics, lower is better.

TABLE I  
COMPUTATION TIMES ( $\mu s$ ) FOR DISTANCE COMPUTATION ( $T_D^\mu$ ) AND BOOLEAN COLLISION CHECKING ( $T_C^\mu$ ) ON THE YCB BENCHMARK FOR CLOSE-PROXIMITY OR SHALLOWLY INTERSECTING SHAPES.  $N$  DENOTES THE NUMBER OF VERTICES FOR EACH MESH.

$N = 240$				$N = 1811$				$N = 3585$			
											
		GJK	Polyak	Nesterov							
	$T_D^\mu$	$1.1 \pm 0.3$	<b><math>0.9 \pm 0.3</math></b>	<b><math>0.9 \pm 0.3</math></b>	$1.9 \pm 0.5$	$1.5 \pm 0.5$	<b><math>1.4 \pm 0.5</math></b>	$3.2 \pm 0.8$	<b><math>2.3 \pm 0.7</math></b>	$2.4 \pm 0.7$	
	$T_C^\mu$	$0.9 \pm 0.4$	<b><math>0.7 \pm 0.4</math></b>	$0.8 \pm 0.4$	$1.5 \pm 0.6$	$1.2 \pm 0.6$	<b><math>1.1 \pm 0.6</math></b>	$2.5 \pm 0.9$	<b><math>1.7 \pm 0.8</math></b>	$1.9 \pm 1.0$	
	$T_D^\mu$				$2.7 \pm 0.8$	$2.1 \pm 0.7$	<b><math>1.9 \pm 0.6</math></b>	$4.0 \pm 1.0$	<b><math>2.9 \pm 0.9</math></b>	$2.9 \pm 1.0$	
	$T_C^\mu$				$2.3 \pm 0.8$	$1.6 \pm 0.8$	<b><math>1.5 \pm 0.8</math></b>	$3.1 \pm 1.2$	<b><math>2.1 \pm 1.0</math></b>	$2.2 \pm 1.3$	
	$T_D^\mu$							$4.4 \pm 1.3$	<b><math>2.9 \pm 1.0</math></b>	$3.0 \pm 0.9$	
	$T_C^\mu$							$3.0 \pm 1.9$	<b><math>2.1 \pm 1.4</math></b>	<b><math>2.1 \pm 1.4</math></b>	

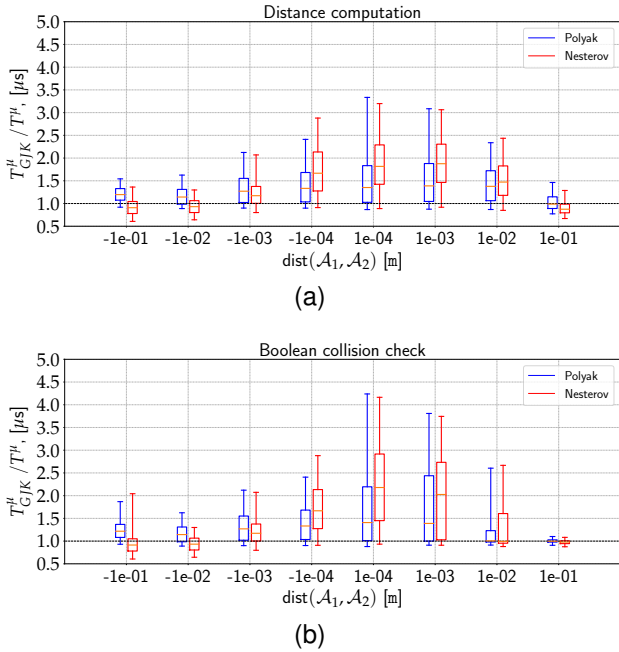


Fig. 11. Speed-ups of Polyak and Nesterov-accelerated GJK over vanilla GJK on the YCB benchmark. The plots show ratios of the number of execution times for (a) distance computation and (b) boolean collision checking of Polyak-accelerated GJK and Nesterov-accelerated GJK (with normalization) against vanilla GJK. Ratios over 1.0 show speed-ups of accelerated GJK over vanilla GJK.

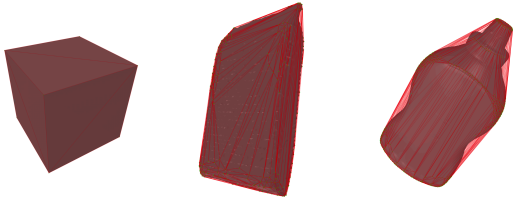
time  $T^\mu$  for Polyak-accelerated GJK, Nesterov-accelerated GJK and vanilla GJK. In Fig. 11, we report relative accelerations  $T_{GJK}^\mu / T_{polyak}^\mu$  and  $T_{GJK}^\mu / T_{Nesterov}^\mu$  of Polyak-accelerated compared to GJK. These relative accelerations are computed on a given collision problem, and Fig. 11 reports their statistical distributions. These relative measures allow analyzing the effects of the studied algorithms on the same collision

problems, which are not captured when using absolute values. Overall, Polyak and Nesterov-accelerated GJK significantly reduce the execution time when compared to GJK in cases where shapes are shallowly intersecting or in close-proximity. It is worth recalling, at this stage, that when two shapes are relatively far from each other, any broadphase algorithm will automatically discard such a pair. Only in a small percentage of cases, Polyak-accelerated GJK and Nesterov-accelerated GJK are slower than GJK. When measuring the absolute performance of the two proposed methods, Polyak-accelerated GJK provides less acceleration than Nesterov-accelerated GJK in critical cases with close proximity and shallowly overlapping collision problems. However, Polyak-accelerated GJK is more robust than Nesterov-accelerated GJK as it is almost always better than vanilla GJK, even when the shapes are distant or overlap.

In Table. I, we select three meshes with an increasing number of vertices to highlight the benefits of the Polyak and Nesterov accelerations. For each pair, we report the mean and the standard deviation of the execution time for distance computation and Boolean collision checking. We consider the challenging set-up of close-by or shallowly intersecting shapes in the range of separation distances  $-0.01 \text{ m} \leq \text{dist}(\mathcal{A}_1, \mathcal{A}_2) \leq 0.01 \text{ m}$ . The lower mean and standard deviation show that Polyak and Nesterov-accelerated GJK are faster than the vanilla GJK and reduce the spread of computation times across the different collision problems in this setting.

From this benchmark involving shapes from the YCB dataset, we can distinguish two use cases in which one would prefer using Polyak-accelerated GJK compared to Nesterov-accelerated and vice-versa. In tasks where the exact distance between the shapes needs to be computed and where this distance separating the shapes can take any value, due to

TABLE II  
SOLVE TIME IN MICRO-SECONDS OF GJK-LIKE SOLVERS VS. SOTA  
QUADRATIC PROGRAMMING PROXQP SOLVER.



	$N_v = 8$ $N_f = 6$	$N_v = 250$ $N_f = 496$	$N_v = 940$ $N_f = 1876$
ProxQP	$5.3 \pm 2.7 \mu s$	$(2 \pm 0.6) \cdot 10^3 \mu s$	$(20 \pm 14) \cdot 10^3 \mu s$
GJK	$0.2 \pm 0.03 \mu s$	$0.8 \pm 0.3 \mu s$	$2.1 \pm 0.5 \mu s$
Nesterov	$0.2 \pm 0.05 \mu s$	$0.7 \pm 0.2 \mu s$	$1.4 \pm 0.3 \mu s$
Polyak	$0.2 \pm 0.05 \mu s$	$0.6 \pm 0.2 \mu s$	$1.4 \pm 0.4 \mu s$

its robustness, the Polyak-accelerated GJK algorithm is better suited than its Nesterov counterpart. However, in a situation involving shapes interacting at close proximity, like in a contact physics simulation, it is preferable to choose the Nesterov-accelerated GJK. Before studying the performance of GJK and our proposed accelerations for physics simulation, we first show the benefits of using GJK-based algorithms for collision detection instead of standard off-the-shelf optimization solvers.

### C. GJK-like algorithms vs. generic quadratic programming solvers

As explained in Sec. II, in the case of two convex meshes, the collision problem can be formulated as a Quadratic Program (2) (QP), which can be solved using any generic QP solver [43], [46]–[49]. In Table II, we compare the performance of GJK and our proposed accelerations against the state-of-the-art ProxQP solver [43]. We report the computation timings in micro-seconds for pairs of identical shapes with an increasing number of vertices ( $N_v$ ) and faces ( $N_f$ ). The results are staggering: for very simple convex meshes like a cube, GJK, and its accelerated variants are already more than 10 times faster than the QP solver. When the complexity of the meshes increases, GJK and its variants are thousands to tens of thousands of times faster than the QP solver, making generic QP solvers prohibitive for collision detection in real-time applications like robotics or computer graphics.

### D. Collision detection for physics simulation

In the previous benchmarks, we have experimentally shown the improvement of our methods, Polyak-accelerated GJK and Nesterov-accelerated GJK, over the vanilla GJK algorithm for collision problems which are important in practice, *i.e.* when the broadphase has not filtered collision pairs and are thus overlapping or in close proximity. So far, the benchmarks have been constructed by randomly selecting poses for our shapes. However, in robotics applications such as trajectory optimization, motion planning, or computer graphics, the successive poses between objects are usually correlated by time. In this sub-section, we study how vanilla GJK, Polyak-accelerated GJK, and Nesterov-accelerated GJK can be warm-

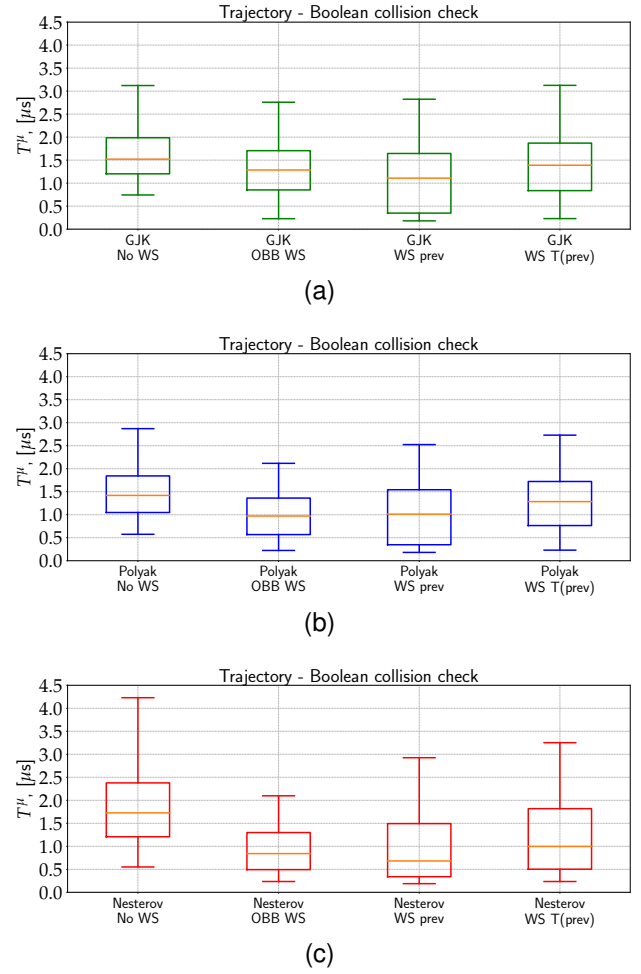


Fig. 12. Boolean collision checking of YCB objects' trajectories (see Fig. 13) for different warm-start strategies for (a) vanilla GJK, (b) Polyak-accelerated GJK, and (c) Nesterov-accelerated GJK (with normalization). In the three figures, WS is an abbreviation of *warm-start*. The *No WS* strategy signifies the algorithm is initialized with  $\mathbf{x}_0 = (1, 0, 0)^T$ . The *OBB WS* strategy uses the objects' current OBBs centers to compute  $\mathbf{x}_0$ . In both *WS prev* and *WS T(prev)*,  $\mathbf{x}_0$  is computed using GJK or EPA's previous solution, when this solution is available (*i.e.*, when the previous collision problem was not discarded by the broadphase). Contrary to *WS prev*, *WS T(prev)* corrects the previous solution using the relative displacement of the shapes between the two considered time steps.

started using the previous time instant, as occurring inside physics simulators.

To do so, we create a dataset of trajectories using pairs of objects from the YCB dataset used in Sec. IV-B. We randomly select 1000 pairs of YCB objects and drop them in a funnel as shown in Fig. 13. At the beginning of the simulation, each object is given a random pose and random translational and rotational velocities. The simulation is then run at 120Hz for 1 second. When a collision occurs, the GJK and EPA (expanding polytope algorithm) algorithms are called to determine the position of the contact points and the corresponding normal for the considered pair of objects. The collision is then resolved using a contact solver based on the Projected Gauss-Seidel [50] algorithm to account for a second-order cone representing friction, following the implementation proposed in [51]. In total, 120k collision problems are generated. For



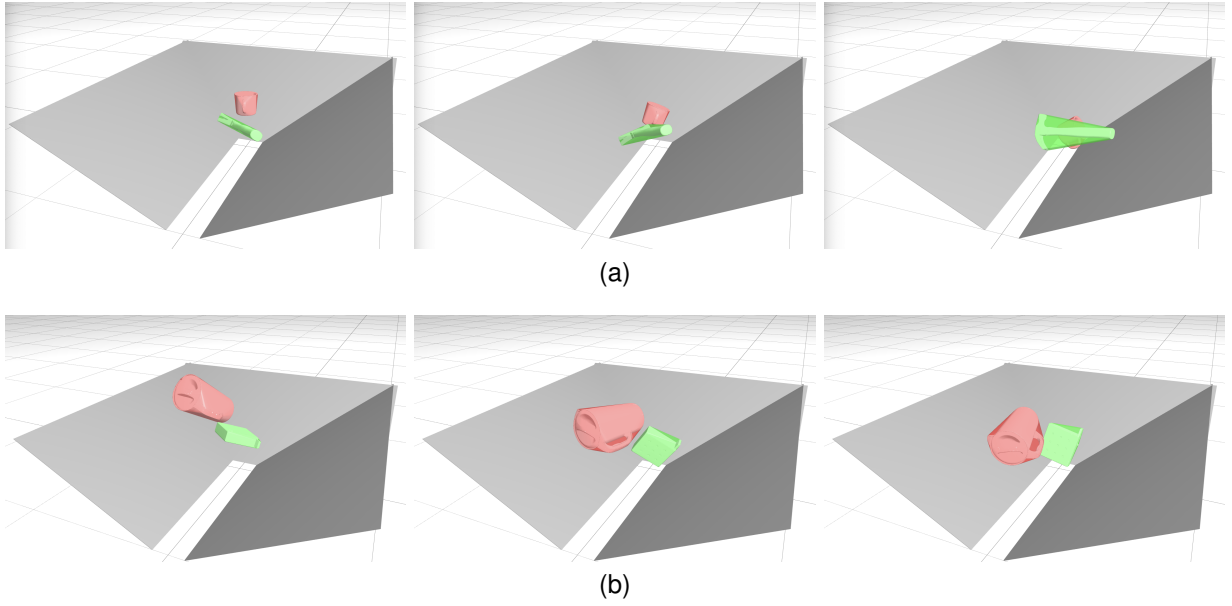


Fig. 13. Two different trajectories (a) and (b) with two different pairs of objects from the YCB dataset. The objects are dropped with a random initial velocity for each trajectory in a funnel (the grey walls). At each time step, if the broadphase cannot discriminate if the shapes are in collision or not, we use the vanilla GJK algorithm or our proposed Polyak and Nesterov-accelerated variants of GJK to determine if a collision occurs between the convex-hulls of the collision pair.

each collision problem, we extract the YCB shapes and their poses.

This dataset allows us to evaluate the vanilla, Polyak-accelerated, and Nesterov-accelerated GJK algorithms on the same collision problems generated by a physics simulation. Interestingly, this dataset allows us to study only the collision problems not filtered by the broadphase of the physics simulator, as explained in Sec. I. During the broad phase, the oriented bounding boxes of the objects (OBBs, as shown in Fig. 1) are used to assess if objects are not in collision. Therefore, if the broad phase does not filter a collision, the GJK algorithm and our proposed accelerations are called and solve the boolean collision check problem. Finally, this dataset allows us to test different strategies to warm-start (WS) the GJK algorithm and our proposed accelerations. We denote by  $x_0^t$  the initial guess given to vanilla, Polyak-accelerated and Nesterov-accelerated GJK at time step  $t$  of the simulation. We also denote by  $x^{t-1}$  the separation vector found by GJK (accelerated or not) or EPA at time-step  $t - 1$  of the simulation. We consider four different warm-start strategies for the vanilla GJK algorithm and our proposed accelerations:

- 1) the first strategy is the *No WS* strategy, where the vanilla, Polyak, and Nesterov GJK algorithms are initialized using  $x_0^t = (1, 0, 0)^T$ . This strategy serves as a baseline for the other warm-start strategies.
- 2) The second strategy is the *OBB WS* strategy, where  $x_0^t = c_1^t - c_2^t$  with  $c_1^t$  and  $c_2^t$  being the centers of the considered objects' oriented bounding boxes. This warm-start is used in all the previous benchmarks, as explained at the beginning of this section.
- 3) The third strategy is the *WS prev* strategy, where  $x_0 = x^{t-1}$  is initialized using the solution found by GJK or EPA in the previous simulation time step.

- 4) The fourth and last strategy is the *WS T(prev)* strategy. The difference with the *WS prev* strategy is that we use the relative transformation of the shapes between time steps  $t$  and  $t - 1$  to anticipate how  $x^{t-1}$  might move between these two time steps.

The last two warm-starting strategies might not always be actionable. Indeed, if at time step  $t - 1$  the broad phase finds no collision between the two considered shapes, the GJK and EPA algorithms are not called, and therefore,  $x^{t-1}$  does not exist. Consequently, if GJK needs to be called at time step  $t$ , it cannot use  $x^{t-1}$ . In such a case, these two strategies fall back to the second strategy, which exploits the objects' OBBs.

We run vanilla, Polyak-accelerated and Nesterov-accelerated GJK on the dataset of trajectories described previously; the results of this benchmark are summed up in Fig. 12. In this figure, we report the computation time of the boolean collision check for GJK and our proposed accelerations. Importantly, this figure only considers the collision problems *which were not filtered by the broad phase*, as GJK or its accelerations would not be called otherwise. In doing so, we aim to provide the clearest possible picture of the computation time dedicated to GJK in a physics computation. Due to the filtering of the broad phase, the typical distance separating the shapes is less than a few centimeters; this corresponds to the overlapping and close-proximity cases described in the previous benchmarks. First, the results show that for the three studied methods, the *No WS* and *WS T(prev)* warm-start strategies provided a worse initial guess than the two other warm-start strategies. It appears that the *WS T(prev)* strategy is often the worse strategy; this observation means that the separation vector computed by GJK and/or EPA moves in a non-trivial manner between time steps  $t - 1$  and  $t$  of the simulation. For vanilla GJK, the



best warm-starting strategy is the *WS prev* strategy, which re-uses the separation vector computed by GJK and EPA at time step  $t - 1$  of the simulation. For Polyak-accelerated GJK, both the *OBB WS* and *WS prev* strategies perform better than vanilla GJK's best warm-starting strategy. However, contrary to GJK, the *OBB WS* strategy is arguably better than the *WS prev* strategy as it greatly reduces the variance of the computation timings distribution. For Nesterov-accelerated GJK, the results are even more significant: both the *OBB WS* and *WS prev* strategy significantly outperform GJK with its best warm-starting strategy. When using the *OBB WS* and *WS prev* strategies, the Nesterov acceleration allows the median of computation times to reach close to  $0.5\mu\text{s}$ , compared to a median above  $1\mu\text{s}$  in the case of GJK's best warm-starting strategy. Like the Polyak acceleration, the Nesterov-accelerated GJK algorithm significantly reduces the spread of the distribution of computation times compared to GJK. This is especially visible when using the *OBB WS* strategy together with the Nesterov acceleration. Finally, this benchmark shows that physics simulation strongly benefits from using Nesterov-accelerated GJK warm-started using the *OBB WS* strategy.

#### E. Importance of the simplex strategy in GJK

We conclude this section by demonstrating the importance of the simplex strategy used in GJK and our method when solving collision problems. To do so, we evaluate the performance of the Frank-Wolfe algorithm (Alg. 1), the recent NESMINO algorithm [20], GJK, and our proposed Nesterov-acceleration of GJK on ellipsoids and cubes and report the results in Table. III. Although the NESMINO algorithm is similar to projected-gradient descent and strongly differs from Frank-Wolfe-like algorithms, it uses the classic Nesterov acceleration, which makes it interesting to compare to our method. FW, GJK, and Nesterov-accelerated GJK stop when a tolerance of  $\epsilon = 10^{-8}$  on the FW duality-gap is met. Therefore, to render the NESMINO algorithm comparable to the other considered methods, we run NESMINO until the distance between its solution and the solution found by GJK is less than  $\sqrt{\epsilon} = 10^{-4}$ .

In Table IVa, we consider 1000 collision problems between pairs of ellipsoids for each distance category (overlapping, close-proximity, and distant). Shapes are in close-proximity when  $0 \leq \text{dist}(\mathcal{A}_1, \mathcal{A}_2) \leq 0.1\text{m}$ . In this first scenario, all algorithms have a comparable number of operations per iteration. Indeed, the projection operation used in NESMINO when the shapes are ellipsoids has the same complexity as the support operation used in the three other algorithms. Although GJK and our method also do a simplex projection at each iteration, this operation has the same complexity as computing the support point. In the case of strictly-convex shapes such as ellipsoids, GJK, and Nesterov-accelerated GJK significantly outperform the FW and NESMINO algorithms. This is especially the case when the shapes are overlapping or in close proximity where GJK and our method take 3 to 10 times fewer iterations compared to FW or NESMINO.

In Table IVb, we repeated the same experiments with collision pairs of cubes. Since cubes are polytopes with a

TABLE III  
NUMBER OF ITERATIONS FOR DISTANCE COMPUTATION BETWEEN ELLIPSOIDS (A) AND BETWEEN CUBES (B).

	FW (Alg. 1)	Nesmino [20]	GJK (Alg. 4)	Ours (Alg. 6)
Overlapping	73 ± 62	50 ± 18	<b>6 ± 2</b>	<b>6 ± 3</b>
Close-proximity	48 ± 42	74 ± 29	16 ± 5	<b>7 ± 2</b>
Distant	4 ± 1	18 ± 2	4 ± 1	13 ± 4

(a)

	FW (Alg. 1)	Nesmino [20]	GJK (Alg. 4)	Ours (Alg. 6)
Overlapping	5.4k ± 4.8k	922 ± 244	6 ± 1	<b>5 ± 1</b>
Close-proximity	14.3k ± 9.7k	828 ± 225	5 ± 1	<b>4 ± 1</b>
Distant	13.1k ± 13.5k	623 ± 219	4 ± 1	<b>3 ± 1</b>

(b)

small number of vertices, GJK, and Nesterov-accelerated GJK only take a few iterations to reach a tolerance of  $\epsilon = 10^{-8}$ . However, because cubes are non-strictly convex shapes, the convergence of the FW algorithm is  $O(1/\epsilon)$ , i.e. it takes on the order of  $1/\epsilon$  iterations to reach an FW duality-gap of  $\epsilon$ . The NESMINO algorithm takes fewer iterations than FW but more than 100 times more iterations than GJK and our method. In the specific case of polytopes, the NESMINO algorithm is also much more costly per iteration than FW, GJK, or Nesterov-accelerated GJK, as it replaces the computation of support points with much more costly projections on the original polytopes.

#### V. CONCLUSION

In this work, we have first established that the well-known GJK algorithm can be understood as a variant of the Frank-Wolfe method, well studied within the convex optimization community, and more precisely, GJK can be identified as a sub-case of fully-corrective Frank-Wolfe. Subsequently, this connection has enabled us to accelerate the GJK algorithm in the sense of Nesterov acceleration by adapting recent contributions on applying Polyak and Nesterov acceleration to the context of Frank-Wolfe. Through extensive benchmarks, we have shown that this acceleration is beneficial for both collision detection and distance computation settings for scenarios where shapes intersect or are close, accelerating collision detection by up to a factor of two. Interestingly, these two scenarios notably encompass the generic contexts of planning and control as well as physical simulation, which are essential areas of modern robotics. Therefore, although the proposed accelerations correspond to improvements of GJK's execution time on the order of a few microseconds, modern robotics applications may solve millions to billions of collision problems when, for instance, learning a policy with RL [52].

The Polyak and Nesterov accelerations for GJK are already included in the HPP-FCL library [45], notably used by the HPP framework [4] for motion planning, the Pinocchio framework [53] dedicated to simulation and modeling, the Crocodyl [54] and the OSC-2 [55] software dedicated to trajectory optimization, to name a few. In future work, we plan to leverage these accelerated collision detection algorithms in the

scope of differentiable collision detection [56], differentiable simulation [57], [58] and constrained optimal control involving contact interactions [54], [59], [60].

Finally, one can expect this work to be largely adopted in the current available GJK implementations, as it only requires minor algorithmic changes. This work should benefit a large audience within robotics (e.g., simulation, planning, control) and beyond by addressing issues shared by other communities, including computer graphics and computational geometry.

#### ACKNOWLEDGMENTS

We warmly thank Francis Bach, Adrien Escande, Joseph Mirabel, and Mehdi Benaïgue for fruitful discussions on the various topics covered by this article. We also warmly thank the cohort of developers who contribute to developing open-source, useful, reproducible, and extensible software, which primarily benefits this project and, more widely, the robotics ecosystem.

This work was partly supported by the European Regional Development Fund under the project IMPACT (reg. no. CZ.02.1.01/0.0/0.0/15 003/0000468), by the French government under the management of Agence Nationale de la Recherche as part of the “Investissements d’avenir” program, reference ANR-19-P3IA-0001 (PRAIRIE 3IA Institute), by the AGIMUS project, funded by the European Union under GA no.101070165 - views and opinions expressed are those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission, neither the European Union nor the European Commission can be held responsible for them - and by the Louis Vuitton ENS Chair on Artificial Intelligence.

#### REFERENCES

- [1] E. Coumans and Y. Bai, “Pybullet, a python module for physics simulation for games, robotics and machine learning,” 2016.
- [2] Nvidia, “Persistent contact manifold,” 2008.
- [3] E. Todorov, T. Erez, and Y. Tassa, “Mujoco: A physics engine for model-based control,” in *2012 IEEE/RSJ International conference on intelligent robots and systems*. IEEE, 2012, pp. 5026–5033.
- [4] J. Mirabel, S. Tonneau, P. Fernbach, A.-K. Seppälä, M. Campana, N. Mansard, and F. Lamiraux, “Hpp: A new software for constrained motion planning,” in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 383–389.
- [5] M. A. Toussaint, K. R. Allen, K. A. Smith, and J. B. Tenenbaum, “Differentiable physics and stable modes for tool-use and manipulation planning,” *Robotics: Science and Systems*, 2018.
- [6] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, “Motion planning with sequential convex optimization and convex collision checking,” *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1251–1270, 2014.
- [7] C. Ericson, *Real-time collision detection*. CRC Press, 2004.
- [8] K. Mamou and F. Ghorbel, “A simple and efficient approach for 3d mesh approximate convex decomposition,” in *2009 16th IEEE international conference on image processing (ICIP)*. IEEE, 2009, pp. 3501–3504.
- [9] E. Gilbert and D. Johnson, “Distance functions and their application to robot path planning in the presence of obstacles,” *IEEE Journal on Robotics and Automation*, vol. 1, no. 1, pp. 21–30, 1985.
- [10] O. Stasse, A. Escande, N. Mansard, S. Miossec, P. Evrard, and A. Kheddar, “Real-time (self)-collision avoidance task on a hrp-2 humanoid robot,” in *2008 IEEE international conference on robotics and automation*. IEEE, 2008, pp. 3200–3205.
- [11] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, “A fast procedure for computing the distance between complex objects in three-dimensional space,” *IEEE Journal on Robotics and Automation*, vol. 4, no. 2, pp. 193–203, 1988.
- [12] G. v. d. Bergen, “A fast and robust gjk implementation for collision detection of convex objects,” *Journal of graphics tools*, vol. 4, no. 2, pp. 7–25, 1999.
- [13] G. Van den Bergen, “Proximity Queries and Penetration Depth Computation on 3D Game Objects,” in *Game Developers Conference*, 2001.
- [14] M. C. Lin and J. F. Canny, “A fast algorithm for incremental distance calculation,” in *ICRA*, vol. 91, 1991, pp. 9–12.
- [15] B. Mirtich, “V-clip: Fast and robust polyhedral collision detection,” *ACM Transactions On Graphics (TOG)*, vol. 17, no. 3, pp. 177–208, 1998.
- [16] S. Cameron, “A comparison of two fast algorithms for computing the distance between convex polyhedra,” *IEEE transactions on Robotics and Automation*, vol. 13, no. 6, pp. 915–920, 1997.
- [17] G. Van Den Bergen, *Collision detection in interactive 3D environments*. CRC Press, 2003.
- [18] M. Montanari, N. Petrinic, and E. Barbieri, “Improving the gjk algorithm for faster and more reliable distance queries between convex objects,” *ACM Transactions on Graphics (TOG)*, vol. 36, no. 3, pp. 1–17, 2017.
- [19] M. Macklin, “Simulation for learning and robotics: Numerical methods for contact, deformation, and identification,” Ph.D. dissertation, University of Copenhagen, 2020.
- [20] X. Qin and N. T. An, “Smoothing algorithms for computing the projection onto a minkowski sum of convex sets,” *Computational Optimization and Applications*, vol. 74, no. 3, pp. 821–850, 2019.
- [21] E. G. Gilbert, “An iterative procedure for computing the minimum of a quadratic form on a convex set,” *SIAM Journal on Control*, vol. 4, no. 1, pp. 61–80, 1966.
- [22] P. Wolfe, “Finding the nearest point in a polytope,” *Mathematical Programming*, vol. 11, pp. 128–149, 1976.
- [23] A. d’Aspremont, D. Scieur, A. Taylor *et al.*, “Acceleration methods,” *Foundations and Trends® in Optimization*, vol. 5, no. 1-2, pp. 1–245, 2021.
- [24] D. Garber and E. Hazan, “Playing non-linear games with linear oracles,” in *2013 IEEE 54th annual symposium on foundations of computer science*. IEEE, 2013, pp. 420–428.
- [25] J. Guélat and P. Marcotte, “Some comments on wolfe’s ‘away step,’” *Mathematical Programming*, vol. 35, no. 1, pp. 110–119, 1986.
- [26] M. Jaggi, “Revisiting frank-wolfe: Projection-free sparse convex optimization,” in *International conference on machine learning*. PMLR, 2013, pp. 427–435.
- [27] T. Kerdreux, A. d’Aspremont, and S. Pokutta, “Projection-free optimization on uniformly convex sets,” in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2021, pp. 19–27.
- [28] S. Lacoste-Julien and M. Jaggi, “On the global linear convergence of frank-wolfe optimization variants,” *Advances in neural information processing systems*, vol. 28, 2015.
- [29] Y. E. Nesterov, “A method of solving a convex programming problem with convergence rate  $O(1/k^2)$ ,” in *Doklady Akademii Nauk*, vol. 269, no. 3. Russian Academy of Sciences, 1983, pp. 543–547.
- [30] B. Li, M. Coutino, G. B. Giannakis, and G. Leus, “A momentum-guided frank-wolfe algorithm,” *IEEE Transactions on Signal Processing*, vol. 69, pp. 3597–3611, 2021.
- [31] M. Frank and P. Wolfe, “An algorithm for quadratic programming,” *Naval research logistics quarterly*, vol. 3, no. 1-2, pp. 95–110, 1956.
- [32] B. Li, A. Sadeghi, and G. Giannakis, “Heavy ball momentum for conditional gradient,” *Advances in Neural Information Processing Systems*, vol. 34, pp. 21 244–21 255, 2021.
- [33] L. Montaut, Q. Lidec, V. Petrik, J. Sivic, and J. Carpentier, “Collision Detection Accelerated: An Optimization Perspective,” in *Proceedings of Robotics: Science and Systems*, New York City, NY, USA, June 2022.
- [34] B. Calli, A. Singh, A. Walsman, S. Srinivasa, P. Abbeel, and A. M. Dollar, “The ycb object and model set: Towards common benchmarks for manipulation research,” in *2015 international conference on advanced robotics (ICAR)*. IEEE, 2015, pp. 510–517.
- [35] J. Nocedal and S. J. Wright, *Numerical optimization*. Springer, 1999.
- [36] F. Bach *et al.*, “Learning with submodular functions: A convex optimization perspective,” *Foundations and Trends® in Machine Learning*, vol. 6, no. 2-3, pp. 145–373, 2013.
- [37] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.
- [38] C. A. Holloway, “An extension of the frank and wolfe method of feasible directions,” *Mathematical Programming*, vol. 6, pp. 14–27, 1974.
- [39] T. Kerdreux, A. d’Aspremont, and S. Pokutta, “Restarting frank-wolfe,” in *The 22nd international conference on artificial intelligence and statistics*. PMLR, 2019, pp. 1275–1283.
- [40] G. Dantzig, *Linear programming and extensions*. Princeton university press, 2016.

- [41] C. Carathéodory, “Über den variabilitätsbereich der koeffizienten von potenzreihen, die gegebene werte nicht annehmen,” *Mathematische Annalen*, vol. 64, no. 1, pp. 95–115, 1907.
- [42] B. T. Polyak, “Some methods of speeding up the convergence of iteration methods,” *Ussr computational mathematics and mathematical physics*, vol. 4, no. 5, pp. 1–17, 1964.
- [43] A. Bambade, S. El-Kazdadi, A. Taylor, and J. Carpentier, “Prox-qp: Yet another quadratic programming solver for robotics and beyond,” in *RSS 2022-Robotics: Science and Systems*, 2022.
- [44] J. Pan, S. Chitta, and D. Manocha, “Fcl: A general purpose library for collision and proximity queries,” in *2012 IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 3859–3866.
- [45] J. Pan, S. Chitta, D. Manocha, F. Lamiriaux, J. Mirabel, J. Carpentier *et al.*, “Hpp-fcl: an extension of the flexible collision library,” <https://github.com/humanoid-path-planner/hpp-fcl>, 2015–2022.
- [46] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, “Osqp: An operator splitting solver for quadratic programs,” *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020.
- [47] D. Goldfarb and A. Idnani, “A numerically stable dual method for solving strictly convex quadratic programs,” *Mathematical programming*, vol. 27, no. 1, pp. 1–33, 1983.
- [48] A. Wächter and L. T. Biegler, “On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming,” *Mathematical programming*, vol. 106, pp. 25–57, 2006.
- [49] K. Tracy, T. A. Howell, and Z. Manchester, “Differentiable collision detection for a set of convex primitives,” 2022.
- [50] B. Brogliato, A. Ten Dam, L. Paoli, F. Génot, and M. Abadie, “Numerical simulation of finite dimensional multibody nonsmooth mechanical systems,” *Appl. Mech. Rev.*, vol. 55, no. 2, pp. 107–150, 2002.
- [51] Q. L. Lidec, W. Jallet, L. Montaut, I. Laptev, C. Schmid, and J. Carpentier, “Contact models in robotics: a comparative analysis.”
- [52] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas *et al.*, “Solving rubik’s cube with a robot hand,” *arXiv preprint arXiv:1910.07113*, 2019.
- [53] J. Carpentier, G. Saurel, G. Buondonno, J. Mirabel, F. Lamiriaux, O. Stasse, and N. Mansard, “The pinocchio c++ library: A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives,” in *2019 IEEE/SICE International Symposium on System Integration (SII)*. IEEE, 2019, pp. 614–619.
- [54] C. Mastalli, R. Budhiraja, W. Merkt, G. Saurel, B. Hammoud, M. Naveau, J. Carpentier, L. Righetti, S. Vijayakumar, and N. Mansard, “Crocodyl: An efficient and versatile framework for multi-contact optimal control,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 2536–2542.
- [55] F. Farshidian *et al.*, “Optimal control for switched systems,” <https://github.com/leggedrobotics/ocs2>, 2018.
- [56] L. Montaut, Q. L. Lidec, A. Bambade, V. Petrik, J. Sivic, and J. Carpentier, “Differentiable collision detection: a randomized smoothing approach,” in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2022.
- [57] K. Werling, D. Omens, J. Lee, I. Exarchos, and C. K. Liu, “Fast and feature-complete differentiable physics engine for articulated rigid bodies with contact constraints,” in *Robotics: Science and Systems*, 2021.
- [58] Q. Le Lidec, I. Laptev, C. Schmid, and J. Carpentier, “Differentiable rendering with perturbed optimizers,” *Advances in Neural Information Processing Systems*, vol. 34, pp. 20 398–20 409, 2021.
- [59] R. Budhiraja, J. Carpentier, C. Mastalli, and N. Mansard, “Differential dynamic programming for multi-phase rigid contact dynamics,” in *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*. IEEE, 2018, pp. 1–9.
- [60] W. Jallet, A. Bambade, N. Mansard, and J. Carpentier, “Constrained differential dynamic programming: A primal-dual augmented lagrangian approach,” in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 13 371–13 378.

# Differentiable Collision Detection: a Randomized Smoothing Approach

Louis Montaut<sup>\*,1,2</sup> Quentin Le Lidec<sup>1</sup> Antoine Bambade<sup>1</sup> Vladimir Petrik<sup>2</sup> Josef Sivic<sup>2</sup> Justin Carpentier<sup>1</sup>

**Abstract**—Collision detection is an important component of many robotics applications, from robot control to simulation, including motion planning and estimation. While the seminal works on the topic date back to the 80s, it is only recently that the question of properly differentiating collision detection has emerged as a central issue, thanks notably to the ongoing and various efforts made by the scientific community around the topic of differentiable physics. Yet, very few solutions have been suggested so far, and only with a strong assumption on the nature of the shapes involved. In this work, we introduce a generic and efficient approach to compute the derivatives of collision detection for *any* pair of convex shapes, by notably leveraging randomized smoothing techniques which have shown to be particularly adapted to capture the derivatives of non-smooth problems. This approach is implemented in the HPP-FCL and Pinocchio ecosystems, and evaluated on classic datasets and problems of the robotics literature, demonstrating few micro-second timings to compute informative derivatives directly exploitable by many real robotic applications, including differentiable simulation.

## I. INTRODUCTION

Collision detection is a crucial stage for many robotic applications, including motion planning, trajectory optimization, or simulation. It is also used in many other related domains, such as computer graphics or computational geometry, just to name a few. In particular for simulation, collision detection is a central component of any physical simulator, allowing to assess of the collision between two geometries, retrieving the contact regions, if any, or computing the closest points between the shapes (also known as *witness points* [1]). In addition to the evaluation of collisions, many problems in robotics also rely on the derivatives of geometric contact information, such as optimal shape design, grasp synthesis, or differentiable simulation.

Over the past few years, differentiable simulation has gained interest within the robotics and machine learning communities thanks to the progress in gradient-based optimization techniques. Differentiable simulation consists in evaluating the gradient (or sub-gradient in case of a non-smooth contact interaction) of the simulation steps, which can then be exploited by any gradient-based optimization algorithm to efficiently solve various robotic problems involv-

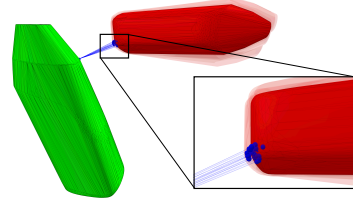


Fig. 1: Illustration of randomized smoothing approximation on two meshes from the YCB dataset. 0<sup>th</sup>-order method, Gaussian distribution,  $M = 25$  samples. The witness points for each perturbed relative pose are drawn on both objects. Randomized smoothing allows for estimating the curvature of object shapes.

ing contact interactions [2], [3], [4], [5], [6], [7]. However, most of the existing works on the topic have only considered the differentiation of the physical principles (Coulomb friction constraints, maximum dissipation principles, Signorini conditions, etc.), without systematically considering the contribution of collision detection in the derivatives, for example, by limiting themselves to a specific type of shapes [4], [7]. In [8], the authors demonstrate that the distance function between two objects is continuously differentiable if and only if at least one of the objects is strictly convex. They proceed to design an algorithm to construct a strictly convex hull for any mesh.

To the best of our knowledge, no systematic procedure to compute collision derivatives has been proposed in the literature, and it remains a challenging problem. In this paper, we propose to address this issue by introducing a generic approach to compute gradient information of collision detection instances in the context of convex shapes (not necessarily strictly convex), without assuming any regularity on the shapes. This includes not only the standard geometric primitives (ellipsoid, sphere, cone, cube, cylinder, capsules, etc.) but also meshes, which are the standard representation of objects in many applications involving physics simulation and a wide range of robotic applications. Our key contributions lie in leveraging randomized smoothing techniques [9] to estimate the gradients of the collision detection procedure through two proposed estimators:

- a 0<sup>th</sup>-order estimator (see Fig. 1), which does not make any assumption on the nature of the collision algorithms in use,
- a 1<sup>st</sup>-order estimator (see Fig. 2), which exploits the optimality conditions of the underlying optimization program inherent to collision detection with convex shapes. Remarkably, this estimator is at the same time more accurate and less expensive to derive than the 0<sup>th</sup>-order estimator.

After stating the problem of collision detection in Sec. II, we evaluate the efficiency of these two derivative estimators

<sup>1</sup>Inria - Département d'Informatique de l'École normale supérieure, PSL Research University. Email: `firstname.lastname@inria.fr`

<sup>2</sup>Czech Institute of Informatics, Robotics and Cybernetics, Czech Technical University. Email: `firstname.lastname@cvut.cz`

This work was partly supported by the European Regional Development Fund under the project IMPACT (reg. no. CZ.02.1.01/0.0/0.0/15 003/0000468), by the French government under the management of Agence Nationale de la Recherche as part of the "Investissements d'avenir" program, reference ANR-19-P3IA-0001 (PRAIRIE 3IA Institute), the AGIMUS project, funded by the European Union under GA no.101070165 and the Louis Vuitton ENS Chair on Artificial Intelligence.

introduced in Sec. III and IV on standard optimization problems (Sec. V). We will release our code as open-source within the Pinocchio [10] framework so that it can be easily disseminated into other existing robotic software and applications. A C++ implementation of our estimators can be found at <https://github.com/lmontaut/RScollision>.

## II. PROBLEM STATEMENT

Collision detection is a very old topic within the robotics community. It consists in determining whether a pair of shapes are in collision or not and finding the contact point locations [1], [11], [12], [13]. To lower the computational burden, the shapes are often assumed to be convex or decomposed as a collection of convex meshes [14]. Such convexity assumptions have led to highly efficient, generic, and robust algorithms to solve collision detection problems, most of which belong to the family of the Gilbert-Johnson-Keerthi (GJK) algorithms [11], [12], [15], [13].

From an optimization perspective, the problem of collision detection can be generically formulated as a convex minimization program of the form:

$$\mathbf{x}_1^*, \mathbf{x}_2^* = \arg \min_{\mathbf{x}_1 \in \mathcal{A}_1, \mathbf{x}_2 \in \mathcal{A}_2} \|\mathbf{x}_1 - \mathbf{x}_2\|_2^2, \quad (1)$$

where  $\|\cdot\|_2$  is the Euclidean norm,  $\mathcal{A}_1$  and  $\mathcal{A}_2$  are convex shapes and  $\mathbf{x}_1^*$  and  $\mathbf{x}_2^*$  are the so-called witness points. The GJK algorithm is typically used to solve (1). When the objects are in collision, most physics simulations extend (1) to also recover the *penetration depth*, in order, for instance, to correct the interpenetration between rigid objects, inherent to the discrete integration scheme used inside physical simulators. The penetration depth corresponds to the length of the smallest translation, which must be applied on the relative configuration between  $\mathcal{A}_1$  and  $\mathcal{A}_2$  to separate them. To compute this quantity, the Expanded Polytope Algorithm (EPA) [1], [12] is typically used and enables the computation of witness points  $\mathbf{x}_1^*$  and  $\mathbf{x}_2^*$  in the case of penetration. In both penetration and non-collision cases, the witness points computed by GJK and EPA lie on the boundaries of the objects considered. Finally, in the presence or absence of collision, the signed distance between  $\mathcal{A}_1$  and  $\mathcal{A}_2$  is defined as:

$$d(\mathcal{A}_1, \mathcal{A}_2) = \begin{cases} \|\mathbf{x}_1^* - \mathbf{x}_2^*\| & \text{if } \mathcal{A}_1 \cap \mathcal{A}_2 = \emptyset, \\ -\|\mathbf{x}_1^* - \mathbf{x}_2^*\| & \text{otherwise.} \end{cases} \quad (2)$$

To parameterize the relative position and orientation of  $\mathcal{A}_1$  and  $\mathcal{A}_2$  in the 3D space, it is convenient to consider their relative configuration,  $T(\mathbf{q}) \in SE(3)$  which is parameterized by a vector  $\mathbf{q} \in \mathbb{R}^7$  (3 coordinates for translation and 4 for rotation, encoded by a quaternion). From a mathematical perspective, the main objective of this paper is to retrieve the partial derivatives of (1) w.r.t  $\mathbf{q}$  namely  $\partial \mathbf{x}_1^* / \partial \mathbf{q}$  and  $\partial \mathbf{x}_2^* / \partial \mathbf{q}$ , in the two cases where the two geometries are in collision or not. It is worth mentioning at this stage that  $\partial \mathbf{x}_1^* / \partial \mathbf{q}$  and  $\partial \mathbf{x}_2^* / \partial \mathbf{q}$  are Jacobian matrices of dimension  $3 \times 6$ , with  $\partial \mathbf{q}$  being an element of the tangent of  $SE(3)$  in  $T(\mathbf{q})$  [16].

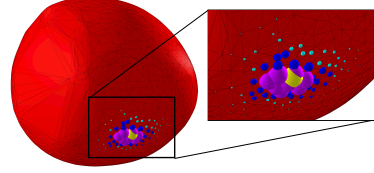


Fig. 2: **Soft-max weights of the Gumbel distribution for the 1<sup>st</sup>-order gradient estimator, mesh from the YCB dataset.** Different colors represent neighbors of different levels. In yellow: the current-witness point of the object. In purple: level 1 neighbors. In blue: level 3 neighbors. In cyan: level 5 neighbors. The size of the neighbors is proportional to the soft-max weight.

Finite differences and other existing methods are likely to fail at capturing informative gradients in most cases. Indeed, even if an object is originally smooth, its approximation as a mesh may present some non-smoothness properties (typically localized around the mesh's vertices), which are difficult to handle for classic optimization-based techniques. More precisely, because they are locally flat, the resulting gradients tend to "forget" the curvature of the original surface they approximate [7], leading to uninformative gradients. To overcome these issues, we introduce hereafter two approaches to retrieve a precise estimator of the partial derivatives  $\partial \mathbf{x}_1^* / \partial \mathbf{q}$  and  $\partial \mathbf{x}_2^* / \partial \mathbf{q}$ , which are extensively cross-validated against finite differences in Sec. V.

## III. 0<sup>TH</sup>-ORDER ESTIMATION OF COLLISION DETECTION DERIVATIVES

In this section, we introduce a generic approach to compute a 0<sup>th</sup>-order estimate of the collision detection derivatives by leveraging randomized smoothing techniques which we first review.

### A. Background on randomized smoothing

Originally used in black-box optimization algorithms [17], [18], randomized smoothing was recently introduced in the machine learning community [19], [9] to integrate discrete operations inside neural networks.

In more details, any function  $g$  can be approximated by convolving it with a probability distribution  $\mu$ :

$$g_\epsilon(x) = \mathbb{E}_{Z \sim \mu} [g(x + \epsilon Z)], \quad (3)$$

which corresponds to the randomly smoothed counterpart of  $g$  and which can be estimated with a Monte-Carlo estimator as follows:

$$g_\epsilon(x) \approx \frac{1}{M} \sum_{i=0}^M g(x + \epsilon z^{(i)}), \quad (4)$$

where  $\{z^{(1)}, \dots, z^{(M)}\}$  are i.i.d. samples and  $M$  is the number of samples. The  $\epsilon$  parameter controls the level of noise injected in  $g_\epsilon$ . Intuitively, the convolution makes  $g_\epsilon$  smoother than its original counterpart  $g$  and, thus, yields better-conditioned gradients [9].

Using an integration by part yields a 0<sup>th</sup>-order estimator of the gradient:

$$\nabla_x^{(0)} g_\epsilon(x) = \frac{1}{M} \sum_{j=0}^M -g(x + \epsilon z^{(j)}) \frac{\nabla \log \mu(z^{(j)})^\top}{\epsilon}. \quad (5)$$

This 0<sup>th</sup>-order estimator can be used even when  $g$  is non-differentiable to obtain first-order information, which can then be exploited by any gradient-based optimization technique, as shown by applications in machine learning [9], computer vision [20], [21] and robotics for the optimal control of non-smooth dynamical systems [22], [23].

#### B. Direct application of randomized smoothing to collision detection

Randomized smoothing can be applied to collision detection by considering the witness points as functions of the configuration vector, i.e.,  $(\mathbf{x}_1^*(\mathbf{q}), \mathbf{x}_2^*(\mathbf{q}))$ , regardless of the method used to compute them. We choose to use the combination of the two complementary, and state-of-the-art algorithms, the Gilbert, Johnson, and Keerthi algorithm (GJK) [11] and Expanding Polytope Algorithm (EPA) [12], to handle collision detection including penetration. Given two convex shapes  $\mathcal{A}_1$  and  $\mathcal{A}_2$  and a relative pose  $T(\mathbf{q})$ , both of these algorithms allow to compute a pair of witness points  $(\mathbf{x}_1^*(\mathbf{q}), \mathbf{x}_2^*(\mathbf{q}))$ .

To compute a 0<sup>th</sup>-order estimator of the gradients of (1), we perturb  $M$  times the configuration vector by sampling from a distribution  $\mu$ :

$$\nabla_{\mathbf{q}}^{(0)} \mathbf{x}_{i,\epsilon}^*(\mathbf{q}) = \frac{1}{M} \sum_{j=0}^M -\mathbf{x}_i^*(\mathbf{q} + \epsilon \mathbf{z}^{(j)}) \frac{\nabla \log \mu(\mathbf{z}^{(j)})^\top}{\epsilon}, \quad (6)$$

for  $i = 1, 2$ . As a consequence, this 0<sup>th</sup>-order estimator requires to run the GJK+EPA procedure  $M$  times. While the choice of distribution may lead to different convolution effects [20], [21], we found that the standard Gaussian distribution is well adapted to sample over  $SE(3)$  adequately. In Fig. 1, we give an intuition of the smoothing which results from the 0<sup>th</sup>-order estimator. To simplify the visualization, we fix the pose of  $\mathcal{A}_1$  (in green) and perturb the pose of  $\mathcal{A}_2$ . The cloud of resulting witness points captures the local geometry of  $\mathcal{A}_2$ . Finally, note that finite differences are a sub-case of randomized smoothing, which considers a specific non-smooth distribution. By construction, finite differences capture less of the underlying geometry, as it deterministically defines the sampling directions and size of the steps taken in such directions. These fundamental differences are quantitatively illustrated in Fig. 5 of Sec. V.

#### IV. 1<sup>ST</sup>-ORDER ESTIMATION OF COLLISION DETECTION DERIVATIVES

In this section, we leverage the optimality conditions of the collision detection problem to introduce a computationally efficient and generic approach to derive a 1<sup>st</sup>-order estimate of the collision detection derivatives. Similarly to the 0<sup>th</sup>-order approach developed in Sec. III, we leverage randomized smoothing to compute the local Hessian information around the witness points required in the computation of specific gradient quantities. In the particular case of meshes, we notably introduce a closed-form Hessian expression which can be directly computed from the vertices located in the neighborhood of the witness points.

**The GJK paradigm for collision detection.** To handle the collision detection problem, we put ourselves in the paradigm of GJK [11], [13], to which EPA [12] also belongs. In this paradigm, the focus is set on computing the *separation vector* between  $\mathcal{A}_1$  and  $\mathcal{A}_2$ . The separation vector is defined as the smallest translation which must be applied to the relative configuration between  $\mathcal{A}_1$  and  $\mathcal{A}_2$  to (i) bring the shapes into collision if they are not in collision, or (ii) separate the shapes if they are in collision. Whether or not the shapes are in collision, the separation vector *always* satisfies the following equation:

$$\mathbf{x}^* = \mathbf{x}_1^* - \mathbf{x}_2^*, \quad (7)$$

where  $\mathbf{x}_1^*$  and  $\mathbf{x}_2^*$  are the witness points obtained as a by-product of the GJK and EPA algorithms.

Although this change in paradigm seems anecdotal, the problem of computing the separation vector  $\mathbf{x}^*$  between  $\mathcal{A}_1$  and  $\mathcal{A}_2$  is conveniently fully encapsulated in a minimization problem over the Minkowski difference of the two shapes  $\mathcal{D} = \mathcal{A}_1 - \mathcal{A}_2 = \{\mathbf{x} = \mathbf{x}_1 - \mathbf{x}_2 \mid \mathbf{x}_1 \in \mathcal{A}_1, \mathbf{x}_2 \in \mathcal{A}_2\}$ :

$$\begin{aligned} \mathbf{x}^* &= \arg \min \|\mathbf{x}\|_2^2 \\ \text{s.t. } \mathbf{x} &\in \delta\mathcal{D}, \end{aligned} \quad (8)$$

where  $\delta\mathcal{D}$  is the boundary of the Minkowski difference. Remarkably, the shapes are in collision if and only if the origin lies inside the Minkowski difference, i.e.,  $\mathbf{0} \in \mathcal{D}$ , as shown in the seminal work of [11] and revisited in [13]. In summary, the separation vector is obtained by projecting the origin onto the surface of the Minkowski difference.

**Optimality conditions of collision detection.** In order to solve (8), GJK and EPA both solve a sequence of simple linear programming sub-problems. Each sub-problem consists in computing the so-called *support function* of  $\mathcal{D}$ :

$$\sigma_{\mathcal{D}}(\mathbf{x}) = \max_{\mathbf{y} \in \mathcal{D}} \langle \mathbf{y}, \mathbf{x} \rangle, \quad (9)$$

where  $\mathbf{x} \in \mathbb{R}^3$ ,  $\mathbf{y} \in \mathcal{D}$  and  $\langle \cdot, \cdot \rangle$  is the Euclidian dot-product. A support point  $\mathbf{s} \in \delta\mathcal{D}$  belongs to the support set  $\partial\sigma_{\mathcal{D}}(\mathbf{x})$ , corresponding to the sub-gradient of  $\sigma_{\mathcal{D}}(\mathbf{x})$ , if and only if, it is a maximizer of (9). Such a point always exists and is *always* a point on the boundary of the Minkowski difference.

Computing  $\sigma_{\mathcal{D}}(\mathbf{x})$  corresponds to minimizing a linearization of the objective function of (8) at point  $-\mathbf{x}$ . Remarkably, when it is impossible to find a point  $\mathbf{x} \in \mathcal{D}$  which further decreases this linearization, both GJK and EPA have reached the optimal solution  $\mathbf{x}^*$ . This optimality condition corresponds to the convergence criterion of both GJK [11] and EPA [12] and can be stated as follows:

$$\begin{cases} \mathbf{x}^* \in \partial\sigma_{\mathcal{D}}(-\mathbf{x}^*) & \text{if } \mathbf{0} \notin \mathcal{D}, \\ \mathbf{x}^* \in \partial\sigma_{\mathcal{D}}(\mathbf{x}^*) & \text{otherwise.} \end{cases} \quad (10)$$

Eq. (10) is directly linked to the Frank-Wolf duality gap, a convergence criterion that allows measuring the progress towards an optimal solution. We refer to [13] for a complete analysis. Eq. (10) is handy as it corresponds to the optimality condition of (8) and characterizes  $\mathbf{x}^*$ . Note that we choose



to compute  $\mathbf{x}^*$  using GJK and EPA, but (10) is true no matter how  $\mathbf{x}^*$  is obtained. For the sake of simplicity, we will assume  $\mathbf{0} \notin \mathcal{D}$  but the rest of this section applies to the case where  $\mathbf{0} \in \mathcal{D}$ .

For now, let us suppose first that  $\mathcal{D}$  is smooth and strictly convex, which corresponds to the case where the shapes  $\mathcal{A}_1$  and  $\mathcal{A}_2$  are smooth and strictly convex (i.e., ellipsoids or spheres). The support set is reduced to a singleton for any  $\mathbf{x}$ :  $\partial\sigma_{\mathcal{D}}(\mathbf{x}) = \nabla\sigma_{\mathcal{D}}(\mathbf{x})$  where  $\nabla\sigma_{\mathcal{D}}$  is the gradient of the support function, i.e., the only maximizer of (9). In such a case, (10) reduces to an equality.

In practice, we use the support functions of the shapes denoted  $\sigma_{\mathcal{A}_1}$  and  $\sigma_{\mathcal{A}_2}$  to compute  $\sigma_{\mathcal{D}}$ , as  $\sigma_{\mathcal{D}} = \sigma_{\mathcal{A}_1} - \sigma_{\mathcal{A}_2}$  [11], [13]. Since we consider the relative pose  $T(\mathbf{q})$  between  $\mathcal{A}_1$  and  $\mathcal{A}_2$ , all vectors are classically expressed in the frame of  $\mathcal{A}_1$ . By decomposing the terms in the support function, we can show that for any  $\mathbf{x}$ , if  $\mathbf{s}_1 \in \partial\sigma_{\mathcal{A}_1}(\mathbf{x})$  and  $\mathbf{s}_2 \in \partial\sigma_{\mathcal{A}_2}(-R(\mathbf{q})^T\mathbf{x})$ , then:

$$\mathbf{s} = \mathbf{s}_1 - \mathbf{s}_2 \in \partial\sigma_{\mathcal{D}}(\mathbf{x}), \quad (11)$$

where  $R(\mathbf{q})$  is the rotation matrix associated to  $T(\mathbf{q})$ .

In practice, evaluating and finding a maximizer of the support function is a simple and computationally cheap operation (this partly explains the large popularity of GJK and related algorithms for collision detection [1]). Since this is true also for  $\mathbf{x}^*$ , we rewrite Eq. (10) to obtain:

$$\mathbf{x}^* - \nabla\sigma_{\mathcal{A}_1}(-\mathbf{x}^*) + T(\mathbf{q})\nabla\sigma_{\mathcal{A}_2}(R(\mathbf{q})^T\mathbf{x}^*) = \mathbf{0}. \quad (12)$$

Remarkably, the witness points  $\mathbf{x}_1^*$  and  $\mathbf{x}_2^*$  appear in (12):

$$\begin{cases} \mathbf{x}_1^*(\mathbf{x}^*) = \nabla\sigma_{\mathcal{A}_1}(-\mathbf{x}^*) \\ \mathbf{x}_2^*(\mathbf{x}^*, \mathbf{q}) = T(\mathbf{q})\nabla\sigma_{\mathcal{A}_2}(R(\mathbf{q})^T\mathbf{x}^*). \end{cases} \quad (13)$$

**Implicit function differentiation.** We define the function  $f: \mathbb{R}^3 \times \mathbb{R}^7 \rightarrow \mathbb{R}^3$  as:

$$f(\mathbf{x}, \mathbf{q}) = \mathbf{x} - \nabla\sigma_{\mathcal{A}_1}(-\mathbf{x}) + T(\mathbf{q})\nabla\sigma_{\mathcal{A}_2}(R(\mathbf{q})^T\mathbf{x}). \quad (14)$$

From (12), the separation vector  $\mathbf{x}^*(\mathbf{q})$ , parameterized by  $\mathbf{q}$ , is thus implicitly described by the equation:

$$f(\mathbf{x}^*, \mathbf{q}) = \mathbf{0}. \quad (15)$$

By expanding the 1<sup>st</sup>-order terms of  $f$ , we can relate the sensitivity of  $\mathbf{x}^*$  to the relative configuration  $\mathbf{q}$  between the shapes:

$$\frac{\partial f(\mathbf{x}^*, \mathbf{q})}{\partial \mathbf{x}^*} \delta \mathbf{x}^* + \frac{\partial f(\mathbf{x}^*, \mathbf{q})}{\partial \mathbf{q}} \delta \mathbf{q} = \mathbf{0}, \quad (16)$$

leading to the following relation:

$$\frac{\partial f(\mathbf{x}^*, \mathbf{q})}{\partial \mathbf{x}^*} \frac{\partial \mathbf{x}^*}{\partial \mathbf{q}} = - \frac{\partial f(\mathbf{x}^*, \mathbf{q})}{\partial \mathbf{q}}, \quad (17)$$

and finally to:

$$\frac{\partial \mathbf{x}^*}{\partial \mathbf{q}} = - \left[ \frac{\partial f(\mathbf{x}^*, \mathbf{q})}{\partial \mathbf{x}^*} \right]^{-1} \frac{\partial f(\mathbf{x}^*, \mathbf{q})}{\partial \mathbf{q}}, \quad (18)$$

if the Jacobian of  $f$  w.r.t.  $\mathbf{x}^*$  is invertible, where:

$$\frac{\partial f(\mathbf{x}^*, \mathbf{q})}{\partial \mathbf{x}^*} = I + \frac{\partial^2 \sigma_{\mathcal{A}_1}(-\mathbf{x}^*)}{\partial \mathbf{x}^{*2}} + R(\mathbf{q}) \frac{\partial^2 \sigma_{\mathcal{A}_2}(\mathbf{y}^*)}{\partial \mathbf{y}^{*2}} R(\mathbf{q})^T, \quad (19)$$

with  $\mathbf{y}^* = R(\mathbf{q})\mathbf{x}^*$ . The terms in  $\partial f(\mathbf{x}^*, \mathbf{q})/\partial \mathbf{q}$  are derivative terms involving elements of  $SE(3)$  and can be simply obtained by following the derivations in [16].

To evaluate both derivative terms of  $f$ , it is necessary to compute the Hessian of the support function  $\partial^2 \sigma_{\mathcal{A}}(\mathbf{x})/\partial \mathbf{x}^2$  at  $\mathbf{x}^*$  (or  $\mathbf{y}^*$ ) where  $\mathcal{A}$  stands for either shape  $\mathcal{A}_1$  or  $\mathcal{A}_2$ . The Hessian of the support function encodes the local curvature of the shape and it is easy to compute for basic smooth and strictly-convex shapes such as spheres or ellipsoids.

Let us now focus on the general case where  $\mathcal{A}_1$  or  $\mathcal{A}_2$  might not be smooth or simply convex. We explain how we can recover and compute the terms of (18). In the general case, Eq. (12) becomes:

$$\mathbf{x}^* - \mathbf{x}_1^*(\mathbf{x}^*) + \mathbf{x}_2^*(\mathbf{x}^*, \mathbf{q}) = \mathbf{0}, \quad (20)$$

with:

$$\begin{cases} \mathbf{x}_1^*(\mathbf{x}^*) \in \partial\sigma_{\mathcal{A}_1}(-\mathbf{x}^*), \\ \mathbf{x}_2^*(\mathbf{x}^*, \mathbf{q}) \in T(\mathbf{q})\partial\sigma_{\mathcal{A}_2}(R(\mathbf{q})^T\mathbf{x}^*), \end{cases}$$

which are computed by the GJK+EPA procedure. By casually writing  $\nabla\sigma_{\mathcal{A}_1}(-\mathbf{x}^*) = \mathbf{x}_1^*(\mathbf{x}^*)$  and  $T(\mathbf{q})\nabla\sigma_{\mathcal{A}_2}(R(\mathbf{q})^T\mathbf{x}_2^*) = \mathbf{x}_2^*(\mathbf{x}^*, \mathbf{q})$ , we recover (12) and ultimately (18).

**Uninformative gradients: the case of meshes.** When a shape is non-smooth or non-strictly convex, the Hessian of its support function may be null (e.g., a flat surface) or even undefined (e.g., the Hessian at the vertex of a cube). Let us consider the example of a mesh representing the convex hull of an arbitrary object to illustrate this phenomenon. A mesh  $\mathcal{A}$  is a list of  $N_v$  vertices  $\{\mathbf{v}_1, \dots, \mathbf{v}_{N_v}\}$  and therefore:

$$\begin{aligned} \sigma_{\mathcal{A}}(\mathbf{x}) &= \max_{\mathbf{v}_i \in \{\mathbf{v}_1, \dots, \mathbf{v}_{N_v}\}} \langle \mathbf{v}_i, \mathbf{x} \rangle, \\ \nabla\sigma_{\mathcal{A}}(\mathbf{x}) &= \mathbf{v}_{i^*}, \end{aligned} \quad (21)$$

for a certain  $i^* \in [1, N_v]$ . We define the matrix  $V \in \mathbb{R}^{3 \times N_v}$  and the vector  $\mathbf{z}(\mathbf{x}) \in \mathbb{R}^{N_v}$  as:

$$\begin{aligned} V^T &= (\mathbf{v}_1 \cdots \mathbf{v}_{N_v})^T, \\ \mathbf{z}(\mathbf{x}) &= V^T \mathbf{x}. \end{aligned} \quad (22)$$

This allows us to define the vector of *argmax weights*  $\mathbf{a}(\mathbf{z}) \in \mathbb{R}^{N_v}$ :

$$\mathbf{a}(\mathbf{z}) = \arg \max_{\|\mathbf{w}\|_1 \leq 1, \mathbf{0} \leq \mathbf{w}} \mathbf{z}^T \mathbf{w}. \quad (23)$$

Therefore, all components of  $\mathbf{a}(\mathbf{z}(\mathbf{x}))$  are null, except the  $i^{*th}$  component which value is 1. As a consequence, we have:

$$\nabla\sigma_{\mathcal{A}}(\mathbf{x}) = \mathbf{v}_{i^*} = \sum_i a_i \mathbf{v}_i = V \mathbf{a}(\mathbf{z}(\mathbf{x})), \quad (24)$$

and:

$$\frac{\partial^2 \sigma_{\mathcal{A}}(\mathbf{x})}{\partial \mathbf{x}^2} = V \frac{\partial \mathbf{a}(\mathbf{z})}{\partial \mathbf{z}} V^T = \mathbf{0}, \quad (25)$$

because  $\partial \mathbf{a}(\mathbf{y})/\partial \mathbf{y}$  is null almost everywhere. As a consequence, the gradients of  $\mathbf{x}^*$  w.r.t  $\mathbf{q}$  obtained after solving (18) fail to capture information regarding the curvature of the underlying object which the mesh approximates.

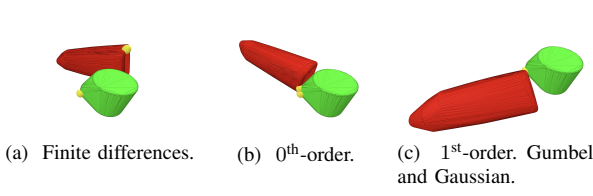


Fig. 3: **Convex hulls of YCB shapes, goal: find a relative pose such that the yellow points are contact points.** We display the final pose each method converged to. Finite differences fail to generate a satisfying pose. The 0<sup>th</sup>-order gradient estimator is better than finite differences but is not as precise as the 1<sup>st</sup>-order gradient estimator.

**Hessian estimation via randomized smoothing.** To overcome this issue, we use a randomized smoothing approach to estimate  $\partial^2 \sigma_{\mathcal{A}}(\mathbf{x})/\partial \mathbf{x}^2$ . We apply (5) to  $g = \nabla \sigma_{\mathcal{A}}$  to obtain:

$$\frac{\partial^2 \sigma_{\mathcal{A}}(\mathbf{x})}{\partial \mathbf{x}^2} \approx \frac{1}{M} \sum_{j=0}^M -\nabla \sigma_{\mathcal{A}}(\mathbf{x} + \epsilon \mathbf{z}^{(j)}) \frac{\nabla \log \mu(\mathbf{z}^{(j)})^\top}{\epsilon}. \quad (26)$$

In practice, we choose  $\mu$  to be a normalized Gaussian centered in  $\mathbf{0}$ . Although this procedure to evaluate the Hessian of the support function requires computing the support function  $M$  times, it is in practice very efficient, as first, the computation of the support function is very cheap, second, it can be warm-started and third, it is highly parallelizable. Finally, this method to estimate the Hessian of a support function is generic as it can be applied to *any* convex shape with a computationally tractable support function.

**Special case of meshes: the Gumbel distribution.** In the specific case of meshes, the structure of the support function allows replacing the Gaussian distribution by the Gumbel distribution [24], resulting in a closed form solution to estimate the mean of  $\partial^2 \sigma_{\mathcal{A}}(\mathbf{x})/\partial \mathbf{x}^2$  and thus removing the need of a Monte-Carlo estimator. Thus, using a Gumbel distribution  $\mu$  with zero mean and identity matrix variance to sample the noise, we get a closed-form solution for (3) when applied to  $g(\mathbf{z}) = \mathbf{a}(\mathbf{z})$  from Eq. (23):

$$\mathbf{a}_\epsilon(\mathbf{z}) = \mathbb{E}_{\mathbf{Z} \sim \mu} [\mathbf{a}(\mathbf{z} + \epsilon \mathbf{Z})] = \frac{1}{\sum_j e^{z_j/\epsilon}} \left( e^{z_1/\epsilon} \dots e^{z_{N_v}/\epsilon} \right)^\top. \quad (27)$$

The smoothed  $\mathbf{a}_\epsilon$  is thus simply a soft-max, which is a smooth and differentiable function of  $\mathbf{y}$  [9]. The  $\epsilon$  parameter serves as a temperature parameter [24]. Due to the nature of the soft-max operator, the  $i^{\text{th}}$  weight in  $\mathbf{a}_\epsilon$  decreases exponentially the further  $z_i = \langle \mathbf{v}_i, -\mathbf{x}^* \rangle$  is from the maximum value  $\sigma_{\mathcal{A}}(-\mathbf{x}^*)$ . This maximum value is attained by the witness point of the considered shape  $\mathcal{A}$ . We illustrate in Fig. 2 the weighting of this soft-max operation on a mesh. Remarkably, the further a vertex is from the current witness point, the less it contributes to the softmax. It is, therefore, only necessary to keep the points of the mesh which belong to a neighborhood around the witness point of shape  $\mathcal{A}$ . This fact renders the use of the Gumbel distribution very efficient on meshes. By choosing the *depth* of neighboring vertices around the witness point, which we denote by  $n_l$ , we can

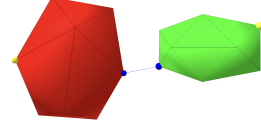


Fig. 4: **Rough shapes, goal: find a relative pose such that the yellow points are contact points.** The current witness points are the blue points.

choose to limit or increase the number of neighbors involved in the computation of (27). For example, a depth of  $n_l = 2$  corresponds to keeping only the neighbors of the witness points and the neighbors of neighbors.

Finally, by applying the chain rule, we get the estimation of  $\partial^2 \sigma_{\mathcal{A}}(\mathbf{x})/\partial \mathbf{x}^2$ :

$$\frac{\partial^2 \sigma_{\mathcal{A}}(\mathbf{x})}{\partial \mathbf{x}^2} \approx \frac{\partial \mathbf{a}_\epsilon(\mathbf{z}(\mathbf{x}))}{\partial \mathbf{x}} = V \frac{\partial \mathbf{a}_\epsilon(\mathbf{z})}{\partial \mathbf{z}} V^\top, \quad (28)$$

where  $\partial \mathbf{a}_\epsilon(\mathbf{z})/\partial \mathbf{z}$  is simply the derivative of the soft-max function.

**Derivatives of the witness points.** In general,  $\mathbf{x}_1^* = \nabla \sigma_{\mathcal{A}_1}(-\mathbf{x}^*)$  so by applying the chain rule we get:

$$\frac{\partial \mathbf{x}_{1,2}^*}{\partial \mathbf{q}} = -\frac{\partial \nabla \sigma_{\mathcal{A}_{1,2}}(-\mathbf{x}^*)}{\partial \mathbf{x}^*} \frac{\partial \mathbf{x}^*}{\partial \mathbf{q}} = -\frac{\partial^2 \sigma_{\mathcal{A}_{1,2}}(-\mathbf{x}^*)}{\partial \mathbf{x}^{*2}} \frac{\partial \mathbf{x}^*}{\partial \mathbf{q}}. \quad (29)$$

To conclude this section, we have introduced a complete approach to retrieve a 1<sup>st</sup>-order estimate of the variation of the witness points lying on the two shapes, according to the relative placements between these two. In particular, in the case of meshes, we have proposed a closed-form formula to compute a local approximation of the Hessian using the neighborhood of the current witness points.

## V. EXPERIMENTS

In this section, we evaluate whether the gradients obtained with the two proposed estimators are meaningful and how computationally efficient it is to compute them compared to finite differences.

**Contact-pose generation benchmark.** To answer the first question, we evaluate the 0<sup>th</sup>-order and 1<sup>st</sup>-order estimators against finite differences on a synthetic benchmark of non-trivial minimization problems. We generate 100 collision pairs with random polyhedral ellipsoids, i.e., ellipsoids whose surfaces are represented by a 12-vertices convex mesh (see Fig. 4). The resulting shapes are rough, i.e., the curvature information of the original shape has been greatly truncated. For each pair of convex shapes  $(\mathcal{A}_1, \mathcal{A}_2)$ , we generate 100 random target points  $\mathbf{x}_{1,\text{des}} \in \mathcal{A}_1$  and  $\mathbf{x}_{2,\text{des}} \in \mathcal{A}_2$  on the shapes' surfaces. For each of the 10000 generated problems, the goal is to find a relative pose  $T(\mathbf{q})$  between  $\mathcal{A}_1$  and  $\mathcal{A}_2$  such that the shapes are in contact and their witness points  $\mathbf{x}_1^*(\mathbf{q})$  and  $\mathbf{x}_2^*(\mathbf{q})$  satisfy  $\mathbf{x}_1^*(\mathbf{q}) = \mathbf{x}_{1,\text{des}}$  and  $\mathbf{x}_2^*(\mathbf{q}) = \mathbf{x}_{2,\text{des}}$ . Mathematically, this corresponds to solving the minimization problem:

$$\min_{\mathbf{q}} \frac{1}{2} \sum_{i=1,2} \|\mathbf{x}_i^*(\mathbf{q}) - \mathbf{x}_{i,\text{des}}^*\|^2 + \frac{1}{2} \|\mathbf{x}_1^*(\mathbf{q}) - \mathbf{x}_2^*(\mathbf{q})\|^2. \quad (30)$$

	Finite differences		0 <sup>th</sup> -order Gaussian	1 <sup>st</sup> -order Gaussian	1 <sup>st</sup> -order Gumbel
	$M = 12$ $\epsilon = 10^{-6}$	$M = 12$ $\epsilon = 10^{-3}$	$M = 50$ $\epsilon = 10^{-2}$	$M = 20$ $\epsilon = 10^{-3}$	$n_l = 1$ $\epsilon = 10^{-4}$
D1	$2 \times 10^{-33}$	$8 \times 10^{-33}$	$4 \times 10^{-23}$	$4 \times 10^{-16}$	$6 \times 10^{-16}$
Q1	$4 \times 10^{-32}$	$7 \times 10^{-23}$	$5 \times 10^{-20}$	$1 \times 10^{-10}$	$3 \times 10^{-10}$
Median	$3 \times 10^{-3}$	$4 \times 10^{-14}$	$2 \times 10^{-13}$	$4 \times 10^{-8}$	$1 \times 10^{-8}$
Q3	$4 \times 10^{-2}$	$1 \times 10^{-2}$	$2 \times 10^{-3}$	$3 \times 10^{-7}$	$7 \times 10^{-8}$
D9	$8 \times 10^{-2}$	$5 \times 10^{-2}$	$5 \times 10^{-3}$	$2 \times 10^{-6}$	$2 \times 10^{-5}$

TABLE I: Rough shapes (see Fig. 4), value of  $C(q)$  after 50 iterations of Gauss-Newton with line search. Q3 and D9: respectively 25% and 10% of problems have a terminal cost worse (higher) than the reported value.

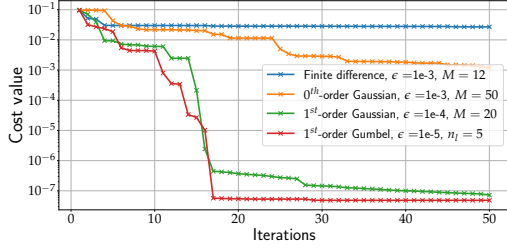


Fig. 5: YCB shape, goal: find a relative pose such that the yellow points are contact points. The finite difference typically gets stuck around a limited precision (depending on the finite difference increment). The 1<sup>st</sup>-order estimator converges rapidly towards a solution with high precision, unlike the 0<sup>th</sup>-order estimator.

To solve this minimization problem, we use the Gauss-Newton algorithm with backtracking line search [25] and run it for 50 iterations. To compute the Jacobian of the cost, we evaluate the terms  $\partial x_{1,2}^*/\partial q$ , with finite differences, and the 0<sup>th</sup> and 1<sup>st</sup>-order estimators proposed in this work. Finally, to compute  $x_1^*(q)$  and  $x_2^*(q)$ , we use the combination of the GJK and EPA algorithms implemented in the HPP-FCL library [26], [27], a fork of the FCL library [28].

We report as quantiles in Table I the value of the terminal cost  $C(q)$ . The lower this quantity, the better the quality of the solution found. We are particularly interested in the quantiles Q3, and D9 of Table I: respectively 25% and 10% of problems have a terminal cost worse (higher) than the reported value. The higher this value is, the less reliable the method is. We observe that finite differences have a high value for Q3 and D9, whereas the two proposed estimators are at least one order of magnitude better. Remarkably, the 1<sup>st</sup>-order estimators, whether the underlying distribution used is Gaussian or Gumbel, are extremely accurate and reliable, with a value of Q3 and D9 at least three orders of magnitude better than finite differences.

As an additional qualitative example, Fig. 3 and Fig. 5 show a typical failure example of finite differences on a collision pair of the YCB dataset - a dataset which contains high-resolution meshes of real-world household objects [29]. Finally, Fig. 6 shows the typical impact of the noise and number of samples on the 1<sup>st</sup>-order estimator using the Gumbel distribution for the same YCB problem. The same kind of behavior is obtained when using a Gaussian distribution. Overall, the higher the number of samples, the better the quality of the estimator; the higher the noise, the faster the convergence at the price of reduced accuracy.

**YCB timings benchmarks.** To evaluate the computational efficiency of the proposed estimators, we generate 10000

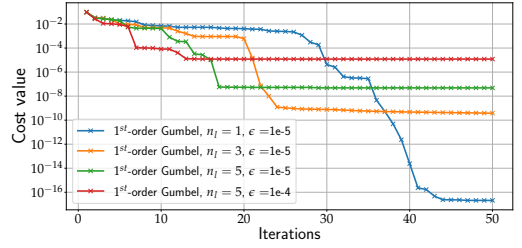


Fig. 6: Goal: find a relative pose such that the yellow points are contact points. With more noise, the faster the convergence is at first, but the estimator ends up being less precise, hence converging to a less optimal solution.

Method	Parameters	Timings (in $\mu s$ )	
		Collision	No collision
Finite differences	$M = 12$	$78 \pm 48$	$11.0 \pm 7.3$
	$M = 10$	$65 \pm 36$	$9 \pm 6$
0 <sup>th</sup> -order Gaussian	$M = 20$	$122 \pm 130$	$17 \pm 10$
	$M = 50$	$338 \pm 391$	$47 \pm 30$
	$M = 100$	$613 \pm 402$	$86 \pm 45$
1 <sup>st</sup> -order Gaussian	$M = 10$	$4.8 \pm 1.4$	$3.1 \pm 0.3$
	$M = 20$	$8.9 \pm 2.8$	$5.8 \pm 0.5$
	$M = 50$	$22 \pm 8$	$15 \pm 3$
	$M = 100$	$42 \pm 13$	$27 \pm 3$
1 <sup>st</sup> -order Gumbel	$n_l = 1$	$1.7 \pm 0.5$	$1.6 \pm 0.5$
	$n_l = 3$	$4.1 \pm 1.8$	$3.9 \pm 1.4$
	$n_l = 5$	$9.6 \pm 7.8$	$9.7 \pm 8.1$

TABLE II: Timings for computing collision detection derivatives, for collision pairs of the YCB dataset. The parameters were selected in the ranges typically used in practice.

collision detection problems (1) using meshes from the YCB real-world objects dataset [29] and measure the time taken to compute the derivatives of witness points for each estimator. For the finite differences and 0<sup>th</sup>-order, GJK+EPA are warm started at each sample  $q + \epsilon z$  to enhance the computational efficiency. The results, reported in Table II show that although the 0<sup>th</sup>-order estimator is often prohibitive compared to finite differences, the 1<sup>st</sup>-order estimators using Gaussian and Gumbel distributions can be obtained extremely efficiently, on the order of the micro-seconds and from 10 to 70 times faster than finite differences.

## VI. CONCLUSION

In this paper, we propose a generic approach for computing 0<sup>th</sup> and 1<sup>st</sup>-order derivatives of collision detection for any convex shapes by leveraging randomized smoothing techniques. While being robust and easy to implement, our approach exhibits strong benefits in terms of speed and accuracy, taking only few micro-seconds to compute informative derivatives of complex shapes, such as meshes with hundred vertices, involved in real robotic applications. Remarkably, the 1<sup>st</sup>-order estimators are also both more accurate and less expensive to compute than the 0<sup>th</sup>-order estimator. All these gradient estimation methods have been implemented in the HPP-FCL and Pinocchio ecosystems. We plan to extend our contributions by applying these methods for differentiable simulation, optimal grasp synthesis and trajectory optimization.

## REFERENCES

- [1] C. Ericson, *Real-Time Collision Detection*. The Morgan Kaufmann Series, 2004.
- [2] F. de Avila Belbute-Peres, K. Smith, K. Allen, J. Tenenbaum, and J. Z. Kolter, “End-to-end differentiable physics for learning and control,” *Advances in neural information processing systems*, vol. 31, 2018.
- [3] J. Carpentier and N. Mansard, “Analytical derivatives of rigid body dynamics algorithms,” in *Robotics: Science and systems (RSS 2018)*, 2018.
- [4] M. Toussaint, K. Allen, K. Smith, and J. Tenenbaum, “Differentiable Physics and Stable Modes for Tool-Use and Manipulation Planning,” in *Robotics: Science and Systems XIV*, Robotics: Science and Systems Foundation, June 2018.
- [5] Q. Le Lidec, I. Kalevatykh, I. Laptev, C. Schmid, and J. Carpentier, “Differentiable simulation for physical system identification,” *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3413–3420, 2021.
- [6] M. Geilinger, D. Hahn, J. Zehnder, M. Bächer, B. Thomaszewski, and S. Coros, “ADD: Analytically Differentiable Dynamics for Multi-Body Systems with Frictional Contact,” *arXiv:2007.00987 [cs]*, July 2020.
- [7] K. Werling, D. Omens, J. Lee, I. Exarchos, and C. K. Liu, “Fast and Feature-Complete Differentiable Physics for Articulated Rigid Bodies with Contact,” *arXiv:2103.16021 [cs, eess]*, June 2021.
- [8] A. Escande, S. Miossec, M. Benallegue, and A. Kheddar, “A strictly convex hull for computing proximity distances with continuous gradients,” *IEEE Transactions on Robotics*, 2014.
- [9] Q. Berthet, M. Blondel, O. Teboul, M. Cuturi, J.-P. Vert, and F. Bach, “Learning with differentiable perturbed optimizers,” in *Advances in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, eds.), vol. 33, pp. 9508–9519, Curran Associates, Inc., 2020.
- [10] J. Carpentier, G. Saurel, G. Buondonno, J. Mirabel, F. Lamiriaux, O. Stasse, and N. Mansard, “The Pinocchio C++ library – A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives,” in *IEEE International Symposium on System Integrations (SII)*, 2019.
- [11] E. Gilbert, D. Johnson, and S. Keerthi, “A fast procedure for computing the distance between complex objects in three-dimensional space,” *IEEE Journal on Robotics and Automation*, 1988.
- [12] G. Van den Bergen, “Proximity Queries and Penetration Depth Computation on 3D Game Objects,” in *Game Developers Conference*, 2001.
- [13] L. Montaut, Q. Lidec, V. Petrik, J. Sivic, and J. Carpentier, “Collision Detection Accelerated: An Optimization Perspective,” in *Proceedings of Robotics: Science and Systems*, (New York City, NY, USA), June 2022.
- [14] K. Mamou and F. Ghorbel, “A Simple and Efficient Approach for 3D Mesh Approximate Convex Decomposition,” in *The 16th IEEE International Conference on Image Processing*, 2009.
- [15] G. Snethen, “Xenocollide: Complex collision made simple,” in *Game Programmings Gems*, 2008.
- [16] J. Solà, J. Deray, and D. Atchuthan, “A micro Lie theory for state estimation in robotics,” *CoRR*, 2021.
- [17] J. Matyas *et al.*, “Random optimization,” *Automation and Remote control*, vol. 26, no. 2, pp. 246–253, 1965.
- [18] B. Polyak, *Introduction to Optimization*. Springer, 07 1987.
- [19] J. C. Duchi, P. L. Bartlett, and M. J. Wainwright, “Randomized smoothing for stochastic optimization,” *SIAM Journal on Optimization*, vol. 22, no. 2, pp. 674–701, 2012.
- [20] Q. Le Lidec, I. Laptev, C. Schmid, and J. Carpentier, “Differentiable rendering with perturbed optimizers,” *Advances in Neural Information Processing Systems*, vol. 34, pp. 20398–20409, 2021.
- [21] F. Petersen, B. Goldluecke, C. Borgelt, and O. Deussen, “Gendr: A generalized differentiable renderer,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4002–4011, 2022.
- [22] Q. L. Lidec, L. Montaut, C. Schmid, I. Laptev, and J. Carpentier, “Leveraging randomized smoothing for optimal control of nonsmooth dynamical systems,” *arXiv preprint arXiv:2203.03986*, 2022.
- [23] H. J. T. Suh, T. Pang, and R. Tedrake, “Bundled gradients through contact via randomized smoothing,” *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4000–4007, 2022.
- [24] E. J. Gumbel, *Statistical theory of extreme values and some practical applications: a series of lectures*, vol. 33. US Government Printing Office, 1954.
- [25] S. Wright, J. Nocedal, *et al.*, “Numerical optimization,” *Springer Science*, vol. 35, no. 67-68, p. 7, 1999.
- [26] J. Mirabel, S. Tonneau, P. Fernbach, A.-K. Seppälä, M. Campana, N. Mansard, and F. Lamiriaux, “HPP: A new software for constrained motion planning,” in *International Conference on Intelligent Robots and Systems*, 2016.
- [27] J. Pan, S. Chitta, D. Manocha, F. Lamiriaux, J. Mirabel, J. Carpentier, *et al.*, “HPP-FCL: an extension of the Flexible Collision Library.” <https://github.com/humanoid-path-planner/hpp-fcl>, 2015–2022.
- [28] J. Pan, S. Chitta, and D. Manocha, “FCL: A General Purpose Library for Collision and Proximity Queries,” in *2012 IEEE International Conference on Robotics and Automation*, IEEE, 2012.
- [29] B. Calli, A. Singh, A. Walsman, S. Srinivasa, P. Abbeel, and A. M. Dollar, “The ycb object and model set: Towards common benchmarks for manipulation research,” in *2015 international conference on advanced robotics (ICAR)*, pp. 510–517, IEEE, 2015.

# Contact Models in Robotics: a Comparative Analysis

Quentin Le Lidec<sup>1,†</sup>, Wilson Jallet<sup>1,2</sup>, Louis Montaut<sup>1,3</sup>, Ivan Laptev<sup>1</sup>, Cordelia Schmid<sup>1</sup>, and Justin Carpentier<sup>1</sup>

**Abstract**—Physics simulation is ubiquitous in robotics. Whether in model-based approaches (e.g., trajectory optimization), or model-free algorithms (e.g., reinforcement learning), physics simulators are a central component of modern control pipelines in robotics. Over the past decades, several robotic simulators have been developed, each with dedicated contact modeling assumptions and algorithmic solutions. In this article, we survey the main contact models and the associated numerical methods commonly used in robotics for simulating advanced robot motions involving contact interactions. In particular, we recall the physical laws underlying contacts and friction (i.e., Signorini condition, Coulomb’s law, and the maximum dissipation principle), and how they are transcribed in current simulators. For each physics engine, we expose their inherent physical relaxations along with their limitations due to the numerical techniques employed. Based on our study, we propose theoretically grounded quantitative criteria on which we build benchmarks assessing both the physical and computational aspects of simulation. We support our work with an open-source and efficient C++ implementation of the existing algorithmic variations. Our results demonstrate that some approximations or algorithms commonly used in robotics can severely widen the reality gap and impact target applications. We hope this work will help motivate the development of new contact models, contact solvers, and robotic simulators in general, at the root of recent progress in motion generation in robotics.

**Index Terms**—Physical simulation, Numerical optimization.

## I. INTRODUCTION

**S**IMULATION is a fundamental tool in robotics. Control algorithms, like trajectory optimization (TO) or model predictive control (MPC) algorithms, rely on physics simulators to evaluate the dynamics of the controlled system. Reinforcement Learning (RL) algorithms operate by trial and error and require a simulator to avoid time-consuming and costly failures on real hardware. Robot co-design aims at finding optimal hardware design and morphology and thus extensively rely on simulation to prevent tedious physical validation. In practice, roboticists also usually perform simulated safety checks before running a new controller on their robots. These applications are evidence for a wide range of research areas in robotics where simulation is critical.

To be effective and valuable in practice, robot simulators must meet some fidelity or efficiency levels, depending on the use case. For instance, trajectory optimization algorithms, e.g. iLQR[1] or DDP [2], [3], use physics simulation to evaluate the

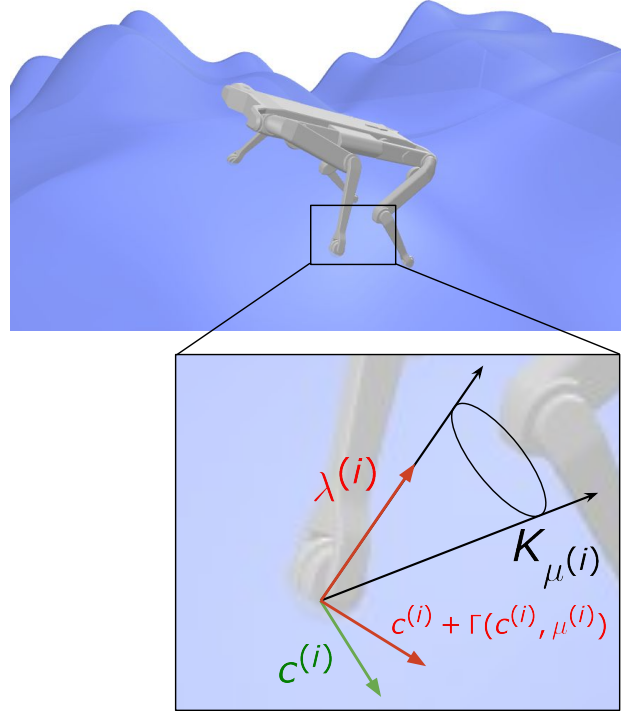


Fig. 1. **Illustration of the dynamics of frictional contacts** between rigid bodies which are governed by the Signorini condition, the Coulomb’s law, and the maximum dissipation principle. The combination of these three principles leads to the Non-linear Complementarity Problem (14).

system dynamics and leverage finite differences or the recent advent of differentiable simulators [4], [5], [6], [7], [8] to compute derivatives. If the solution lacks precision, the real and planned trajectories may quickly diverge, impacting *de facto* the capacity of such control solutions to be deployed on real hardware. To absorb such errors, the Model Predictive Control (MPC) [9], [10] control paradigm exploits state feedback by repeatedly running Optimal Control (OC) algorithms at high-frequency rates (e.g., 1kHz) [11], [12]. The frequency rate is one factor determining the robustness of this closed-loop algorithm to modeling errors and perturbations; thus, the efficiency of the simulation becomes critical. Although RL [13] is considered as a model-free approach, physical models are still at work to generate the samples that are indispensable for learning control policies. In fact, the vast number of required samples is the main bottleneck during training, as days or years of simulation, which corresponds to billions of calls to a simulator, are necessary [14], [15], [16]. Therefore, the efficiency of the simulator directly determines

<sup>1</sup>Inria - Département d’Informatique de l’École normale supérieure, PSL Research University. Email: [firstname.lastname@inria.fr](mailto:firstname.lastname@inria.fr)

<sup>2</sup>LAAS-CNRS, 7 av. du Colonel Roche, 31400 Toulouse

<sup>3</sup>Czech Institute of Informatics, Robotics and Cybernetics, Czech Technical University, Prague, Czech Republic

<sup>†</sup>Corresponding author

the computational and, thus, the energetic cost of learning a control policy. Physical accuracy plays an important role after training as well, as more physically accurate simulations will result in a smaller reality-gap to cross for the learned policy to transfer to a real robot [14].

Many manipulation tasks can be tackled by assuming quasi-staticity and by considering only a restricted variety of contact events [17], [18]. The recent robotics efforts, highlighted, for instance, by the athletic motions performed by the humanoid robots of Boston Dynamics [19], focus on very dynamic tasks for which these simplification hypotheses cannot hold. In fact, tasks like agile locomotion or dexterous manipulation require the robot to quickly plan and finely exploit, at best, the contact interactions with its environment to shape the movements [20], [21], [22]. In this respect, the ability to handle impacts and frictions, physical phenomena at the core of contact interactions, becomes fundamental for robotic simulators.

Physics simulation is often considered a solved problem with several well-known simulators which are available off-the-shelf. However, simulating a physical system raises several and complex issues that are usually circumvented at the cost of approximations or costly computation. When simulating a system evolving freely, rigid body dynamics algorithms [23], [24] are now established as the way to go due to their high efficiency. For robotics, one has to consider interactions through contact between the robot and its environment, thus constraining the movement. However, due to the creation or the breaking of contacts along a trajectory, the dynamics switch from one mode to the other, making the problem of simulating a system with contacts and frictions highly non-smooth and non-convex [25], [26], [27]. More precisely, contact dynamics between rigid objects are governed by three main principles: the Signorini condition specifies the unilaterality nature of contact interactions, while the Coulomb’s law of friction and the maximum dissipation principle (MDP) of Moreau state that friction force should lie inside a second-order cone and oppose the movement.

Altogether, these three principles correspond to a so-called nonlinear complementarity problem (NCP), which is nonconvex and thus difficult to solve in general [28]. Numerical methods to solve this NCP fall into two main categories: event-driven and time-stepping methods [28]. Most modern robotics simulators are part of the latter category because predicting collisions is intractable due to the complexity of the scenes. Therefore, we will restrain our study to this type of method. Historically, the Open Dynamic Engine (ODE) [29] is one of the first open-source simulators with a large impact on the community, which was then followed by Bullet [30]. Both of them rely on maximal coordinates to depict the state of the objects and kinematic constraints imposed by the articulations are tackled explicitly. Such a choice leads to large-dimensional problems to solve, impacting *de facto* the computational performances. To lower the computational burden, alternative simulators rooted in generalized coordinates, like DART [31] and MuJoCo [32], appeared shortly after. More recently, RaiSim [33] emerged as one of the first simulators enabling RL policies to transfer to real quadrupedal robots. Its implementation being closed source, we provide what constitutes, to the best of our knowledge,

the first in-depth study and open-source re-implementation of this contact solver. Still today, the number of alternative algorithms available is growing fast [34], [6], [8]. In general, these simulators differ at their very core: one should be aware of the contact modeling embedded in the simulator they are using and how it can impact the applications they aim at. Some high-level benchmarks of simulators exist [35], evaluating the whole simulation pipeline and its multiple internal routines, *e.g.* rigid-body dynamics algorithms, collision detection, and contact problem-solving. Our work closely relates to [36]. It separately assesses the various contact models and their associated algorithms. We achieve this by decoupling the contact models from their implementations and re-implemented the solvers with a unique back-end based on the Pinocchio toolbox [24], [37] for evaluating the dynamic quantities and on HPP-FCL [38], [39], [40] for computing the collisions. We pursue the effort of [36] by studying recent algorithms and adding advanced evaluation criteria. Our experiments are done in both illustrative and realistic robotics setups.

We made the following contributions:

- we make a detailed survey of contact models and their associated algorithms, including established and more recent robotics simulators;
- we expose the main limitations of existing simulators by inspecting both the physical approximations and the numerical methods that are at work;
- we develop an open source and generic implementation of the main robotic contact solvers in C++;
- based on our implementation and the theoretical study, we propose quantitative criteria which allow performing an in-depth evaluation of both physical and computational aspects of contact models and solvers.
- we explore the impacts of the simulation choices on the practical application of MPC for quadruped locomotion.

The article is organized as follows: we first recall the background of contact simulation: the physical principles behind contact modeling (Sec. II) and the numerical algorithms allowing us to solve the resulting equations (Sec. III). In the experimental part (Sec. IV), we propose an exhaustive empirical evaluation of the various existing contact models and solvers to assess both their physicality (Sec. IV-A) and computational efficiency (Sec. IV-C). At last, Sec. IV-D investigates the consequences of the contact models in the context of quadruped locomotion. It is finally worth mentioning that the authors are linked to the Pinocchio and HPP-FCL open-source projects.

## II. RIGID CONTACT MODELLING

We start by stating the physical principles commonly admitted for rigid body simulation with point contact. If these principles remain hypothetical and can still be discussed, they have been, in general, empirically tested and are arguably better than their relaxations. Once the modeling is done, we transcribe these physical laws into a numerical problem, which should be solved via optimization-based techniques to simulate a system with contacts and frictions. We also present the various open-source tools that allow computing all the intermediate quantities necessary to build a physics simulator.



In this paper, we describe the state of a system with its generalized coordinates  $q \in \mathcal{Q} \cong \mathbb{R}^{n_q}$ . We denote by  $v \in \mathcal{TQ} = \mathbb{R}^{n_v}$  the joint velocity.

**Free motion.** The principle of least action states that the path followed by a dynamical system should minimize the action functional [41]. This principle induces the celebrated Lagrangian equations of motion:

$$M(q)\dot{v} + C(q, v)v + g(q) = \tau \quad (1)$$

where  $M \in \mathbb{R}^{n_v \times n_v}$  represents the joint space inertia matrix of the system,  $C(q, v)v$  accounts for the centrifugal and Coriolis effects, and  $g$  is the generalized gravity. This Lagrangian equation of motion naturally accounts for the kinematic constraints induced by the articulations of the rigid-body dynamical system. When applied to a robot, *i.e.*, a system of multiple rigid bodies, the inertia matrix  $M$  becomes sparse. Rigid body dynamic algorithms exploit this sparsity at best [23], [24] making it possible to compute the free acceleration in less than  $4\mu s$  for the robots as complex as a 36-dof humanoid. As done by time-stepping approaches [28], we will express the problem in terms of velocities rather than acceleration, thus discretizing (1) into:

$$Mv^{t+1} = Mv^t + (\tau - Cv - g)\Delta t. \quad (2)$$

We note  $v^f$  the free velocity which is defined as the solution of (2).

**Bilateral contact.** When the system is subject to constraints, *e.g.* kinematic loop closures or anchor points, it is convenient to represent them implicitly:

$$\Phi(q) = 0. \quad (3)$$

where  $\Phi : \mathbb{R}^{n_q} \mapsto \mathbb{R}^m$  is the implicit constraint function of dimension  $m$ , which depends on the nature of the constraint. For resolution, it is more practical to proceed to an index reduction [42] by deriving w.r.t. time (3) to express it as a constraint on joints velocities:

$$c - c^* = 0. \quad (4)$$

where  $c = Jv^{t+1}$  is the constraint velocity,  $J = \partial\Phi/\partial q$  is the constraint Jacobian which can be computed efficiently via the rigid body dynamics algorithms [23], [43] and  $c^*$  is the reference velocity of the constraint. Such a constraint (7) is enforced by the action of the environment on the system via the contact vector impulse  $\lambda \in \mathbb{R}^m$ . These considerations lead to Gauss's principle of least constraint [44], [45], *i.e.* the acceleration of a constrained physical system is as similar as possible to that of the unconstrained system. By duality, the contact impulses are spanned by the adjoint of the constraint Jacobian and should be incorporated in the Lagrangian equations (2) via:

$$Mv^{t+1} = Mv_f + J^\top \lambda \quad (5)$$

Regarding bilateral contacts, the contact efforts, corresponding to the Lagrange multipliers associated with the constraint (3), are unconstrained. If a bilateral constraint is well

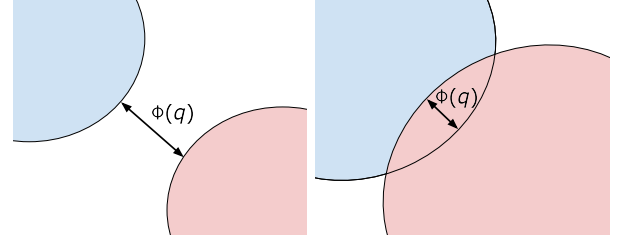


Fig. 2. **The separation vector  $\Phi$**  allows to formulate the non-penetration constraint which leads to the *Signorini condition* (8). This vector is computed by the GJK or EPA algorithms which are internal blocks of the simulator. We refer to [46] for a tutorial introduction on the topic.

suited to model kinematic closures, it is not to model interactions between the robot and its environment, which are better represented by unilateral contacts. This paper focuses on the latter, for which we provide a more detailed presentation.

**Unilateral contact.** When a system is in contact with its environment, the rigid body hypothesis enforces the normal component of the separation vector between the two objects [40], *i.e.* the signed distance function, to be non-negative. Expanding the bilateral case, the constraint function  $\Phi$  now coincides with the separation vector (Fig. 2) and describes a unilateral constraint:

$$\Phi(q)_N \geq 0 \quad (6)$$

where  $\Phi(q) \in \mathbb{R}^{3n_c}$ ,  $n_c$  is the number of contacts and the subscripts  $N$  and  $T$  respectively account for the normal and tangential components. In practice,  $\Phi$  can be computed efficiently via the Gilbert-Johnson-Keerthi (GJK) [47], [40] and the Expanding Polytope Algorithm (EPA) algorithms[48]. In order to ease the resolution, one can write (6) in terms of velocities, and the Taylor expansion of this condition leads to:

$$c_N - c_N^* \geq 0 \quad (7)$$

where  $c = Jv^{t+1} \in \mathbb{R}^{3n_c}$  is the velocity of contact points. We explain later how  $c_N^*$  is set to model physical effects or improve the numerical accuracy of the solutions. As in the bilateral case, the adjoint operator of the contact Jacobian  $J$  spans the contact forces, which leads again to (5) the constrained equations of motion. Unlike the bilateral case, unilateral contacts constrain the possible efforts  $\lambda$ . In a frictionless situation, the tangential forces are null, which implies that  $\lambda_T = 0$ . In addition, the contact forces  $\lambda$  can only be repulsive as they should not act in a glue-like fashion (the environment can only push and not pull on the feet of a legged robot) and, thus, are forced to be non-negative. An impulse cannot occur when an object takes off, *i.e.*, the normal velocity and impulse cannot be non-null simultaneously. Combining these conditions, we obtain the so called *Signorini condition* [49]:

$$0 \leq \lambda_N \perp c_N - c_N^* \geq 0. \quad (8)$$

However, such a condition does not define a mapping between  $\lambda_N$  and  $c_N$ , *i.e.* the contact forces are not a function of the penetration error. Indeed, its representation is an infinitely steep graph that may be relaxed into a mapping via a spring

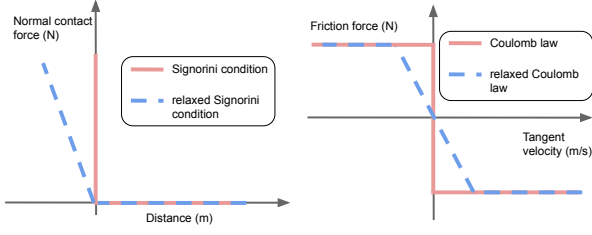


Fig. 3. Both the *Signorini condition* (Left) and the *Coulomb's law* (Right) induce infinitely steep graphs, which make the contact problem hard to solve.

damper accounting for local deformation of the materials (see Fig. 3). Substituting,  $v^{t+1}$  by its expression from the Lagrangian equations (5), we obtain a Linear Complementarity Problem (LCP) [50]:

$$0 \leq \lambda_N \perp (G\lambda + g)_N - c_N^* \geq 0 \quad (9)$$

where  $G = JM^{-1}J^\top$  is the so-called Delassus matrix, and  $g = Jv^f$  is the free velocity of contact points (the velocity of the contact points in the unconstrained cases). It is worth mentioning at this stage that several approaches [23], [51], [43] have been developed in the computational dynamics and robotics literature to efficiently evaluate the Delassus matrix.

In the case of rigid bodies - purely inelastic impacts, and exact collision detection, *i.e.*  $\Phi(q)_N = 0$  - the reference velocity  $c_N$  can be set to 0. However, setting this velocity to a non-null quantity may be useful to improve the modeling on both physical and numerical aspects. A first benefit is a possibility of accounting for impacts that may occur when two objects collide with non-null relative normal velocity. The most common impact law stipulates  $c^* = -ec^t$  where  $e$  is the restitution coefficient which adjusts the quantity of energy dissipated during the collision. Another benefit is when time stepping methods are employed, one cannot avoid penetration errors, *i.e.*  $\Phi(q)_N < 0$ , but it is possible to prevent these errors from growing over time via a Baumgarte correction [52] which sets  $c^* = K \max(0, -\Phi(q)_N)$ . In addition, in most cases in robotics, the Delassus' matrix  $G$  is rank deficient. Such physical systems are said to be hyperstatic, and because  $\text{rank}(J) > n_v$ , several  $\lambda$  values may lead to the same trajectory. This under-determination can be circumvented by relaxing the rigid-body hypothesis, *i.e.* the *Signorini condition*, and considering compliant contacts via a reference velocity linearly depending on  $\lambda$  as represented in Fig. 3. Indeed, with  $c^* = -R\lambda$  where  $R$  is a diagonal matrix with non-null and positive elements only on the normal components, called compliance and whose value is a property of the material, the original Delassus matrix  $G$  is replaced by the damped matrix  $\tilde{G} = G + R$  which is full rank.

**Friction phenomena** are at the core of contact modeling as they precisely allow manipulation or locomotion tasks. Coulomb's law for dry friction represents the most common way to model friction forces. This phenomenological law states that the friction forces  $\lambda_T$  should be proportional to the normal contact forces  $\lambda_N$  and the friction coefficient  $\mu$ . Mathematically, this

suggests that contact forces should lie inside an ice cream cone whose aperture is set by the coefficient of friction  $\mu$ :

$$\lambda \in K_\mu = \prod_{i=1}^{n_c} K_{\mu^{(i)}} \quad (10)$$

where the superscript  $(i)$  refers to the  $i^{th}$  contact point and  $K_{\mu^{(i)}} = \{\lambda | \lambda_N \geq 0, \|\lambda_T\|_2 \leq \mu^{(i)} \lambda_N\}$ . Additionally, when sliding occurs, the maximum dissipation principle formulated by Jean-Jacques Moreau [25] implies that the frictional forces should maximize the dissipated power:

$$\forall i, \lambda_T^{(i)} = -\mu^{(i)} \lambda_N^{(i)} \frac{c_T^{(i)}}{\|c_T^{(i)}\|}, \text{ if } \|c_T^{(i)}\| > 0. \quad (11)$$

As for the *Signorini condition*, Coulomb's law does not describe a mapping but an infinitely steep graph (Fig. 3). Relaxing this law via viscous frictions, *i.e.* assuming the tangent contact forces to be proportional to the tangent velocities, allows defining a mapping between  $\lambda_T$  and  $c_T$ .

Combining the Coulomb's law for friction with the *Signorini condition* evoked earlier, we finally get three distinct cases corresponding to a sticking contact point (12a), a sliding contact point (12c) or a take-off (12b):

$$\begin{cases} \lambda^{(i)} \in K_{\mu^{(i)}}, \text{ if } c^{(i)} = 0 & (12a) \\ \lambda^{(i)} = 0, \text{ if } c_N^{(i)} > 0 & (12b) \\ \lambda^{(i)} \in \partial K_{\mu^{(i)}}, \exists \alpha > 0, \lambda_T^{(i)} = -\alpha c_T^{(i)} \text{ otherwise.} & (12c) \end{cases}$$

where  $\partial K$  indicates the border of the cone. The equations (12) are referred to as the disjunctive formulation of the contact problem. However, such a formulation is not suited for solving as the switching condition depends on the contact point velocity  $c$ . As this quantity is an unknown of the problem, one cannot know in which case of (12) they are standing. For this reason, the problem is often reformulated as a nonlinear complementarity problem (NCP). Indeed, using de Saxcé's bipotential function [53] defined as:

$$\Gamma : (c, \mu) \in \mathbb{R}^3 \times \mathbb{R} \mapsto [0, 0, \mu \|c_T\|_2] \quad (13)$$

one can show that (12) is equivalent to the following [27], [54] (Fig. 1):

$$\forall i, K_{\mu^{(i)}} \ni \lambda^{(i)} \perp c^{(i)} + \Gamma(c^{(i)}, \mu^{(i)}) \in K_{\mu^{(i)}}^*. \quad (14)$$

In (14),  $K_\mu^*$  refers to the dual cone of  $K_\mu$  such that if  $\lambda \in K_\mu$  and  $c \in K_\mu^*$ , then  $\langle \lambda, c \rangle \geq 0$ , where  $\langle \cdot, \cdot \rangle$  is the canonical scalar product.

Eq. (14) allows to define, for each contact  $i \in \llbracket 1, n_c \rrbracket$ , the primal and dual residuals as  $\epsilon_p^{(i)} = \text{dist}_{K_{\mu^{(i)}}}(\lambda^{(i)})$  and  $\epsilon_d^{(i)} = \text{dist}_{K_{\mu^{(i)}}^*}(c^{(i)} + \Gamma(c^{(i)}, \mu^{(i)}))$  respectively, where  $\text{dist}_C$  is the distance function w.r.t. a convex set  $C$ . It also induces a contact complementarity criterion  $\epsilon_c^{(i)} = |\langle \lambda^{(i)}, c^{(i)} + \Gamma(c^{(i)}, \mu^{(i)}) \rangle|$ . From these per-contact criteria, it is then possible to introduce a well-posed absolute convergence criterion  $\epsilon_{\text{abs}}$  for (14), as the maximum of  $\epsilon_p^{(i)}, \epsilon_d^{(i)}$  and  $\epsilon_c^{(i)}$  for all  $i$ . We use this criterion as a stopping criterion in our implementation of NCP solvers, but also as a measure of physical accuracy in our experiments of Section IV.

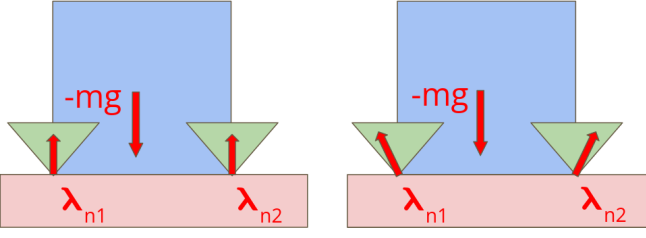


Fig. 4. **Underdetermined contact problem.** The left and right contact forces are solutions of the NCP (14) and lead to the same system velocity. Such an undetermined problem can also occur on normal forces.

At this point, it is worth mentioning that the problem (14), which we refer to as NCP, does not derive from a convex optimization problem, thus making its resolution complex. Alternatively, one can see the frictional contact problem as two interleaved convex optimization problems [34], [55], [6] whose unknowns,  $\lambda$  and  $v$ , also define each other problems. Practically, this can induce the existence of multiple, or even an infinite number of contact forces satisfying (14). As mentioned earlier, this can be due to normal forces, but tangential components can also cause under-determination (Fig. 4). Even though it does not impact the trajectories of rigid bodies, a simulator should not exhibit internal forces compressing or stretching the objects. Indeed, such forces would not coincide with the forces observed by force sensors and would rather correspond to some internal deformations of the objects, which should thus be considered soft and no more rigid. Additionally, these internal forces might also be problematic as it is impossible to characterize them. This may induce inconsistent derivatives, which become critical in the context of differentiable simulation.

**Open-source frameworks for contact simulation.** To conclude this section, we propose to review the main open-source software that has been developed by the robotics community and can be used for simulating contact. Simulating contact interactions, as illustrated in Fig. 5, involved two main stages, corresponding to the collision detection step (which shapes are in contacts) and the collision resolution (which contact forces are applied through the contact interaction). These frameworks are enumerated in Tab. I.

More precisely, at each time step, a simulator must first detect which geometries are colliding and compute their separation vector  $\Phi$ . The GJK and EPA algorithms are widely adopted for their low computational cost. HPP-FCL [38], [39], [40], the extension of Flexible Collision Library (FCL) [56] and libccd [57] implement them efficiently. Some simulators such as Bullet [30] or ODE [29] also re-implement the same algorithm as an internal routine.

Once collisions are evaluated, one still requires the contact points free velocity  $v_f$  and Jacobians  $J$  to formulate (14). These two quantities are efficiently computed via the rigid body algorithms [23]. The RBDL [58] or the Pinocchio library [24] provide efficient implementations to evaluate them. In addition, Pinocchio proposes a direct and robust way to compute Cholesky's decomposition of the Delassus' matrix  $G$  [43].

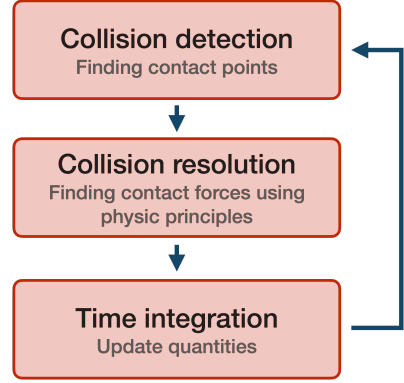


Fig. 5. **Simulation routines.** When simulating rigid bodies with frictional contacts, a physics engine goes through a suite of potentially challenging sub-problems: the collision detection, the collision resolution, and the time integration step.

TABLE I  
OPEN SOURCE TOOLS FOR PHYSICS SIMULATION IN ROBOTICS.

	License	API	Used by
<b>Collision detection</b>			
FCL [56]	BSD	C++	DART, Drake
libccd [57]	BSD	C++, Python	MuJoCo, Drake, FCL, Bullet, ODE
HPP-FCL [38]	BSD	C++, Python	Pinocchio
Bullet [30]	BSD	C++, Python	DART
ODE [29]	BSD/GPL	C++, Python	DART
<b>Rigid body dynamics algorithms</b>			
Pinocchio [24]	BSD	C++, Python	
RBDL [58]	zlib	C++, Python	
<b>Collision resolution</b>			
MuJoCo [32]	Apache 2.0	C++, Python	
DART [31]	BSD 2	C++, Python	
Bullet [30]	BSD	C++, Python	
Drake [59]	BSD 3	C++, Python	
ODE [59]	BSD/GPL	C++, Python	

These algorithms are also embedded as internal routines in various simulators such as MuJoCo [32], DART[31], Drake [59], Bullet [30] or ODE[29].

Eventually, when all quantities necessary to formulate the NCP (14) are computed, the simulator has to call a solver. Every simulator, *i.e.* MuJoCo[32], DART [31], Bullet[30], Drake[59] and ODE[29], proposes its own implementation. This procedure varies greatly depending on the physics engine, as each has its own physical and numerical choices. We detail in the next section the existing algorithms.

### III. ALGORITHMIC DERIVATIONS OF THE CONTACT PROBLEM

As explained in the previous section, the nonlinear complementarity problem (14) does not derive from a variational principle. Thus, classical numerical optimization solvers will not be of any help in solving it. This section studies the various approximations and algorithmic techniques in the literature to tackle this problem. As summarized in Tab. II, this section is organized in subsections describing the four contact models most commonly used in robotics, namely the linear complementarity problem (LCP), the cone complementary

TABLE II  
CHARACTERISTICS OF VARIOUS CONTACT MODELS.

	Signorini	Coulomb	MDP	No shift	No internal forces	Robust	Convergence guarantees
<b>LCP</b>							
PGS [30], [29], [60], [31]	✓			✓			✓
Staggered projections [34]	✓			✓	✓	✓	
<b>CCP</b>							
PGS [61]		✓	✓	✓			✓
MuJoCo [32]		✓	✓		✓	✓	✓
ADMM (Alg. 3)		✓	✓	✓	✓	✓	✓
<b>RaiSim</b> [33]	✓	✓		✓			
<b>NCP</b>							
PGS	✓	✓	✓	✓			✓
Staggered projections [6]	✓	✓	✓	✓	✓	✓	

problem (CCP), RaiSim, and the nonlinear complementarity problem (NCP). For each contact model, we also report the related algorithmic variants. If each tick in Tab. II represents a positive point for the concerned algorithm, Sec. IV shows that even one default may be prohibitive and can cause a solver to be unusable in practice. Finally, we also mention a set of useful implementation tricks which should be used to build an efficient simulator.

**Linear Complementarity Problem.** A first way to simplify the resolution of problem (14) is to linearize the NCP problem by approximating the second-order cone constraint from Coulomb's law with a pyramid, typically composed of four facets. This is done by replacing  $K_{\mu^{(i)}}$  by  $\tilde{K}_{\mu^{(i)}} = \{\lambda | \lambda_N \geq 0, \|\lambda_T\|_{\infty} \leq \mu^{(i)} \lambda_N\}$ . Doing so allows retrieving a linear complementarity problem (LCP), often easier to solve [50]. Such a problem is more standard and better-studied than its nonlinear counterpart [62]. Indeed, the standard Lemke's algorithm [50] or the projected Gauss-Seidel (PGS) algorithm (Alg. 1) allow to solve it and even come with convergence guarantees. Due to its easy implementation, the PGS algorithm may seem attractive and was widely adopted by modern physics engines, such as in Bullet [30], PhysX [60], ODE [29], and DART [31], [7] simulators.

This iterative algorithm loops on contact points and successively updates the normal and tangent contact forces. Because PGS works separately on each contact point, the update compensates for the current errors due to the estimated forces from other contact points. Yet, as illustrated in the experimental section IV, this process induces the emergence of internal forces during the resolution. Moreover, Gauss-Seidel-like approaches are similar to what is also known as block coordinate descent in the optimization literature. As first-order algorithms, they do not benefit from improved convergence rates or robustness with respect to their conditioning, unlike second-order algorithms. In parallel, the linearization of the second-order cone causes the loss of the isotropy for frictions, as stated by Coulomb's law. By choosing the axes for the facets of the pyramid and due to the maximum dissipation principle, one incidentally biases the friction forces towards the corners, as illustrated in Fig. 6. This error is sometimes mitigated by increasing the number of facets which also comes

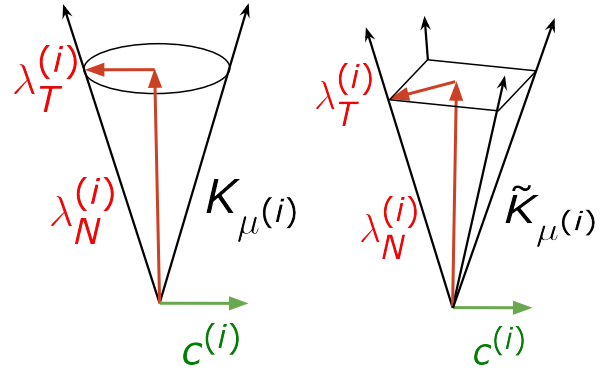


Fig. 6. **Cone linear approximation.** Linearizing the friction cone induces a bias on the direction of friction forces. The MDP tends to push tangential forces toward the corners of the pyramid.

**Algorithm 1:** Pseudo-code of the projected Gauss-Seidel (PGS) algorithm for solving LCPs.

---

**Input:** Delassus matrix:  $G$ , free velocity:  $g$ , friction cones  $K_{\mu}$

**Output:** Contact forces:  $\lambda$ , velocity:  $v$

```

1 for  $k = 1$  to  $M$  do
2   for  $i = 1$  to  $n_c$  do
3      $\lambda_N^{(i)} \leftarrow \lambda_N^{(i)} - \frac{1}{G_{iN}} (G\lambda + g)_N^{(i)}$ ;
4      $\lambda_N^{(i)} \leftarrow \max(0, \lambda_N^{(i)})$ ;
5      $\lambda_T^{(i)} \leftarrow \lambda_T^{(i)} - \frac{1}{\min(G_i, G_i)} (G\lambda + g)_T^{(i)}$ ;
6      $\lambda_T^{(i)} \leftarrow \text{clamp}(\lambda_T^{(i)}, \mu_i \lambda_N^{(i)})$ ;
7   end
8 end

```

---

at the cost of more computations.

**Cone Complementarity Problem.** An alternative approach consists in approximating the NCP problem in order to transform it into a more classical convex optimization problem. By relaxing the complementarity constraint from (14), one can obtain a Cone Complementarity Problem (CCP) [61]:

$$K_{\mu} \ni \lambda \perp c \in K_{\mu}^* \quad (15)$$

If this relaxation preserves the Maximum Dissipation Principle (MDP) and the second-order friction cone, it loses the *Signorini condition* (8). Indeed, re-writing the complementarity of (15) in the case of a sliding contacts yields:

$$\lambda_N c_N + \lambda_T^\top c_T = 0, \quad (16)$$

and in the case of sliding contact, the MDP leads to:

$$\lambda_N c_N - \mu \lambda_N \|c_T\|_2 = 0, \quad (17)$$

which is equivalent to the following complementarity condition:

$$\lambda_N \perp (c_N - \mu \|c_T\|_2). \quad (18)$$

(18) indicates that the CCP approximation allows for simultaneous normal velocity and forces, contrary to (8). The problem (15) can in fact be viewed as the Karush Kuhn Tucker conditions of an equivalent Quadratically Constrained Quadratic Programming (QCQP) problem:

$$\min_{\lambda \in K_\mu} \frac{1}{2} \lambda^\top G \lambda + g^\top \lambda \quad (19)$$

Once the contact problem is formulated as an optimization problem, any optimization algorithms can be employed to solve it. Here, we propose to study an ADMM algorithm [63] to solve (19), but Interior Point algorithms [64] can be used as in the latest Mujoco's version [32]. A benefit of using the family of proximal algorithms like ADMM is their natural ability to handle ill-conditioned and hyper-static cases. Another by-product of such methods is the implicit regularization they induce on the found solution, which removes the potential internal forces.

One may argue that such algorithms require to compute  $G^{-1}$  (Alg. 3, line 3) while per contact approaches repeatedly solve for each contact point individually, and thus only require the cheap inverse of diagonal blocks from  $G$  (Alg. 1, lines 3,5, Alg. 2, line 3, Alg. 4, line 4, Alg. 5, lines 3,5). However, the recent progress [43] demonstrated the Cholesky decomposition of  $G$  can be computed efficiently and robustly. We detail this point later when discussing implementation tricks, at the end of this section. Exploiting the knowledge of  $G^{-1}$  and not the block components as in the "per-contact" approaches mentioned earlier allows us to capture the coupling between all contact points. The most recent version of MuJoCo [32] adopts this approach with a Newton solver. In parallel, MuJoCo replaces  $G$  by  $\tilde{G} = G + R$  in the quadratic part of (19), which is justified by a compliant contact hypothesis. As evoked earlier,  $R$  is homogeneous to a compliance which should be a material property of the objects involved in the collision. However, MuJoCo arbitrarily set this to the diagonal of  $\alpha G$ , where  $\alpha \in [0, 1]$  is close to 0. This choice has no physical justification (at least, without making strong assumptions that are not met in practice), and its only intent is to improve the conditioning of the problem to ease the resolution and artificially make the solution unique. Moreover,  $R$  has non-null tangential components and thus may also introduce some tangential "compliance" which corresponds to the relaxation of Coulomb's law (Fig. 3). In fact, this should instead be interpreted as a Tikhonov regularization term enforcing the strict convexity of the problem to facilitate the numerics and the

---

**Algorithm 2:** Projected Gauss-Seidel (PGS) algorithm for Cone Complementarity Problem (CCP)

---

**Input:** Delassus matrix:  $G$ , free velocity:  $g$ , friction cones  $K_\mu$

**Output:** Contact forces:  $\lambda$ , velocity:  $v$

```

1 for  $k = 1$  to  $M$  do
2   for  $i = 1$  to  $n_c$  do
3      $\lambda^{(i)} \leftarrow \lambda^{(i)} - \frac{3}{G_i + G_i + G_i} (G\lambda + g)^{(i)}$ ;
4      $\lambda^{(i)} \leftarrow \text{proj}_{K_{\mu_i}}(\lambda^{(i)})$ ;
5   end
6 end
```

---



---

**Algorithm 3:** ADMM algorithm for Cone Complementarity Problem (CCP)

---

**Input:** Delassus matrix:  $G$ , free velocity:  $g$ , friction cones  $K_\mu$

**Output:** Contact forces:  $\lambda$ , velocity:  $v$

```

1  $\tilde{G}^{-1} \leftarrow (G + \rho \text{Id})^{-1}$ ;
2 for  $k = 1$  to  $M$  do
3    $\lambda \leftarrow -\tilde{G}^{-1}(g - \rho z + \gamma)$ ;
4    $z \leftarrow \text{proj}_{K_\mu}(\lambda + \frac{\gamma}{\rho})$ ;
5    $\gamma \leftarrow \gamma + \rho(\lambda - z)$ ;
6 end
```

---

existence of both the forward and inverse dynamics computation at the cost of shifting, even more, the solution.

**RaiSim contact model.** A contact model introduced in [65] and implemented in the RaiSim simulator [33] aims at partially correcting the drawbacks from the CCP contact model exploited in MuJoCo [32]. As explained earlier, the CCP formulation relaxes the Signorini condition for sliding contacts, leading to positive power from normal contact forces. The contact model proposed in [65] fixes this by explicitly enforcing the Signorini condition by constraining  $\lambda^{(i)}$  to remain in the null normal velocity hyper-plane  $V_N^{(i)} = \{\lambda | G_N^{(i)} \lambda + g_N^{(i)} = 0\}$ . For a sliding contact point, the problem (19) becomes:

$$\min_{\lambda \in K_{\mu^{(i)}} \cap V_N^{(i)}} \frac{1}{2} \lambda^\top G^{(i)} \lambda + \tilde{g}^{(i)\top} \lambda \quad (20)$$

where  $\tilde{g}^{(i)} = g^{(i)} + \sum_{j \neq i} G_{ij} \lambda^{(j)}$  is the  $i^{\text{th}}$  contact point velocity as it was free. The new problem (20) remains a QCQP and [33] leverages the analytical formula of the ellipse  $K_{\mu^{(i)}} \cap V_N^{(i)}$  in polar coordinates to tackle it as a 1D problem via the bisection algorithm [66] (Alg. 4, line 10). We refer to the original publication for a more detailed description of the bisection routine [33].

This approach implies several drawbacks. Indeed, it requires knowing whether a contact point is sliding, which cannot be known in advance as the contact point velocity depends on the contact forces. Thus, some heuristics, based on the disjunctive formulation of the contact problem (12), are introduced to try to guess the type of contact which will occur, *i.e.* take-off (Alg. 4, line 5), sticking (Alg. 4, line 7) or sliding (Alg. 4, line 9).



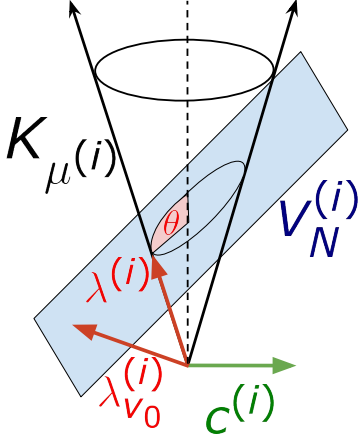


Fig. 7. **Bisection algorithm.** When the contact point is sliding,  $\lambda_{v_0}^{(i)}$  (Alg. 4, line 4) lies outside of the friction cone  $K_{\mu^{(i)}}$ , leading to a non-null tangential contact velocity  $c^{(i)}$ . In this case, RaiSim solves (20). This is equivalent to finding the  $\lambda \in K_{\mu^{(i)}} \cap V_N^{(i)}$  which is the closest to  $\lambda_{v_0}^{(i)}$  under the metric defined by  $G^{(i)}$ . The constraint set being an ellipse, the problem boils down to a 1D problem on  $\theta$  using polar coordinates. This figure is inspired from Fig. 2 of [33].

Obviously, such heuristics may be wrong, which may cause the algorithms to get stuck and lose convergence guarantees. This effect is strengthened by the caveats of the per-contact loop, which additionally make RaiSim not robust to conditioning and prone to internal forces. Eventually, if adding the constraint  $\lambda^{(i)} \in V_N^{(i)}$  allows to retrieve the Signorini condition from the CCP model, it also induces the loss of the maximum dissipation principle. Writing the Karush Kuhn Tucker (KKT) conditions of the problem (20) and some algebra manipulations yields:

$$c_T^{(i)} \propto -\lambda_T^{(i)} - \frac{\mu^{(i)^2} \lambda_N^{(i)}}{G_{NN}^{(i)}} G_{NT}^{(i)} \quad (21)$$

which contradicts (11).

**Tackling the NCP.** Despite the non-smooth and non-convex issues described previously, some simulation algorithms aim to directly solve the original NCP problem [67], [54]. The PGS algorithm exploited for LCP and CCP problems can easily be adapted to the NCP case by changing the clamping step (Alg. 1, line 6) or the normal projection (Alg. 2, line 4) for a horizontal projection on the cone (Alg. 5, line 6). However, it is worth noting that such approaches have fewer convergence guarantees than their relaxed counterpart [27]. As with every Gauss-Seidel approach, the methods inherited from the sensitivity to ill-conditioning and spurious internal forces. The staggered projections [34], [6] (Alg. 6) approach proceeds by rewriting the NCP as two interleaved optimization problems. This interconnection is solved via a fix-point algorithm that repeatedly injects one problem's solution into the formulation of the other. The staggered projections algorithm has no convergence guarantees but was heavily tested and seems, in practice, to converge most of the time in a few iterations (typically five iterations [6]). However, solving a cascade of optimization problems allows the use of robust optimization

---

**Algorithm 4:** Per-contact bisection algorithm

---

**Input:** Delassus matrix:  $G$ , free velocity:  $g$ , friction cones  $K_{\mu}$

**Output:** Contact forces:  $\lambda$ , velocity:  $v$

```

1 for  $k = 1$  to  $M$  do
2   for  $i = 1$  to  $n_c$  do
3      $\tilde{g}^{(i)} \leftarrow g^{(i)} + \sum_{j \neq i} G_{ij} \lambda^{(j)}$ ;
4      $\lambda_{v_0}^{(i)} \leftarrow -G_{ii}^{-1} \tilde{g}^{(i)}$ ;
5     if  $\tilde{g}_N^{(i)} > 0$  then
6       // takeoff
7        $\lambda^* \leftarrow 0$ ;
8     else if  $\lambda_{v_0}^{(i)} \in K_{\mu^{(i)}}$  then
9       // stiction
10       $\lambda^* \leftarrow \lambda_{v_0}^{(i)}$ ;
11    else
12      // sliding
13       $\lambda^* \leftarrow \text{bisection}(G_{ii}, \tilde{g}^{(i)}, K_{\mu^{(i)}}, \lambda_{v_0}^{(i)})$ ;
14    end
15  end
16 end
```

---



---

**Algorithm 5:** Projected Gauss-Seidel (PGS) algorithm for Non-linear Complementarity Problem (NCP)

---

**Input:** Delassus matrix:  $G$ , free velocity:  $g$ , friction cones  $K_{\mu}$

**Output:** Contact forces:  $\lambda$ , velocity:  $v$

```

1 for  $k = 1$  to  $M$  do
2   for  $i = 1$  to  $n_c$  do
3      $\lambda_N^{(i)} \leftarrow \lambda_N^{(i)} - \frac{1}{G_{iN}} (G\lambda + g)_N^{(i)}$ ;
4      $\lambda_N^{(i)} \leftarrow \max(0, \lambda_N^{(i)})$ ;
5      $\lambda_T^{(i)} \leftarrow \lambda_T^{(i)} - \frac{1}{\min(G_i, G_i)} (G\lambda + g)_T^{(i)}$ ;
6      $\lambda_T^{(i)} \leftarrow \text{proj}_{\mu_i \lambda_N^{(i)}}(\lambda_T^{(i)})$ ;
7   end
8 end
```

---

algorithms (e.g., ADMM), but remains more costly than other approaches.

**Additional practical tricks.** In practice, the performances of contact solvers can be improved by a few simple tricks. The most important one is warm-starting the solver by providing the contact forces from the previous time step. Indeed, in the case of a persisting contact between two objects, the contact forces can be cached and reused as an initial guess when solving for the contact forces of the next time step. This relies on the ability of the contact solver algorithm to be warm-started. This excludes Interior Point algorithms as they expect a feasible initial guess. In contrast, the feasible set of contact forces may change from one time step to the other, even in the case of a persisting contact point. On the opposite, augmented lagrangian (AL) methods can naturally be warm started: not only the primal (i.e., contact forces) and dual (i.e. contact velocities)



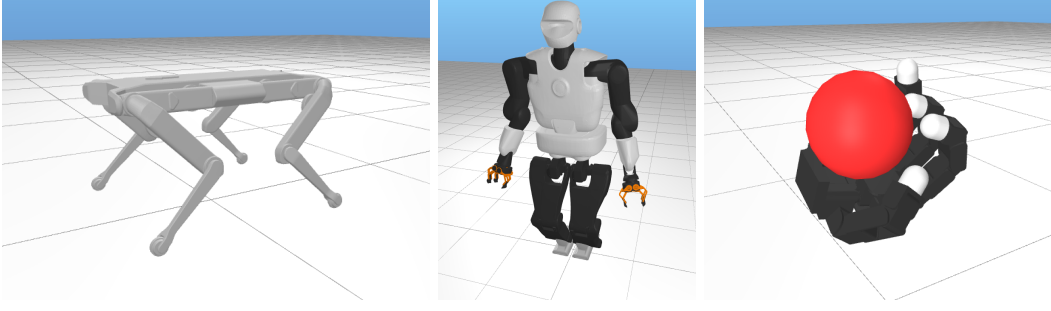


Fig. 8. **Robotics systems used for the experiments.** The Solo-12 quadruped (**Left**), the Talos humanoid (**Center**) and the Allegro hand (**Right**) allow to respectively exhibit locomotion, high-dimensional, and manipulation contact scenarii.

---

**Algorithm 6:** Staggered projections algorithm

---

**Input:** Delassus matrix:  $G$ , free velocity:  $g$ , friction cones  $K_\mu$

**Output:** Contact forces:  $\lambda$ , velocity:  $v$

```

1 for  $k = 1$  to  $M$  do
2    $\tilde{g}_N \leftarrow g_T + G_{NT} \lambda_T$ ;
3    $\lambda_N \leftarrow \arg \min_{\lambda \geq 0} \frac{1}{2} \lambda^\top G_N \lambda + \tilde{g}_N^\top \lambda$ ;
4    $\tilde{g}_T \leftarrow g_T + G_{TN} \lambda_N$ ;
5    $\lambda_T \leftarrow \arg \min_{\|\lambda^{(i)}\| \leq \mu_i \lambda_N^{(i)}} \frac{1}{2} \lambda^\top G_T \lambda + \tilde{g}_T^\top \lambda$ ;
6 end
```

---

variables but also the proximal parameter can be initialized with the previous values.

In addition, second-order algorithms can further exploit the recent progress in rigid body algorithms [43]. This work takes advantage of the sparse structure of the kinematic chains in order to efficiently compute the Cholesky decomposition of the Delassus matrix  $G$ . This approach is robust enough to handle the case of hyperstatic systems and reduces the cost of the computation of matrix-vector products involving  $G^{-1}$  (Alg. 3, line 3). This also indicates that evaluating  $G$  from its Cholesky decomposition, as required by per-contact approaches, actually constitutes an additional cost. In the context of ADMM (Alg. 3, the algorithm from [43] can also be favorably combined with the adaptation of the proximal parameter [68]. Indeed, updating the regularized Cholesky can be done at almost no cost by using [43].

Additionally, over-relaxation is often employed to accelerate the convergence of both Gauss-Seidel or ADMM algorithms. This technique applies the following update:

$$\lambda \leftarrow \alpha \lambda^- + (1 - \alpha) \lambda, \quad (22)$$

where  $\alpha \in ]0, 2[$  and  $\lambda^-$  denotes the previous iteration. For  $\alpha > 1$ , over-relaxing consists in an extrapolation step and should be carefully used as it may also hinder convergence.

#### IV. EXPERIMENTS

In this section, we thoroughly evaluate the performances and behaviors of the formulations explained in Sec. III. To fairly compare and benchmark the various algorithmic formulations, we have implemented them in a unified C++ framework

called ContactBench. This framework extensively relies on the Pinocchio library [24] for rigid body algorithms and HPP-FCL [38], [46] implementation of GJK and EPA for collision detection. It is worth noting that some popular simulators have closed-source implementations, such as RaiSim [33]. In this case, we strove to keep our implementation as close as possible to the information in the corresponding papers. ContactBench will be released as open-source upon article acceptance.

Several factors may hinder the correctness and accuracy of simulators based on time-stepping methods:

- i) the low accuracy of the solver of the contact problem;
- ii) the limitation from the contact model itself;
- iii) or the numerical integration due to the time discretization scheme.

In this section, we first evaluate the error from sources i) and ii) (Sec. IV-A). The source of error i) is evaluated by measuring the time taken to reach a given accuracy. The errors from ii) are analyzed by measuring the residual for an (approximately) infinite time budget. We further assess i) and iii) by examining the sensitivity of the contact solvers with respect to respectively the stopping criterion value  $\epsilon_{\text{abs}}$  and the time-step  $\Delta t$  (Sec. IV-B). Sec. IV-C evaluates their computational efficiency. Eventually, Sec. IV-D explores how the contact models and their implementations can impact the final robotics applications, in the case of the MPC for quadruped locomotion.

##### A. Evaluation of physical correctness

**LCP relaxation.** The linearization of the friction cone loses the isotropy and biases the friction forces towards some specific directions, as shown in Fig. 6). This observation has already been raised in the literature [6], [8]. As expected, the bias on the contact forces significantly impairs the simulation by deviating the trajectory of the simulated system (Fig. 9).

**CCP relaxation.** As detailed previously, the CCP contact model relaxes the *Signorini condition*. As shown in Fig. 10, this results in non-null normal contact forces and velocities when a contact point is sliding. As a consequence, the contact points start to bump, which modifies the trajectory of the system (Fig. 10, left), which also impacts the overall dissipated energy (Fig. 10, right). The model adopted by Raisim aims at correcting this

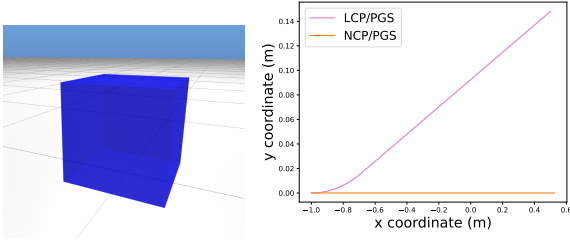


Fig. 9. **Trajectory of a cube sliding on a plane.** The cube is initialized with an initial tangential velocity along the y-axis. **Right:** The bias of friction forces (Fig.6) introduces a tangential velocity along the x-axis, which deviates the cube from the expected straight-line trajectory.

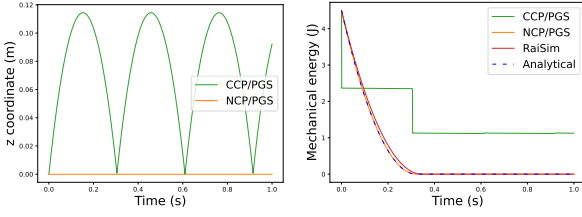


Fig. 10. **A cube is initialized on a plane with a tangential velocity along the y-axis,** similarly to the case studied in Fig. 9. **Left:** The CCP contact model relaxes the Signorini condition, which induces unphysical forces leading to the vertical bouncing of the cube. **Right:** From the MDP, it is possible to determine the evolution of the energy of the system analytically and compare it to what is computed by the various simulation algorithms. The CCP relaxation induces a significant gap with the analytical solution. The RaiSim contact model narrows this gap but dissipates less power than expected, as it does not enforce the MDP. The NCP formulation, solved using the PGS solver, is the only formulation which closely matches the expected analytical behavior of the system.

undesired phenomenon by enforcing the Signorini condition but still does not match the analytical solution due to its relaxation of the MDP (21) (Fig. 10, right).

**Underdetermined contact problems.** Underdetermination occurs when infinite combinations of contact forces lead to the same trajectory. These artifacts happen on the normal and tangential components of contact forces, as depicted in Fig. 11. As shown in Fig. 11, the solution found depends on the numerical scheme. We observe that the per-contact approaches (Alg. 2,4 and 5) exhibit spurious internal forces at stiction, values which are not controlled by the algorithms. On the opposite, the algorithms working directly on the global contact problem with a proximal regularization (Alg. 3 and 6) avoid injecting such artifacts in the contact forces (Fig. ,11).

This phenomenon may seem innocuous as forward dynamics are not affected. However, it makes the inverse dynamics ill-posed, as there is no way to predict such numerical artifacts. Additionally, in the context of differentiable physics, we believe these spurious contact forces may catastrophically impact the computation of derivatives, but we leave this study as a future work. Finally, it is worth mentioning that such under-determined cases are ubiquitous in robotics (e.g., legged robots making redundant contact with their environments).

**Robustness to ill-conditioning contact problems.** More generally, the contact problem becomes challenging when the

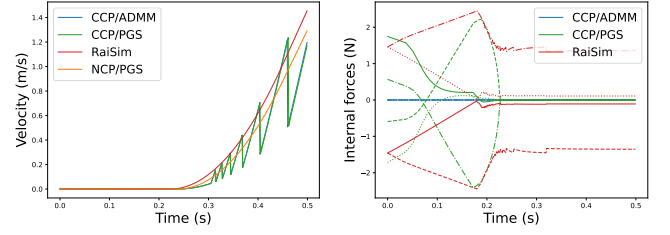


Fig. 11. **Applying a linearly growing horizontal force to a cube on a plane.** **Left:** The cube is at stiction before it starts sliding after approximately 0.25s. The tangential velocity differs depending on the contact model: CCP induces some bumps which occasionally dissipate energy during impacts, while RaiSim violates the MDP leading to contact points sliding faster than in the case of NCP. **Right:** At stiction, multiple combinations of tangential forces may lead to the same trajectory. There are four curves for each contact model, each curve accounting for one of the four contact forces on the cube. CCP solvers exhibit internal forces "stretching" the cube at stiction before the MDP enforces these forces to disappear when the cube starts to slide. RaiSim relaxes the maximum dissipation principle so the friction forces are not opposed to the movements, and internal forces persist when the cube is sliding. Eventually, ADMM avoids injecting spurious internal forces even at stiction.

ratio between the biggest and the smallest eigenvalue of the Delassus matrix grows. The experiment of Fig. 12 exhibits the convergence issues of per-contact approaches when simulating systems with a strong coupling between the different contact points, which causes large off-diagonal terms on the matrix  $G$ . Such a behavior can be expected as supposing the matrix to be diagonal dominant is a classical hypothesis ensuring the convergence of Gauss-Seidel methods. On the contrary, the proximal algorithms account for off-diagonal terms of  $G$  and only rely on a regularized inverse of  $G$  (Alg. 3, line 1) and thus robustly converge towards an optimal solution.

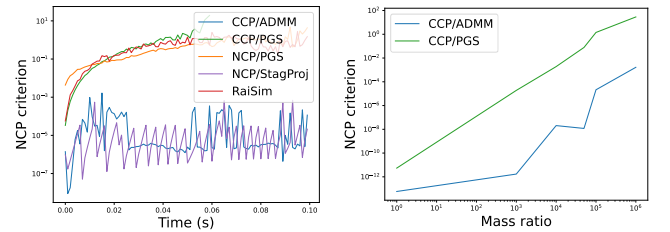


Fig. 12. **Simulation of ill-conditioned systems.** **Left:** Stacking a heavy cube ( $10^3\text{kg}$ ) on a light one ( $10^{-3}\text{kg}$ ) makes the problem ill-conditioned and, therefore, not solvable via per-contact algorithms. **Right:** The accuracy of the simulators improves when the ratio between the masses of the two cubes gets close to one. The ADMM algorithm remains more accurate than PGS in all cases.

**Effects of compliance.** As demonstrated by Fig. 13, the normal forces vary linearly with the compliance parameter  $R$ . Moreover, adding compliance to the tangential components induces the vanishing of dry frictions, resulting in tangential oscillations instead of a null velocity. These compliant effects regularize the infinitely steep graphs due to the Signorini condition and Coulomb's law and replace them with locally linear mapping, which also eases the numerics. Therefore, the compliance added in MuJoCo has no physical purpose and should be considered a numerical trick designed to circumvent

the issues due to hyper-staticity or ill-conditioning at the cost of impairing the simulation.

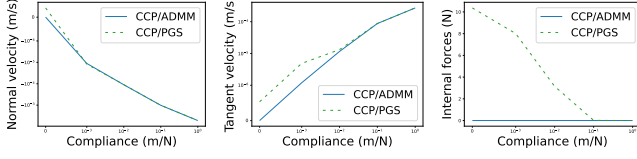


Fig. 13. **Simulation of Solo-12 with varying compliance for the contacts with the floor.** **Left:** Adding a compliance to the contact model relaxes both the Signorini condition. **Center:** This compliance also relaxes the Coulomb's law of friction. **Right:** Compliance also regularizes the problem, removing the spurious internal forces in under-determined cases.

### B. Self-consistency of the solvers

The accuracy of simulators can be affected by the numerical resolution induced by two "hyper-parameters": the value of the stopping criterion for the contact solver desired accuracy ( $\epsilon_{abs}$ ) and the time-step value ( $\Delta t$ ). We measure their effect on the simulation quality when varying them independently. A simulator is said to be self-consistent when this deviation remains limited.

**Time stepping.** As already mentioned, time-stepping simulators are sensitive to the choice of the time-step  $\Delta t$ . Here, we intend to assess the self-consistency of the various contact solvers by examining their deviation when  $\Delta t$  grows. Because time discretization also affects the collision detection process, our study is done on the trajectory of a cube sliding on a plane whose contact points should remain constant. A trajectory obtained by simulating the system with a small time-step ( $\Delta t = 0.1ms$ ) serves as a reference to compute the state consistency error along the trajectories simulated with larger time-steps (Tab. III).

TABLE III  
INTEGRAL CONSISTENCY ERROR FOR A CUBE SLIDING ON A PLANE.

Solver	Time-step (ms)	Integral consistency error (m.s)
CCP/PGS	1	$1.6 \times 10^{-3}$
CCP/PGS	10	$1.7 \times 10^{-2}$
CCP/PGS	100	$1.6 \times 10^{-1}$
RaiSim	1	$1.5 \times 10^{-3}$
RaiSim	10	$1.6 \times 10^{-2}$
RaiSim	100	$1.5 \times 10^{-1}$
NCP/PGS	1	$1.5 \times 10^{-3}$
NCP/PGS	10	$1.6 \times 10^{-2}$
NCP/PGS	100	$1.6 \times 10^{-1}$

Looking at Tab. III, we observe that the sensitivities with respect to the time-step are similar for the various solvers. Moreover, as shown by Fig. 14, the energy evolution of the system simulated via NCP and RaiSim models is only a little affected when increasing  $\Delta t$ . On the contrary, the CCP relaxes the *Signorini condition*, which results in the cube taking off and causing a nonphysical and inconsistent evolution of the mechanical energy.

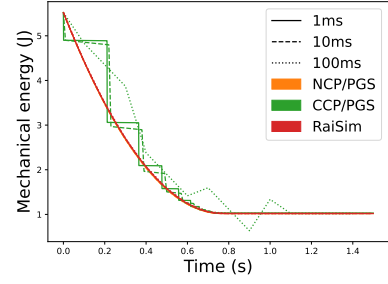


Fig. 14. **Self-consistency w.r.t. time-stepping when simulating a cube initialized on a plane with a tangential velocity.** The sensitivity to the time-stepping can also be observed through the evolution of mechanical energy.

**Stopping criterion.** As done in the case of the time-stepping, the sensitivity to the stopping criterion is evaluated by measuring the integrated consistency error with respect to a high-resolution ( $\epsilon_{abs} = 10^{-9}$ ) reference trajectory (Tab. IV). Contrarily to the time-stepping case, the solvers appear to have different behaviors when their stopping criteria are modified. Indeed, trajectories generated via the RaiSim and CCP contact models are significantly modified when  $\epsilon_{abs}$  is increased (Tab. IV, Fig. 15). The figure 15 demonstrates that the NCP model simulates robustly the mechanical energy evolution, which is not true for the CCP and RaiSim models. In particular, as the accuracy of RaiSim is relaxed, due to its approximation (21), the error on the MDP becomes completely uncontained leading to almost no energy dissipation for low accuracies ( $\epsilon_{abs} = 10^{-2}$ ).

TABLE IV  
INTEGRAL CONSISTENCY ERROR FOR A CUBE SLIDING ON A PLANE.

Solver	Stopping criterion	Integral consistency error (m.s)
CCP/PGS	$10^{-6}$	$6.7 \times 10^{-7}$
CCP/PGS	$10^{-4}$	$4.3 \times 10^{-5}$
CCP/PGS	$10^{-2}$	$4.8 \times 10^{-2}$
RaiSim	$10^{-6}$	$9.0 \times 10^{-8}$
RaiSim	$10^{-4}$	$4.8 \times 10^{-5}$
RaiSim	$10^{-2}$	$7.0 \times 10^{-4}$
NCP/PGS	$10^{-6}$	$6.0 \times 10^{-10}$
NCP/PGS	$10^{-4}$	$9.9 \times 10^{-8}$
NCP/PGS	$10^{-2}$	$5.3 \times 10^{-6}$

### C. Performance benchmarks

As evoked earlier, in addition to being physically accurate, it is also essential for a simulator to be fast, which, in general, constitutes two adversarial requirements. To evaluate the computational footprint of the various solvers, we measure the time taken to reach a fixed accuracy on dynamic trajectories involving robotics systems (Fig. 8). When the contact solvers are cold started, we observe that the second-order optimization techniques like ADMM are less efficient than the PGS solvers and their cheap per-contact iterations (Fig. 16, left). However, leveraging the solution from the previous time step to warm-start the solvers, a common strategy in practice, allows for

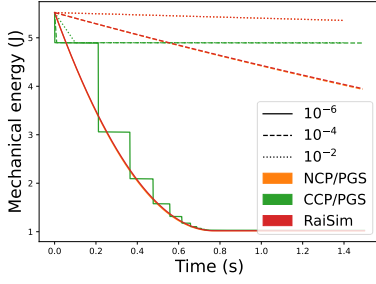


Fig. 15. Self-consistency w.r.t. stopping criteria when simulating a cube initialized on a plane with a tangential velocity. Dropping the solver accuracy hardly affect the energy evolution when using NCP/PGS. On the contrary, the energy of trajectories from the CCP/PGS and RaiSim algorithms is spectacularly impacted.

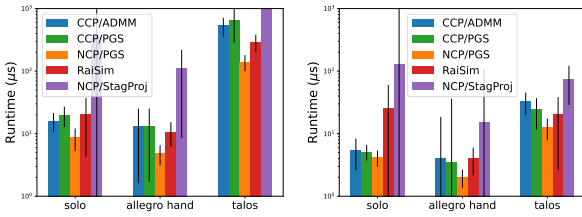


Fig. 16. Computational timings measured along a trajectory for three robotics systems (c.f. Fig. 8). Solo-12 and Talos are perturbed by applying an external force at their center of mass, while a ball is dropped in the Allegro hand, so the trajectories are not static. The contact solvers are tested in both cold-start (Left) and warm-start (Right) modes. We simulate the same trajectories to evaluate the benefit of warm-starting, but we use the solution of the previous time step as an initial guess. This leads to significant improvements in the computational timings.

significantly reducing this gap (Fig. 16, right). Therefore, regarding the study of Sec. IV-A, a trade-off appears for algorithms like ADMM and Staggered Projections, which treat all the contact points globally. In practice, they might be slower than their PGS counterpart while they benefit from better behaviors on ill-conditioned problems.

#### D. MPC for quadruped locomotion

The previous examples already illustrate the differences among the various simulators in terms of both physical accuracy and numerical efficiency. However, such scenarios may not represent the richness of contacts in practical robotics situations. For this purpose, we use the implementation of MPC on the Solo-12 system introduced in [69] to generate locomotion trajectories on flat and bumpy terrains. These experiments are designed to involve a wide variety of contacts (*i.e.*, sticking, sliding, and taking-off) and see how the simulation choices impact the final task (*i.e.*, horizontal translation of the robot).

For a flat and barely slippery ( $\mu = 0.9$ ) ground, we observe that the choice of simulator hardly affects the base velocity tracked by the MPC controller (Fig. 16, top right). In this case, the contacts are mainly sticking, leading to low violation of the NCP criterion (14) (Fig. 16, bottom left).

When the terrain is bumpy (roughness of  $10^{-1}m$ ) and slippery ( $\mu = 0.3$ ), the locomotion velocity generated from the RaiSim and CCP models significantly deviates from the NCP

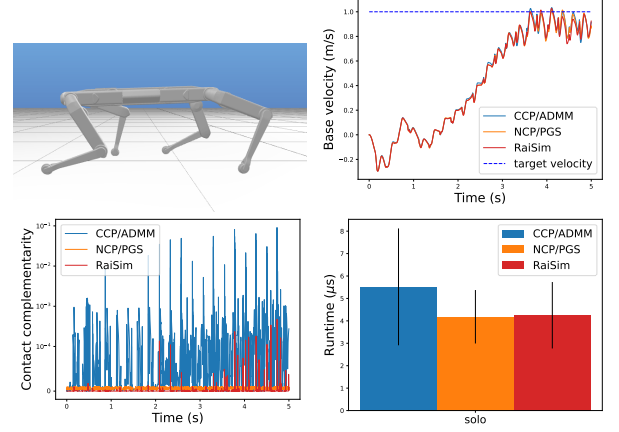


Fig. 17. MPC for locomotion on a flat terrain (Top left). The target horizontal translation velocity of the base is similarly reached by the controller with the different simulators (Top right). However, they do not equally respect the contact complementarity criterion is not (Bottom left). Per-contact approaches, *e.g.* are more efficient (Bottom right).

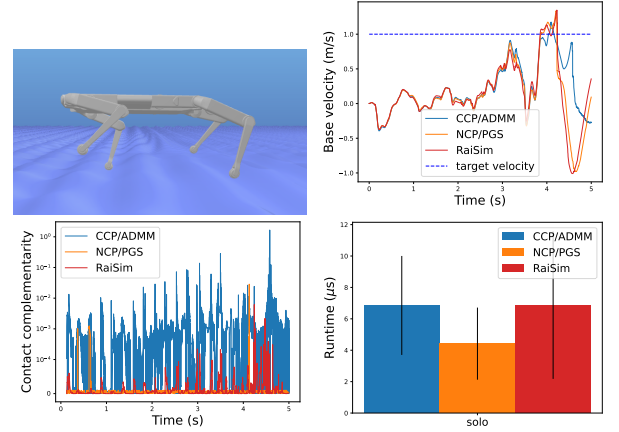


Fig. 18. MPC for locomotion on a bumpy terrain (Top left). The tracked velocity of the base quickly differs depending on the used simulator (Top right). Slippery contact points violate the contact complementarity criterion for the RaiSim and CCP contact modelings (Bottom left). The complexity of contacts also hampers the solvers and reduces the gap between per-contact and ADMM approaches (Bottom right).

one (Fig. 16, top right). This can be explained in the light of our previous study as both the RaiSim and CCP contact models make physical approximations when contact points are sliding (Fig. 16, bottom left). In addition, the simulation is more challenging than the flat case, which causes increased computations from the solvers, particularly for RaiSim (Fig. 16, bottom right). These observations indicate that the low-level choices of the contact solver may induce significant differences in the high-level behaviors of locomotion controllers on complex terrains.

## V. DISCUSSION AND CONCLUSION

NCP is known to be complex to solve and thus is often relaxed to find approximate solutions. In this article, we report a deeper study on how the various rigid contact models commonly employed in robotics and their associated solvers can impact

the resulting simulation. We have notably established and experimentally highlighted that these choices may induce unphysical artifacts, thus widening the reality gap, leading to unrealistic behaviors when the simulator is later used for practical robotics applications. Our experiments show that there is no fully satisfactory approach at the moment, as all existing solutions compromise either accuracy, robustness, or efficiency. This also indicates that there may still be room for improvements in contact simulation.

Beyond contact simulation, differentiable physics constitutes an emergent and closely related topic. However, the impact of forward simulation artifacts on gradient computation remains unexplored. In particular, some of the relaxations at work, e.g. the artificial compliance added in MuJoCo, result in crucial differences in gradients, which then affect downstream applications like trajectory optimization [70], [71]. We leave the study of the various existing differentiable simulators [5], [6], [7], [72], [8] through this lens as a future work.

For all these reasons, we believe it would be highly beneficial for the robotics community to take up such low-level topics around simulation, as they could lead to substantial progress in the field. The work of [36] is an inspiring first step in this direction. With this article, we intend to go further by providing open-source implementations and benchmarks to the community.

#### ACKNOWLEDGMENTS

We thank Jemin Hwangbo for providing useful details on the algorithm of the RaiSim simulator, and Stephane Caron for helpful discussions. This work was supported in part by L'Agence d'Innovation Défense, the French government under the management of Agence Nationale de la Recherche through the project INEXACT (ANR-22-CE33-0007-01) and as part of the "Investissements d'avenir" program, reference ANR-19-P3IA-0001 (PRAIRIE 3IA Institute), by the European Union through the AGIMUS project (GA no.101070165) and the Louis Vuitton ENS Chair on Artificial Intelligence. Views and opinions expressed are those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the European Commission can be held responsible for them.

#### REFERENCES

- [1] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems," vol. 1, pp. 222–229, 01 2004.
- [2] D. Mayne, "A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems," *International Journal of Control*, vol. 3, no. 1, pp. 85–95, 1966.
- [3] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4906–4913, IEEE, 2012.
- [4] J. Carpentier and N. Mansard, "Analytical derivatives of rigid body dynamics algorithms," in *Robotics: Science and systems (RSS 2018)*, 2018.
- [5] F. de Avila Belbute-Peres, K. Smith, K. Allen, J. Tenenbaum, and J. Z. Kolter, "End-to-end differentiable physics for learning and control," *Advances in neural information processing systems*, vol. 31, 2018.
- [6] Q. Le Lidec, I. Kalevtykh, I. Laptev, C. Schmid, and J. Carpentier, "Differentiable simulation for physical system identification," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3413–3420, 2021.
- [7] K. Werling, D. Omens, J. Lee, I. Exarchos, and C. K. Liu, "Fast and feature-complete differentiable physics engine for articulated rigid bodies with contact constraints," in *Robotics: Science and Systems*, 2021.
- [8] T. Howell, S. Le Cleac'h, J. Brüdigan, Z. Kolter, M. Schwager, and Z. Manchester, "Dojo: A differentiable simulator for robotics," *arXiv preprint arXiv:2203.00806*, 2022.
- [9] M. Diehl, H. G. Bock, H. Diedam, and P.-B. Wieber, "Fast direct multiple shooting algorithms for optimal robot control," in *Fast motions in biomechanics and robotics*, pp. 65–93, Springer, 2006.
- [10] D. Q. Mayne, "Model predictive control: Recent developments and future promise," *Automatica*, vol. 50, no. 12, pp. 2967–2986, 2014.
- [11] S. Kleff, A. Meduri, R. Budhiraja, N. Mansard, and L. Righetti, "High-frequency nonlinear model predictive control of a manipulator," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7330–7336, IEEE, 2021.
- [12] E. Dantec, M. Taix, and N. Mansard, "First order approximation of model predictive control solutions for high frequency feedback," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4448–4455, 2022.
- [13] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
- [14] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," in *Robotics: Science and Systems*, 2018.
- [15] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, et al., "Solving rubik's cube with a robot hand," *arXiv preprint arXiv:1910.07113*, 2019.
- [16] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, vol. 4, no. 26, p. eaau5872, 2019.
- [17] M. T. Mason and J. K. Salisbury Jr, *Robot hands and the mechanics of manipulation*. The MIT Press, Cambridge, MA, 1985.
- [18] J. Ponce, S. Sullivan, A. Sudsang, J.-D. Boissonnat, and J.-P. Merlet, "On computing four-finger equilibrium and force-closure grasps of polyhedral objects," *The International Journal of Robotics Research*, vol. 16, no. 1, pp. 11–35, 1997.
- [19] B. Dynamics, "Atlas gets a grip | boston dynamics." [https://youtu.be/-e1\\_QhJIEhQ](https://youtu.be/-e1_QhJIEhQ).
- [20] I. Mordatch, E. Todorov, and Z. Popović, "Discovery of complex behaviors through contact-invariant optimization," *ACM Transactions on Graphics (ToG)*, vol. 31, no. 4, pp. 1–8, 2012.
- [21] M. Posa, C. Cantu, and R. Tedrake, "A direct method for trajectory optimization of rigid bodies through contact," *The International Journal of Robotics Research*, vol. 33, no. 1, pp. 69–81, 2014.
- [22] M. A. Toussaint, K. R. Allen, K. A. Smith, and J. B. Tenenbaum, "Differentiable physics and stable modes for tool-use and manipulation planning," 2018.
- [23] R. Featherstone, *Rigid body dynamics algorithms*. Springer, 2014.
- [24] J. Carpentier, G. Saurel, G. Buondonno, J. Mirabel, F. Lamiraux, O. Stasse, and N. Mansard, "The pinocchio c++ library: A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives," in *2019 IEEE/SICE International Symposium on System Integration (SII)*, pp. 614–619, IEEE, 2019.
- [25] J. J. Moreau, "Unilateral Contact and Dry Friction in Finite Freedom Dynamics," in *Nonsmooth Mechanics and Applications* (M. J.J. and P. P.D., eds.), vol. 302 of *International Centre for Mechanical Sciences (Courses and Lectures)*, pp. 1–82, Springer, 1988.
- [26] M. Jean, "The non-smooth contact dynamics method," *Computer methods in applied mechanics and engineering*, vol. 177, no. 3–4, pp. 235–257, 1999.
- [27] V. Acary, F. Cadoux, C. Lemaréchal, and J. Malick, "A formulation of the linear discrete Coulomb friction problem via convex optimization," *Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik*, vol. 91, pp. 155–175, Feb. 2011.
- [28] B. Brogliato, A. Ten Dam, L. Paoli, F. Génot, and M. Abadie, "Numerical simulation of finite dimensional multibody nonsmooth mechanical systems," *Appl. Mech. Rev.*, vol. 55, no. 2, pp. 107–150, 2002.
- [29] R. Smith, "Open dynamics engine," 2008. <http://www.ode.org/>.
- [30] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," <http://pybullet.org>, 2016–2021.
- [31] J. Lee, M. X. Grey, S. Ha, T. Kunz, S. Jain, Y. Ye, S. S. Srinivasa, M. Stilman, and C. K. Liu, "DART: Dynamic animation and robotics toolkit," *The Journal of Open Source Software*, vol. 3, p. 500, Feb 2018.
- [32] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ international conference on intelligent robots and systems*, pp. 5026–5033, IEEE, 2012.
- [33] J. Hwangbo, J. Lee, and M. Hutter, "Per-contact iteration method for solving contact dynamics," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 895–902, 2018.



- [34] D. M. Kaufman, S. Sueda, D. L. James, and D. K. Pai, "Staggered projections for frictional contact in multibody systems," pp. 1–11, 2008.
- [35] T. Erez, Y. Tassa, and E. Todorov, "Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4397–4404, 2015.
- [36] P. Horak and J. C. Trinkle, "On the similarities and differences among contact models in robot simulation," *IEEE Robotics and Automation Letters*, vol. 4, pp. 493–499, 2019.
- [37] J. Carpentier, F. Valenza, N. Mansard, *et al.*, "Pinocchio: fast forward and inverse dynamics for poly-articulated systems," <https://stack-of-tasks.github.io/pinocchio>, 2015–2021.
- [38] J. Pan, S. Chitta, D. Manocha, F. Lamiraux, J. Mirabel, J. Carpentier, *et al.*, "HPP-FCL: an extension of the Flexible Collision Library," <https://github.com/humanoid-path-planner/hpp-fcl>, 2015–2022.
- [39] J. Mirabel, S. Tonneau, P. Fernbach, A.-K. Seppälä, M. Campana, N. Mansard, and F. Lamiraux, "HPP: A new software for constrained motion planning," in *International Conference on Intelligent Robots and Systems*, 2016.
- [40] L. Montaut, Q. Le Lidec, V. Petřík, J. Sivic, and J. Carpentier, "Collision Detection Accelerated: An Optimization Perspective," in *Proceedings of Robotics: Science and Systems*, (New York City, NY, USA), June 2022.
- [41] P.-L. M. de Maupertuis, *Essais de cosmologie*. 1751.
- [42] S. L. Campbell and C. W. Gear, "The index of general nonlinear daes," *Numerische Mathematik*, vol. 72, pp. 173–196, 1995.
- [43] J. Carpentier, R. Budhiraja, and N. Mansard, "Proximal and sparse resolution of constrained dynamic equations," in *Robotics: Science and Systems*, 2021.
- [44] C. F. Gauß, "Über ein neues allgemeines grundgesetz der mechanik," 1829.
- [45] F. E. Udvardi and R. E. Kalaba, "A new perspective on constrained motion," *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, vol. 439, no. 1906, pp. 407–410, 1992.
- [46] L. Montaut, Q. Le Lidec, V. Petřík, J. Sivic, and J. Carpentier, "Collision detection accelerated: An optimization perspective," in *RSS 2022- Robotics: Science and Systems*, 2022.
- [47] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space," *IEEE Journal on Robotics and Automation*, vol. 4, no. 2, pp. 193–203, 1988.
- [48] C. Ericson, *Real-Time Collision Detection*. The Morgan Kaufmann Series, 2004.
- [49] A. Signorini, "Questioni di elasticità non linearizzata e semilinearizzata," *Rendiconti di Matematica e delle sue applicazioni*, vol. 18, no. 5, pp. 95–139, 1959.
- [50] R. W. Cottle, J.-S. Pang, and R. E. Stone, *The Linear Complementarity Problem*. Society for Industrial and Applied Mathematics, 2009.
- [51] P. Wensing, R. Featherstone, and D. E. Orin, "A reduced-order recursive algorithm for the computation of the operational-space inertia matrix," in *2012 IEEE International Conference on Robotics and Automation*, pp. 4911–4917, IEEE, 2012.
- [52] J. Baumgarte, "Stabilization of constraints and integrals of motion in dynamical systems," *Computer methods in applied mechanics and engineering*, vol. 1, no. 1, pp. 1–16, 1972.
- [53] G. de Saxcé and Z.-Q. Feng, "The bipotential method: A constructive approach to design the complete contact law with friction and improved numerical algorithms," *Mathematical and Computer Modelling*, vol. 28, pp. 225–245, Aug. 1998.
- [54] V. Acary, M. Brémont, and O. Huber, "On solving contact problems with Coulomb friction: formulations and numerical comparisons," Research Report RR-9118, INRIA, Nov. 2017.
- [55] K. Erleben, "Rigid body contact problems using proximal operators," in *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, SCA '17, (New York, NY, USA), Association for Computing Machinery, 2017.
- [56] J. Pan, S. Chitta, and D. Manocha, "FCL: A General Purpose Library for Collision and Proximity Queries," in *2012 IEEE International Conference on Robotics and Automation*, IEEE, 2012.
- [57] D. Fiser, "libccd," <https://github.com/danfis/libccd>.
- [58] M. L. Felis, "Rbdl: an efficient rigid-body dynamics library using recursive algorithms," *Autonomous Robots*, vol. 41, no. 2, pp. 495–511, 2017.
- [59] R. Tedrake and the Drake Development Team, "Drake: Model-based design and verification for robotics," 2019.
- [60] M. Macklin, K. Storey, M. Lu, P. Terdiman, N. Chentanez, S. Jeschke, and M. Müller, "Small steps in physics simulation," in *Proceedings of the 18th Annual ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '19, (New York, NY, USA), Association for Computing Machinery, 2019.
- [61] M. Anitescu and A. Tasora, "An iterative approach for cone complementarity problems for nonsmooth dynamics," *Computational Optimization and Applications*, vol. 47, no. 2, pp. 207–235, 2010.
- [62] K. Erleben, "Numerical methods for linear complementarity problems in physics-based animation," in *ACM SIGGRAPH 2013 Courses*, SIGGRAPH '13, (New York, NY, USA), Association for Computing Machinery, 2013.
- [63] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.
- [64] S. Mehrotra, "On the implementation of a primal-dual interior point method," *SIAM Journal on optimization*, vol. 2, no. 4, pp. 575–601, 1992.
- [65] T. Preclik, *Models and algorithms for ultrascale simulations of non-smooth granular dynamics*. Friedrich-Alexander-Universitaet Erlangen-Nuernberg (Germany), 2014.
- [66] S. Boyd and L. Vandenberghe, "Localization and cutting-plane methods," *From Stanford EE 364b lecture notes*, 2007.
- [67] F. Jourdan, P. Alart, and M. Jean, "A gauss-seidel like algorithm to solve frictional contact problems," *Computer methods in applied mechanics and engineering*, vol. 155, no. 1-2, pp. 31–47, 1998.
- [68] R. Nishihara, L. Lessard, B. Recht, A. Packard, and M. Jordan, "A general analysis of the convergence of admm," in *International conference on machine learning*, pp. 343–352, PMLR, 2015.
- [69] P.-A. Léziart, T. Flayols, F. Grimmering, N. Mansard, and P. Souères, "Implementation of a Reactive Walking Controller for the New Open-Hardware Quadruped Solo-12," in *2021 IEEE International Conference on Robotics and Automation - ICRA*, (Xi'an, China), May 2021.
- [70] H. J. T. Suh, T. Pang, and R. Tedrake, "Bundled gradients through contact via randomized smoothing," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4000–4007, 2022.
- [71] Q. Le Lidec, L. Montaut, C. Schmid, I. Laptev, and J. Carpentier, "Leveraging randomized smoothing for optimal control of nonsmooth dynamical systems," *arXiv preprint arXiv:2203.03986*, 2022.
- [72] E. Heiden, D. Millard, E. Coumans, Y. Sheng, and G. S. Sukhatme, "NeuralSim: Augmenting differentiable simulators with neural networks," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2021.