Trajectory Planning in high dimensional robotic systems with geometric collision avoidance

Rebecca Richter

Vollständiger Abdruck der von der Fakultät für Luft- und Raumfahrttechnik der Universität der Bundeswehr München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

angenommenen Dissertation.

Gutachter: 1. Prof. Dr. rer. nat. Matthias Gerdts
2. Prof. Dr. rer. nat. Jürgen Pannek

Die Dissertation wurde am 03.06.2025 bei der Universität der Bundeswehr München eingereicht und durch die Fakultät für Luft- und Raumfahrttechnik am 15.10.2025 angenommen. Die mündliche Prüfung fand am 17.10.2025 statt.

Abstract

In this thesis, we focus on high dimensional trajectory planning problems subject to geometric obstacle avoidance, mainly arising in robotic motion planning. Due to their non-differential and non-convex structure, we identify the modeling of the collision avoidance constraints as an essential first step towards a feasible solution. Focusing especially on their efficient evaluation, we theoretically verify a well known collision detection strategy based on separating axis. Subsequently, we propose an extension to efficiently compute the penetration distance of two intersecting convex polygons.

Striving for a global solution, we derive a discrete dynamic programming algorithm based on a backward Semi-Lagrangian approximation proposed in [21] and generalize the corresponding convergence proof for a weaker set of assumptions. Characterizing the *curse of dimensionality* as our main issue in the arising numerical procedure, we investigate parallelisation, as well as adaptive state space refinement to lower its effect. In variable end time problems, we additionally exploit the representation in terms of optimal stopping to derive an efficient discrete dynamic programming strategy equipped with proofs on convergence. We proof a reasonable execution time of our accelerated algorithm on an industrial four dimensional robotic arm, as well as real time capability on a three dimensional task arising from autonomous driving.

As this methods never the less remains limited to tasks of dimension three to four, we subsequently propose a planning method combining dynamic and nonlinear programming to handle high dimensional systems. Exploiting dynamic programming to handle collision avoidance as well as nonlinear programming to deal with the high dimensional system dynamics, our strategy aims to combine best of both worlds. Based on the original strategy, we additionally propose an extension to allow consideration of moving environments. Considering the example of an acceleration controlled robotic manipulator with six degrees of freedom, we proof efficiency of our combined strategy as well as its variant compared to state of the art solvers. Finally, we demonstrate the extended version to allow trajectory planning towards a tumbling target within the context of space debris removal.

Zusammenfassung

In dieser Arbeit betrachten wir hochdimensionale Probleme zur Planung von Trajektorien, die explizit die Kollisionsvermeidung zwischen geometrisch modellierten Objekten mit einbeziehen. Sie treten in der Praxis in erster Linie im Bereich der Robotik auf. Da die Beschränkungen zur Kollisionsvermeidung in diesem Fall im Allgemeinen nicht differenzierbar und nicht konvex sind, ist die Modellierung und Auswertung dieser Nebenbedingung ein wichtiger Schritt auf dem Weg zu einer zulässigen Lösung. Wir betrachten daher eine bekannte numerische Methode zur Kollisionsidentifikation basierend auf separierbaren Achsen und leiten einen Beweis für ihre theoretische Korrektheit her. Im Anschluss erweitern wir unseren Algorithmus, sodass auch die Überlappung zwischen zwei konvexen Polygonen effizient berechnet werden kann.
Auf der Suche nach einer globalen Methode zur Lösung unseres Bewegungsplanungsproblems, wird der diskrete Algorithmus basierend auf der rückwärts Semi-Lagrange Approximation der Hamilton-Jacobi-Bellmann Differenzialgleichung aus [21] verwendet. Der dort vorgestellte Konvergenzbeweis, wird im Rahmen dieser Arbeit auf allgemeinere Annahmen erweitert. Um die Auswirkungen des *Fluchs der Dimensionalität* einzudämmen, werden sowohl Parallelisierung als auch adaptive Zustandsgitterverfeinerung diskutiert. Für Probleme mit variabler Endzeit, erweist sich zudem eine Darstellung in sogenannter *optimal stopping* Form als vorteilhaft um eine alternative, effiziente Strategie herzuleiten die die Konvergenzbedingungen erfüllt. Der aus der Kombination der Beschleunigungsstrategien entstandene Algorithmus ist in der Lage ein Industrieszenario basierend auf einem Roboterarm mit vier Freiheitsgraden in akzeptabler Zeit zu lösen und kann ein dreidimensionales Problem aus dem Bereich des autonomen Fahrens sogar in Echtzeit behandeln.
Da der Algorithmus basierend auf dynamischer Programmierung in der Praxis auf Probleme der Dimensionen drei bis vier limitiert bleibt, entwickeln wir im Rahmen dieser Arbeit außerdem eine Strategie, die dynamische und nichtlineare Programmierung kombiniert. Konkret verwenden wir dynamische Programmierung zur erfolgreichen Kollisionsvermeidung und nichtlineare Programmierung zur Berücksichtigung der hochdimensionalen System Dynamik mit dem Ziel die Vorteile beider Ansätze zu vereinen. Eine Erweiterung des daraus resultierenden Algorithmus, erlaubt zudem auch die Betrachtung von zeitabhängigen Umgebungen. In einem numerischen Industrieszenario verwenden wir einen Robotierarm mit sechs Freiheitsgraden und stellen fest, das sowohl unsere kombinierte Strategie, als auch die Erweiterung in der Lage sind bessere Ergebnisse zu liefern, als aktuelle Vergleichsstrategien. Zuletzt testen wir unsere erweiterte Strategie an einem dynamischen Szenario das die Planung von Bewegungen zur Erfassung von taumelndem Weltraumschrott beinhaltet.

Acknowledgments

First and foremost I would like to thank my supervisor *Matthias Gerdts*, not only for giving me the opportunity to work as a doctoral candidate in his group but also for his endless patience and support on the road towards this thesis. Thank you for giving me the freedom to follow my own ideas and interests while always providing an open door for trouble shooting.
I also want to extend my appreciation to Jürgen Pannek for taking the time to read this manuscript carefully as a second corrector.
A warm thank you goes to all of the guys from our *lunch crew* for lots of inspiring talks and motivating words that made the time at the office as enjoyable as it was.
Especially, I want to point out my former colleague *Andreas*, who introduced me into the world of dynamic programming. Thank you, not only for answering a thousand stupid questions, but also for guiding me through my first steps in academic writing.
Additionally, I want to express my deep gratitude to my former office buddy and co-author *Alberto*. It was a pleasure to work with a human lexicon on optimal control literature in my back and your skills in academic writing have always been a role model for me.
Thank you, to my colleague *Björn* for providing his expert knowledge on mathematic proofs as well as to *Thomas* for all kinds of trouble shooting in practical experiments.
I would like to extend my gratitude to my family, Oma, Papa und Pati, who always encouraged and supported me to pursue my goals including this thesis.
Finally, Flo: THANK YOU !

Contents

1 Introduction **3**

1.1 Related Work 5

1.2 Contributions and Outline 7

2 Geometric Collison Models **9**

2.1 Collision Avoidance in Optimization 11

2.2 Collision Avoidance Evaluation 13

2.2.1 Collision Detection 13

2.2.2 Separation Distance Evaluation 13

2.2.3 Penetration Distance Evaluation 15

2.2.4 SAT for Penetration distance 15

2.3 Numerical Comparison 21

2.3.1 Collision Detection 21

2.3.2 Distance Evaluation 22

3 Dynamic Programming **25**

3.1 The method of Dynamic Programming 27

3.1.1 Restricted State Space 30

3.1.2 Numerical Approximation 31

3.1.3 Time and Control Approximation 32

3.1.4 Full Discretisation 35

3.1.5 DP Algorithm 40

3.2 DP for Trajectory Planning 42

3.3 DP with Optimal stopping 43

3.3.1 Theoretic Value function Properties 44

3.3.2 Numerical Approximation 56

3.4 Iterative State Space Splitting 62

3.4.1 Geometry based State grid refinement 62

3.5 Parallelization using CPU/GPU 66

3.5.1 DP Parallelization Capabilities 66

3.5.2 DP Parallelization in practice 67

3.6 Experiment: KUKA YouBot Trajectory Planning 70

3.6.1 Setup . . . 70
3.6.2 Optimal Stopping . . . 72
3.6.3 Parallelisation . . . 73
3.6.4 Adaptive State Space Refinement . . . 75
3.6.5 Conclusion . . . 77
3.7 Experiment: Autonomous Driving with DP based MPC . . . 78

4 Dynamic and Nonlinear Programming 81
4.1 Low Dimensional Level . . . 83
4.2 Inverse Dimensional Mapping . . . 85
4.3 Trajectory Generation . . . 86
4.4 Penalty adaptation . . . 87
4.5 Iterative Dynamic and Nonlinear Programming (IDNP) Algorithm . . . 89
4.6 Extension: Non-autonomous Systems . . . 90
4.7 Experiment: Kinova Mico Trajectory Planning . . . 94
4.7.1 Experiment Setup . . . 94
4.7.2 Algorithm Setup . . . 96
4.7.3 Results . . . 98
4.8 Experiment: Satellite Trajectory Planning . . . 104
4.8.1 Experiment Setup . . . 104
4.8.2 Algorithm Setup . . . 107
4.8.3 Results . . . 108

5 Summary and Future Work 113

Chapter 1

Introduction

Throughout the last centuries robotic systems were designed for a huge range of applications to execute previously infeasible tasks or simply take off work from peoples hands and make their life easier. Starting from the sector of healthcare, there are not only robotic arms assisting disabled people within their everyday life [146, 30], but also robotic surgery systems allowing human surgeons to exceed their natural limits regarding precision[107, 117] and force [129]. Also in industrial applications, robotic arms are used to execute assembly[4], welding [148] or cutting [43] tasks on a production line, while mobile robots can transport components within a factory[3]. In the area of space, systems especially designed for the tasks of on orbit servicing and space debris removal aim to increase lifetime and therefore sustainability of current satellite systems[118]. While this list of examples could be extended for pages, automated motion is ether considered beneficial or essential in most of them, making robotic motion planning an important challenge in current research.

Starting from the origins, we can characterize Shakey [93] as the first mobile robot to allow for fully autonomous motion planning. Combining camera based environment perception as well as image processing, with the graph based path planning algorithm of A* [72], it was able to perform simple tasks, such as locating and pushing a rectangular target object. Throughout the following years, the robotic systems, alongside with the corresponding planning algorithms became more complex. Initially using deterministic and complete strategies based on graph search [101, 95], their practical application was usually limited to robotic systems acting in two or three dimensional configuration spaces due to computational complexity. Facing this problem mainly arising from full sampling of the configuration space, the so-called sampling based methods build on the introduction of random samples in space to find a valid path connecting initial and goal configurations. Well known examples are the strategies of Probabilistic Roadmaps (PRMs) [87] as well as Rapidly Exploring Random Trees (RRT) [96]. While these early strategies were mainly designed to examine a feasible path for the manipulator with respect to the geometric environment, more recent strategies arising from the field of optimization [136, 77], allow consideration of complex constraints and therefore an integration of the systems dynamics. Following their spirit, we formulate the general trajectory planning task to be faced within this work as an optimal control problem



$$
\begin{aligned}
\text{Minimize } & \int_0^{t_f} c(x(t), u(t))dt && && (1.1)\\
\text{subject to } & \dot{x}(t) = f(x(t), u(t)) && a.e. \text{ in } [0, t_f]\\
& g(x(t)) \geq 0 && \forall t \in [0, t_f]\\
& b(x(t_f)) = 0\\
& x(0) = z\\
& x(t) \in X,\ u(t) \in U && \forall t \in [0, t_f]\\
& t_f \in [0, T]
\end{aligned}
$$

with respect to states $x : [0, t_f] \to X$ and controls $u : [0, t_f] \to U$ from bounding sets $X \subseteq \mathbb{R}^{n_x}$ and $U \subseteq \mathbb{R}^{n_u}$ using state and control dimensionality $n_x, n_u \in \mathbb{N}$. Depending on the structure of the system dynamics, we let the state function x ether directly track the robotic systems configuration, or in case of a second order differential equation describing the dynamics, its configuration and velocities combined into a single vector. As motion planning in practical applications is usually not limited to a certain final time, but allows the target task to be executed within a certain time interval, we search for an optimal end time t_f within time bounds $[0, T]$ here. We let $f : X \times U \to X$ describe the dynamics of our system expressed as a first order differential equation, highlighting again that an appropriate transformation of the state space allows also consideration of practically relevant second order system dynamics, such as toque control. Function $g : X \to \mathbb{R}^{n_g}$ forms the possibly multidimensional inequality constraint of our problem that is used within the context of this thesis to represent geometric collision avoidance. Nevertheless, the problem structure allows for an extension of the function to cover additional inequality constraints. We assume our motion plan to initially start from a given state $z \in X$, while fulfilling an equality constraint with respect to function $b : X \to \mathbb{R}^{n_b}$ at the end. In contrast to the use of an explicit final configuration constraint, this formulation of a goal function allows integration of more practical constraints i.e. a goal region for the end-effector of a robotic arm in configuration space planning. Finally, we can filter feasible solutions according to various application specific criteria i.e. minimum time or minimum control using incremental objective function $c : X \times U \to \mathbb{R}$.



1.1 Related Work

Before we provide an overview on strategies from current literature to solve Problem (1.1), we briefly want to highlight two important problem properties: On the one hand, due to their physical setup, many robotic systems, i.e. robotic arms, provide a lot of degrees of freedom. As a consequence, their explicit modeling within the arising optimal control problem often results in large state and control space dimensionalities. On the other hand, many robotic applications require accurate navigation of the manipulator around the environment. Therefore, a geometric instead of a spherical representation of the objects involved becomes necessary. The arising distance functions included into the collision avoidance constraints of Problem (1.1) are therefore in general considered highly-nonlinear and non-differentiable [64].

Regarding the structure of our trajectory planning Problem (1.1), general purpose nonlinear solvers such as Ipopt [147], SNOPT [66] or WORHP [24] can be considered to find an optimal solution. While designed to handle nonlinear constraints, the consideration of highly non-convex and even non-differential collision avoidance constraints often causes these solvers to struggle in practice.

Differential reformulations of the collision avoidance constraints using polyhedral object representations, such as proposed in [62] based on Farka's Lemma as well as in [157] based on the dual formulation of the distance function, are designed to overcome this issue. While the latter strategy has already been further optimized for specific geometric shapes in [70, 44], all these reformulations require a large number of additional variables to represent a single polyhedral set, subject to collision avoidance. Consequently, their application in practice is, especially in crowded environments, not always considered beneficial.

As an alternative, the method of CHOMP[126] optimizes a cost function balancing between the properties of smoothness and collision avoidance of the resulting trajectory, without any additional constraints. By approximating the collision avoidance constraints using interpolation of pre-computed signed distance samples, this strategy manages to successfully include geometric obstacle avoidance properties into gradient based optimization without expensive reformulations. It is extended to a stochastic framework in [86]. Building upon their idea of approximating collision avoidance constraints, TrajOpt [136] exploits the method of sequential convex programming using a linear approximation of the signed distance function as a penalty term. In the following, sequential convexification of planning problems with non-convex constraints became a popular approach in the robotics community [19, 103].

Exploiting the time structure of the trajectory planning problem, indirect strategies such as differential dynamic programming can propose an opportunity to accelerate convergence compared to general purpose solvers. While the original approach presented in [106] was not able to handle others than dynamics constraints, multiple extensions have been proposed over the years addressing this issue [143, 154, 76]. In practice, strategies based on differential dynamic programming do still lack reliable constraint handling in some scenarios, but are especially popular due to their efficient computation time[142, 55].

All the methods based on optimization discussed so far are highly dependent on the initial guess



and provide, if any, local solutions. In contrast, global path finding strategies such as graph search [101] or sampling based methods [87, 96] are (probabilistically) complete and not at risk to get stuck in local minima. While there exist multiple attempts for so-called kinodynamic extensions using precomputed motion primitives [132, 120], the original, efficient strategies are unfortunately not able to consider system dynamics. A possible alternative provides the method of dynamic programming based on Bellmann's principle of optimality[10], as it is able to handle highly non-linear collision, as well as dynamics constraints, while striving towards a global solution [9]. Unfortunately, it suffers from the curse of dimensionality [14], characterizing its asymptotic runtime to be exponentially dependent on state and control dimensionality. As a result, even though there exist multiple approaches to lower the curses impact[33, 111], application of the method of dynamic programming in high dimensional systems is in practice considered ether very expensive or impossible.

During the last years, methods combining multiple strategies together, ideally exploiting best of their properties have gained popularity. While strategies proposed in [82, 153] exploit capabilities of neuronal networks to make configuration space sampling in sampling based planners more efficient, [119] relies on a learning based approach to accelerate the generation of motion primitives within their kinodynamic extension. Interleaving graph search with trajectory optimization, [112] proposed a global kinodynamic planner for the specific application of Quadrotor flight. In the same spirit of exploiting benefits from local and global planners, the Graphs of Convex Sets (GCS) strategy proposed in [105] models a graph out of safe, interleaving, configuration space regions and uses a mix of graph search and convex optimization to achieve a globally-optimal, collision-free trajectory.



1.2 Contributions and Outline

Starting from our robotic motion planning Problem (1.1) we already identified the geometric collision avoidance as an essential component. As its formulation can heavily influence the performance of the motion planning process, we dedicate Chapter 2 to approaches to model and evaluate proximity between convex objects. Initially recapitulating the concept of a signed distance function, we subsequently provide an overview on state of the art strategies to integrate the originally non-differential structure of geometric collision avoidance into gradient based optimization, exploiting i.e. reformulation with respect to additional variables or approximation. Bearing in mind motion planning strategies independent from the gradient, we shift the focus from smooth modeling to efficient evaluation of collision detection, separation distance and signed distance in the proceeding sections. As a first contribution, we therein proof correctness of the efficient method of separating axis (SAT) for collision checking in convex polytopes, as it is heavily exploited in practical algorithms so far, but was to the authors knowledge never theoretically verified. Additionally, we propose an extension of the existing (SAT) algorithm to allow not only collision checking, but also evaluation of the penetration distance. Initializing a small geometric example, we proof numerical efficiency of our extended SAT strategy compared to state of the art algorithms.
Striving for a globally optimal motion planner, Chapter 3 of this thesis investigates and combines strategies to lower the curse of dimensionalities impact on the method of dynamic programming. Initially deriving a discrete numerical algorithm, we extent the convergence guarantees provided for a Semi-Langrangian discretisation scheme with linear interpolation in [21], to a problem formulation using weaker assumptions. Subsequently, we exploit the concept of optimal stopping and proof the convergence properties of our discretisation to remain valid. The arising discrete dynamic programming algorithm allows handling of a free end time problem without an additional state or control dimension. To further improve efficiency of the numerical method, we investigate adaptive state space refinement, recapitulating the geometric state space splitting strategy presented in

[128] Rebecca Richter, Andreas Britzelmeier, and Matthias Gerdts. *An adaptive mesh dynamic programming algorithm for robotic manipulator trajectory planning.* 2023 European Control Conference (ECC). IEEE, 2023.

Additionally, we propose an intuitive parallelisation strategy for our discrete dynamic programming algorithm, exploiting capabilities of recent multi-kernel central processing units (CPU) as well as graphics processing unit (GPU) architectures. Finally, we demonstrate effectivity of our acceleration strategies on a robotic manipulator motion planning example considering four degrees of freedom, as well as in real-time path planning required for autonomous driving.
Following the trend in current literature to combine existing planners, the main contribution of this thesis arises from Chapter 4 where we propose an algorithm combining dynamic and nonlinear programming based on our work in



[22] Britzelmeier, Andreas, Alberto De Marchi, and Rebecca Richter. *Dynamic and nonlinear programming for trajectory planning.* IEEE Control Systems Letters 7 (2023): 2569-2574.

As the original algorithm is limited to autonomous problems, we additionally provide, based on our work in

[127] Richter, Rebecca. *Dynamic and Nonlinear Programming for Trajectory Planning in Moving Environments.* IFAC-PapersOnLine 59.1 (2025): 265-270,

an extension to handle problems involving time dependent collision avoidance as well as terminal restrictions while at the same time improving the overall strategies robustness. We proof numerical efficiency of both variants on a task involving second order control of a robotic manipulator with seven degrees of freedom as well as geometric collision avoidance. Finally, we use the extended variant for non-autonomous problems, to plan a feasible motion for a chaser object in space towards a tumbling piece of space debris within the scope of project SeRANIS [90].
At the end, we summarize our main contributions and discuss ideas for future work in Chapter 5.



Chapter 2

Geometric Collison Models

Collision avoidance is a problem arising from most practical applications of trajectory planning including autonomous driving and space manipulations as well as industrial and robotics applications. In any case, a successful collision avoidance requires for every object in the scene to not collide with each other. While there exist various numerical algorithms in literature for pure collision checking returning yes or no for given objects [138],[134],[131] this can also be modeled with respect to some distance function measuring the space between objects. In contrast to the Hausdorff distance metric, the Euclidean distance function returns zero on overlapping sets. Consequently, for successful collision avoidance it is requested to be greater than zero for all object pairs in the scene. Focusing on objects that can be represented by a collection of convex sets, the Euclidean distance function can, according to [64], be described by

$$d^+(A,B) = \min_{t \in A-B} ||t||_2$$

for convex sets $A, B \subset \mathbb{R}^n$ and their Minkowski Difference

$$A - B \coloneqq \{a - b | a \in A, b \in B\}.$$

Following [27], it can be extended by the penetration distance,

$$d^-(A,B) = \min_{t \in (A-B)/\text{int}(A-B)} ||t||_2$$

using $\text{int}(A)$ as the interior of set A, to the Signed distance function

$$d(A,B) = \min(d^+(A,B), -d^-(A,B))$$

to additionally measure the overlap on intersecting objects. In the context of motion planning, this can be useful to find least intrusive trajectories in infeasible settings [157] or to penalize infeasible regions [22].

As pointed out in [64], the resulting (signed) distance function, as well as results from collision detection methods, are in general highly non-linear and non-differentiable. Regarding the



trajectory planning task, this causes gradient or hessian based nonlinear optimization solvers to struggle with problems containing them. As a result, multiple strategies on reformulation and approximation of the collision avoidance constraint and the distance function have been proposed and will be reviewed within Section 2.1. On the other hand, sampling or search based planning methods as well as dynamic programming can handle non-differentiable constraints and simply require efficient ways to evaluate them. Depending on the geometric structure of the considered objects, Section 2.2 presents existing algorithms to efficiently evaluate collision detection as well as separation and penetration distance. Furthermore, we introduce a strategy to use the method of Separating Axis (SAT) known from collision detection in Computer graphics [134] to efficiently evaluate the penetration distance on simple polyhedra. Overall, this chapter is meant to illustrate the strong dependency between chosen planning method and efficient collision avoidance, underlined in Section 2.3 by a numerical comparison of most common algorithms.



2.1 Collision Avoidance in Optimization

As already pointed out above, collision avoidance constraints based on distance formulations, are in general non-smooth, causing trouble when given to hessian or gradient based solvers. Exploring ways to integrate collision avoidance into the problems of such solvers is therefore a subject to ongoing research with multiple solution strategies suggested in current literature, each bringing its pros and cons.

The work presented in [97] proposes differentiable collision avoidance constraints especially designed for the case of two rectangular bodies, based on geometric and trajectory motion properties. In [62] the more general case of collision avoidance on two convex polyhedra, specified by linear systems of their faces, is reformulated using Farkas Lemma. They achieve exact collision checking from smooth differentiable constraints. This approach, as well as its more recent extension in [73], is however limited to collision detection. In contrast, the differentiable reformulation based on the dual formulation of the distance function for convex sets presented in [157], is not only able to consider the separation distance, but additionally models penetration and is therefore, when used within optimal trajectory planning, able to find "least-intrusive" solutions. The proposed constraints have already been applied in real world scenarios [137],[156] and optimized with respect to time efficiency on specific groups of geometric shapes [70], [44]. Nevertheless, the main drawback of these exact reformulations, is the large number of additional constraints and states required. Considering the case of two convex polyhedra of dimension n described in total by ether N vertices or M linear constraints, the method from [62] results in M additional variables and $2M+1$ constraints, the one from [157] in M variables and $2+M+n$ constraints, and still $2+n$ additional variables and $2+N$ additional constraints are used in [70]. As environments of increasing geometric complexity require large numbers of collision checks, the enlargement of the problem statement resulting from collision avoidance reformulation can slow down the solution process significantly. The exact differentiable collision avoidance strategy based on the separating hyperplane theorem recently proposed in [53] is facing this issue, lowering the required additional variables on three dimensional polygons to four independent from the shapes complexity. However, it is limited to collision detection without any indications on penetration or separation distance

During their theoretical analysis in [64], Gilbert et. al. have shown that the distance function is actually differentiable if considered between a convex and a strictly convex set instead. Consequently, by outer approximating one of their 2D target sets with a strictly convex set, they achieved a differentiable distance representation. Also algorithms proposed more recently [11],[47] take up the idea to 'bulge' target objects and extend it to the three dimensional space. Unfortunately, the resulting differentiable distance function lacks a closed form representation, consequently also for the derivative. During optimization, it can therefore not directly be included into the problem statement as a constraint, but requires numerical algorithms for evaluation.

Considering a Euclidean ball target and a polyhedral obstacle, collision avoidance can also be represented with respect to the objects Minkowski sum [71]. More precisely, if the sum does



not contain the balls center representing the current target position, the objects do not suffer from collision and vice versa. Unfortunately, the Minkowski sum of two convex objects is in general non-smooth and can not be written in a closed form. As a results, Guthrie et. al.[71] outer approximate the Minkowski sum with a smooth closed form representation, which allows to directly include the resulting differential collision avoidance constraint into an optimization problem. Numerical results prove this approximated collision avoidance to be very efficient compared to [157]. Nevertheless, it is yet limited to targets represented by collections of spheres.
Another way of approximating the signed distance is proposed in [136] building a penalty from a linear approximation of the signed distance function, considered for collision avoidance within sequential convex programming on convex polyhedra. Collision avoidance in [126] also relies on penalization, this time based on a potential field approximated from signed distance evaluations on equidistant state space grids. While both of these approaches provide impressive numerical results and are common benchmarks used within robotic motion planning in practice [36],[123], they lack theoretical analysis of the error resulting from the distance approximation.



2.2 Collision Avoidance Evaluation

Instead of focusing on smoothness and differentiability, this section contains a selection of methods to efficiently evaluate collision avoidance between convex sets. The first part will cover boolean collision checking algorithms, only indicating overlap appearance. Especially in hierarchical environment representations [67],[69], they are used to filter candidate objects for more expensive exact distance evaluations. Afterwards, an overview on most common numerical algorithms for separation as well as for penetration distance computation will be given. Finally, we introduce a strategy to efficiently compute the penetration distance for Polygons and Polyhedra with small numbers of faces, based on the method of separating axis [134].

2.2.1 Collision Detection

Depending on the geometric structure of the considered objects, different algorithms are proposed. Considering simple polytopic shapes in two or three dimensional space, the method of separating axis [134] is usually considered most efficient. The algorithm identifies a finite set of hyperplanes/axis based on the objects geometric properties and iteratively checks each of them for separating the candidate shapes. It terminates on successful separation found or implies overlapping objects when every candidate fails. Unfortunately, the maximum number of separating axis candidates is quadratically dependent on the number of polygon features (faces, edges), making the method computationally expensive on more complex structures. In these cases, incremental methods also used for distance computation such as Gilbert-Johnson-Keerthi Algorithm (GJK) [65] or Lin-Canny Algorithm (LC) [98] as well as derivatives especially designed for collision detection [12],[138],[80] can be used, since their computational complexity is not directly related to the number of object features. Moving from convex polyhedra to quadratic structures, the ASC algorithm proposed in [149] is an efficient opportunity for collision detection on ellipses and ellipsoids. Based on a closed form notation of the objects Minkowski Sum, Ruan et. al. in [131] and [130] recently presented methods for collision detection between ellipsoid and superquadrics, and two superquadrics respectively.

2.2.2 Separation Distance Evaluation

As also done within the review in [85], we divide the field of existing separation distance evaluation methods into the ones based on geometric properties and those based on optimization, starting with the latter. In the early work of Brobrow [15], the distance evaluation between two convex polyhedral sets P and Q

$$P \coloneqq \{p \in \mathbb{R}^n | \langle p, n_i^P \rangle \leq d_i^P \, i = 1, \ldots, k_i^P\}$$
$$Q \coloneqq \{q \in \mathbb{R}^n | \langle q, n_i^Q \rangle \leq d_i^Q \, i = 1, \ldots, k_i^Q\}$$



defined with respect to normal vectors $n_i \in \mathbb{R}^n$ and face positions $d_i \in \mathbb{R}$ is formulated as a convex optimization problem

$$\min \frac{1}{2}||p-q||^2$$
$$\text{subject to } p \in P, q \in Q$$

with quadratic objective function and linear constraints. Rosen's gradient projection algorithm is used to obtain the global solution. As pointed out in [155] this solving process can end up in a convergence problem if the Kuhn-Tucker conditions, used as stopping criterion within the iterative procedure, are satisfied alternatively on each polyhedron. To overcome this issue, the algorithm proposed by [155] uses an additional optimal search direction. An extension to implicit surfaces, containing in addition to polyhedrons for example ellipsoids and parabolic surfaces, is proposed in [35]. As this results in non-linear constraints $p \in P, q \in Q$, an interior point solver is used to solve the optimization problem.

In the field of geometry based algorithms, we find two different strategies on evaluating the separation distance: feature and vertex based algorithms. The well known LC Algorithm presented in [98] is based on the features (vertices, edges, faces) of two 3D convex polyhedra. Starting from an arbitrary feature pair, the Voronoi region of each of the features is used to check if it contains the closest points of the polygons. If not, the algorithm adapts the pair to use a closer, neighboring feature on one of the polygons within the next iteration. The algorithm is shown to be complete, as well as efficient (linear runtime with respect to the product of polygon features) especially when provided a good initial guess (almost constant runtime). Unfortunately, this original algorithm, lacks termination in the case of intersecting polyhedra as well as robustness on degenerate configurations. These drawbacks are faced within the improved V-Clip algorithm [108].

On the other hand, the GJK algorithm proposed in [65] is based on the vertices of the polyhedra. Starting from their Minkowski difference, the algorithm iteratively searches among the resulting points for those that form the convex simplex closest to the origin. In case of originally separated polygons, the resulting minimal origin to simplex distance matches exactly their separation distance. Otherwise, the origin is either located inside or on the boundary of the Minkowski difference simplex, indicating penetration or contact of the original polyhedra respectively. While this original algorithm already provides a runtime of linear order, the enhanced variant, based on additional edge information, introduced by Cameron [28] is able to run within almost constant time on good initial guess. Additionally, on overlapping polyhedra, the resulting witness points can as well as the resulting witness features from [108] be used to give an approximation of the penetration distance. The GJK extension provided in [145] demonstrates not only applicability to a more general set of convex objects including i.e quadric primitives, but also provides a follow up routine based on GJK's result to determine the exact penetration distance on intersection. More recent algorithm improvement suggestions can be found in [109], replacing the distance subalgorithm for shorter runtime and improved robustness and [110], bringing GJK back to its



origins in optimization for incorporating Nesterov accelerations and therefore lower the number of iterations. Its efficiency, as well as its applicability to general convex objects, make GJK the most common algorithm used for distance computation in various applications [159], [37].
While the aforementioned algorithms require discrete vertices or polyhedral features, such a representation is only an approximation for ellipses or quadric surfaces. An exact distance estimation in these cases can be achieved by solving the nonlinear system of equations arising from the common normal concept [99] or even more efficiently when combined with a closed form Minkowski sum representation [130].

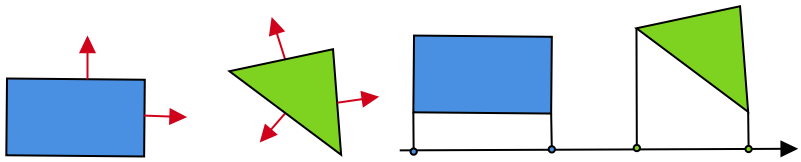
2.2.3 Penetration Distance Evaluation

Compared to the large literature on the separation distance, the number of proposals for evaluating the penetration distance is small. Regarding convex polyhedra, the methods proposed in [27],[89] and [145] all exploit the equality of the penetration distance and the minimum distance from the origin to the border of their Minkowski difference. The latter one is proposed as an extension to the GJK procedure, evaluating the penetration distance based on GJK's resulting simplex. This combination of GJK and the so called Expanding Polytope Algorithm (EPA) is due to its incremental structure and computational efficiency most commonly used for signed distance evaluations on polyhedra. Using the Minkowski Portal Refinement (MPR) Algorithm developed in [138] for collision detection, evaluates an approximation of the penetration distance more efficiently [56], as it does not require preceding separation evaluation. Furthermore, we propose an algorithm based on the method of separating axis [134], that is especially designed to evaluate the pure and exact penetration distance on simple polyhedra.
As the penetration and separation part of the Euclidean signed distance function always required separate description and evaluation, a more general measure on distance called *growth distance* was introduced in [113]. While its original evaluation by convex optimization was computationally expensive, a more efficient geometry-based algorithm was proposed in [159].

2.2.4 SAT for Penetration distance

Following the naming in [134], we define a *separating axis* of convex polytopes $P, Q \subset \mathbb{R}^n$ with $n = 2, 3$ as a direction vector $v \in \mathbb{R}^n$, where the projections of P and Q on v do not intersect. The method of SAT [134] builds on the separating axis theorem implying separation if and only if a separating axis is found. Furthermore, instead of searching all possible directions, checking a finite set of axis arising from geometric properties is according to [134] sufficient. It is obtained in dimension two from the edges unit normals, while in three dimensional space the normalized cross products of the polygon edges are used in addition to the unit normals of the faces. Figure 2.1 illustrates the separation axis candidates on polygons in 2D. While this statement is exploited extensively within practical algorithms for collision checking, we were not able to find a formal proof within current literature. Consequently, we aim to mathematically justify the procedure in the following





(a) (b)

Figure 2.1: Illustration of the separating axis candidate directions for collision detection between a rectangle and a triangle (red arrows in (a)) as well as the minimum and maximum values arising from their projection on a separating axis (b).

Lemma 1 *Let $P, Q \subset \mathbb{R}^n$ be two convex polyhedral sets such that $P \cap Q = \emptyset$. Then, there exists a separating axis v that is perpendicular to one of the facets of the Minkowski sum $P + (-Q)$*

Proof. From convex set properties, we can observe that the Minkowski sum

$$K \coloneqq P + (-Q) = \{p - q \mid p \in P, q \in Q\}$$

is itself a convex polyhedral set [150] and can therefore be represented by a collection of half-spaces

$$K \coloneqq \{x \in \mathbb{R}^n | \langle x, n_i \rangle \leq d_i \quad i = 1, \ldots, m\}.$$

Without loss of generality, we can assume n_i to represent the outward pointing normal vectors, on each of the faces of the resulting polytope. Furthermore, we can obtain $0 \notin K$ from $P \cap Q = \emptyset$. As a result, there exists at least one half-space H_i

$$H_i \coloneqq \{x \in \mathbb{R}^n | \langle x, n_i \rangle \leq d_i\}$$

that does not contain zero. Let point $v \in \mathbb{R}^n$ represent the closest point out of such a half space to the origin

$$v = \arg\min_{x \in H_i} \|x\| \tag{2.1}$$

with $\|v\| > 0$ as zero is not part of the half-space. Furthermore, let unit direction vector $w \in \mathbb{R}^n$ be perpendicular to normal n_i and therefore parallel to the half spaces separating hyperplane. As a result we can shift v around in this direction without ever leaving the half-space

$$v - sw \in H_i \quad \forall s \in \mathbb{R}.$$

Using equation (2.1) we can derive

$$\begin{aligned} \|v\|^2 \leq \|v - sw\|^2 &= \langle v - sw, v - sw \rangle = \|v\|^2 - 2s\langle v, w \rangle + s^2 \\ 0 &\leq -2s\langle v, w \rangle + s^2 \end{aligned}$$



and by choosing $s = \langle v, w \rangle$ above expression evaluates to $0 \leq -\langle v, w \rangle^2$. Please note that this inequality can only be fulfilled in case of $0 = \langle v, w \rangle$. We can therefore obtain for any $w \in \mathbb{R}^n$ perpendicular to normal n_i that it is also orthogonal to v. Consequently, v and n_i are parallel. Additionally, as n_i is pointing outward of H_i and is of unit length, while v is pointing inward of H_i, we can even write $n_i = -\frac{v}{\|v\|}$.
It remains to proof that v is a separating axis of Polygons A,B. Due to the definition of K

$$\langle p - q, n_i \rangle \leq d_i$$

holds for any $p \in P$ and $q \in Q$. Exploiting property $n_i = -\frac{v}{\|v\|}$ from above, we can furthermore derive

$$\begin{aligned} \langle p - q, -\frac{v}{\|v\|} \rangle &\leq d_i \\ -\frac{1}{\|v\|}(\langle p, v \rangle - \langle q, v \rangle) &\leq d_i \\ \langle p, v \rangle &\geq -d_i \|v\| + \langle q, v \rangle \end{aligned}$$

As $0 \notin H_i$ implies $d_i < 0$ we can follow

$$\langle p, v \rangle > \langle q, v \rangle \quad \forall p \in P, q \in Q.$$

Consequently, the projections of P and Q on axis v do not intersect, characterizing v as a separating axis. □
In a next step, we aim to characterize the normal vectors of the half-spaces constructing the Minkowski sum $P + (-Q)$ to be parallel to the separating axis candidates suggested within SAT algorithm in two and three dimensional space. The following Lemma heavily depends on results from [100] and [27].

Lemma 2 *Let $P, Q \subset \mathbb{R}^n$ be two convex polyhedral sets of dimension $n = 2, 3$. Then, for every facet of their Minkowski sum $P + (-Q)$, there exists at least one perpendicular separating axis within the set of candidate axis arising from SAT method.*

Proof. Initially we can observe from the definition of the Minkowski sum

$$K \coloneqq P + (-Q) = \{p - q \mid p \in P, q \in Q\}$$

that every point on its boundary $k \in \partial K$ represents a translation such that the sets $P, Q + k$ are exactly in contact. Consequently, there exist $p \in \partial P$ and $q \in \partial Q$ with $k = p - q$ [39]. Furthermore, in [100], boundary ∂K in case of $n = 2$ is generated from sliding polygon Q around P while keeping them in touch but not overlapping. The arising translations of Q form the boundary of K. Sliding counterclockwise around P, we either slide a point af P along an edge



of Q or a point of Q along an edge of P. Consequently the arising boundary of K is formed by a succession of rescaled and translated edges of P and Q [100],[39]. As the SAT method suggests all the edges normals arising from P and Q as separating axis candidates, this closes the proof for case $n = 2$.
Using the same sliding strategy, [100] also characterizes the faces of K in case of $n = 3$ to arise from different modes of contact of P and Q:

- A part of Q sliding against a face of P
- A part of P sliding against a face of Q
- An edge of P sliding against an edge of Q

As a result, following [27] the convex polyhedral set K can be constructed out of a collection of half-spaces where the corresponding normal vector of every separating hyperplane is

- orthogonal to a face of P
- and/or orthogonal to a face of Q
- and/or orthogonal to a plane spanned by an edge of P and an edge of Q

Regarding the method of SAT in dimension three, we consider all face unit normals in P and Q as well as cross products of edge combinations from P and Q as separating axis candidates. Thus, our proof is completed. □
We now formulate the fundamental theorem of the SAT method with $D_{SAT}(P, Q) \subset \mathbb{R}^n$ representing the finite set of candidate axis arising from geometric object properties:

Theorem 3 *Let $P, Q \subset \mathbb{R}^n$ be two convex polyhedral sets of dimension $n = 2, 3$.*

$$P \cap Q = \emptyset \iff \exists \text{ separating axis } v \in D_{SAT}(P, Q)$$

Proof. For the backward direction, let $v \in D_{SAT}(P, Q)$ represent a separating axis and therefore

$$\langle p, v \rangle < \langle q, v \rangle \quad \forall p \in P, q \in Q.$$

Consequently, P and Q are separated and thus $P \cap Q = \emptyset$.
For the second case, we initially assume P and Q separated. Using Lemma 1 we can follow existence of a separating axis w orthogonal to a facet of the Minkowski sum $P + (-Q)$. Furthermore, with Lemma 2 we can even follow, that there exists a vector $v \in D_{SAT}(P, Q)$ parallel to w and thus v is also a separating axis of P and Q. □
The resulting algorithm for collision detection iteratively evaluates all candidate axis projections and returns separation on separating axis found and intersection otherwise.



In order to extend this algorithm to give the penetration distance, instead of just checking for projection overlay on every candidate axis, we evaluate the minimum intersection of the projected polytopes. Let us therefore write the projected polytope sets along any direction $d \in \mathbb{R}^n$ as

$$A_d = \left\{ \left\langle p, \frac{d}{|d|} \right\rangle | p \in P \right\}, \quad B_d = \left\{ \left\langle q, \frac{d}{|d|} \right\rangle | q \in Q \right\}$$

with minimum $a^- = \min_{a \in A_d} a$, $b^- = \min_{b \in B_d} b$ and maximum $a^+ = \max_{a \in A_d} a$, $b^+ = \max_{b \in B_d} b$ values. Consequently, we define the minimum overlap of the projections as directional penetration,

$$\mathrm{dp}(A_d, B_d) = \begin{cases} 0 & a^- > b^+ \text{ or } a^+ < b^- \\ \min(a^+ - b^-, b^+ - a^-) & \text{else} \end{cases} \tag{2.2}$$

holding zero on separated projections and the minimum translation required in positive or negative direction along d to bring them from overlap to contact otherwise. Please note that as we project along normalized direction vectors we successfully prevent axis length indicated scaling on the resulting directional penetration. As a result, *dp* holds the minimum Euclidean translation along direction d to bring the polygons from penetration into contact. Finally, we minimize over directional penetrations on all SAT candidate axes and receive the penetration distance. Thus, the result of our algorithm can be formalized into

$$d^-_{SAT}(P, Q) = \min_{d \in D_{SAT}} \mathrm{dp}(A_d, B_d).$$

In the following, we prove that our algorithms result indeed matches the penetration distance.

Theorem 4 *Let $P, Q \subset \mathbb{R}^n$ be two convex polyhedral sets of dimension $n = 2, 3$ that are in a collision (contact or intersection). Then, it holds that*

$$d^-(P, Q) = d^-_{SAT}(P, Q).$$

Proof. Proof by contradiction:
As the overall penetration distance is given by the minimum translation required in any direction to bring the objects out of intersection, we can reformulate it with respect to the directional penetration from Equation (2.2) as

$$d^-(P, Q) = \min_{d \in \mathbb{R}^n} \mathrm{dp}(A_d, B_d).$$

As a result, we can directly obtain from $D_{SAT} \subset \mathbb{R}^n$ that

$$d^-(P, Q) = \min_{d \in \mathbb{R}^n} \mathrm{dp}(A_d, B_d) \leq \min_{d \in D_{SAT}} \mathrm{dp}(A_d, B_d) = d^-_{SAT}(P, Q)$$



holds. We show the remaining inequality $d^-(P,Q) \geq d^-_{SAT}(P,Q)$ by contradiction:
Assume $d^-(P,Q) < d^-_{SAT}(P,Q)$. Then, there exists some $\epsilon > 0$ with

$$d^-(P,Q) + \epsilon \leq d^-_{SAT}(P,Q).$$

Since the penetration distance is defined as the minimum directional overlap, we can deduce

$$\mathrm{dp}(A_d, B_d) \geq d^-(P,Q) + \epsilon \qquad \forall d \in D_{SAT} \tag{2.3}$$

and $d_{\min} \notin D_{SAT}$ for all directions of minimum overlap

$$d_{\min} \in \mathbb{R}^n : d^-(P,Q) = \mathrm{dp}(A_{d_{\min}}, B_{d_{\min}}).$$

We apply an Euclidean translation

$$t = \frac{d_{\min}}{|d_{\min}|} \cdot (d^-(P,Q) + \epsilon)$$

to Polyhedron P. By definition of the penetration distance, the resulting polyhedra $P + t$, Q are no longer in a collision but separated.
On the other hand, as $d_{\min} \notin D_{SAT}$ holds, the translation of P by t induces a change on all SAT candidate directions smaller than $|t|$. Using projected polytope set

$$At_d = \{\langle p + t, \frac{d}{|d|}\rangle | p \in P\}$$

we can therefore conclude from equation (2.3)

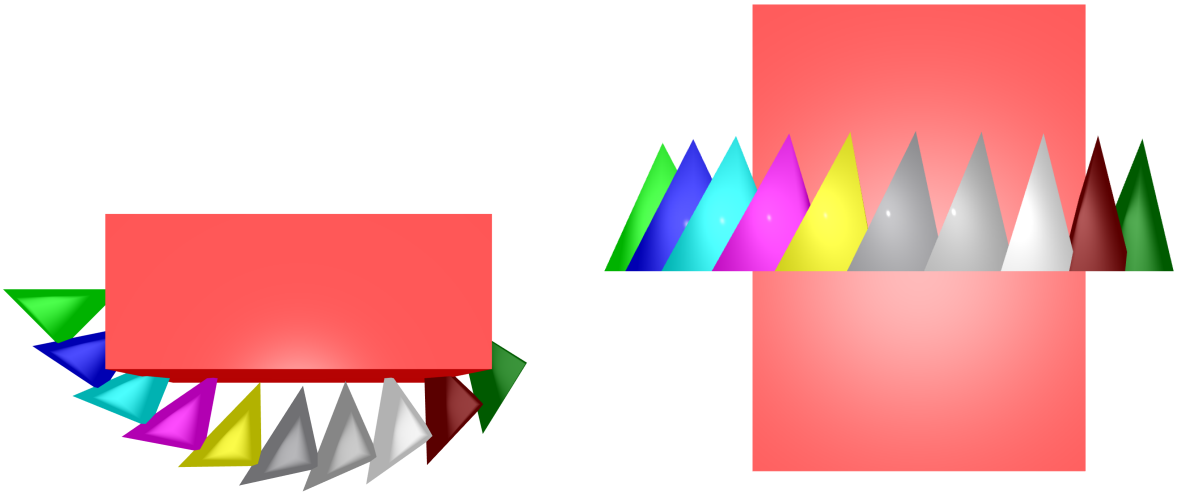
$$\mathrm{dp}(At_d, B_d) > \mathrm{dp}(A_d, B_d) - |t| \geq 0 \qquad \forall d \in D_{SAT} \tag{2.4}$$

Following Theorem 3, this is only the case for Polyhedra in contact or collision, contradicting above implication of separation. □



2.3 Numerical Comparison

In order to proof efficiency of our newly developed penetration distance algorithm as well as giving a performance impression on some of the algorithms mentioned above, we initialize a simple scene consisting of a moving tetrahedron and a static axis aligned box. Starting with tetrahedron vertices ($\{-0.5, 0.3, 0.3\}$, $\{-0.3, 0.3, 0.3\}$, $\{-0.4, 0.2, 0.3\}$, $\{-0.4, 0.25, 0.5\}$), we consider 200 time steps, each translating the tetrahedron with 0.005m in origins x direction as well as 0.01rad around its z axis, while the axis aligned box is kept static within lower left $\{-0.25, 0.15, 0.0\}$ and upper right $\{0.25, 0.35, 0.7\}$ points. In total, this generates 200 different scene configurations, with 99 bringing the objects into overlap, while 101 are collision-free. Figure 2.2 illustrates the scene at 10 different configurations.



(a) XY Plane (b) ZY Plane

Figure 2.2: Distance evaluation scene with static box (red) and dynamic tetrahedron illustrated every 20 iterations moving from left to right, including configurations 0 (light green), 20(dark blue), 40(turquoise), 60(pink) and 80(yellow).

All of the numerical results discussed in the following were implemented in C++ and executed on an AMD Ryzen Threadripper 3390X system with 256GB RAM. To lower the effect of CPU throttling, we executed every computation 100 times and considered the average of the 90% lowest runtimes in our evaluations.

2.3.1 Collision Detection

In a first experiment, we compare different strategies for collision detection with respect to numerical efficiency. We use the pure GJK implementation included in the hpp-fcl library [116], its Nesterov improvement proposed in [110], the MPR and GJK implementations from LibCCD [57] as well as our SAT algorithm implementation. To additionally compare against an optimization based strategy, we formulate the collision avoidance constraints proposed in [62] and



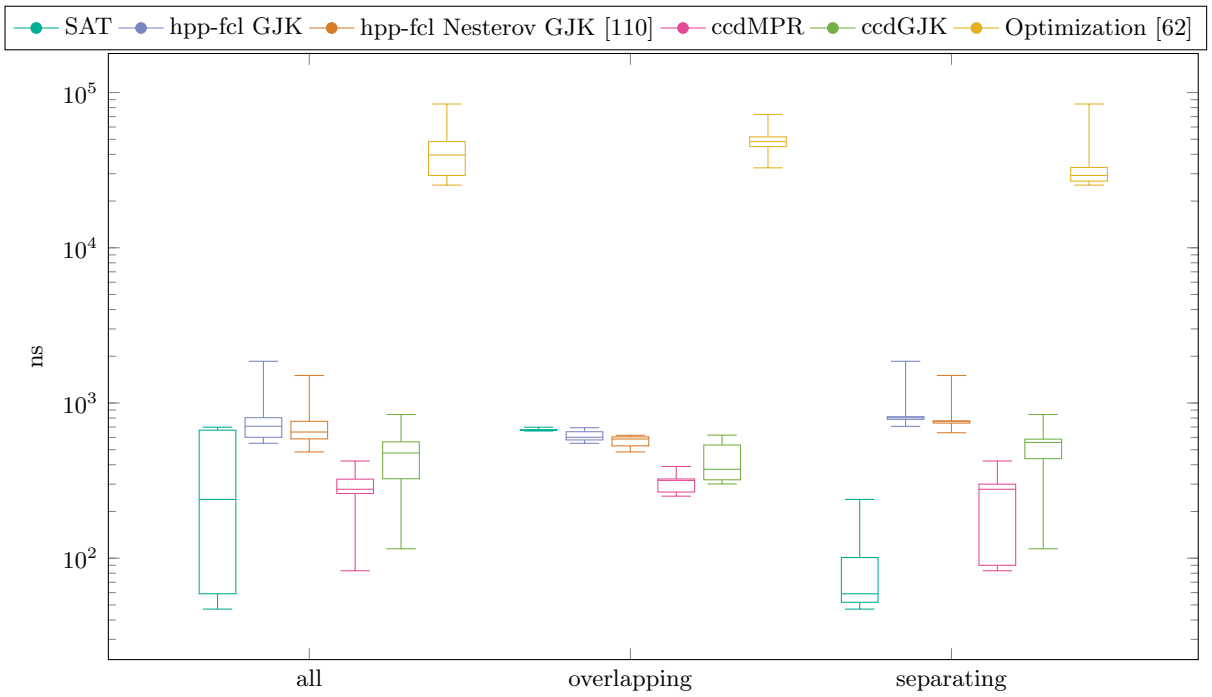


Figure 2.3: Comparison of Algorithms for collision detection on 200 sample configurations including 101 separating and 99 overlapping cases

evaluate the arising problem using ProxSuite [8] QP solver. We illustrate the results in Figure 2.3 and interpret them as follows: As the SAT algorithm allows early termination once a valid separation axis is found, its runtimes are especially low in separated polygon cases. Otherwise, a constant number of checks is required to successfully detect overlap, explaining the approximately constant runtime on overlapping polygon cases. Furthermore, we can observe a slightly faster termination of the Nesterov acceleration compared to vanilla GJK from hpp-fcl in all cases, confirming the observations from [110]. While in [110] the hpp-fcl GJK implementation outperformed MPR and GJK from ccd on more complex objects (100-8000 vertices), we observe an inverse result here on our simple polyhedra. In general, the SAT algorithm and ccds MPR algorithm seem to be the most efficient overall, with the SAT algorithm performing especially well on separated object cases, while the fastest results in overlapping scenarios can be achieved with ccds MPR. Finally, comparing the geometry based algorithms mentioned above to optimization based collision detection results, as expected, in huge time differences. In conclusion, we confirm the SAT algorithm as the method of choice on simple polygon collision detection tasks if separation is expected and MPR otherwise.

2.3.2 Distance Evaluation

Starting with the evaluation of the full signed distance d, we compare the GJK with EPA implementation of the hpp-fcl library with our penetration distance algorithm based on SAT combined with a custom GJK implementation. As can be obtained from Figure 2.4, the algo-

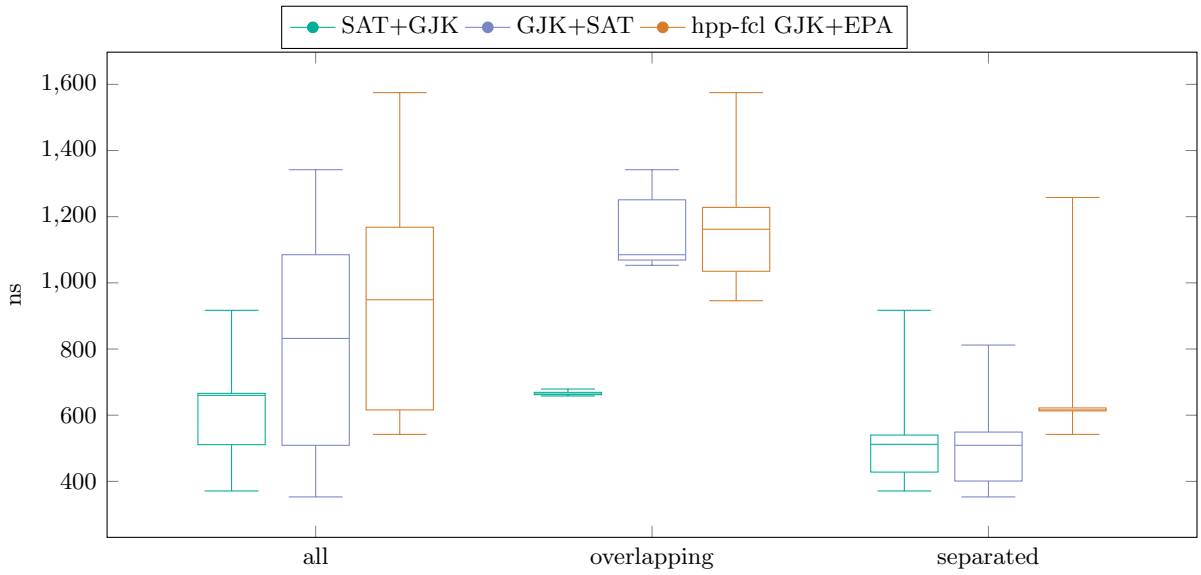


Figure 2.4: Comparison of Algorithms for signed distance evaluation on 200 sample configurations including 101 separating and 99 overlapping cases

rithm starting from the SAT method and proceeding with GJK if necessary, requires by far the smallest amounts of time on overlapping scenarios. It can be explained by its ability to directly evaluate the penetration distance without preceding GJK collision checks. Also with respect to the full set of scene configurations, this combination of SAT with GJK seems to be efficient. Surprisingly, the variant starting with GJK is only a little bit faster in separated cases and the hpp-fcl GJK with EPA even slower, even though they do not require preceding SAT algorithm collision checking. Looking back at Figure 2.3 we can explain that with the small runtime of the SAT algorithm in separating cases resulting from early stopping on separation axis found. In total, we observe the SAT algorithm combined with GJK to be most efficient for signed distance evaluation in our simple polyhedron scenario.

In some applications, such as physics simulations, we are interested in evaluating the penetration distance, to e.g model contact forces, but do not really care about the separation distance. The GJK combined with EPA algorithm shipped with the ccd library is especially dedicated to that case (GJK is just used for collision detection without distance evaluation), as well as our SAT based penetration distance evaluation. Naturally, the penetration distance can also be extracted from a full signed distance as evaluated by the GJK with EPA implementation from hpp-fcl. Figure 2.5 provides a runtime comparison of these three algorithms for penetration distance evaluation. It can be obtained that the SAT based algorithm clearly outperfroms the GJK with EPA implementations in overlapping as well as in separated object scenarios. Based on these results we suggest the SAT based algorithm as the method of choice for pure penetration distance evaluations on simple polyhedron scenarions.



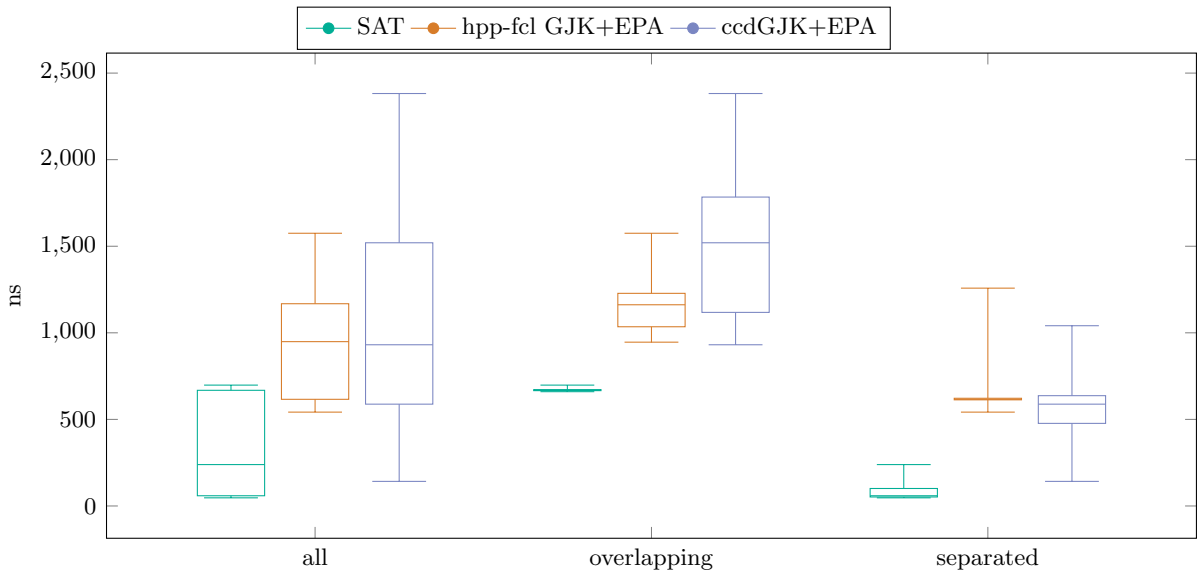


Figure 2.5: Comparison of Algorithms for penetration distance d^- evaluation on 200 sample configurations including 101 separating and 99 overlapping cases.



Chapter 3

Dynamic Programming

The method of dynamic programming (DP) was introduced by Richard Bellmann in 1957 as one of the early strategies to solve optimal control problems. It is based on the principle of optimality, stating that a sub-trajectory of an optimal trajectory is necessarily itself an optimal trajectory [10], as illustrated in Figure 3.1. In Section 3.1 we introduce the method of dynamic programming exploiting this principle for a generic final time nonlinear optimal control problem. Starting from the definition of a value function holding for every initial point in space and time the optimal resulting cost, we obtain its description in terms of a recursive equation as well as its characterization as a viscosity solution to the first order differential Hamilton Jacobi Bellman equation. In order to enable a numeric evaluation, we approximate the value function first in time and control, afterwards by full discretisation and combine results from literature [21] and [51] to proof convergence. From the resulting numeric algorithm, we obtain the main drawback of dynamic programming: The curse of dimensionality. Basically, the algorithms memory requirement as well as its run-time are exponentially dependent on the number of discretisation points used in time and state space. In terms of execution times, this additionally holds for the number of discrete controls. In high dimensional settings, such as considered in this thesis, these

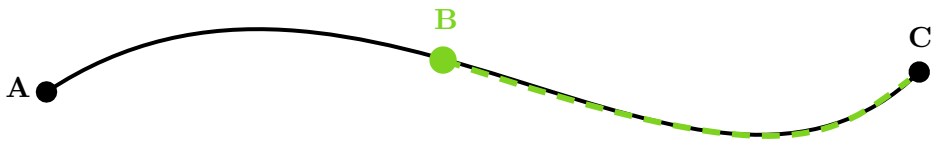


Figure 3.1: Given an optimal trajectory (black) connecting points A and C crossing B, the optimal connection between B and C is according to Bellmann's principle of optimality the respective sub-trajectory (dashed green).

restrictions often make the DP method inapplicable or at least very expensive. Even though it provides extraordinary advantages such as global solutions and simplified handling of non-differentiable constraints, it is therefore limited to low dimensional applications in practice. After transferring the original Trajectory planning Problem (1.1) in a way that it can be solved by the numerical DP algorithm in Section 3.2, the remaining parts of this chapter will focus on methods to reduce the impact of the curse of dimensionality. In Section 3.3, we propose the method of optimal stopping already known from DP literature [14, 9] for optimal end time problems. In contrast to the common transformation from optimal control where the optimal time is treated as an additional state [60], this formulation does not require any dimensional increase, which is especially beneficial in terms of DP algorithms performance. Furthermore, instead of using equidistant state space sampling in Section 3.4, we provide our algorithm originally presented in [128] to iteratively refine the grid where needed. As this allows a coarser initial state grid the approach results in less grid points used overall and therefore a reduction in run-times. Finally, we exploit the heavy improvement in hardware nowadays compared to Richard Bellmann's time in Section 3.5 and propose a highly parallelized variant of the dynamic programming algorithm, executable on recent multi kernel CPU structures, as well as in GPU supported environments. To demonstrate numerical effectiveness, a mixture of the acceleration strategies is applied to receive optimal trajectories for a four degrees of freedom robotic manipulator moving within a cluttered environment in Section 3.6. Furthermore, we prove the accelerated DP algorithm to be real time capable as part of a Model Predictive Control algorithm (MPC) for path planning in autonomous driving using a three dimensional state space.



3.1 The method of Dynamic Programming

This section introduces the main concept and characteristics of the method of dynamic programming based on multiple sources from literature [9, 22, 6] for an exemplary task.
Let

$$\begin{aligned} &\text{Minimize } \int_{t_0}^{T} c(x(s), u(s))ds + m(x(T)) && && (3.1)\\ &\text{subject to } \dot{x}(s) = f(x(s), u(s)) && a.e. \text{ in } [t_0, T]\\ &\qquad x(t_0) = z\\ &\qquad u(s) \in \mathbb{U} && a.e. \text{ in } [t_0, T] \end{aligned}$$

define a finite horizon optimal control problem using nonlinear first order dynamics function $f : \mathbb{R}^{n_x} \times \mathbb{U} \to \mathbb{R}^{n_x}$, running cost $c : \mathbb{R}^{n_x} \times \mathbb{U} \to \mathbb{R}$ and terminal cost $m : \mathbb{R}^{n_x} \to \mathbb{R}$ with respect to vectorised state function $x : [t_0, T] \to \mathbb{R}^{n_x}$ as well as a set of admissible controls

$$\mathcal{U} \coloneqq \{u : [t_0, T] \to \mathbb{R}^{n_u} \text{ measurable }, u(t) \in \mathbb{U} \text{ a.e in } [t_0, T]\}.$$

We impose a compactness assumption on the control space

$$\mathbb{U} \subset \mathbb{R}^{n_u} \text{ is a compact set} \tag{A1}$$

as well as Lipschitz continuity

$$\exists L_f \geq 0 : \quad |f(x_1, u_1) - f(x_2, u_2)| \leq L_f(|x_1 - x_2| + |u_1 - u_2|) \tag{A2}$$
$$\exists L_m \geq 0 : \quad |m(x_1) - m(x_2)| \leq L_m|x_1 - x_2| \tag{A3}$$
$$\exists L_c \geq 0 : \quad |c(x_1, u) - c(x_2, u)| \leq L_c|x_1 - x_2| \tag{A4}$$

and boundedness on Mayer, Lagrange and dynamics functions

$$\exists M_f \geq 0 : \quad |f(x_1, u)| \leq M_f \tag{A5}$$
$$\exists M_m \geq 0 : \quad |m(x_1)| \leq M_m \tag{A6}$$
$$\exists M_c \geq 0 : \quad |c(x_1, u)| \leq M_c \tag{A7}$$

for all $x_1, x_2 \in \mathbb{R}^{n_x}$ and $u, u_1, u_2 \in \mathbb{U}$. Given a control function as well as an initial constraint, we can follow from properties (A1) and (A2) the existence of a unique state function solving the dynamics constraint [6]. In the following, we denote this solution with respect to a certain $u \in \mathcal{U}$, $z \in \mathbb{R}^{n_x}$ and initial time t_0 as function $x(\cdot; t_0, z, u) : [t_0, T] \to \mathbb{R}^{n_x}$, which allows to



summarize (3.1) into

$$\underset{u\in\mathcal{U}}{\text{Min}} \int_{t_0}^{T} c(x(s;t_0,z,u),u(s))ds + m(x(T;t_0,z,u)).$$

Remark: Even though especially the global boundedness assumptions impose strong restrictions on practical applications, they are crucial for convergence of the discrete dynamic programming algorithm derived in the following.

The standard dynamic programming approach starts from the definition of the value function

$$v(t_0,z) = \inf_{u\in\mathcal{U}} \int_{t_0}^{T} c(x(s;t_0,z,u),u(s))ds + m(x(T;t_0,z,u)).$$

Given an initial state and time pair, it holds the cost arising from the optimal control.

Remark: Please note that there exist multiple approaches in literature where the objective function and consequently also the value function contain the final instead of the initial time as an argument. As pointed out in [9][Chapter 3, Remark 3.10], this does not imply any changes with respect to the theoretic properties presented in the following and is therefore considered as a matter of taste. We decided to follow the formulation in [22] with respect to the initial time here, as it is more intuitive in the context of optimal end times (see Section 3.3).

In the following, we recapitulate some of the value functions theoretic properties shown in detail in [9], that are fundamental for the DP method. Let us start from the Lipschitz property

Proposition 5 *[9, Chapter 3, Proposition 3.1]*
Assume (A1) - (A7). *Then, for all compact sets* $\mathcal{K} \subseteq \mathbb{R}^{n_x}$ *there exist* $L_v > 0$ *such that*

$$|v(s_1,z_1) - v(s_2,z_2)| \leq L_v(|z_1 - z_2| + |s_1 - s_2|)$$

holds for any $z_1, z_2 \in \mathcal{K}$ *and* $t_0 \leq s_1, s_2 \leq T$.

In combination with an exploitation of the principle of optimality, this allows to derive a recursive definition of the value function, commonly known as the Dynamic Programming Principle (DPP)



Proposition 6 *[9, Chapter 3, Proposition 3.2]. Assume* (A1) - (A7). *Then, for all* $z \in \mathbb{R}^{n_x}$ *and* $t < \tau \leq T$ *we have*

$$v(t,z) = \inf_{u \in \mathcal{U}} \left\{ \int_t^\tau c(x(s;t,z,u),u(s))ds + v(\tau, x(\tau;t,z,u)) \right\}$$

Please note that in case of $\tau = T$ the second term $v(T, x(T;t,z,u))$ simply evaluates to $m(x(T;t,z,u))$. Furthermore, extending the DPP to its infinitesimal space, the value function can be described by a partial differential equation: the so-called Hamilton-Jacobi-Bellman (HJB) equation

$$\begin{cases} -\partial_t v(t,z) + \sup_{u \in \mathbb{U}} \left\{ -f(z,u)^\top \nabla_z v(t,z) - c(z,u) \right\} = 0 & \text{in } \mathbb{R}^{n_x} \times \,]-\infty, T] \\ v(T,z) = m(z) & \text{in } \mathbb{R}^{n_x} \end{cases} \tag{3.2}$$

Consequently, we can determine the value function v by solving equation (3.2) if is differentiable. Unfortunately, v is usually non-smooth and therefore in most cases not a solution to the HJB equation in the classical sense.

From the theoretical perspective, this issue can be faced using the weaker definition of a viscosity solution introduced in [40]. It allows to characterize a continuous function as a solution to a partial differential equation at least in the viscosity sense, even though it is not differentiable everywhere. Indeed, as pointed out in [9] the value function can be characterized as the unique viscosity solution of the HJB equation (3.2) at least in unrestricted state spaces. Furthermore, using the even weaker concept of constraint viscosity solutions we can also verify the value function as a unique solution of the HJB on a restricted state space $\mathcal{K} \subset \mathbb{R}^{n_x}$ as described in Section 3.1.1. In the following, we will mainly focus on the implications arising for numerical evaluation of the value function from these properties. Nevertheless, we refer the interested reader to [9, 40] for further details on viscosity solutions.

From the practical perspective on the other hand, a non-smooth value function prohibits its evaluation by analytically solving the HJB. Instead, a numerical scheme to approximate the equation is required to be able to obtain a solution. In the following, we first derive an approximation with respect to time, combine it with a control space discretisation and prove convergence towards the continuous value function v. Afterwards, we additionally introduce a state space discretisation combined with an interpolation scheme, derive its convergence towards the time and control discrete approximate and finally formulate the numerical algorithm to evaluate the value function and consequently the optimal trajectory in the fully discretised setup.



3.1.1 Restricted State Space

As it is required in many practical applications due to, e.g., physical joint limitations in robotics, a lot of problems solved by the method of Dynamic Programming contain additional state constraints. In this section, we consider Problem 3.1 with a state space limited by closed non-empty subset $\mathcal{K} \subset \mathbb{R}^{n_x}$ with non-empty interior. As we are searching for a control that implies a certain state function, this limits the set of admissible controls to

$$\mathcal{U}_{\mathcal{K}}(t_0, z) \coloneqq \{u \in \mathcal{U}(t_0),\ x(t; t_0, z, u) \in \mathcal{K}\ a.e.\ \text{in}\ [t_0, T]\}.$$

Consequently, the value function of the state constrained problem evaluates to

$$v(t_0.z) = \inf_{u \in \mathcal{U}_{\mathcal{K}}(t_0,z)} \{m(x(T; t_0, z, u)) + \int_{t_0}^{T} c(x(s; t_0, z, u), u(s))ds\}$$

following convention $v(t_0, z) = \infty$ if $\mathcal{U}_{\mathcal{K}}(t_0, z) = \emptyset$. Without any additional restrictions, the set of admissible controls can be empty for some initial point in $\mathcal{K}$, resulting in a discontinuous value function. We therefore follow the argumentation in [139] and impose the so-called *inward pointing constraint*

$$\exists \beta > 0,\ \forall z \in \partial\mathcal{K},\ \exists u \in \mathbb{U} :\ f(z, u)\eta(z) < -\beta \tag{A9}$$

on the dynamics where $\eta(z)$ is the exterior normal vector of $\mathcal{K}$. Basically, it ensures that for any given point on the boundary, there exists a trajectory staying in the interior of set $\mathcal{K}$. In fact, it represents a strengthened viability condition and therefore implies the intuitive statement

$$\mathcal{U}_{\mathcal{K}}(t, z) \neq \emptyset \quad \forall (z, t) \in \mathcal{K} \times [t_0, T].$$

Additionally, as stated in [5],[16] we can follow Lipschitz continuity of the value function from Assumption (A9). Further tracking the arguments in [139] we define the state constraint HJB equation

$$\begin{cases} -\partial_t v(t, z) + \sup\limits_{u \in \mathbb{U}} \left\{-f(z, u)^\top \nabla_z v(t, z) - c(z, u)\right\} = 0 & \text{in } \mathcal{K} \times\ ]-\infty, T] \\ v(T, z) = m(z) & \text{in } \mathcal{K} \end{cases} \tag{3.3}$$

to characterize the value function as a unique solution with respect to its definition of a *constraint viscosity solution*. This property can also be achieved by applying alternative controllability assumptions such as weaker formulations of the inward pointing constraint [9, 83] or an outward pointing constraint [59, 58]. In practice, these assumptions are impossible to fulfill for lots of dynamic functions, including e.g., second order formulations. The approach of exact penalty introduced in [16, 5] penalizes the state constraint within an auxiliary problem and characterizes the original value function to track a level set on the auxiliary value function. As the auxiliary problem is formulated with respect to an unconstrained state space, the HJB equation and the viscosity solution property can be obtained without any further controllability restrictions,



making this approach more generally applicable. As it comes with the drawback of an additional state dimension, we apply the inward pointing constraint within the numerical examples of this section. Please note, that it implies Lipschitz continuity of the value function as well as a viscosity relation to the HJB equation. As a result, on time, control and state discretisation, the convergence results obtained in the following subsections can be extended to the state constrained case as long as the additional assumption (A9) is fulfilled.

3.1.2 Numerical Approximation

Due to the characterization of the value function as a solution of the HJB equation only in the viscosity sense, numerical approximation schemes instead of analytic strategies are required to propose a solution estimate. Over the years, two distinct discretisation approaches have established them self within the literature on HJB approximation, namely the method of finite difference and the Semi-Lagrangian strategy.

Starting our review from the development of the viscosity solution concept in [40], the class of first order finite difference schemes proposed in [41] characterizes the arising approximated value function to converge towards the weak solution of the HJB with an explicit error estimate provided. The same property is shown in [140] for a more general set of approximations, including some explicit and implicit finite difference schemes. Moving beyond fist order approximation accuracy, also higher order schemes are proposed such as the methods of ENO [114, 115] as well as its more robust extension WENO [84]. An adaptation of the latter to unstructured grids is provided in [158]. A common weakness of methods based on finite difference is the dependency of their stability on a time to state step-size ratio, commonly referenced to as Courant number [38, 50]. Thus, especially in applications with long time horizons, the fine time discretisation required for numerical stability, can make evaluations of the arising algorithm expensive in practice.

Capuzzo-Dolcetta in [45, 46] proposed an alternative scheme to discretise the HJB equation arising from an infinite horizon problem with respect to time, providing results on convergence towards a HJB viscosity solution. In [48], it is additionally equipped with a finite element approximation in state space, to allow numerical evaluation. Furthermore, extended in terms of an evolutive discounted finite horizon problem to variable step size time grids in [51], these schemes are characterized as early examples of Semi-Lagrangian schemes for HJB approximation in [2, 50]. In contrast to finite differences, approaches based on Semi-Lagrangian scheme turned out to be unconditionally stable [50, 21]. Consequently, they are a common choice in current literature [18, 21, 31, 32].



3.1.3 Time and Control Approximation

Using a backward formulation for the objective function and consequently for DPP and HJB equations, we derive a backward Semi-Lagrangian scheme similar to [21] based on [49]. Let $\mathbb{G}_t$ describe a time grid with

$$\mathbb{G}_t = \{t_i = t_0 + h_t \cdot i,\ i = 0, \cdots, K_t\}$$

using $K_t \in \mathbb{N}$ time intervals with step size $h_t = \frac{T-t_0}{K_t}$ and let the dynamics function be discretised with respect to explicit Euler method

$$x_h(t_{i+1}; t_i, z, u) = z + h_t \cdot f(z, u(t_i)) \qquad i = 0, \cdots, K_t - 1 \tag{3.4}$$

in every grid point $t_i \in \mathbb{G}_t$ with function $x_h : \mathbb{G}_t \to \mathbb{R}^{n_x}$ tracking the discrete time solution. Approximating the first line of the HJB equation (3.2) at initial time t_{i+1} and state z, we apply forward finite difference on the gradient with respect to state, as well as backward finite difference on the partial time derivative. The result can be written as

$$-\frac{v(t_{i+1}, z) - v(t_i, z)}{h_t} + \sup_{u \in \mathbb{U}} \left\{ -c(z, u) + \sum_{j=1}^{n_x} -f_j(z, u) \frac{v_j(t_{i+1}, z + h_t f(z, u)) - v_j(t_{i+1}, z)}{z_j + h_t f_j(z, u) - z_j} \right\} \approx 0$$

using a component wise representation of the gradient with respect to state, and further reformulated to

$$-\frac{v(t_{i+1}, z) - v(t_i, z)}{h_t} + \sup_{u \in \mathbb{U}} \left\{ -\frac{v(t_{i+1}, z + h_t f(z, u)) - v(t_{i+1}, z)}{h_t} - c(z, u) \right\} \approx 0.$$

Subsequently, solving for $v(t_i, z)$ yields

$$\begin{aligned} v(t_i, z) &\approx v(t_{i+1}, z) - \sup_{u \in \mathbb{U}} \{-v(t_{i+1}, z + h_t f(z, u)) + v(t_{i+1}, z) - h_t \cdot c(z, u)\} \\ &\approx -\sup_{u \in \mathbb{U}} \{-v(t_{i+1}, z + h_t f(z, u)) - h_t \cdot c(z, u)\} \\ &\approx \inf_{u \in \mathbb{U}} \{v(t_{i+1}, z + h_t f(z, u)) + h_t \cdot c(z, u)\} \end{aligned}$$

characterizing the Semi-Lagrangian scheme.
Staying on the track of [21], we additionally introduce a discretisation of the control space according to grid $\mathbb{G}_u$ using $K_u \in \mathbb{N}$ equidistant intervals of size h_u and define function $v_h : \mathbb{G}_t \times \mathbb{R}^{n_x} \to \mathbb{R}$ to describe the resulting time and control discrete approximated value function

$$\begin{aligned} v_h(t_i, z) &= \min_{u \in \mathbb{G}_u} \{v_h(t_{i+1}, z + h_t f(z, u)) + h_t \cdot c(z, u)\} \qquad i = 0, \cdots, K_t - 1 \\ v_h(t_{K_t}, z) &= m(z). \end{aligned} \tag{3.5}$$



Remark: Please note that we are able to replace the infimun by the minimum here as the grid $\mathbb{G}_u$ contains a finite number of elements.

Given this approximation, we are interested in a quantification of the error compared to its continuous counterpart as well as in the behavior of this error on step size reduction $h_t, h_u \to 0$. As a result, we compare the continuous value function v characterized as the unique viscosity solution of the HJB Equation (3.2) to its Semi-Lagrangian approximation derived in Equation (3.5). While the error estimate and convergence order derived in [21] require several additional assumptions on the problem structure, the theorem introduced in the following derives an error bound in a very general setup.

Theorem 7 *Let Assumptions* (A1)-(A7) *hold and let* $\mathcal{K} \subseteq \mathbb{R}^{n_x}$ *be an arbitrary compact set. Then, for all* $t_i \in \mathbb{G}_t$ *and* $z \in \mathcal{K}$

$$|v(t_i, z) - v_h(t_i, z)| = \mathcal{O}(h_t).$$

Proof. Let $z \in \mathcal{K}$ be arbitrary. We prove this theorem by induction, starting from

$$|v(t_{K_t}, z) - v_h(t_{K_t}, z)| = |m(z) - m(z)| = 0 = \mathcal{O}(h_t)$$

which can be obtained from the functions definitions in (3.5) and Proposition 6. For the induction step we move backward in time from $t_i \to t_{i-1}$. To unburden the notation, we make use of abbreviations $x(\tau; u) \coloneqq x(\tau; t_{i-1}, z, u)$ and $x_h(\tau; u) \coloneqq x_h(\tau; t_{i-1}, z, u)$. Exploiting the DPP we derive

$$\begin{aligned} |v(t_{i-1}, z) - v_h(t_{i-1}, z)| = \Bigg| &\inf_{u \in \mathcal{U}} \left\{ \int_{t_{i-1}}^{t_i} c(x(\tau; u), u(\tau)) d\tau + v(t_i, x(t_i; u)) \right\} \\ &- \min_{u \in \mathbb{G}_u} \{ h_t \cdot c(z, u) + v_h(t_i, z + h_t f(z, u)) \} \Bigg|. \end{aligned} \tag{3.6}$$

For further transformations, we distinguish two cases:
Case 1: $v(t_{i-1}, z) \geq v_h(t_{i-1}, z)$
In this case, we introduce the function $\hat{u} \in \mathcal{U}$ defined by

$$\hat{u}(\tau) = \arg\min_{u \in \mathbb{G}_u} \{ h_t \cdot c(z, u) + v_h(t_i, z + h_t f(z, u)) \} \qquad \tau \in [t_{i-1}, t_i).$$

By definition, inserting this function into the first part, leads to an overall absolute value increase,



which allows us to follow

$$
\begin{aligned}
&|v(t_{i-1}, z) - v_h(t_{i-1}, z)| \\
&\leq |\int_{t_{i-1}}^{t_i} c(x(\tau; \hat{u}), \hat{u}(\tau))d\tau + v(t_i, x(t_i; \hat{u})) - h_t c(z, \hat{u}(t_{i-1})) - v_h(t_i, x_h(t_i; \hat{u}))|.
\end{aligned}
$$

Furthermore, exploiting Proposition 5, Assumptions (A5), (A7) as well as the induction assumption, we can reformulate above inequality as

$$
\begin{aligned}
&|v(t_{i-1}, z) - v_h(t_{i-1}, z)| \\
&\leq |v(t_i, x(t_i; \hat{u})) - v_h(t_i, x_h(t_i; \hat{u}))| + |\int_{t_{i-1}}^{t_i} c(x(\tau; \hat{u}), \hat{u}(\tau))d\tau - h_t c(z, \hat{u}(t_{i-1}))| \\
&\overset{(A7)}{\leq} |\ v(t_i, x(t_i; \hat{u})) - v(t_i, x_h(t_i; \hat{u}))| + |v(t_i, x_h(t_i; \hat{u})) - v_h(t_i, x_h(t_i; \hat{u}))| + 2h_t M_c \\
&\overset{Prop.5}{\leq} L_v |x(t_i; \hat{u}) - x_h(t_i; \hat{u})| + \mathcal{O}(h_t) \\
&\leq L_v |\int_{t_{i-1}}^{t_i} f(x(\tau; \hat{u}), \hat{u}(\tau)) - f(z, \hat{u}(t_{i-1}))d\tau| + \mathcal{O}(h_t) \\
&\overset{(A5)}{\leq} L_v\ 2M_f h_t + \mathcal{O}(h_t) = \mathcal{O}(h_t)
\end{aligned}
$$

Case 2: $v(t_{i-1}, z) < v_h(t_{i-1}, z)$
Exploiting the infimum definition, for any $\epsilon > 0$ there exists a function $\hat{u} \in \mathcal{U}$ such that

$$
v(t_{i-1}, z) \geq \int_{t_{i-1}}^{t_i} c(x(\tau; \hat{u}), \hat{u}(\tau))d\tau + v(t_i, x(t_i; \hat{u})) - \epsilon
$$

Again, this allows to derive an upper bound on the overall equation

$$
\begin{aligned}
&|v(t_{i-1}, z) - v_h(t_{i-1}, z)| \\
&\leq \left| \int_{t_{i-1}}^{t_i} c(x(\tau; \hat{u}), \hat{u}(\tau))d\tau + v(t_i, x(t_i; \hat{u})) - \epsilon - h_t c(z, \hat{u}(t_{i-1})) - v_h(t_i, x_h(t_i; \hat{u})) \right|,
\end{aligned}
$$

this time by increasing $v_h(t_{i-1}, z)$ while decreasing $v(t_{i-1}, z)$. Following the arguments used in Case 1 for the upper bound reformulation we obtain

$$
|v(t_{i-1}, z) - v_h(t_{i-1}, z)| \leq L_v 2M_f h_t + \mathcal{O}(h_t) + \epsilon = \mathcal{O}(h_t) + \epsilon
$$

This completes the proof, since $\epsilon > 0$ was chosen arbitrarily. □

Finally, using the theorem above, we have proven our time and control approximated value function $v_h(t_i, \cdot)$ to indeed converge towards the continuous function $v(t_i, \cdot)$ on infinitely dense discretisation $h_t \to 0$, in every state $z \in \mathcal{K}$ on any compact set $\mathcal{K} \subseteq \mathbb{R}^{n_x}$.



3.1.4 Full Discretisation

As the approximated function $v_h(t_i, \cdot)$ is still defined on a continuous state space, it can not be fully evaluated within an algorithmic procedure. In order to extend to a full discretisation, we therefore additionally introduce an equidistant state space grid $\mathbb{G}_x \subset \mathbb{R}^{n_x}$ with step sizes $h_x \in \mathbb{R}^{n_x}$ in each dimension respectively. When defining the respective fully discretised approximate value function according to (3.5) an issue arises as we cannot ensure that each dynamics step result $x_j + h_t f(x_j, u)$ hits a point on the grid $\mathbb{G}_x$. As a solution, we follow [21] and [49] and introduce an interpolation scheme to evaluate the value function at non-grid points. For any given non-grid point $z \in \mathbb{R}^{n_x}$ we define functions $\Pi_j : \mathbb{R}^{n_x} \to \mathbb{G}_x$ to return each of the $j = 1, \dots, 2^{n_x}$ neighbored grid points and their weights $\alpha_j : \mathbb{R}^{n_x} \to \mathbb{R}$ where

$$\sum_{j=1}^{2^{n_x}} \alpha_j(z) = 1 \qquad \forall z \in \mathbb{R}^{n_x}$$

holds. Furthermore, given a grid point $\tilde{z} \in \mathbb{G}_x$ we expect the interpolation to return exactly the value at this point which can be formalized into

$$\Pi_j(\tilde{z}) = \tilde{z} \implies \alpha_j(\tilde{z}) = 1 \text{ and } \alpha_i(\tilde{z}) = 0 \; \forall i \neq j$$

As a result we can define a fully discretised approximated value function $\tilde{v}_h : \mathbb{G}_t \times \mathbb{R}^{n_x} \to \mathbb{R}$

$$\tilde{v}_h(t_i, z) = \sum_{j=1}^{2^{n_x}} \alpha_j(z) \cdot \tilde{v}_h(t_i, \Pi_j(z)) \tag{3.7}$$

such that we can adapt the DPP from equation (3.5) to

$$\begin{aligned} \tilde{v}_h(t_i, \Pi_j(z)) &= \min_{u \in \mathbb{G}_u} \{\tilde{v}_h(t_{i+1}, z + h_t f(\Pi_j(z), u)) + h_t \cdot c(t_i, \Pi_j(z))\} \\ \tilde{v}_h(t_{K_t}, \Pi_j(z)) &= m(\Pi_j(z)) \end{aligned} \tag{3.8}$$

for all $z \in \mathbb{R}^{n_x}$ and $t_i \in \mathbb{G}_t$. Again, the following parts of this section will focus on the convergence of the fully approximated value function $\tilde{v}_h$ defined in (3.8) towards the time and control approximate v_h characterized by equation (3.5) on infinitely dense space discretisation $h_x \to 0$ as well as towards its fully continuous counterpart. Following [21] in Lemma 4.16 we start with a Lemma on the time and control discretised approximates Lipschitz property, but instead of following their argumentation, we adapt the one from [51] in Lemma 2.2

Lemma 8 *Assume* (A1) - (A4). *Then, there exists a constant* $L_{v_h} > 0$ *such that*

$$|v_h(t_i, z) - v_h(t_i, y)| \leq L_{v_h} |z - y|$$

holds for any $z, y \in \mathbb{R}^{n_x}$ *and* $t_i \in \mathbb{G}_t$



Proof. Initially, we show

$$|v_h(t_{K_t-n}, z) - v_h(t_{K_t-n}, y)| \leq [(1 + L_f h_t)^n \cdot (L_c h_t n + L_m)]|z - y| \tag{3.9}$$

using induction over n. For the base case $n = 0$ we obtain

$$|v_h(t_{K_t}, z) - v_h(t_{K_t}, y)| = |m(z) - m(y)| \leq L_m|z - y|$$

using the Lipschitz property (A3) of the Mayer term. Continuing with the induction step, we move from $n \to n + 1$ and obtain

$$\begin{aligned}|v_h(t_{K_t-(n+1)}, z) - v_h(t_{K_t-(n+1)}, y)| =& |\min_{u_1 \in \mathbb{G}_u} \{v_h(t_{K_t-n}, z + h_t f(z, u_1)) + h_t \cdot c(z, u_1)\} \\ &- \min_{u_2 \in \mathbb{G}_u} \{v_h(t_{K_t-n}, y + h_t f(y, u_2)) + h_t \cdot c(y, u_2)\}|.\end{aligned}$$

Without loss of generality we can assume $v_h(t_{K_t-(n+1)}, z) \geq v_h(t_{K_t-(n+1)}, y)$ and let $\hat{u} \in \mathbb{G}_u$ be defined by

$$\hat{u} = \arg\min_{u_1 \in \mathbb{G}_u} \{v_h(t_{K_t-n}, z + h_t f(z, u_1)) + h_t \cdot c(z, u_1)\}.$$

Consequently, we get an upper bound

$$\begin{aligned}&|v_h(t_{K_t-(n+1)}, z) - v_h(t_{K_t-(n+1)}, y)| \\ \leq &|v_h(t_{K_t-n}, z + h_t f(z, \hat{u})) - v_h(t_{K_t-n}, y + h_t f(y, \hat{u}))| + |h_t \cdot c(z, \hat{u}) - h_t \cdot c(y, \hat{u})|\end{aligned}$$

by increasing the larger component of the absolute value while leaving the smaller one unchanged. Subsequently, we can derive

$$\begin{aligned}&|v_h(t_{K_t-(n+1)}, z) - v_h(t_{K_t-(n+1)}, y)| \\ \leq &[(1 + L_f h_t)^n \cdot (L_c h_t n + L_m)] \cdot |z + h_t f(z, u) - y - h_t f(y, u)| + h_t L_c|z - y| \\ \leq &[(1 + L_f h_t)^{n+1} \cdot (L_c h_t n + L_m)] \cdot |z - y| + h_t L_c|z - y| \\ \leq &[(1 + L_f h_t)^{n+1} \cdot (L_c h_t (n + 1) + L_m)] \cdot |z - y|\end{aligned}$$

using the assumption (3.9) as well as Lipschitz properties of the dynamics (A2) and the cost function (A4) to complete the proof by induction. Finally, as

$$(1 + L_f h_t)^n \cdot (L_c h_t n + L_m) \leq e^{T L_f}(L_c T + L_m)$$

holds for any $n = 0, \ldots, K_t$ we can choose $L_{v_h} = e^{T L_f}(L_c T + L_m)$ which in combination with equation (3.9) implies the Lemma. □

We can now move on to approximate the local error arising from the state discretisation of the value function. We rewrite Lemma 4.17 from [21] for our general problem and follow their arguments to proof correctness.



Lemma 9 *Assume* (A1) - (A7). *For any time point* $t_i \in \mathbb{G}_t$ *and any state* $z \in \mathbb{R}^{n_x}$ *holds*

$$|\tilde{v}_h(t_i, z) - v_h(t_i, z)| \leq L_{v_h}(K_t + 1 - i)|h_x| \tag{3.10}$$

using Lipschitz constant L_{v_h} *from Lemma 8.*

Proof. Following the proof of Lemma 4.17 in [21], we initially show

$$|\tilde{v}_h(t_i, \tilde{z}) - v_h(t_i, \tilde{z})| \leq L_{v_h}(K_t - i)|h_x| \tag{3.11}$$

for any $t_i \in \mathbb{G}_t$ and any grid point $\tilde{z} \in \mathbb{G}_x$ by induction, starting from base $i = K_t$ with

$$|\tilde{v}_h(t_{K_t}, \tilde{z}) - v_h(t_{K_t}, \tilde{z})| = |m(\tilde{z}) - m(\tilde{z})| = 0.$$

For the induction step we move from $i \to i - 1$ and use the respective definitions (3.5) and (3.7),(3.8) to derive

$$\begin{aligned} |\tilde{v}_h(t_{i-1}, \tilde{z}) - v_h(t_{i-1}, \tilde{z})| \leq & |\min_{u_1 \in \mathbb{G}_u} \{\tilde{v}_h(t_i, \tilde{z} + h_t f(\tilde{z}, u_1)) + h_t \cdot c(\tilde{z}, u_1)\} \\ & - \min_{u_2 \in \mathbb{G}_u} \{v_h(t_i, \tilde{z} + h_t f(\tilde{z}, u_2)) + h_t \cdot c(\tilde{z}, u_2)\}|. \end{aligned}$$

We let $\hat{u} \in \mathbb{G}_u$ be defined by

$$\hat{u} = \arg\min_{u_1 \in \mathbb{G}_u} \{\tilde{v}_h(t_i, \tilde{z} + h_t f(\tilde{z}, u_1)) + h_t \cdot c(\tilde{z}, u_1)\}$$

if $\tilde{v}_h(t_{i-1}, \tilde{z}) \geq v_h(t_{i-1}, \tilde{z})$ holds and by

$$\hat{u} = \arg\min_{u_2 \in \mathbb{G}_u} \{v_h(t_i, \tilde{z} + h_t f(\tilde{z}, u_2)) + h_t \cdot c(\tilde{z}, u_2)\}$$

otherwise. Consequently, we get the upper bound

$$\begin{aligned} & |\tilde{v}_h(t_{i-1}, \tilde{z}) - v_h(t_{i-1}, \tilde{z})| \\ \leq & |\tilde{v}_h(t_i, \tilde{z} + h_t f(\tilde{z}, \hat{u})) - v_h(t_i, \tilde{z} + h_t f(\tilde{z}, \hat{u}))| + |h_t \cdot c(\tilde{z}, \hat{u}) - h_t \cdot c(\tilde{z}, \hat{u})| \end{aligned}$$

as in ether case we increase the larger component while leaving the smaller one unchanged. Additionally, to unburden the notation, we use abbreviation $z_i := \tilde{z} + h_t f(\tilde{z}, u)$.



As a result, we can follow

$$
\begin{aligned}
&|\tilde{v}_h(t_{i-1},\tilde{z}) - v_h(t_{i-1},\tilde{z})| \\
\leq&|\sum_{j=1}^{2^{n_x}} \alpha_j(z_i)\tilde{v}_h(t_i,\Pi_j(z_i)) - \sum_{j=1}^{2^{n_x}} \alpha_j(z_i)v_h(t_i,z_i)| \\
\leq&\sum_{j=1}^{2^{n_x}} \alpha_j(z_i)\Big[|\tilde{v}_h(t_i,\Pi_j(z_i)) - v_h(t_i,\Pi_j(z_i))| + |v_h(t_i,\Pi_j(z_i)) - v_h(t_i,z_i)|\Big] \\
\leq&\sum_{j=1}^{2^{n_x}} \alpha_j(z_i)\Big[L_{v_h}(K_t-i)|h_x| + L_{v_h}|\Pi_j(z_i) - z_i|\Big] \\
\leq&L_{v_h}(K_t-(i-1))|h_x|
\end{aligned}
$$

using the Lipschitz property from Lemma 8 as well as the assumption. Additionally, the state grid and interpolation properties $\sum_{j=1}^{2^{n_x}} \alpha_j = 1$ and $|\Pi_j(z_i) - z_i| < |h_x|$ are exploited. As this completes the proof for statement (3.11) we can now derive (3.10) with

$$
\begin{aligned}
|\tilde{v}_h(t_i,z) - v_h(t_i,z)| &= |\sum_{j=1}^{2^{n_x}} \alpha_j(z)\cdot\tilde{v}_h(t_i,\Pi_j(z)) - v_h(t_i,z)| \\
&\leq \sum_{j=1}^{2^{n_x}} \alpha_j(z)\cdot[|\tilde{v}_h(t_i,\Pi_j(z)) - v_h(t_i,\Pi_j(z))| + |v_h(t_i,\Pi_j(z)) - v_h(t_i,z)|] \\
&\leq \sum_{j=1}^{2^{n_x}} \alpha_j(z)\cdot[L_{v_h}(K_t-i)|h_x| + L_{v_h}|\Pi_j(z) - z|] \\
&\leq L_{v_h}(K_t+1-i)|h_x|
\end{aligned}
$$

□

So far, we already gave an estimate on the local error arising from time and control discretisation of the value function compared to its continuous counterpart as well as a quantification of the error arising from additional state space discretisation with respect to the time and control discrete function v_h. As numerical DP algorithms typically approximate the fully discretised value function $\tilde{v}_h$, we are especially interested in the behavior of the fully discretised to continuous error estimate, when deriving convergence properties for algorithms. Consequently, the theorem introduced in the following allows to derive convergence of this fully discretised value function $\tilde{v}_h$ towards its continuous counterpart for $h_x, h_t \to 0$.



Theorem 10 *Assume* (A1) - (A7)*, an arbitrary compact set* $\mathcal{K} \subseteq \mathbb{R}^{n_x}$*, as well as*

$$h_t^p > |h_x|$$

with $p > 1$*. Then, for an arbitrary time point* $t_i \in \mathbb{G}_t$*, and state* $z \in \mathcal{K}$

$$\lim_{h_t \to 0} |v(t_i, z) - \tilde{v}_h(t_i, z)| = 0$$

holds.

Proof. Initially, we can derive a limit on the overall local error using the time and control discrete value function formulation

$$|v(t_i, z) - \tilde{v}_h(t_i, z)| \leq |v(t_i, z) - v_h(t_i, z)| + |v_h(t_i, z) - \tilde{v}_h(t_i, z)|.$$

Subsequently, we can exploit Theorem 7 and Lemma 9 to derive

$$|v(t_i, z) - \tilde{v}_h(t_i, z)| \leq c \cdot h_t + L_{v_h}(K_t + 1)|h_x|$$

for any constant $c > 0$. Furthermore, using the additional assumption, we write $|h_x|$ in terms of h_t resulting in

$$|v(t_i, z) - \tilde{v}_h(t_i, z)| \leq c \cdot h_t + L_{v_h}(\frac{T}{h_t} + 1)h_t^p$$

Finally, we can exploit this inequality to derive

$$\lim_{h_t \to 0} |v(t_i, z) - \tilde{v}_h(t_i, z)| \leq \lim_{h_t \to 0} \left(c \cdot h_t + L_{v_h}(\frac{T}{h_t} + 1)h_t^p \right) = \lim_{h_t \to 0} \left(c \cdot h_t + L_{v_h}(T h_t^{p-1} + h_t^p) \right) = 0$$

concluding the proof. □



3.1.5 DP Algorithm

With all the results on discretisation and convergence, we can now initialize the numeric DP algorithm. Its structure is rather standard (see [63, 14]) and contains two main steps: a backward and a forward pass. During the backward pass, we exploit the recursive structure of the DPP to evaluate the approximated value function backward in time for every point on the grid $(t, z) \in \mathbb{G}_t \times \mathbb{G}_x$. Based on that, we can in the forward pass start from the initial condition (t_0, z) and track the value function forward in time, such that we finally obtain a discrete control trajectory with an associated set of states. Algorithm 1 illustrates the strategy in more detail. Obviously, the state grid $\mathbb{G}_x$ must be restricted to a finite number of elements to enable numeric

Algorithm 1: Discrete Dynamic Programming

Initialize grids $\mathbb{G}_t$, $\mathbb{G}_u$, $\mathbb{G}_x$

Backwardpass

Initialize for all $z \in \mathbb{G}_x$

$$\tilde{v}_h(t_{K_t}, z) = m(z)$$

for $i \leftarrow K_t - 1, \dots, 0$ **do**

  Evaluate for all $z \in \mathbb{G}_x$

$$\tilde{v}_h(t_i, z) = \min_{u \in \mathbb{G}_u} \{\tilde{v}_h(t_{i+1}, z + h_t f(z, u)) + h_t \cdot c(z, u)\}$$

  w.r.t. equation (3.7)

Forwardpass

Start from initial value $x_h(t_0) = z$

for $i \leftarrow 0, \dots, K_t - 1$ **do**

  Evaluate the minimum control from

$$u_h(t_i) \coloneqq \arg\min_{u \in \mathbb{G}_u} \{\tilde{v}_h(t_{i+1}, x_h(t_i) + h_t f(x_h(t_i), u)) + h_t \cdot c(x_h(t_i), u)\}$$

  w.r.t. equation (3.7) and store the corresponding state

$$x_h(t_{i+1}) = x_h(t_i) + h_t f(x_h(t_i), u_h(i))$$

Return state function x_h and piece wise constant interpolated controls u_h

trajectory evaluation. While this is naturally the case when considering restricted state spaces from Section 3.1.1, the unrestricted case operates on $\mathbb{R}^{n_x}$, which does not allow an equidistant discretisation using a finite number of elements. Luckily, the trajectory evaluation within the forward pass requires only value functions at states reachable by the dynamics in a certain time. Assuming a closed set of reachable states for a certain initial value z, we can limit the value function evaluation to a finite state grid containing this set. During algorithm evaluation, we basically evaluate every single time, state, and control combination admissible for the discrete problem. Choosing the minimum among them, the resulting trajectory is naturally the



globally optimal solution with respect to time, control, and state discretisation. Indeed, if the approximated value function $\tilde{v}_h$ on infinitely dense time and state grids converges according to Theorem 10 towards its continuous counterpart, the result of the algorithm converges towards the continuous time solution. Regarding numerical evaluation, the overall algorithms runtime is within

$$\mathcal{O}(K_t \cdot \prod_{k=0}^{n_u} K_u^k \cdot \prod_{k=0}^{n_x} K_x^k)$$

function calls for the backward pass and in

$$\mathcal{O}(K_t \cdot \prod_{k=0}^{n_u} K_u^k)$$

function calls for the forward pass on increasing numbers of grid points. Furthermore, the number of value function entries to be stored during the backward pass is in

$$\mathcal{O}(K_t \cdot \prod_{k=0}^{n_x} K_x^k)$$

imposing, in addition to the exponential growth in execution time, an exponential increase in memory requirement. As these dependencies become a burden relatively fast when increasing the grids, literature called this phenomenon the *curse of dimensionality*. In practice, it quite often prevents the usage of the DP method as it is either impossible w.r.t memory requirement or unacceptably expensive regarding execution time. Nevertheless, the method provides some great advantages such as global optimality and the usage of non-differentiable functions, such that the remaining part of this chapter will focus on strategies to lower the curses impact.



3.2 DP for Trajectory Planning

Given a numerical algorithm as well as corresponding results on convergence, we aim to use the method of dynamic programming to solve our overall trajectory planning problem. Consequently, a reformulation of Problem (1.1) introduced in Chapter 1 into the form of a dynamic programming Problem (3.1) is required. Starting from the additional collision avoidance and goal constraints, we can apply the penalty method [63] to include them into the cost and Mayer terms such that the reformulated problem reads as

$$
\begin{aligned}
\text{Minimize } & \int_0^{t_f} c(x(t),u(t)) + p(x(t))dt + \rho_1 \|b(x(t_f))\| && && (3.12)\\
\text{subject to } & \dot{x}(t) = f(x(t),u(t)) && a.e \text{ in } [0,t_f]\\
& x(0) = z\\
& x(t) \in X && \forall t \in [0,t_f]\\
& u(t) \in U && a.e \text{ in } [0,t_f]\\
& t_f \in [0,T]
\end{aligned}
$$

using function $p(x(t)) = \rho_2 \max(-g(x(t)),0)$ as well as penalty scaling factors $\rho_1, \rho_2 > 0$. Please note that, if the functions c, g, and b are continuous, the introduced penalty terms fulfill the requirements stated in [63, Chapter 7] and consequently the solutions of Problem (3.12) do converge towards solutions of the original Problem (1.1) on increasing penalty factors $\rho_1, \rho_2 \to \infty$. Unfortunately, the problem described above still depends on a free final time, while we introduced the method of dynamic programming in (3.1) for a fixed time interval. To solve that issue, we can apply the time transformation from classical optimal control as described in [60] and obtain with introduction of additional constant state $t_f \in [0,T]$ the fixed time scale problem

$$
\begin{aligned}
\text{Minimize } & t_f \int_0^1 c(x(t_f\cdot\tau),u(t_f\cdot\tau)) + p(x(t_f\cdot\tau))d\tau + \rho_1 \|b(x(t_f))\| && && (3.13)\\
\text{subject to } & \dot{x}(t_f\cdot\tau) = t_f \cdot f(x(t_f\cdot\tau),u(t_f\cdot\tau)) && a.e \text{ in } [0,1]\\
& x(0) = z\\
& x(t_f\cdot\tau) \in X \times \mathbb{R} && \forall \tau \in [0,1]\\
& u(t_f\cdot\tau) \in U && a.e \text{ in } [0,1]\\
& t_f \in [0,T]
\end{aligned}
$$

matching the structure required for dynamic programming application.



3.3 DP with Optimal stopping

As already demonstrated in the previous section, the standard approach to deal with an optimal end time in optimal control uses an additional state to represent it [60]. Unfortunately, in the context of dynamic programming, due to the curse of dimensionality this results in a massive increase in execution time as well as storage requirement, compared to the fixed end time case. As free final time problems nevertheless often appear in practice, the dynamic programming community has found another more efficient way to handle these so-called *optimal stopping* problems: In contrast to the time transformation, the variable end time problem is given to the method of dynamic programming as is. Naturally, this results in a change in the definitions of value function and Hamilton Jacobi Bellman equation, but as pointed out in [9] and [124] the theoretic properties required for the method of dynamic programming remain untouched. The main advantage from the numerical perspective arises if we discretise this adapted value function and approximate it using a strategy closely related to Algorithm 1. As the end time in this case is neither treated as an additional state nor control but is implicitly part of the dynamic programming principle, the arising algorithms asymptotic requirements on execution time and storage match the ones for the static final time strategy. Consequently, the DP algorithm for optimal stopping can be seen as an effective tool to lower the curse of dimensionalities impact in free final time problems.
Let us equip the example problem used in Section 3.1 with an optimal final time

$$\begin{aligned} \text{Minimize } & \int_{t_0}^{t_f} c(x(s),u(s))ds + m(x(t_f)) && && (3.14)\\ \text{subject to } & \dot{x}(s) = f(x(s),u(s)) && a.e. \text{ in } [t_0,t_f]\\ & x(t_0) = z\\ & t_f \in [t_0,T]\\ & u(s) \in \mathcal{U} && \text{in } [t_0,t_f] \end{aligned}$$

and adopt assumptions (A1) - (A7) on its components. From (A1) and (A2) we can again derive the existence of a unique state function solving the dynamics constraint, denoted by $x(s;t_0,z,u)$ with respect to a certain control function $u \in \mathcal{U}$, initial state $z \in \mathbb{R}^{n_x}$, and initial time t_0. Furthermore, we introduce the function

$$J(t_f,u;t_0,z) = \int_{t_0}^{t_f} c(x(s;t_0,z,u),u(s))ds + m(x(t_f;t_0,z,u))$$

to track the cost of a certain control and end time solution candidate, such that we can compress



the problem to

$$\underset{u\in\mathcal{U},t_f\in[t_0,T]}{\text{Minimize}} \; J(t_f,u;t_0,z). \tag{3.15}$$

In the following, we derive the method of dynamic programming for the above generic optimal stopping problem. Regarding current literature, analysis of optimal stopping problems obtained in optimal control is either limited to the infinite horizon case [9, 50] or discrete time problems [14]. Luckily, it turns out that searching for a barrier characterization in differential games, the arising variational inequality to be solved [124, 122] shares some similarities with the HJB equation obtained in optimal stopping problems. As a consequence, even though we can not directly adapt results on the theoretic properties of the arising value function, such as Lipschitz continuity and unique viscosity solution of a partial differential equation, from current literature, we can combine results from games [122, 124] and optimal control [9, 5] to explicitly derive them in Section 3.3.1. Afterwards, we focus in Section 3.3.2 on the numerical evaluation of a solution starting from a full discretisation of the value function. As already done in Section 3.1 and heavily based on the arguments therein, we proceed with a convergence proof of the approximate value function to its continuous counterpart. Finally, we point out the changes required in Algorithm 1 to handle the optimal stopping case.

3.3.1 Theoretic Value function Properties

In this section, we derive the main theoretic building blocks of the method of dynamic programming, when applied in the context of optimal stopping Problem (3.15). As in the fixed end time case, we start from the definition of the value function

$$v(t_0,z) = \inf_{u\in\mathcal{U}} \min_{t_f\in[t_0,T]} J(t_f,u;t_0,z) \tag{3.16}$$

for any $(t_0,z)\in[t_0,T]\times\mathbb{R}^{n_x}$, returning the cost of the optimal solution for each initial time and state pair. Please note that using assumptions (A4) and (A3) we can follow continuity of J in argument t_f and consequently, exploiting the Weierstrass extreme value theorem, the optimal time infimum arising within the value function definition is actually a minimum. Following the arguments in [122, Proposition 6] we can also derive a recursive definition of this function, representing the dynamic programming principle in the optimal stopping case.



Proposition 11 *Assume* (A1)-(A7). *Then, for all* $(t, z) \in [t_0, T] \times \mathbb{R}^{n_x}$ *and* $\alpha > 0$ *such that* $t < t + \alpha \leq T$ *holds*

$$v(t,z) = \inf_{u \in \mathcal{U}} \min[\min_{t_f \in [t, t+\alpha]} J(t_f, u; t, z), \int_t^{t+\alpha} c(x(s; t, z, u), u(s))ds + v(t + \alpha, x(t + \alpha; t, z, u))] \tag{3.17}$$

Proof. To unburden the notation, we use abbreviations $x(t_f; u) \coloneqq x(t_f; t, z, u)$ and $t_\alpha \coloneqq t + \alpha$ within the scope of this proof. Initially, we reformulate the value function definition (3.16) to

$$\begin{aligned} v(t,z) &= \inf_{u \in \mathcal{U}} \min_{t_f \in [t,T]} J(t_f, u; t, z) \\ &= \inf_{u \in \mathcal{U}} \min \left[\min_{t_f \in [t, t_\alpha]} J(t_f, u; t, z), \min_{t_f \in [t_\alpha, T]} J(t_f, u; t, z) \right] \\ &= \inf_{u \in \mathcal{U}} \min \left[\min_{t_f \in [t, t_\alpha]} J(t_f, u; t, z), \int_t^{t_\alpha} c(x(s; u), u(s))ds + \min_{t_f \in [t_\alpha, T]} J(t_f, u; t_\alpha, x(t_\alpha; u)) \right]. \end{aligned} \tag{3.18}$$

Then, exploiting the infimum definition in (3.16), for any $\epsilon > 0$ and any state $z_\alpha \in \mathbb{R}^{n_x}$ exists $\bar{u} : [t_\alpha, T] \to \mathbb{U}$ such that

$$v(t_\alpha, z_\alpha) + \epsilon \geq \min_{t_f \in [t_\alpha, T]} J(t_f, \bar{u}; t_\alpha, z_\alpha).$$

Consequently, we can initialize a combined function $\tilde{u} : [t, T] \to \mathbb{U}$ as

$$\tilde{u}(t) = \begin{cases} u(t) & t \in [t, t_\alpha) \\ \bar{u}(t) & t \in [t_\alpha, T] \end{cases}$$

for any $u \in \mathcal{U}$. Plugging control function $\tilde{u}$ into equation (3.18) yields

$$\begin{aligned} v(t,z) &\leq \min \left[\min_{t_f \in [t, t_\alpha]} J(t_f, \tilde{u}; t, z), \int_t^{t_\alpha} c(x(s; \tilde{u}), \tilde{u}(s))ds + \min_{t_f \in [t_\alpha, T]} J(t_f, \tilde{u}; t_\alpha, x(t_\alpha; \tilde{u})) \right] \\ &\leq \min \left[\min_{t_f \in [t, t_\alpha]} J(t_f, u; t, z), \int_t^{t_\alpha} c(x(s; u), u(s))ds + v(t_\alpha, x(t_\alpha; u)) + \epsilon \right] \end{aligned}$$

for any $u \in \mathcal{U}$. For ϵ tending to zero we get

$$v(t,z) \leq \min \left[\min_{t_f \in [t, t_\alpha]} J(t_f, u; t, z), \int_t^{t_\alpha} c(x(s; u), u(s))ds + v(t_\alpha, x(t_\alpha; u)) \right]$$



for any $u \in \mathcal{U}$, which implies

$$v(t,z) \leq \inf_{u\in\mathcal{U}} \min\left[\min_{t_f\in[t,t+\alpha]} J(t_f,u;t,z), \int_t^{t_\alpha} c(x(s;u),u(s))ds + v(t_\alpha, x(t_\alpha;u))\right].$$

On the other hand, we can exploit the definition of the infimum in (3.16) once more, indicating for any $\epsilon > 0$, existence of a function $\hat{u} \in \mathcal{U}$ such that

$$v(t,z) \geq \min_{t_f\in[t,T]} J(t_f,\hat{u};t,z) - \epsilon$$

holds. Plugging the specific control into equation (3.18), results this time in the lower bound

$$v(t,z) \geq \min\left[\min_{t_f\in[t,t_\alpha]} J(t_f,\hat{u};t,z), \int_t^{t_\alpha} c(x(s;\hat{u}),\hat{u}(s))ds + \min_{t_f\in[t_\alpha,T]} J(t_f,\hat{u};t_\alpha,x(t_\alpha;\hat{u}))\right] - \epsilon.$$

Furthermore, as we can directly follow from definition (3.16) that

$$v(t_\alpha, z_\alpha) \leq \min_{t_f\in[t_\alpha,T]} J(t_f,u;t_\alpha,z_\alpha))$$

holds for any initial state $z_\alpha \in \mathbb{R}^{n_x}$ and any control $u \in \mathcal{U}$, we can further lower this bound to

$$v(t,z) \geq \min\left[\min_{t_f\in[t,t_\alpha]} J(t_f,\hat{u};t,z), \int_t^{t_\alpha} c(x(s;\hat{u}),\hat{u}(s))ds + v(t_\alpha, x(t_\alpha;\hat{u}))\right] - \epsilon.$$

As we have chosen $\epsilon > 0$ arbitrarily, we can follow

$$v(t,z) \geq \inf_{u\in\mathcal{U}} \min\left[\min_{t_f\in[t,t+\alpha]} J(t_f,u;t,z), \int_t^{t_\alpha} c(x(s;u),u(s))ds + v(t_\alpha, x(t_\alpha;u))\right]$$

and the proof is complete. □

Another important property of the value function pointed out in Section 3.1 is its Lipschitz continuity. For the value function of the optimal stopping case, we can derive it following the arguments in Proposition 2.2 and Corollary 2.1 [124].

Proposition 12 *Assume* (A1)-(A7). *Then, there exists* $L_v > 0$ *such that for any* $z_1, z_2 \in \mathbb{R}^{n_x}$ *and* $t_1, t_2 \in [t_0, T]$

$$|v(t_1,z_1) - v(t_2,z_2)| \leq L_v(|z_1 - z_2| + |t_1 - t_2|)$$

Proof. We start from the uniform Lipschitz continuity in the initial state domain, searching for $L_{v1} > 0$ such that

$$|v(t,z_1) - v(t,z_2)| \leq L_{v1}|z_1 - z_2|$$

holds for any $z_1, z_2 \in \mathbb{R}^{n_x}$ and $t \in [t_0, T]$. Initially, plugging the value function's definition (3.16)



into the left hand side leads to

$$|v(t,z_1) - v(t,z_2)| = |\inf_{u_1 \in \mathcal{U}} \min_{t_{f1} \in [t,T]} J(t_{f1}, u_1; t, z_1) - \inf_{u_2 \in \mathcal{U}} \min_{t_{f2} \in [t,T]} J(t_{f2}, u_2; t, z_2)|.$$

Without loss of generality, we can assume $v(t,z_1) \geq v(t,z_2)$. Subsequently, we can exploit infimum properties to obtain for any $\epsilon > 0$ existence of a control $\tilde{u} \in \mathcal{U}$ such that

$$v(t,z_2) \geq \min_{t_{f2} \in [t,T]} J(t_{f2}, \tilde{u}; t, z_2) - \epsilon$$

holds. As plugging control $\tilde{u}$ into the above equation lowers the smaller part of the absolute value, while increasing the larger one, we can derive an upper bound

$$0 \leq v(t,z_1) - v(t,z_2) \leq \min_{t_{f1} \in [t,T]} J(t_{f1}, \tilde{u}; t, z_1) - \min_{t_{f2} \in [t,T]} J(t_{f2}, \tilde{u}; t, z_2) + \epsilon.$$

By choosing

$$\tilde{t}_f = \arg \min_{t_{f2} \in [t,T]} J(t_{f2}, \tilde{u}; t, z_2)$$

we can increase this upper bound even further to

$$\begin{aligned} 0 \leq v(t,z_1) - v(t,z_2) &\leq J(\tilde{t}_f, \tilde{u}; t, z_1) - J(\tilde{t}_f, \tilde{u}; t, z_2) + \epsilon \\ &\leq |J(\tilde{t}_f, \tilde{u}; t, z_1) - J(\tilde{t}_f, \tilde{u}; t, z_2)| + \epsilon. \end{aligned} \tag{3.19}$$

Using abbreviations $x_1(\tau) \coloneqq x(\tau; t, z_1, \tilde{u})$, $x_2(\tau) \coloneqq x(\tau; t, z_2, \tilde{u})$, we can furthermore obtain

$$\begin{aligned} |x_1(\tau) - x_2(\tau)| &\leq |z_1 - z_2| + \int_t^\tau |f(x_1(s), \tilde{u}(s)) - f(x_2(s), \tilde{u}(s))| ds \\ &\leq |z_1 - z_2| + \int_t^\tau L_f |x_1(s) - x_2(s)| ds \end{aligned}$$

from assumption (A2), which allows deduction of

$$|x_1(t) - x_2(t)| \leq |z_1 - z_2| e^{(\tau - t)L_f} \leq |z_1 - z_2| e^{(T - t_0)L_f}.$$

using the Gronwall inequality. We can use the arising inequality as well as Lipschitz properties (A4),(A3) and $\tilde{t}_f \leq T$ to further reformulate equation (3.19) to

$$\begin{aligned} v(t,z_1) - v(t,z_2) &\leq |m(x_1(\tilde{t}_f)) - m(x_2(\tilde{t}_f)) + \int_t^{\tilde{t}_f} c(x_1(\tau), \tilde{u}(\tau)) - c(x_2(\tau), \tilde{u}(\tau)) d\tau| + \epsilon \\ &\leq L_m |x_1(\tilde{t}_f) - x_2(\tilde{t}_f)| + L_c \int_t^{\tilde{t}_f} |x_1(\tau) - x_2(\tau)| d\tau + \epsilon \\ &\leq |z_1 - z_2| e^{(T - t_0)L_f} (L_m + L_c(T - t_0)) + \epsilon. \end{aligned}$$



Finally, for ϵ tending to zero, we get

$$|v(t, z_1) - v(t, z_2)| \leq L_{v_1}(|z_1 - z_2| + |t_1 - t_2|)$$

with

$$L_{v1} = e^{(T-t_0)L_f}(L_m + L_c(T - t_0))$$

concluding the first part of the proof.
To show the Lipschitz property in time we search for $L_{v2} \geq 0$ such that

$$|v(t_1, z) - v(t_2, z)| \leq L_{v2}|t_2 - t_1| \tag{3.20}$$

holds for any $t_1, t_2 \in [t_0, T]$ and $z \in \mathbb{R}^{n_x}$. Without loss of generality, we can assume $t_1 < t_2$ in the following and use abbreviation $T_1 \coloneqq T - (t_2 - t_1)$. Given any control $u : [t_2, T] \to \mathbb{U}$, we define its time transformed $\bar{u} : [t_1, T_1] \to \mathbb{U}$ with

$$\bar{u}(t) = u(t + (t_2 - t_1)).$$

As the dynamics function f is not explicitly dependent on time t this allows us to write

$$x(t; t_1, z, \bar{u}) = x(t + (t_2 - t_1); t_2, z, u)$$

for any $t \in [t_1, T_1]$. Consequently, exploiting the autonomous definitions of running cost and Mayer term we can derive

$$\begin{aligned}
J(t, \bar{u}; t_1, z) &= \int_{t_1}^{t} c(x(\tau; t_1, z, \bar{u}), \bar{u}(\tau))d\tau + m(x(t; t_1, z, \bar{u})) \\
&= \int_{t_1}^{t} c(x(\tau + (t_2 - t_1); t_2, z, u), \bar{u}(\tau))d\tau + m(x(t + (t_2 - t_1); t_2, z, u)) \\
&= \int_{t_2}^{t+(t_2-t_1)} c(x(\tau; t_2, z, u), \bar{u}(\tau - (t_2 - t_1))d\tau + m(x(t + (t_2 - t_1); t_2, z, u)) \\
&= J(t + (t_2 - t_1), u; t_2, z)
\end{aligned}$$

for any $t \in [t_1, T_1]$, resulting in

$$\min_{t_f \in [t_1, T_1]} J(t_f, \bar{u}; t_1, z) = \min_{t_f \in [t_1, T_1]} J(t_f + (t_2 - t_1), u; t_2, z) = \min_{t_f \in [t_2, T]} J(t_f, u; t_2, z)$$

for any $u \in \mathcal{U}$. In combination with the value function's definition (3.16), this allows us to rewrite

$$v(t_2, z) = \inf_{u \in \mathcal{U}} \min_{t_f \in [t_2, T]} J(t_f, u; t_2, z) = \inf_{u \in \mathcal{U}} \min_{t_f \in [t_1, T_1]} J(t_f, \bar{u}; t_1, z) = \inf_{u \in \mathcal{U}} \min_{t_f \in [t_1, T_1]} J(t_f, u; t_1, z)$$

exploiting the definition of the control function space $\mathcal{U}$ in the last step. Keeping the arising



equation, as well as Proposition 11 in mind, we plug the value function's definition (3.16) into the left hand side of equation (3.20) and derive

$$\begin{aligned}|v(t_1,z)-v(t_2,z)| &= |\inf_{u_1\in\mathcal{U}}\min_{t_{f1}\in[t_1,T]} J(t_{f1},u_1;t_1,z)-\inf_{u_2\in\mathcal{U}}\min_{t_{f2}\in[t_2,T]} J(t_{f2},u_2;t_2,z)|\\ &= |\inf_{u\in\mathcal{U}}\min[\min_{t_f\in[t_1,T_1]} J(t_f,u;t_1,z),\min_{t_f\in[T_1,T]} J(t_f,u;t_1,z)]\\ &\quad -\inf_{u\in\mathcal{U}}\min_{t_f\in[t_1,T_1]} J(t_f,u;t_1,z)|.\end{aligned}$$

In case of

$$\inf_{u\in\mathcal{U}}\min_{t_f\in[T_1,T]} J(t_f,u;t_1,z)\geq\inf_{u\in\mathcal{U}}\min_{t_f\in[t_1,T_1]} J(t_f,u;t_1,z)$$

above equation evaluates to zero and hence the assumption is true for any $L_{v_2}>0$.
In the complementary case, we can write

$$\begin{aligned}|v(t_1,z)-v(t_2,z)| &= |\inf_{u\in\mathcal{U}}\min_{t_f\in[T_1,T]} J(t_f,u;t_1,z)-\inf_{u\in\mathcal{U}}\min_{t_f\in[t_1,T_1]} J(t_f,u;t_1,z)|\\ &= \inf_{u\in\mathcal{U}}\min_{t_f\in[t_1,T_1]} J(t_f,u;t_1,z)-\inf_{u\in\mathcal{U}}\min_{t_f\in[T_1,T]} J(t_f,u;t_1,z).\end{aligned}$$

Exploiting the definition of the infimum, there exists for every $\epsilon>0$ a control function $\tilde{u}\in\mathcal{U}$ such that

$$\inf_{u\in\mathcal{U}}\min_{t_f\in[T_1,T]} J(t_f,u;t_1,z)\geq\min_{t_f\in[T_1,T]} J(t_f,\tilde{u};t_1,z)-\epsilon$$

holds which allows to derive upper bound

$$|v(t_1,z)-v(t_2,z)|\leq\min_{t_f\in[t_1,T_1]} J(t_f,\tilde{u};t_1,z)-\min_{t_f\in[T_1,T]} J(t_f,\tilde{u};t_1,z)+\epsilon.$$

Subsequently, we can further increase the bound using abbreviation $x_1(t)\coloneqq x(t;t_1,z,\tilde{u})$ to

$$\begin{aligned}0\leq|v(t_1,z)-v(t_2,z)| &\leq J(T_1,\tilde{u};t_1,z)-\min_{t_f\in[T_1,T]} J(t_f,\tilde{u};t_1,z)+\epsilon\\ &\leq\int_{t_1}^{T_1} c(x_1(s),\tilde{u}(s))ds+m(x_1(T_1))\\ &\quad-\int_{t_1}^{T_1} c(x_1(s),\tilde{u}(s))ds-\min_{t_f\in[T_1,T]} J(t_f,\tilde{u};T_1,x_1(T_1))+\epsilon\\ &\leq m(x_1(T_1))-\min_{t_f\in[T_1,T]} J(t_f,\tilde{u};T_1,x_1(T_1))+\epsilon.\end{aligned}$$

Subsequently, we can use

$$\tilde{t}_f=\arg\min_{t_f\in[T_1,T]} J(t_f,\tilde{u};T_1,x_1(T_1))$$



to reformulate the arising inequality into

$$\begin{aligned}
|v(t_1,z)-v(t_2,z)| &\leq |m(x_1(T_1))-J(\tilde{t}_f,\tilde{u};T_1,x_1(T_1))|+\epsilon \\
&\leq |m(x_1(T_1))-m(x(\tilde{t}_f;T_1,x_1(T_1),\tilde{u}))| \\
&\quad +|\int_{T_1}^{\tilde{t}_f} c(x(s;T_1,x_1(T_1),\tilde{u}),\tilde{u}(s))ds|+\epsilon \\
&\leq L_m|x_1(T_1)-x(\tilde{t}_f;T_1,x_1(T_1),\tilde{u}))|+M_c(t_2-t_1)+\epsilon \\
&\leq L_m|\int_{T_1}^{\tilde{t}_f} f(x(\tau;T_1,x_1,\tilde{u}),\tilde{u}(\tau))d\tau|+M_c(t_2-t_1)+\epsilon \\
&\leq (L_mM_f+M_c)(t_2-t_1)+\epsilon
\end{aligned}$$

exploiting bounds (A5),(A7) and Lipschitz property (A3). As a result, for ϵ tending to zero, we can choose $L_{v_2}=(L_mM_f+M_c)$ and conclude Lipschitz continuity in time. Finally, letting L_v track the maximum out of L_{v_1} and L_{v_2}, we can follow uniform Lipschitz continuity of the value function in state and time. □

As done for the standard DP method, we characterize the value function as a viscosity solution to some partial differential equation. Unfortunately, we were not able to find a statement for our specific case in literature, so we need to explicitly prove it here. For this purpose, let us recapitulate the concept of viscosity solutions developed in [40] analogue to [9].

Definition: A continuous function v is a viscosity solution of the nonlinear partial differential equation

$$F(t,z,v(t,z),Dv)=0$$

if the following conditions are satisfied:

- **i) v is viscosity subsolution:** for any $\varphi\in C^1$, if (t,z) is a local maximum point of $v-\varphi$, then
$$F(t,z,v(t,z),D\varphi)\leq 0$$
- **ii) v is viscosity supersolution:** for any $\varphi\in C^1$, if (t,z) is a local minimum point of $v-\varphi$, then
$$F(t,z,v(t,z),D\varphi)\geq 0$$

Using this definition, we can now follow a mixture of the arguments in [124, Proposition 2.4] and [9, Proposition 3.5] to formulate the HJB and characterize the value function as its solution.

Proposition 13 *Let assumptions* (A1)-(A7) *hold. Then, the value function, defined in* (3.16) *is a viscosity solution of the partial differential equation*

$$\max[-\partial_t v(t,z)+\max_{u\in\mathbb{U}}\{-\nabla_z v(t,z)f(z,u)-c(z,u)\};v(t,z)-m(z)]=0 \tag{3.21}$$



a.e. in $(0,T)$ *and for all* $\mathbb{R}^{n_x}$ *with*

$$v(T,z) = m(z)$$

for all $z \in \mathbb{R}^{n_x}$.

Proof. Initially, let us observe from the continuity of the value function shown in Proposition 12 that it is indeed a candidate function to satisfy the above criteria for a viscosity solution. We start from the proof of the subsolution property.
Following the arguments in [9, Proposition 3.5], we let $\varphi \in C^1(\mathbb{R}^{n_x}\times]0,\infty[)$ and point (t_0,z) characterize a local maximum of $v-\varphi$. According to the definition we aim to proof

$$\max[-\partial_t\varphi(t_0,z) + \max_{u\in\mathbb{U}}\{-\nabla_z\varphi(t_0,z)f(z,u) - c(z,u)\}; v(t_0,z) - m(z)] \leq 0$$

in the following using abbreviation $x_0(t;u) \coloneqq x(t;t_0,z,u)$. Initially, as (t_0,z) characterizes a local maximum of $v-\varphi$ there exists $r>0$ such that

$$v(t,y) - \varphi(t,y) \leq v(t_0,z) - \varphi(t_0,z)$$

holds for all x,y with $|z-y|<r$, $|t-t_0|<r$. Additionally, due to the continuity of $x_0(\cdot;u)$, there exists $\eta_1 \in (0,r)$ such that for any $u\in\mathcal{U}$ and $t\in[t_0,t_0+\eta_1]$ we have $|x_0(t;u)-z|<r$ and consequently

$$v(t,x_0(t;u)) - \varphi(t,x_0(t;u)) \leq v(t_0,z) - \varphi(t_0,z)$$

holds. Exploiting the DPP from Proposition 11, we can further reformulate the inequality to

$$\begin{aligned}\varphi(t_0,z) - \varphi(t,x_0(t;u)) &\leq v(t_0,z) - v(t,x_0(t;u))\\ &\leq \int_{t_0}^{t} c(x_0(s;u),u(s))ds + v(t,x_0(t;u)) - v(t,x_0(t;u))\\ &= \int_{t_0}^{t} c(x_0(s;u),u(s))ds\end{aligned}$$

for any $u\in\mathcal{U}$. Subsequently, we fix the control function u to remain constant in time with value $\mu\in\mathbb{U}$. As the resulting state function $x_0(\cdot,u)$ is continuous and differentiable, we can divide by $\tau = t-t_0 > 0$ and pass to the limit $\tau\to 0$ which yields

$$-\partial_t\varphi(t_0,z) - \nabla_z\varphi(t_0,z)x'(t_0) = -\partial_t\varphi(t_0,z) - \nabla_z\varphi(t_0,z)f(z,\mu) \leq c(z,\mu)$$

using the differentiability of φ as well as the continuity of f and c. As this holds for an arbitrary μ, the equation

$$-\partial_t\varphi(t_0,z) + \max_{\mu\in\mathbb{U}}\{-\nabla_z\varphi(t_0,z)f(z,\mu) - c(z,\mu)\} \leq 0$$



is also valid. Additionally, we can directly obtain from the value function definition (3.16) that

$$v(t_0, z) - m(z) \leq 0$$

holds for any $(t_0, z) \in [0, T] \times \mathbb{R}^{n_x}$. Thus, the subsolution property holds.
For the supersolution property, we start from function a $\varphi \in C^1(\mathbb{R}^{n_x} \times]0, \infty[)$ and let point (t_0, z) characterize a local minimum of $v - \varphi$. Then, we aim to prove

$$\max[-\partial_t \varphi(t_0, z) + \max_{u \in \mathbb{U}} \{-\nabla_z \varphi(t_0, z) f(z, u) - c(z, u)\}; v(t_0, z) - m(z)] \geq 0.$$

Due to the minimum property, there exists $r > 0$ such that

$$v(t, y) - \varphi(t, y) \geq v(t_0, z) - \varphi(t_0, z)$$

for all t, y with $|z - y| < r$, $|t - t_0| < r$ as well as $\eta_2 \in (0, r)$ such that for any $u \in \mathcal{U}$ and $t \in [t_0, t_0 + \eta_2]$

$$v(t, x_0(t; u)) - \varphi(t, x_0(t; u)) \geq v(t_0, z) - \varphi(t_0, z).$$

Consequently

$$\varphi(t_0, z) - \varphi(t, x_0(t; u)) \geq v(t_0, z) - v(t, x_0(t; u)) \tag{3.22}$$

holds.
Case1: $v(t_0, z) < m(z)$
Due to the continuity of v in Proposition 12 as well as m in Assumption (A3), there exists $\beta > 0$ such that

$$v(t, x_0(t; u)) < m(x_0(t; u))$$

for any $u \in \mathcal{U}$ and $t \in [t_0, t_0 + \beta]$. Exploiting above inequality, as well as the value function definition from equation (3.16), we can conclude

$$v(t_0, z) = \inf_{u \in \mathcal{U}} \left\{ \min_{t_f \in [t_0 + \beta, T]} J(t_f, u; t_0, z) \right\}$$

Furthermore, regarding the principle of dynamic programming in Proposition 11, the minimum evaluates in case of $\alpha \in (0, \beta]$ to its second argument. Consequently, choosing $\alpha = \beta$ we can rewrite the equation to

$$v(t_0, z) = \inf_{u \in \mathcal{U}} \left\{ \int_{t_0}^{t_0 + \beta} c(x_0(s; u), u(s)) ds + v(t_0 + \beta, x_0(t_0 + \beta; u)) \right\}$$

Subsequently, for any $\epsilon > 0$ exists a control $\tilde{u} \in \mathcal{U}$ with

$$v(t_0, z) \geq \int_{t_0}^{t_0 + \beta} c(x_0(s; \tilde{u}), \tilde{u}(s)) ds + v(t_0 + \beta, x_0(t_0 + \beta; \tilde{u})) - (t_0 + \beta - t_0)\epsilon.$$



due to the infimum definition. Restricting $t \in [t_0, t_0 + \min(\eta_2, \beta)]$ we can plug above equation into (3.22) and derive

$$\varphi(t_0, z) - \varphi(t, x_0(t; \tilde{u})) \geq \int_{t_0}^{t} c(x_0(s; \tilde{u}), \tilde{u}(s))ds - (t - t_0)\epsilon. \tag{3.23}$$

Following the arguments in [9][Proposition 3.5] we exploit properties of the dynamics and cost function (A2),(A4),(A5), to obtain

$$|x_0(s; \tilde{u}) - z| \leq M_f(s - t_0)$$

and thus

$$|c(x_0(s; \tilde{u}), \tilde{u}(s)) - c(z, \tilde{u}(s))| \leq L_c M_f(s - t_0)$$

as well as

$$|f(x_0(s; \tilde{u}), \tilde{u}(s)) - f(z, \tilde{u}(s))| \leq L_f M_f(s - t_0)$$

for any $s \in [t_0, t]$. We can therefore rewrite the right hand side of inequality (3.23) to

$$\int_{t_0}^{t} c(z, \tilde{u}(s))ds + \int_{t_0}^{t} c(x(s; \tilde{u}), \tilde{u}(s)) - c(z, \tilde{u}(s))ds = \int_{t_0}^{t} c(z, \tilde{u}(s))ds + \mathcal{O}(\tau) \tag{3.24}$$

using $\tau = t - t_0$ and the asymptotic notation with respect to limit $\tau \to 0$. Additionally, we reformulate the left hand side to

$$\begin{aligned}
\varphi(t, z) - \varphi(t, x_0(t; \tilde{u})) &= -\int_{t_0}^{t} \frac{d}{ds}\varphi(t, x_0(s; \tilde{u}))ds \\
&= -\int_{t_0}^{t} \nabla_z \varphi(t, x_0(s; \tilde{u}))x_0'(s; \tilde{u})ds \\
&= -\int_{t_0}^{t} \nabla_z \varphi(t, x_0(s; \tilde{u}))f(x_0(s; \tilde{u}), \tilde{u}(s))ds \\
&= -\int_{t_0}^{t} \nabla_z \varphi(t, z)f(z, \tilde{u}(s))ds \\
&\quad + \int_{t_0}^{t} -\nabla_z \varphi(t, x_0(s; \tilde{u}))f(x_0(s; \tilde{u}), \tilde{u}(s)) + \nabla_z \varphi(t, z)f(z, \tilde{u}(s))ds \\
&= -\int_{t_0}^{t} \nabla_z \varphi(t, z)f(z, \tilde{u}(s))ds + \mathcal{O}(\tau)
\end{aligned} \tag{3.25}$$

using $\varphi \in \mathcal{C}^1$ as well as above inequality on the dynamics. Exploiting inequalities (3.24) and (3.25) we can reformulate (3.23) to

$$\varphi(t_0, z) - \varphi(t, z) + \int_{t_0}^{t} -\nabla \varphi_z(t, z)f(z, \tilde{u}(s)) - c(z, \tilde{u}(s))ds \geq \mathcal{O}(\tau) - \tau\epsilon.$$



Subsequently, we further increase the left hand side and obtain

$$\varphi(t_0,z)-\varphi(t,z)+\int_{t_0}^{t}\max_{\mu\in\mathbb{U}}\{-\nabla\varphi_z(t,z)f(z,\mu)-c(z,\mu)\}ds\geq\mathcal{O}(\tau)-\tau\epsilon.$$

Finally, using division by τ and passing the limit $\tau\to 0$ we end up with

$$-\partial_t\varphi(t_0,z)+\max_{\mu\in\mathbb{U}}\{-\nabla\varphi_z(t_0,z)f(z,\mu)-c(z,\mu)\}\geq-\epsilon.$$

This proves the supersolution property, since $\epsilon>0$ was chosen arbitrarily.
Case 2: $v(t_0,z)=m(z)$
In this case the second part of the max is exactly zero, leading to a valid statement independent of the first part. □

Finally, we want to characterize any viscosity solution to be unique. For the proof of the arising Theorem stated below, we can heavily exploit the results from [5, Theorem A.1].

Theorem 14 *Let Assumptions* (A1)-(A7) *hold and* a,b *be two functions in* $\mathcal{C}([0,T]\times\mathbb{R}^{n_x})$ *that describe a sub- and super-solution of* (3.21) *such that*

$$\max[-\partial_t a(t,z)+\max_{u\in\mathbb{U}}\{-\nabla_z a(t,z)f(z,u)-c(z,u)\};a(t,z)-m(z)]\leq 0 \tag{3.26}$$

$$\max[-\partial_t b(t,z)+\max_{u\in\mathbb{U}}\{-\nabla_z b(t,z)f(z,u)-c(z,u)\};b(t,z)-m(z)]\geq 0 \tag{3.27}$$

holds for all $(t,z)\in(0,T)\times\mathbb{R}^{n_x}$. *Then, for any* $t\in[0,T]$, *we have*

$$\sup_{z\in\mathbb{R}^{n_x}}(a(t,z)-b(t,z))\leq\max(\sup_{z\in\mathbb{R}^{n_x}}(a(T,z)-b(T,z)),0), \tag{3.28}$$

and the value function described in (3.16) *can be characterized as the unique viscosity solution of equation* (3.21).

Proof. Exploiting assumptions (A1),(A2),(A4) and (A7) we can apply Lemma 2.11 from [9] Chapter III and derive for Hamiltonian $H(z,p):=\max_{u\in\mathbb{U}}\{-p\cdot f(z,u)-c(z,u)\}$ the properties

$$|H(z,p)-H(z,q)|\leq M_f|p-q|$$

$$|H(z,p)-H(y,p)|\leq L(|p|+1)|z-y|$$

for any $z,y\in\mathbb{R}^{n_x}$ and $p,q\in\mathbb{R}$ with $L:=\max(L_c,L_f)$. Furthermore, we define a transformed



Hamiltonian function $\tilde{H}(z,p) = -H(z,-p)$ and prove above properties to remain valid

$$\begin{aligned}
|\tilde{H}(z,p) - \tilde{H}(z,q)| &= |-H(z,-p) - (-H(z,-q))| \\
&= |H(z,-p) - H(z,-q)| \leq M_f|-p-(-q)| = M_f|p-q| \\
|\tilde{H}(z,p) - \tilde{H}(y,p)| &= |-H(z,-p) - (-H(y,-p))| \\
&= |H(z,-p) - H(y,-p)| \leq L(|p|+1)|z-y|.
\end{aligned}$$

In order to be able to exploit the results from [5]Theorem A.1, we reformulate our HJB equation (3.21) into

$$\begin{aligned}
\min[-\partial_t u(t,z) + \tilde{H}(z, \nabla_z u(t,z)); u(t,z) - g(z)] = 0 \qquad & \text{on } (0,T) \times \mathbb{R}^{n_x} \\
u(T,z) = g(z) \qquad & z \in \mathbb{R}^{n_x}
\end{aligned}$$

using $u(t,z) = -v(t,z)$ as well as $g(z) = -m(z)$. In addition, we can rewrite equations (3.26) and (3.27) to

$$\begin{aligned}
\min[-\partial_t \tilde{a}(t,z) + \tilde{H}(z, \nabla_z \tilde{a}(t,z)); \tilde{a}(t,z) - g(z)] \geq 0 \\
\min[-\partial_t \tilde{b}(t,z) + \tilde{H}(z, \nabla_z \tilde{b}(t,z)); \tilde{b}(t,z) - g(z)] \leq 0
\end{aligned}$$

such that $\tilde{a}(t,z) = -a(t,z)$ and $\tilde{b}(t,z) = -b(t,z)$ characterize a sub- and supersolution respectively. As a result, we can apply Theorem A.1 from [5] and obtain

$$\begin{aligned}
\sup_{z \in \mathbb{R}^{n_x}} (\tilde{b}(t,z) - \tilde{a}(t,z)) \leq \max(\sup_{z \in \mathbb{R}^{n_x}} (\tilde{b}(T,z) - \tilde{a}(T,z)), 0) \\
\sup_{z \in \mathbb{R}^{n_x}} (a(t,z) - b(t,z)) \leq \max(\sup_{z \in \mathbb{R}^{n_x}} (a(T,z) - b(T,z)), 0)
\end{aligned}$$

which concludes the prove of statement (3.28).
In Proposition 13 we have characterized the value function as a sub- and supersolution of equation (3.21). By definition (3.21), for any subsolution a and supersolution b holds $a(T,z) \leq m(z) \leq b(T,z)$ for any $z \in \mathbb{R}^{n_x}$. We can therefore conclude from inequality (3.28)

$$a(t,z) - b(t,z) \leq 0 \quad \text{on} [0,T] \times \mathbb{R}^{n_x}.$$

Consequently, in order to fulfill both properties a viscosity solution is necessarily the maximum subsolution as well as the minimum supersolution and therefore unique. □

Remark: To conclude uniqueness of a viscosity solution from sub and super solution differences as done above is a common strategy in proofs. In literature it is often called a proof using the comparison principle (see [9],[5]).



3.3.2 Numerical Approximation

Now that we have defined and proven all the theoretic properties of the value function for the optimal stopping case, we can move on to derive the numeric approximation scheme. Starting from the Time and Control Discretisation we derive a Semi-Lagrangian scheme as done for the case of a static end time in Chapter 3.1. Unfortunately, we cannot directly derive an approximated value function, but instead we introduce

$$\begin{aligned} v(t,z) - m(z) &\leq 0 \\ -\partial_t v(t,z) + \max_{u\in\mathbb{U}}\{-f(z,u)^\top \nabla v_z(t,z) - c(z,u)\} &\leq 0 \\ (v(t,z) - m(z)) \cdot (-\partial_t v(t,x) + \max_{u\in\mathbb{U}}\{-f(z,u)^\top \nabla v_z(t,z) - c(z,u)\}) &= 0 \end{aligned} \tag{3.29}$$

closely related to [9][Equation (4.50)] for any $(t,z) \in [0,T) \times \mathbb{R}^{n_x}$. Regarding the proof of Theorem 14, the first two equations hold for any viscosity subsolution of equation (3.21), while the third restricts v to additionally fulfill the supersolution property. As a result, above system of equations is equivalent to the Hamilton-Jacobi-Bellman equation (3.21) and can therefore be used for the approximation. Let time and control grids $\mathbb{G}_t$,$\mathbb{G}_u$, discrete points t_i, u, and discretised state function x_h follow the definitions in Section 3.1.3, we can reformulate above equations (using the same arguments as in Section 3.1.3) for the time and control discretised value function $v_h : \mathbb{G}_t \times \mathbb{R}^{n_x} \to \mathbb{R}$ to

$$\begin{aligned} &v_h(t_i,z) \leq m(z), \\ &v_h(t_i,z) \leq \inf_{u\in\mathbb{G}_u}\{v(t_{i+1}, z + h_t f(z,u)) + h_t \cdot c(z,u)\}, \\ &v_h(t_i,z) = [\inf_{u\in\mathbb{G}_u}\{v(t_{i+1}, z + h_t f(z,u)) + h_t \cdot c(z,u)\} \ \vee \ m(z).] \end{aligned}$$

with $\vee$ representing the logical *or* operation. To make the approximated value function satisfy all of these conditions, we fix it to

$$\begin{aligned} v_h(t_i,z) &= \min(m(z), \min_{u\in\mathbb{G}_u}\{v_h(t_{i+1}, z + h_t f(z,u)) + h_t \cdot c(z,u)\}) \\ v_h(T,z) &= m(z) \end{aligned} \tag{3.30}$$

for all $(t_i,z) \in \mathbb{G}_t \times \mathbb{R}^{n_x}$. As already done for the fixed end time in Section 3.1.3, we want to make sure our discretised value function does converge towards its continuous counterpart for $h_t, h_u \to 0$. In order to do so, we start from a short Lemma, that will be heavily exploited within the convergence proofs:

Lemma 15 *Let $a,b,c,d \in \mathbb{R}$. Then,*

$$|\min(a,b) - \min(c,d)| \leq |a-c| + |b-d|.$$



Proof. Exploiting the minimum definition, we can rewrite

$$|\min(a,b) - \min(c,d)| = \left|\frac{1}{2}(a+b-|a-b|) - \frac{1}{2}(c+d-|c-d|)\right|.$$

Furthermore, using the triangular inequality we can derive

$$\begin{aligned}
|\min(a,b) - \min(c,d)| &= \frac{1}{2}|a+b-c-d+|c-d|-|a-b|| \\
&\leq \frac{1}{2}(|(a-c)+(b-d)|+|c-d-a+b|) \\
&\leq \frac{1}{2}(|a-c|+|b-d|+|c-a|+|b-d|) \\
&\leq |a-c|+|b-d|
\end{aligned}$$

to conclude the proof. □

Compared to the original DP discretisation in 3.1.3, we are using exactly the same schemes to approximate time and control. As a result, the following extension of Theorem 7 is heavily based on its original arguments.

Theorem 16 *Let assumptions* (A1) - (A7) *hold. Then, for any* $t_i \in \mathbb{G}_t$ *we have*

$$|v(t_i,z) - v_h(t_i,z)| \in \mathcal{O}(h_t)$$

Proof. Following the arguments on Theorem 7, we proof this statement by induction backward in time. We start from

$$|v(T,z) - v_h(T,z)| = |m(z) - m(z)| = 0 \in \mathcal{O}(h_t)$$

obtained from the definition of the value functions (3.16) and (3.30). For the induction step, we move backward in time from $t_i \to t_{i-1}$. Again, to unburden the notation, we make use of abbreviations $x(\tau;u) \coloneqq x(\tau;t_{i-1},z,u)$ and $x_h(t_i;u) \coloneqq z + h_t f(z,u(t_{i-1}))$. Plugging in the functions definitions, as well as Proposition 11 we obtain

$$\begin{aligned}
&|v(t_{i-1},z) - v_h(t_{i-1},z)| = \\
&|\inf_{u\in\mathcal{U}} \min[\min_{t_f\in[t_{i-1},t_i]} J(t_f,u;t_{i-1},z), \int_{t_{i-1}}^{t_i} c(x(s;u),u(s))ds + v(t_i,x(t_i;u))] \\
&- \min[m(z), \min_{u\in\mathbb{G}_u}\{v_h(t_i,z+h_t f(z,u)) + h_t\cdot c(z,u)\}]|
\end{aligned}$$



Subsequently, we can exploit Lemma 15 to derive

$$|v(t_{i-1},z)-v_h(t_{i-1},z)| \leq |\inf_{u\in\mathcal{U}} \min_{t_f\in[t_{i-1},t_i]} J(t_f,u;t_{i-1},z)-m(z)|$$
$$+|\inf_{u\in\mathcal{U}} \int_{t_{i-1}}^{t_i} c(x(s;u),u(s))ds + v(t_i,x(t_i;u)) - \min_{u\in\mathbb{G}_u}\{v_h(t_i,z+h_t f(z,u))+h_t\cdot c(z,u)\}|$$

Please note that the structure of the second part of the addition matches Equation (3.6) in Theorem 7. As a result, the corresponding arguments can be fully applied here, such that we end up with

$$|v(t_{i-1},z)-v_h(t_{i-1},z)| \leq |\inf_{u\in\mathcal{U}} \min_{t_f\in[t_{i-1},t_i]} J(t_f,u;t_{i-1},z)-m(z)| + \mathcal{O}(h_t).$$

Additionally, as t_{i-1} is considered as a possible optimal end time in the first part, we can directly observe

$$\inf_{u\in\mathcal{U}} \min_{t_f\in[t_{i-1},t_i]} J(t_f,u;t_{i-1},z) \leq m(z)$$

from the definition of function J. Consequently, since for any $\epsilon>0$ exists $\bar{u}\in\mathcal{U}$ such that

$$\inf_{u\in\mathcal{U}} \min_{t_f\in[t_{i-1},t_i]} J(t_f,u;t_{i-1},z) \geq \min_{t_f\in[t_{i-1},t_i]} J(t_f,\bar{u};t_{i-1},z)-\epsilon,$$

we additionally choose

$$\bar{t}_f := \arg\min_{t_f\in[t_{i-1},t_i]} J(t_f,\bar{u};t_{i-1},z)$$

and derive upper bound

$$\begin{aligned}|v(t_{i-1},z)-v_h(t_i,z)| &\leq |\int_{t_{i-1}}^{\bar{t}_f} c(x(s;\bar{u}),\bar{u}(s))ds| + |m(x(\bar{t_f};\bar{u}))-m(z)| + \epsilon + \mathcal{O}(h_t)\\ &\leq M_c(\bar{t}_f-t_{i-1}) + L_m|\int_{t_{i-1}}^{\bar{t}_f} f(x(s;\bar{u}),\bar{u}(s))ds| + \epsilon + \mathcal{O}(h_t)\\ &\leq (M_c+L_mM_f)h_t + \epsilon + \mathcal{O}(h_t).\end{aligned}$$

For ϵ tending to zero, we obtain the assumption. □

We can therefore observe that the optimal stopping value function discretised with respect to control and time does indeed converge towards its continuous counterpart for $h_t, h_u \to 0$.
To finally apply the method of dynamic programming numerically as illustrated in algorithm 1 to the optimal stopping case, a state discretisation is required. We use the same as in Section 3.1 resulting, with state grid $\mathbb{G}_x$ as well as interpolation functions Π_j,α_j, in the fully discretised value function $\tilde{v_h} : \mathbb{G}_t \times \mathbb{R}^{n_x} \to \mathbb{R}$

$$\tilde{v}_h(t_i,z) = \sum_{j=1}^{2^{n_x}} \alpha_j(z)\cdot \tilde{v}_h(t_i,\Pi_j(z)) \tag{3.31}$$



such that we can adapt the DPP from equation (3.30) into

$$\tilde{v}_h(t_i, \Pi_j(z)) = \min(m(\Pi_j(z)), \min_{u \in \mathbb{G}_u} \{\tilde{v}_h(t_{i+1}, z + h_t f(\Pi_j(z), u)) + h_t \cdot c(t_i, \Pi_j(z))\})$$
$$\tilde{v}_h(t_{K_t}, \Pi_j(z)) = m(\Pi_j(z)) \tag{3.32}$$

for all $z \in \mathbb{R}^{n_x}$ and $t_i \in \mathbb{G}_t$. To prove its convergence towards v_h for $h_x \to 0$ we follow the arguments of Section 3.1 adapting Lemma 8 to the optimal stopping case.

Lemma 17 *Assume* (A1) - (A4). *Then, there exists a constant* $L_{v_h} > 0$ *such that*

$$|v_h(t_i, z) - v_h(t_i, y)| \leq L_{v_h}|z - y|$$

holds for any $z, y \in \mathbb{R}^{n_x}$ *and* $t_i \in \mathbb{G}_t$

Proof. We follow the arguments in Lemma 8 and verify the inequality

$$|v_h(t_{K_t-n}, z) - v_h(t_{K_t-n}, y)| \leq [(1 + L_f h_t)^n \cdot (L_c h_t n + L_m + n L_m)]|z - y| \tag{3.33}$$

using induction over n. For the base case $n = 0$ we again obtain

$$|v_h(T, z) - v_h(T, y)| = |m(z) - m(y)| \leq L_m|z - y|$$

using the Mayer term Lipschitz property (A3). Proceeding with the induction step, we move from $n \to n + 1$ and obtain

$$\begin{aligned}
&|v_h(t_{K_t-(n+1)}, z) - v_h(t_{K_t-(n+1)}, y)| \\
&\leq |\min[m(z), \min_{u_1 \in \mathbb{G}_u} \{v_h(t_{K_t-n}, z + h_t f(z, u_1)) + h_t \cdot c(z, u_1)\}] \\
&\quad - \min[m(y), \min_{u_2 \in \mathbb{G}_u} \{v_h(t_{K_t-n}, z + h_t f(y, u_2)) + h_t \cdot c(y, u_2)\}]| \\
&\leq |m(z) - m(y)| + |\min_{u_1 \in \mathbb{G}_u} \{v_h(t_{K_t-n}, z + h_t f(z, u_1)) + h_t \cdot c(z, u_1)\} \\
&\quad - \min_{u_2 \in \mathbb{G}_u} \{v_h(t_{K_t-n}, z + h_t f(y, u_2)) + h_t \cdot c(y, u_2)\}|
\end{aligned}$$

exploiting the value function definition as well as Lemma 15. For the second part, we can directly apply the arguments from original Lemma 8 such that

$$\begin{aligned}
&|v_h(t_{K_t-(n+1)}, z) - v_h(t_{K_t-(n+1)}, y)| \\
&\leq L_m|z - y| + [(1 + L_f h_t)^{n+1} \cdot (L_c h_t (n + 1) + L_m + n L_m)]|z - y| \\
&\leq [(1 + L_f h_t)^{n+1} \cdot (L_c h_t (n + 1) + L_m + (n + 1) L_m)]|z - y|
\end{aligned}$$

concludes the proof. □



Finally, we adapt Lemma 9 to the optimal stopping case

Lemma 18 *Assume* (A1) - (A7). *For any time point* $t_i \in \mathbb{G}_t$ *and any state* $z \in \mathbb{R}^{n_x}$ *holds*

$$|\tilde{v}_h(t_i, z) - v_h(t_i, z)| \leq L_{v_h}(K_t + 1 - i)|h_x| \tag{3.34}$$

using Lipschitz constant L_{v_h} *from Lemma 17.*

Proof. Following the proof of Lemma 9, we initially show

$$|\tilde{v}_h(t_i, \tilde{z}) - v_h(t_i, \tilde{z})| \leq L_{v_h}(K_t - i)|h_x| \tag{3.35}$$

for any $t_i \in \mathbb{G}_t$ and any grid point $\tilde{z} \in \mathbb{G}_x$ by induction. We start from base $i = K_t$ with

$$|\tilde{v}_h(T, \tilde{z}) - v_h(T, \tilde{z})| = |m(\tilde{z}) - m(\tilde{z})| = 0.$$

For the induction step we move backward in time from $i \to i - 1$ and exploit the value function definitions (3.30) and (3.31),(3.32)

$$\begin{aligned}|\tilde{v}_h(t_{K_t-(n+1)}, \tilde{z}) - v_h(t_{K_t-(n+1)}, \tilde{z})| = \\ |\min(m(\tilde{z}), \min_{u_1 \in \mathbb{G}_u} \{\tilde{v}_h(t_{K_t-n}, \tilde{z} + h_t f(\tilde{z}, u_1)) + h_t \cdot c(\tilde{z}, u_1)\}) \\ - \min(m(\tilde{z}), \min_{u_2 \in \mathbb{G}_u} \{v_h(t_{K_t-n}, \tilde{z} + h_t f(\tilde{z}, u_2)) + h_t \cdot c(\tilde{z}, u_2)\})|\end{aligned}$$

to begin the proof. Subsequently, exploiting Lemma 15 results in

$$\begin{aligned}|\tilde{v}_h(t_{K_t-(n+1)}, \tilde{z}) - v_h(t_{K_t-(n+1)}, \tilde{z})| \leq |m(\tilde{z}) - m(\tilde{z})| \\ + |\min_{u_1 \in \mathbb{G}_u} \{\tilde{v}_h(t_{K_t-n}, \tilde{z} + h_t f(\tilde{z}, u_1)) + h_t \cdot c(\tilde{z}, u_1)\} \\ - \min_{u_2 \in \mathbb{G}_u} \{v_h(t_{K_t-n}, \tilde{z} + h_t f(\tilde{z}, u_2)) + h_t \cdot c(\tilde{z}, u_2)\}|\end{aligned}$$

Here, the structure is equivalent to the one considered in Lemma 9. We can therefore follow the arguments therein to obtain

$$|\tilde{v}_h(t_{K_t-(n+1)}, \tilde{z}) - v_h(t_{K_t-(n+1)}, \tilde{z})| \leq L_{v_h}(K_t - (i-1))|h_x|$$

and close the proof.
Proven inequality (3.35) valid, we can exactly follow the arguments in Lemma 9 to derive inequality (3.34). □
Having a quantification for the errors from v to v_h as well as from v_h to $\tilde{v}_h$ we can finally develop an optimal stopping counterpart of Theorem 10 to derive convergence of the fully discretised value function towards its continuous counterpart. As we can directly follow the arguments of



the original theorem, we do not explicitly state the proof here.

Theorem 19 *Assume* (A1) - (A7) *as well as*

$$h_t^p > |h_x|$$

with $p > 1$. *Then, for an arbitrary time point* $t_i \in \mathbb{G}_t$, *and state* $z \in \mathbb{R}^{n_x}$

$$\lim_{h_t \to 0} |v(t_i, z) - \tilde{v}(t_i, z)| = 0$$

holds.

Using all of the properties derived in this section, we can now formulate an Approximate Dynamic Programming algorithm by replacing the approximate value function definition in Algorithm 1 with its optimal stopping variant. Following the same arguments as in Section 3.1.5, we can characterize the discrete solution trajectory to be globally optimal with respect to time, control and state discretisation. Additionally, we obtain exactly the same asymptotic boundaries on execution time and memory requirements as in Section 3.1.5. As a consequence, the method of dynamic programming with optimal stopping enables solution of a variable end time problem in the same order of magnitude than a fixed end time problem.



3.4 Iterative State Space Splitting

As pointed out before, the main issue arising from the method of dynamic programming is the exponential dependency of its costs on the number of time, control and state grid points (see Algorithm 1). In other words, a small decrease in reference points can cause a large improvement in the algorithms performance. In contrast, a decrease in the number of grid points results, at least on equidistant grids, in less accurate solutions. As a result, there exist various strategies in the current literature exploiting specialized non-equidistant grid structures, to lower the DP algorithms computational requirements on equivalent quality of solutions. In [33, 34], a quasi-random sampling strategy is proposed that achieves balanced covering of the state space, without the exponential growth in the number of grid points, while [18] employs sparse grid strategies [23]. A possible alternative is the strategy of adaptive grids, where we start from a coarse initial grid and subsequently add reference points where needed. Originally introduced in the context of finite elements [7, 151], it is applied for optimistic planning methods [17, 25] as well as approximate dynamic programming [111, 133]. The existing strategies mainly differ in the criteria proposed to identify interesting regions for refinement. While [68] and [135] use large local Bellman residuals, the authors of [111] introduce various criteria depending on local properties of the value function. Subsequently, they compare their performance to a global strategy taking also the effect of a single reference point on the overall function into account. This is further developed in [133]. Regarding all of these different criteria, we can observe that each of them is dependent on the value function. As a result, its (usually expensive) evaluation is required in every iteration of the grid adaptation. Instead, the refinement strategy introduced in the following is especially designed for trajectory planning with respect to collision avoidance and depends solely on geometric scene properties instead of the value function. While the arising strategy ignores local problem properties arising from the dynamics, it can especially be a more efficient alternative for high dimensional crowded scenes. In the following, we develop the adaptive state grid approach using our criterion as an extension to the standard Algorithm 1. We point out convergence of the splitting procedure towards the continuous state space on infinitely many sampling iterations, resulting in convergence to the continuous value function on $h_u, h_t \to 0$ and finally convergence of the approximated solution trajectory to the globally optimal solution of the initial problem.

3.4.1 Geometry based State grid refinement

Instead of focusing on properties of the value function, the refinement criterion proposed in this section characterizes regions of interest in the current state grid based on their geometry. For this purpose, we initially impose some assumptions on the adaptive grid structure before moving on to the criterion definition. Given a state grid $\mathbb{G}_x$ that covers the compact state space $\mathcal{K} \subset \mathbb{R}^{n_x}$ with maximum cell size h_x, we define set $\mathbb{S}$ to represent the cells formed by its vertices. Additionally, we assume existence of a refinement operation breaking a cell apart by introduction of new vertices into $\mathbb{G}_x$. Denoting the size of the original cell with $\Delta\mathcal{S} > 0$, as well



as the maximum size of any arising subcell by $\Delta\tilde{\mathcal{S}}$ we impose the assumption

$$\Delta\mathcal{S} > \Delta\tilde{\mathcal{S}} \tag{3.36}$$

on the splitting process. Finally, to measure geometric properties at a certain state, we initialize the total signed distance function $\delta : \mathcal{K} \to \mathbb{R}$

$$\delta(z) = \min_{a \in A(z), b \in B(z)} d(a, b)$$

with state dependent polygon sets A and B representing the geometric environment and the signed distance function d from Chapter 2.

Remark: Above restrictions are introduced to allow the usage of various grid structures in practice. While especially in two and three dimensional state spaces, triangulation methods have been proven beneficial [111, 133], we use the d-dimensional cube grid that we introduced in [128] within our numerical examples due to its easy scalability on high dimensional spaces.

Out of this definition of an appropriate adaptive grid structure, we are looking for a strategy to detect regions with interesting geometric properties for refinement. On the one hand, cells containing small absolute distance values might be a good choice due to their proximity to collision avoidance violation. On the other hand, it might also increase the performance to refine a large cell further away from the obstacles boundaries, instead of a small one that is close. Keeping these intuitions in mind, we combine the two arguments, namely large cell size and small absolute distance value, with a small constant $\epsilon > 0$ to formulate our criterion $r : \mathbb{S} \to \mathbb{R}$

$$r(\mathcal{S}) = \frac{\max_{z_l \in \mathcal{S}} |\delta(z_l)| + \epsilon}{(\Delta\mathcal{S})^2} \tag{3.37}$$

with variable z_l iterating over all grid vertices spanning cell $\mathcal{S}$. The arising splitting strategy illustrated in Algorithm 2, starts from an initial grid $\mathbb{G}_x^0$ and uses equation (3.37) in every iteration to identify the smallest τ percent of the cells to be refined. Subsequently, the arising additional reference points are added into the grid and the set of cells is adapted. Algorithm 2 terminates after a fixed number of iterations which allows us to provide an upper bound on the overall reference points in the resulting grid. Afterwards, the standard dynamic programming algorithm from Algorithm 1 can be evaluated using the non-equidistant state grid. As the maximum state space error arising within the initial grid is still bounded by h_x, the convergence of the interpolated approximation towards the continuous value function from Theorem 10 remains valid.

To additionally show our refinement algorithm to result in a balanced covering of the state space on large numbers of iterations, the remaining part of this section will focus on its convergence to the continuous state space on infinitely many iterations. Let use start with the definition of



Algorithm 2: Geometry Based State Space refinement

Initialize state space grid $\mathbb{G}_x^0$

for $\eta \leftarrow 0, \ldots, N$ **do**

  Evaluate $r(S)$ for every $S \in \mathbb{S}^\eta$

  Refine the τ percent lowest criterion cells such that $\mathbb{S}^\eta \rightarrow \mathbb{S}^{\eta+1}$

Return grid $\mathbb{G}_x^N$

a lower bound on the refinement criterion in every cell.

Lemma 20 *For any cell $\mathcal{S} \in \mathbb{S}$ the refinement criterion r is bounded from below by*

$$\frac{\epsilon}{(\Delta\mathcal{S})^2} \leq r(\mathcal{S}).$$

Furthermore, let $\mathcal{S}^\eta$ denote a cell arising from $\eta \in \mathbb{N}$ refinement operations on $\mathcal{S}$,

$$\lim_{\eta \to \infty} r(S^\eta) = \infty$$

holds.

Proof. Using the positive definiteness of absolute and square function as well as the criterion definition (3.37), we can derive

$$0 \leq \frac{\max_{z_l \in \mathcal{S}} |\delta(z_l)|}{(\Delta\mathcal{S})^2}$$

leading directly to the inequality. Subsequently, we obtain

$$\lim_{\eta \to \infty} \frac{\epsilon}{(\Delta\mathcal{S}^\eta)^2} = \infty$$

from assumption (3.36) and combine it with the lower bound inequality on the criterion to finish the proof. □

Subsequently, we can show the algorithm to converge towards the continuous state space on infinitely many refinement iterations. We note that this is the case if and only if the maximum cell size converges to zero. In the following theorem, we proof the latter statement.



Theorem 21 *Given a state space grid $\mathbb{G}_x$ consisting of cells $\mathbb{S}$ that is refined $\eta \in \mathbb{N}$ times with respect to Algorithm 2. Then for $\eta \to \infty$*

$$\lim_{\eta \to \infty} \max_{\mathcal{S} \in \mathbb{S}^\eta} \Delta \mathcal{S} = 0$$

holds for any cell in the refined grid $\mathbb{G}^\eta$ with $\mathbb{S}^\eta$.

Proof. For the proof by contradiction let us suppose there exists a level set of cells $\hat{\mathbb{S}} \subseteq \mathbb{S}$ that is no longer subject to refinement in subsequent iterations and let $c = r(\hat{\mathcal{S}})$ denote the refinement criterion value of all $\hat{\mathcal{S}} \in \hat{\mathbb{S}}$. Considering now infinitely many refinement iterations with respect Algorithm 2 on the grid, there exists a least one cell $\tilde{S}_j$ that is refined infinitely many times. Following the arguments in Lemma 20 it therefore satisfies $\lim_{\eta \to \infty} r(\mathcal{S}_j^\eta) > c$. We can repeat this procedure until every cell in $\mathbb{S} \setminus \hat{\mathbb{S}}$ was split enough times to have criterion values larger c. As this requires the subsequent algorithm iteration to refine at least one cell in $\hat{\mathbb{S}}$ it contradicts the assumption. □



3.5 Parallelization using CPU/GPU

Initially, let us recapitulate the overall goal of this section, namely to limit the curse of dimensionalities impact on runtime and memory requirements. The approach of optimal stopping, as well as the introduction of an optimal state space grid, focused on the reduction of grid points to reduce the overall number of computation steps. In contrast, the strategy of parallelization proposed in this section, simply tries to reduce the impact of the number of operations on the computation time by executing them in parallel where possible. Regarding DP literature, early works propose domain decomposition [52, 29] to allow parallel evaluation of smaller problem parts. More recently, the huge parallelisation capabilites of current graphics processing units (GPUs) are exploited in various applications based on dynamic programming such as combinatorial optimization [20, 13] and bioinformatics [104, 88]. Also within the context of robotic motion planning, parallelisation based acceleration of common planning techniques, including differential dynamic programming [121] and nonlinear model predictive control [125, 81] are proposed. Recently, [21] also suggested GPU based acceleration in discrete dynamic programming. In the following, we initially analyze our standard discrete dynamic programming algorithm to identify structural parallelization capabilities. Subsequently, we introduce the two main hardware structures used for parallelization, namely CPU and GPU, and highlight their benefits and limits. Finally, we propose a parallel dynamic programming algorithm that, in contrast to the one proposed in [21] incorporates limitations arising from common hardware structures, giving rise to an increase in scalability to high dimensional problems, as well as in performance with respect to runtime.

3.5.1 DP Parallelization Capabilities

In order to make the parallel version of a sequential strategy evaluate exactly the same result on every possible input, thus modeling the same operation, the parts executed in parallel are required to be fully independent from each other. Hence, on analyzing the parallelization capabilities, we are actually searching for parts of an algorithm that can be executed independently. Regarding the Backward Pass of the Dynamic Programming Algorithm 1 we can identify three nested loops over time, state and control grid elements. The outermost one iterates the elements of the time grid, and due to the exploitation of the recursive principle of optimality its evaluation does naturally depend on the previous time step. In contrast, the evaluation of the value function for every point in the space grid is independent, and can therefore be executed in parallel. Furthermore, in order to find the minimum control, all candidates from grid $\mathbb{G}_u$ can independently be evaluated before comparison, giving again rise to parallelization. These independent control grid entries are also the only resource for parallelization within the forward pass. The DP Algorithm with all these possibilities for parallel execution highlighted in blue is illustrated in Algorithm 4. Interestingly, considering the possibility to execute as many operations in parallel as needed, the asymptotic runtime of the arising algorithm reduces to $\mathcal{O}(K_t)$. Nevertheless, the practical acceleration achieved is limited by the capability of current hard-



ware. Finally, we should point out that parallelization of the control does increase the temporal memory requirements, due to the additional introduction of vector C in Algorithm 4.

Algorithm 3: Approximate Dynamic Programming Parallelization Capabilities

Initialize grids $\mathbb{G}_t$, $\mathbb{G}_u$, $\mathbb{G}_x$

Backwardpass

for $z_l \in \mathbb{G}_x$ do in parallel $\tilde{v}_h(t_{K_t}, z_l) = m(z_l)$

for $i \leftarrow K_t - 1, \dots, 0$ **do**

  for $z_l \in \mathbb{G}_x$ do in parallel

    Init vector $C \in \mathbb{R}^{|\mathbb{G}_u|}$

    for $u_j \in \mathbb{G}_u$ do in parallel

      $C_j = \{\tilde{v}_h(t_{i+1}, z_l + h_t f(z_l, u_j)) + h_t \cdot c(z_l, u_j)\}$

    $\tilde{v}_h(t_i, z_l) = \min_{j \in 0,\dots,|\mathbb{G}_u|} C_j$

Forwardpass

Start from initial value $x_h(t_0) = z$

for $i \leftarrow 0, \dots, K_t - 1$ **do**

  Init vector $C \in \mathbb{R}^{|\mathbb{G}_u|}$

  for $u_j \in \mathbb{G}_u$ do in parallel

    $C_j = \{\tilde{v}_h(t_{i+1}, x_h(t_i) + h_t f(x_h(t_i), u)) + h_t \cdot c(x_h(t_i), u)\}$

  Evaluate $u_h(t_i) := \arg\min_{j \in 0,\dots,|\mathbb{G}_u|} C_j$

  Store the corresponding state $x_h(t_{i+1}) = x_h(t_i) + h_t f(x_h(t_i), u_h(t_i))$

Return state function x_h and piecewise constant interpolated controls u_h

3.5.2 DP Parallelization in practice

The analysis for parallelization capabilities in the dynamic programming algorithm within the last section gives, at least in theory, huge rooms for runtime improvements. As a result, this section provides an overview on practical possibilities to implement parallelization. In recent architectures, we can either exploit the structure of a processor containing multiple central processing units (CPU), or outsource parallel processes to the graphics processing unit (GPU). While typical processors allow a maximum of 8 (on regular of the shelf PCs) to 128 (on high performance gaming processors) operations executed in parallel, the structure of the GPU is especially optimized for parallelization and therefore allows execution of thousands of concurrent threads. On the other hand, a single CPU kernel is optimized for maximum speed at every operation, making the evaluation of a single sequence much more efficient than on GPU. As a result, in order to make GPU based parallelization beneficial, the considered algorithm should allow large numbers of operations executed in parallel to compensate for the drawback on every single execution. Considering high dimensional discrete state and control grids this is naturally the case for our Dynamic Programming algorithm.

In addition to the cost for executing the operations itself on the threads, some management



overhead arises from parallelization in practice. In case of using the CPU structure, this mostly contains the time for initializing and synchronizing the arising threads. In contrast, as GPU can not access regular memory, input and output data of operations executed on the GPU have to be explicitly copied before and after computations. Depending on the size of the data, this memory transfer can be very costly in practice.

Keeping these properties in mind, we can now analyze the theoretic parallelization capabilities of dynamic programming for their practical efficiency. Starting from the backward pass, the theoretic algorithm suggests parallel evaluation of the Mayer term at every final time grid point. Due to the large number of state space grid points obtained in high dimensional systems, not only parallelization on CPU but even on GPU can be beneficial here. Subsequently, within a parallelized state grid traversal, an additional parallel loop over the control grid is requested. Regarding CPU structures and high dimensional state grids, all threads available are already used to traverse the state grid. Thus, additional parallelization requests on the control grid imply, instead of performance increase, additional management cost and less optimal thread synchronization that possibly can result in idle computational resources and an overall increased evaluation time. On huge state grids, the same effect can even occur on GPU structures. Additionally, the enhanced memory requirement resulting from introduction of vector C in Algorithm 3 can be problematic as memory resources on GPU are usually limited. Finally, implementation of such a nested loop parallelization structure proposes, even though possible, additional challenges resulting in increasing management overhead on the GPU. As a result, we consider an algorithm traversing only the state grid in parallel most beneficial within the Backward pass of the DP algorithm. Regarding the forward pass, the parallelization of the control space loop is no longer nested and can therefore also be beneficial in practice. As a result, we propose the Algorithm 4 for efficient DP evaluation and compare its performance arising from CPU as well as GPU based parallelization to the parallel Algorithm proposed in [21] within example Section 3.6.



Algorithm 4: Parallel Approximate Dynamic Programming

Initialize grids $\mathbb{G}_t$, $\mathbb{G}_u$, $\mathbb{G}_x$

Backwardpass

for $z_l \in \mathbb{G}_x$ do in parallel $\tilde{v}_h(t_{K_t}, z_l) = m(z_l)$

for $i \leftarrow K_t - 1, \ldots, 0$ **do**

  for $z_l \in \mathbb{G}_x$ do in parallel

    $\tilde{v}_h(t_i, z_l) = \min_{u_j \in \mathbb{G}_u} \{\tilde{v}_h(t_{i+1}, z_l + h_t f(z_l, u_j)) + h_t \cdot c(z_l, u_j)\}$

Forwardpass

Start from initial value $x_h(t_0) = z$

for $i \leftarrow 0, \ldots, K_t - 1$ **do**

  Init vector $C \in \mathbb{R}^{|\mathbb{G}_u|}$

  for $u_j \in \mathbb{G}_u$ do in parallel

    $C_j = \{\tilde{v}_h(t_{i+1}, x_h(t_i) + h_t f(x_h(t_i), u)) + h_t \cdot c(x_h(t_i), u)\}$

  Evaluate $u_h(t_i) \coloneqq \operatorname{argmin}_{j \in 0, \ldots, |\mathbb{G}_u|} C_j$

  Store the corresponding state $x_h(t_{i+1}) = x_h(t_i) + h_t f(x_h(t_i), u_h(t_i))$

Return state function x_h and piecewise constant interpolated controls u_h



3.6 Experiment: KUKA YouBot Trajectory Planning

In the previous sections we introduced the method of dynamic programming to solve optimal control problems. As the resulting algorithm suffers from the curse of dimensionality, its runtime and memory requirements can become impractical or at least expensive in high dimensional systems. We therefore suggested multiple strategies to lower the curses impact. In this section, we want to prove that at least for a relatively small system using dimension four in state and control space, dynamic programming, in combination with the acceleration strategies proposed above, can still be efficient in practice. Our experimental setup, described in detail in Section 3.6.1 asks a robotic manipulator to perform a pick and place task in a crowded environment. We compare the original dynamic programming algorithm to the suggested improvement strategies with respect to runtime and solution quality on this exemplary problem. As we aim to proof efficiency of every single improvement strategy before combining them all together at the end, we start from the original algorithm using time transformation and compare to its optimal stopping adaptation in Section 3.6.2. Subsequently, we evaluate the performance of different strategies suggested for parallelisation in Section 3.6.3 and finally compare the geometric adaptive state grid splitting algorithm to strategies using different splitting criteria as well as equidistant grids in Section 3.6.4.

3.6.1 Setup

In this experiment, we model a KUKA youBot robotic arm as illustrated in Figure 3.2. We plan its trajectory from an initial joint state towards a desired goal position, while avoiding collision between robotic parts and obstacles from the scene. Figure 3.3 illustrates the scene configuration used throughout this section.

As we are only interested in a desired goal position, independent from the orientation, we use a simplified model of the robotic structure considering its first four joint angles as states, while keeping the last one static at configuration zero. Considering velocity control at each joint, we obtain a system with dimension four on state and control domain. Due to the mechanical structure of the robot, our state and control spaces Q and U are bounded by

$$Q \coloneqq \{q \in \mathbb{R}^4 | (-2.95, -1.13, -2.55, -1.78)^\top \text{rad} \leq q \leq (2.95, 1.57, 2.62, 1.78)^\top \text{rad}\}$$

and

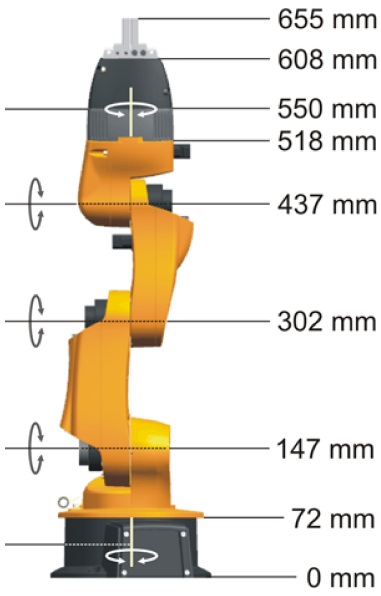
$$U \coloneqq \{u \in \mathbb{R}^4 | -1.57\,\text{rad/s} \leq u_i \leq 1.57\,\text{rad/s}, \quad i = 1, \ldots, 4\}$$

respectively. Furthermore, we model its system dynamics as

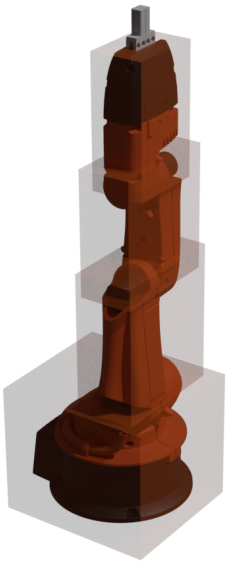
$$x'(t) = f(x(t), u(t)) = u(t)$$

and can directly obtain Assumptions (A1),(A2),(A5) as well as (A9) valid. To model collision





(a) YouBot Geometry



(b) Collision Geometry

Figure 3.2: Geometric setup of the *KUKA youBot* with joints and link lengths highlighted (left) and the collision geometry considered in the model(right). Figure 3.2a is taken from the robots user guide [94] and (3.2b) was generated based on the Blender model of the robot downloaded from [1].

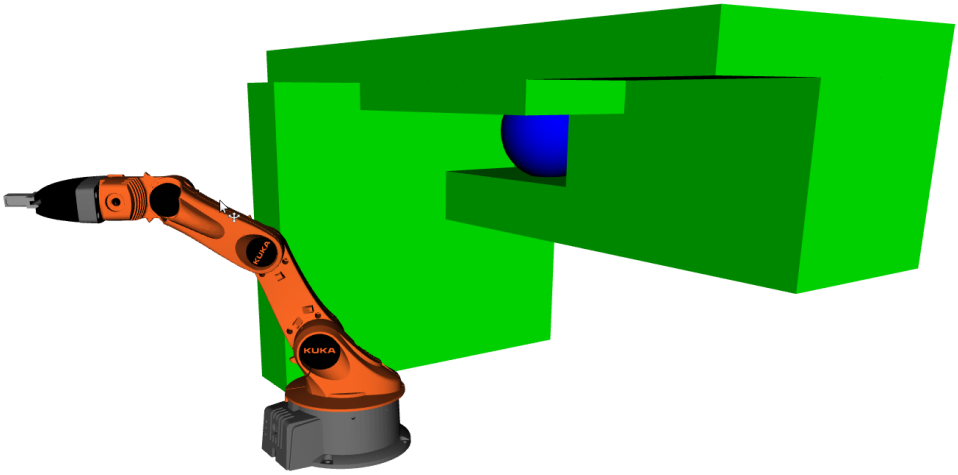


Figure 3.3: The KUKA youBot robotic arm in its initial configuration(orange), the obstacles in the scene approximated by convex polytopes (green), as well as the desired goal region for the robots TCP(blue)

avoidance, we initialize the total signed distance function $\delta : Q \to \mathbb{R}$

$$\delta(x(t)) = \min_{a \in A(x(t)), b \in B} d(a, b)$$

with polygon set $A(x(t))$ representing the robots collision geometry (see Figure 3.2b) at a certain joint configuration, environment polygons B and the signed distance function d from Chapter 2. Subsequently, we can characterize penalty function p as

$$p(x(t)) = \rho_2 \max(0, -\delta(x(t)) + \epsilon)$$

with safety distance $\epsilon = 0.01\,\text{m}$ and $\rho_2 = 5000$. In order to filter feasible results for small overall execution times as well as small changes in joint positions, we use an optimal end time structure and choose running cost function

$$c(x(t), u(t)) = 20 + \|u(t)\| + p(x(t)).$$

As the signed distance function is considered non-smooth but Lipschitz continuous [64] assumptions (A4) and (A7) hold. As suggested in Section 3.2, we incorporate the goal sphere (see Figure 3.3) centered at point $p_g \in \mathbb{R}^3$ with radius $r = 0.075\,\text{m}$ into the Mayer term, such that we end up with

$$m(x(t)) = \rho_1 \max(0, \|\kappa(x(t)) - p_g\| - r) + p(x(t))$$

using penalty scaling $\rho_1 = 5000$ as well as function $\kappa : Q \to \mathbb{R}^3$ representing the forward kinematics of the robot. Exploiting the smoothness of κ, as well as properties of state space Q, we can verify assumptions (A3) and (A6). Finally we restrict the optimal end time to range $[0, 3\,\text{s}]$.

All the results presented in the following, are based on a C++ implementation and executed on an AMD Ryzen Threadripper 3390X system with 256GB RAM. To lower the effect of CPU throttling, we executed every computation 20 times and considered the average of the 90% lowest runtimes in our evaluations.

3.6.2 Optimal Stopping

To proof efficiency of the optimal stopping algorithm on our optimal end time problem, we compare its runtime to the standard dynamic programming algorithm given the time transformed problem as described in Section 3.2. Unfortunately, in the latter case, we lack an initial guess for the arising additional end time state and consequently can not apply Algorithm 1 as is. Instead, we search for the minimum value function end time grid point directly after the backward pass, use it as an initial guess and subsequently execute the forwardpass. Given a constant control discretisation $K_u \coloneqq (11, 11, 11, 11)^\top$ as well as the number of time points $K_t = 25$, we use the optimal joint space discretisation strategy introduced in [74] to initialize a sequence of state grids with increasing numbers of points. We therefore lower the maximum TCP position offset arising from neighbored joint state grid points step by step from $d_m = 0.9\,\text{m}$ to $d_m = 0.7\,\text{m}$. As a result, we obtain initially $K_x = (15, 7, 9, 4)^\top$ and gradually refine the grid until we end up with $K_x = (19, 9, 11, 5)^\top$. Please note that in case of the time transformed problem, we discretise the end time state grid as well as the time grid with $K_t = 25$. In order to accelerate the



evaluation process, we applied CPU parallelization according to Algorithm 4 to both strategies. Figure 3.4 illustrates the arising run-times, plotted against the number of joint state grid points considered. As expected, the additional dimension arising from the time transformation, causes

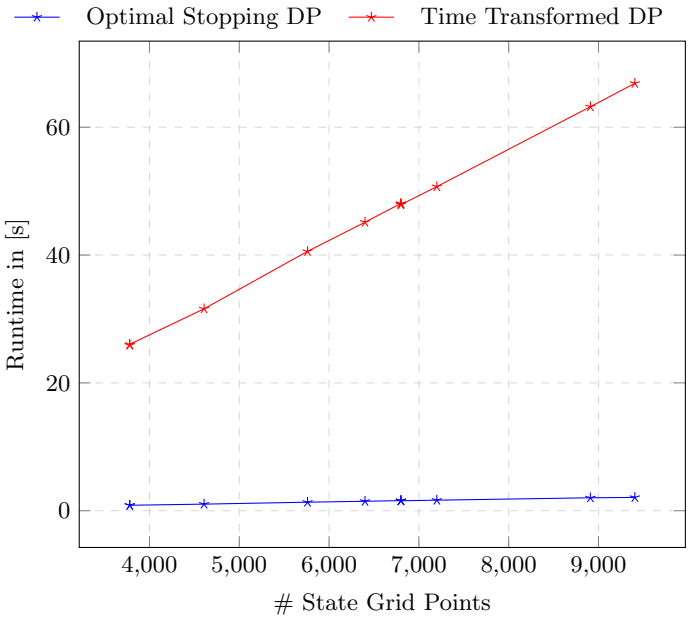


Figure 3.4: Runtime of the Optimal Stopping DP Algorithm compared to the Standard DP Algorithm given a time transformed problem, evaluated for different numbers of joint state grid points.

higher execution times compared to the optimal stopping strategy. Furthermore, on increasing numbers of grid points, the gap in between them is widening as expected from the runtime analysis given in Section 3.1.5. We can therefore conclude the optimal stopping DP algorithm as an efficient strategy to increase the DP algorithms performance on optimal end time problems.

3.6.3 Parallelisation

In Section 3.5 we introduced various strategies for parallelization of the original dynamic programming algorithm, exploiting GPU as well as CPU structures. Using our PCs 128 CPU cores as well as its local GPU (GeForce RTX 3090) with 25GB storage we apply Algorithms 3 and 4 to the YouBot trajectory planning problem. Since our GPU is only optimized for single precision operations, we stick to this datatype during our evaluations, despite the decrease in accuracy when compared to double precision. As in the previous section, we consider a constant number of discrete time points $K_t = 70$, as well as constant control space discretisation $K_u := (11, 11, 11, 11)^\top$. Subsequently, we evaluate 21 different joint space discretisations resulting from maximum TCP offset values $d_m \in [0.3\,\mathrm{m}, 0.5\,\mathrm{m}]$ using the optimal joint space discretisation proposed in [74]. The results are illustrated in Figure 3.5.

Initially, we should point out here that already given the smallest number of joint space grid points considered (32760), the original DP Algorithm, executed sequentially on CPU took on



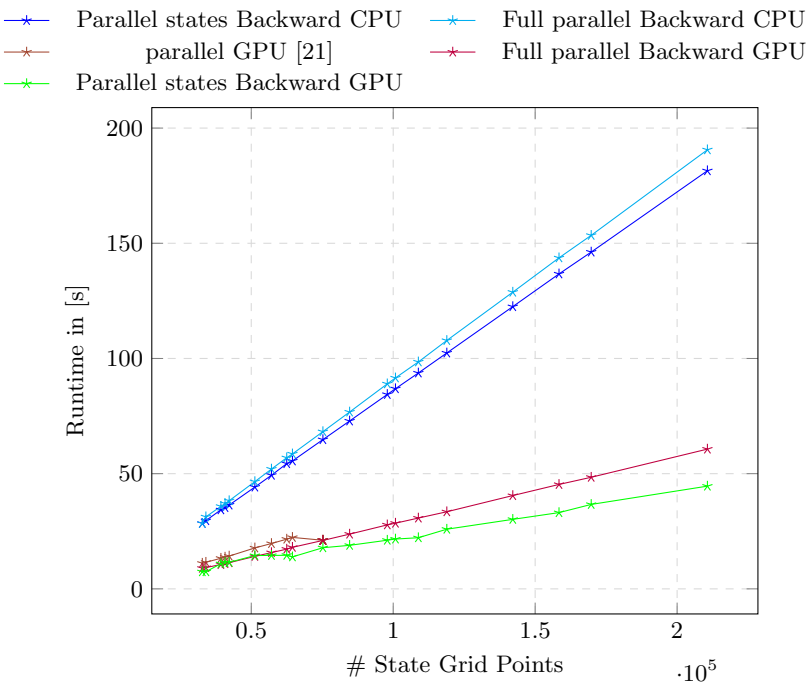


Figure 3.5: Runtime of different parallelisation strategies applied to the DP Algorithm.

average 1603 s. As a consequence, we did not include it into Figure 3.5 for readability, but can directly follow that any of the parallel algorithms compared in the following, results in a huge runtime improvement compared to sequential execution. Furthermore, we discussed different variants of parallel DP algorithms in Section 3.5 in Algorithms 3 and 4. Regarding Figure 3.5, we can obtain for execution on CPU, that the parallelization of the nested-control loop as suggested in Algorithm 3 results in sightly larger execution times than leaving it sequential. We therefore successfully verified, that parallelization of the inner loop, in case of fully exploited parallelization capabilities on the outer loop (number of states $>$ maximum number of CPU threads) results in higher run-times due to thread scheduling overhead. In contrast, we can obtain on GPU, where huge numbers of threads can be executed in parallel, that a further parallelization of the inner control loop does indeed increase the algorithms performance on small numbers of state grid points. When increasing the number of state grid points, also the parallelization capabilities of the GPU can be fully exploited by the outer loop. As a result we can again obtain a thread scheduling overhead, making the execution of Algorithm 4 even on GPU more efficient than Algorithm 3. For comparison, we also implemented the parallel GPU Algorithm proposed in [21]. As can be obtain from Figure 3.5, we were only able to execute it for small numbers of state space discretisation points, as the precomputed interpolation coefficients suggested therein require huge memory storage, resulting in a shortage of memory on our GPU very fast. While this could be solved by iteratively evaluating batches of states, the coefficients stored in global GPU memory are additionally quite costly to access on linear interpolation evaluation. Consequently, we obtain an increase in runtime when compared to the execution of Algorithm 4 on GPU using the grid structure proposed in [128]. In total, we can conclude



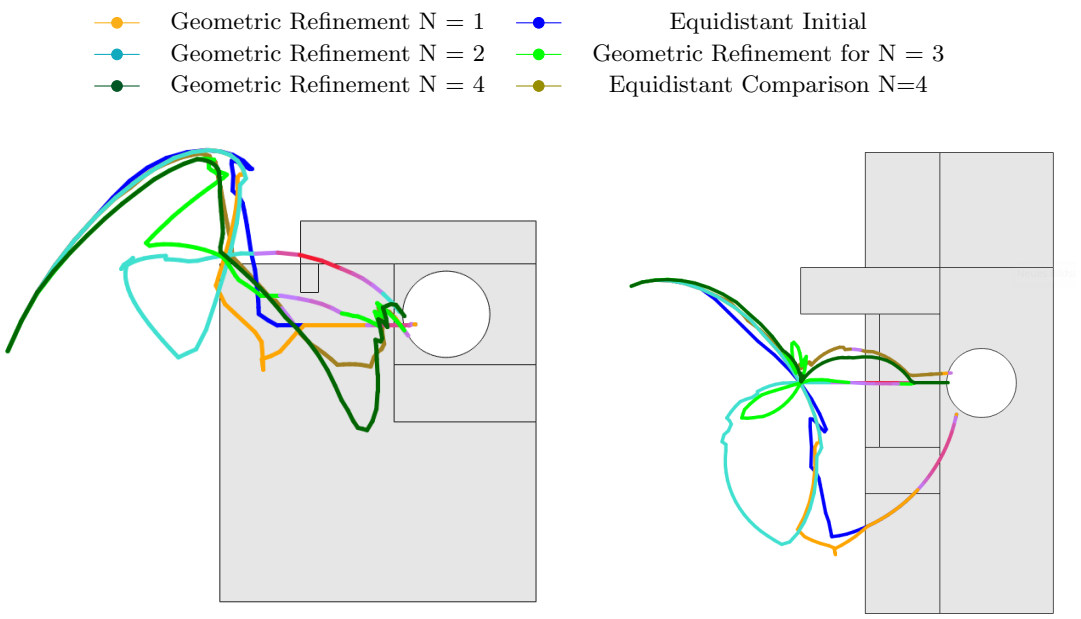
this section with three main takeaways: First of all, parallel execution of the DP Algorithm in general results in huge performance increases in practice. Furthermore, we obtain Algorithm 4 to be especially beneficial when executed on CPU, as well as on GPU for large numbers of state grid points. Finally, the most efficient solver for the YouBot Trajectory Planning problem considered here, turns out to be the execution of Algorithm 4 on GPU.

3.6.4 Adaptive State Space Refinement

Adaptive state space splitting was introduced as an additional strategy to improve performance of the DP Algorithm in Section 3.4. Here, we want to compare the refinement criterion based on geometric scene properties, to those based on the value function from literature [68, 111] as well as to equidistant state space sampling. We use the hypercube based structure introduced in [128] to represent a refinable grid. Please note that it does not allow refinement with respect to a single hyperplane, but only supports to refine the cell along n_x hyperplanes, such that we end up with 2^{n_x} new subcells. Consequently, we use the directional average when evaluating the *average corner-value difference* as well as the *Value non-linearity* criterion from [111]. Additionally, we compare against the refinement criterion proposed in [68], filtering especially for regions with large Bellman errors. Again, we use a fixed number of time points $K_t = 100$ as well as fixed control space discretisation $K_u := (11, 11, 11, 11)^\top$ and start from a coarse joint space discretisation $K_x := (21, 10, 12, 5)^\top$. In every iteration, we refine 7 % of the cells and execute in total 4 iterations. To compare against equidistant grids, we use optimal joint space discretisation as proposed in [74] and initialize 20 grids matching approximately the number of points resulting from the refinement process. In order to accelerate computations, we use optimal stopping, as well as Algorithm 3 executed on GPU.

First of all, we can obtain from Figure 3.6 that using our geometric refinement criterion, the algorithm is able to recover from heavy collision at the beginning to a feasible trajectory after four splitting iterations. It therefore proves our concept of reducing collision by refining cells with high proximity to intersection valid. Additionally, using an optimal joint space discretisation equidistant grid with approximately the same number of points results in a trajectory that still suffers from collision. Keeping this initial proof of concept for our criterion in mind, we move on to Figure 3.7 and compare against existing splitting criteria from literature. Again, we can obtain our algorithm to step by step lower its overall cost by decreasing collision avoidance violation, until we end up with a small cost after four refinement iterations. Also the *Value non-linearity* and *average corner-value difference* criterion from [111] as well as the Bellmann Error based criterion proposed in [68], were able to reduce the resulting trajectories cost with every iteration, but a lot slower than ours (see Figure 3.7a). A possible reason for that behavior is the time dependent value function evaluated by value iteration on a coarse time grid. Compared to the fix point iteration used in [111] and [68] this produces highly nonlinear value function parts not only at regions of interest, but also at the boundaries of reachability. Regarding the cost arising from equidistant grids, we can obtain, especially for coarse discretisations, large differences on small changes in the number of grid points. This effect mainly arises from the





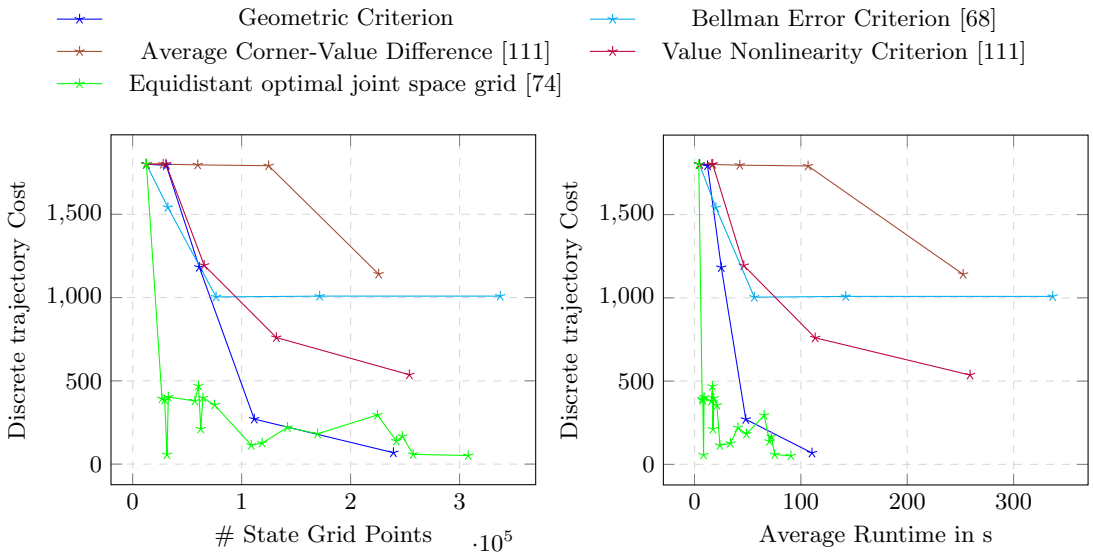
(a) XZ Plane View

(b) YZ Plane View

Figure 3.6: Trajectories arising from the DP Algorithm after N = 0,1,2,3,4 refinement iterations and an respective equidistant grid for N = 4 shown in the XZ and XY plane. Severity of collision is indicated for a low penalty (purple) to a high penalty (red). Static obstacles (gray) and the target region (white) are marked.

location change of the sampling points in a dimension on increasing numbers. Roughly speaking some discrete space point locations are more beneficial for collision avoidance and arising low trajectory costs than others. Consequently, increasing the number of points in a dimension, we switch between locations, which can result on coarse discretisations in oscillations as illustrated in Figure 3.7a. As the discretisation becomes finer, the effect vanishes leading on average to lower trajectory costs on increasing numbers of state space points. Nevertheless, we can obtain from Figure 3.7a that using our geometric splitting criterion, we were able to find a feasible trajectory using a smaller number of points than required from equidistant grids. Overall, comparing solely the number of grid points to the resulting trajectories cost, our algorithm is able to outperform existing splitting criteria, as well as equidistant grids resulting from optimal joint space discretisation [74]. As already, pointed out in Section 3.4, the main motivation for our splitting criterion was its independent evaluation from the value function, which should be beneficial regarding computation times. Hence, we compare in Figure 3.7b the resulting trajectories cost against their algorithm runtime. As expected, we can observe an acceleration in evaluation times comparing the geometric criterion to those based on the value function. On four splitting iterations, we can already obtain time savings of more then 50% in this example. Nevertheless, compared to pure equidistant grids, the evaluation of the geometric criterion and the splitting process do indeed require some time. Consequently, even though using slightly more state space points, the equidistant grid is able to provide a feasible solution in 75 s on average, while the evaluation of four splitting iterations using the geometric criterion takes on





(a) Cost over Number of Points

(b) Cost over Runtime

Figure 3.7: Comparison of trajectories resulting from adaptive state space refinement using 4 different splitting criteria as well as equidistant grids. Every refinement iteration is marked by a star.

average 110 s.

Remark: The Trajectory arising from the fourth iteration of our geometric refinement criterion is illustrated at the bottom right corner of this manuscript.

3.6.5 Conclusion

Finally wrapping up all the acceleration strategies seen so far, the most efficient way to determine a feasible solution to our youBot trajectory planning problem, turns out to be a combination of equidistant optimal joint space grids, the transformed strategy for optimal stopping as well as execution of Algorithm 4 on the GPU. The arising computation time of approximately 55 s is by far not real time capable, but acceptable e.g. for offline trajectory planning in industrial scenarios such as factories. After all, we have therefore successfully proven our accelerated dynamic programming algorithm to be applicable in medium dimensional problems within reasonable evaluation times.



3.7 Experiment: Autonomous Driving with DP based MPC

As part of the project MORE (Munich Mobility Research Campus) a StreetScooter B16 as illustrated in Figure 3.8 is transformed into a research platform for autonomous driving. While this includes among others the attachment of various sensors, we refer to [102] for a detailed description of the hardware setup. Within the project, the task of autonomous driving is realized using an hierarchical approach: At the top, an offline path planner evaluates an optimal path, based on start and goal position as well as road map data. Subsequently, for actually traveling the path, a fast path planning strategy is required that allows dynamic obstacle avoidance as well as control error compensation in real time. Finally, given a valid reference path, a low level path following controller is used to ensure the actual vehicle to stay on track. While we assume a global road map path as well as the low level controller to be given, we propose a model predictive control (MPC) strategy [60] executing the dynamic programming algorithm accelerated by optimal stopping and CPU parallelisation in every iteration, for the real-time trajectory planning part in the following.



Figure 3.8: Street Scooter used as a research platform for autonomous driving within project MORE

Given a spline representation of our global reference path, we plan our local trajectory in every iteration of MPC in a 3D state space $x \coloneqq (s, r, \phi)^\top$ using the global path arc length s, its orthogonal deviation r as well as the local vehicle rotation angle ϕ. Hence, for an MPC planning horizon of 100 m, we limit our state space in arc length dimension by $s \in [s_0, s_0 + 110\,\mathrm{m}]$ using initial point on spline s_0, in deviation dimension by $r \in [-4\,\mathrm{m}, 4\,\mathrm{m}]$ as well as by $\phi \in [-0.785\,\mathrm{rad}, 0.785\,\mathrm{rad}]$ in relative orientation. Due to the low level controller, we can neglect velocity and acceleration control in the DP vehicle model, but use a simplified representation, adopting only its restrictions on curvature as well as its limitation to forward motion. Consequently, our dynamics

function reads as

$$\begin{aligned} s'(t) &= \cos(\phi(t)) \\ r'(t) &= \sin(\phi(t)) \\ \phi'(t) &= u(t). \end{aligned}$$

using orientation change controls $u \in [-0.05\,\mathrm{rad/s}, 0.05\,\mathrm{rad/s}]$ limited with respect to the hardware capabilities of the vehicle. Furthermore, we aim to minimize deviations from the global trajectory, as well as changes in vehicle rotation and therefore use cost function

$$c(x(t), u(t)) = 100 \cdot (|u(t)| + |r(t)|).$$

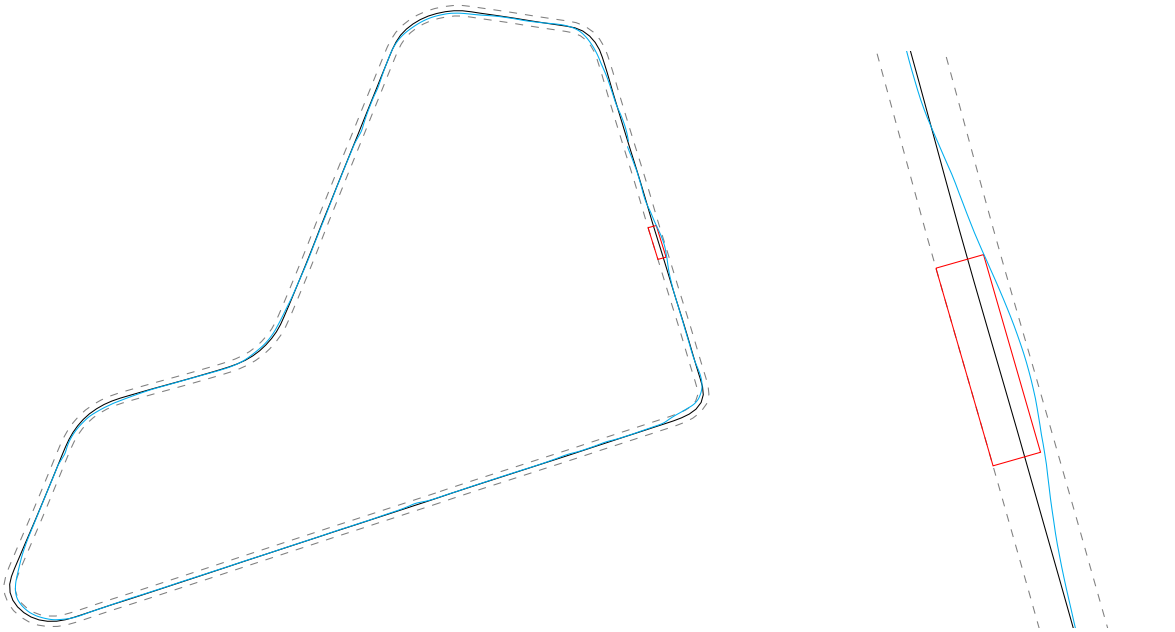
We model the goal as a rectangular region around state $x_g \coloneqq (s_g, r_g, \phi_g)^\top$ and include it into the Mayer term

$$\begin{aligned} m(x(t)) = \rho_1 \Bigg[&\max\left(0, \frac{\|s(t) - s_g\|}{|h_s|} - 1\right) + \max\left(0, \frac{\|r(t) - r_g\|}{|h_r|} - 1\right) \\ &+ \max\left(0, \frac{\|\phi(t) - \phi_g\|}{|h_\phi|} - 1\right)\Bigg] \end{aligned}$$

with respect to state space grid cell size $h_x \coloneqq (h_s, h_r, h_\phi)^\top$ as well as penalty scaling $\rho_1 = 50000$. As we rely on sensor based environment observations, we obtain a discrete point cloud representing the state space that stores for each point if there is an obstacle or not. We can therefore intuitively model collision avoidance as an additional restriction in discrete state space here. While this combination of dynamics function and state space restrictions violates Assumption (A9) in theory, it has been proven efficient in practice. Finally, we consider $K_x \coloneqq (31, 31, 31)^\top$ discrete state grid points, $K_u \coloneqq 11$ control grid points, $K_t \coloneqq 30$ sampling points on time scale $[t_0, t_0 + 150]$ and an MPC update rate of 150 ms. As within the previous example, we use a c++ implementation of the proposed algorithm, combining our parallel Discrete Dynamic Programming Algorithm for optimal stopping with the MPC implementation developed by A. Huber within [79].

As a proof of concept, we accomplished an autonomously driven test track using the real vehicle. Regarding the results illustrated in Figure 3.9 we can obtain the driven trajectory of the STS to remain, most of the time, close to the reference track. Additionally, even though cutting some curves, it always stays within the considered boundaries of the street. As our MPC strategy is originally intended to allow obstacle avoidance on the global reference track, we placed a virtual rectangular object blocking it. As illustrated in Figure 3.9b, the real STS was able to successfully avoid this obstacle during the test drive. While these results have revealed a lot of potential for further tuning to enhance the planners performance, e.g, within curvatures, we have successfully proven our accelerated discrete dynamic programming algorithm to be applicable with an update frequency of 4 Hz in MPC on a real vehicle.





(a) overall test drive

(b) focus on obstacle avoidance

Figure 3.9: Plotting the global reference trajectory (black,solid), the left and right road boundaries (gray, dashed) as well as the trajectory driven by our real vehicle (cyan,solid), for the vehicle's center point. The virtual obstacle is highlighted in red.

Chapter 4

Dynamic and Nonlinear Programming

As already pointed out in the introduction, general purpose nonlinear programming (NLP) solvers usually struggle with high dimensional problems subject to obstacle avoidance, due to the highly nonlinear and non-differentiable structure of the collision avoidance constraints. In contrast, we have seen in the previous Chapter that the method of Dynamic Programming is able to handle these constraints, but it is limited by to the curse of dimensionality. In the following, we describe iterative methods, originally proposed in [22, 127], combining them together such that we can use best of both.
Searching for a state function $x : [0, t_f] \to \mathcal{K}$ as well as controls $u : [0, t_f] \to \mathbb{U}$ subject to state and control space bounds $\mathcal{K} \subseteq \mathbb{R}^{n_x}$ and $\mathbb{U} \subseteq \mathbb{R}^{n_u}$ respectively, we formulate a general high dimensional trajectory planning problem as

$$
\begin{aligned}
& x'(t) = f(x(t), u(t)) && a.e \text{ in } [0, t_f] \\
& g(x(t)) \geq 0 && \forall\ t \in [0, t_f], \\
& x(t) \in \mathcal{K},\ u(t) \in \mathbb{U} && \forall\ t \in [0, t_f], \\
& x(0) = z, \quad b(x(t_f)) = 0, && \\
& t_f \in [0, T], &&
\end{aligned}
\tag{4.1}
$$

using dynamics function $f : \mathcal{K} \times \mathbb{U} \to \mathbb{R}^{n_x}$, free final time t_f limited by $T > 0$ as well as function $g : \mathcal{K} \to \mathbb{R}^{n_g}$ representing the collision avoidance constraint. Furthermore, we denote the initial by z and use function $b : \mathcal{K} \to \mathbb{R}^{n_b}$ to represent the terminal condition. As is, Problem (4.1) is due to its large dimensionality not solvable by dynamic programming in reasonable time. In order to nevertheless exploit DP capabilities to handle the collision avoidance, we initially introduce function $\Omega : \mathcal{K} \to \mathcal{W}$, mapping every high dimensional state x to a lower dimensional



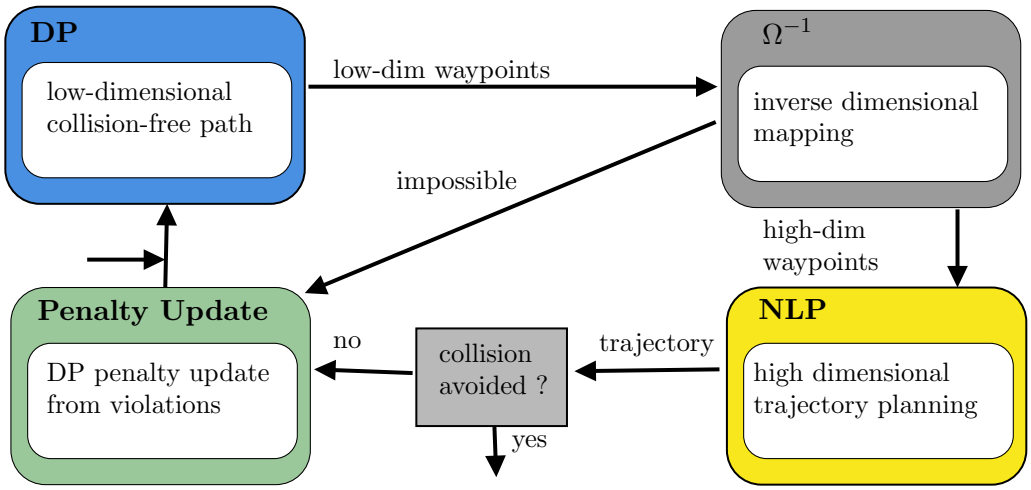


Figure 4.1: Schematic overview of our strategy combining DP and NLP iteratively for trajectory planning subject to collision avoidance

representation $w \in \mathcal{W} \subseteq \mathbb{R}^{n_w}$ with $n_w < n_x$. Based on that, we can bring the information necessary for collision avoidance to a lower dimensional space and formulate a lower dimensional trajectory planning problem. Using the method of Dynamic Programming, we can solve that problem and obtain a discrete low dimensional trajectory successfully avoiding obstacles. In a next step, we bring the trajectory back into the original dimension, accounting for the ill-posed structure of the inverse mapping by solving an optimization problem with reasonable objective function filtering. Subsequently, we solve the original Problem (4.1) using NLP, but instead of considering the difficult collision avoidance constraints, we just track the discrete high dimensional way-points resulting from DP. Naturally, this approach does not guarantee for a collision free trajectory in between the way-points. We therefore verify the obstacle avoidance constraint carefully on every trajectory point and terminate on success. Otherwise, we modify the penalty function responsible for collision avoidance in the DP problem, to avoid problematic regions in the next iteration. Figure 4.1 provides a schematic overview of the strategy.
In the following, we introduce the four core components of our method, namely the low dimensional level using DP, the inverse dimensional mapping, the trajectory generation based on NLP, and the DP penalty adaption, in detail in Sections 4.1,4.2,4.3 and 4.4 respectively. Furthermore, the extension of our method to non-autonomous systems, originally published in [127], is described in Section 4.6. It does not only allow our strategy to handle moving environments as well as time dependent goal constraints, but also increases the methods robustness with respect to infeasible NLP results. Finally, we compare the proposed combination of DP and NLP, as well as its extension to existing methods from literature on a high dimensional robotic trajectory planning task in Section 4.7, and demonstrate the extensions effectivity on a non-autonomous space application in Section 4.8.

Remark: Actually, Problem (4.1) matches our overall Problem (1.1) without its objective function. The reason for that arises from the structure of the iterative dynamic and nonlinear

programming algorithm which does not allow for optimization of a global objective function. Nevertheless, individual cost functions can be added to the DP and NLP problems respectively, filtering the resulting trajectories according to e.g. minimum time or minimum effort.

4.1 Low Dimensional Level

As already pointed out above, we want to exploit DP's capability to handle highly nonlinear and non-differentiable constraints, as used i.e. to model collision avoidance, without running into the curse of dimensionality. As a result, we initialize mapping function $\Omega : \mathcal{K} \to \mathcal{W}$ and transfer the initially high dimensional collision avoidance task in (4.1) to the low dimensional Problem

$$
\begin{aligned}
\text{Minimize } & \int_0^{\tau_f} \bar{c}(w(\tau), v(\tau))\mathrm{d}\tau & \\
\text{subject to } & w'(\tau) = \bar{f}(w(\tau), v(\tau)) & \quad a.e. \text{ in } [0, \tau_f] \\
& w(0) = \Omega(z) & \\
& \exists x \in \mathcal{K} : b(x) = 0, \Omega(x) = w(\tau) & \quad \forall \tau \in [0, \tau_f] \\
& \exists x \in \mathcal{K} : g(x) \geq 0, \Omega(x) = w(\tau) & \quad \forall \tau \in [0, \tau_f] \\
& v(\tau) \in V & \quad \forall \tau \in [0, \tau_f] \\
& \tau_f \in [0, \tau_{\max}] &
\end{aligned}
$$

using state function $w : [0, \tau_f] \to \mathcal{W}$, control function $v : [0, \tau_f] \to V$ as well as reasonable dynamics $\bar{f} : \mathcal{W} \times V \to \mathbb{R}^{n_w}$ and cost function $\bar{c} : \mathcal{W} \times V \to \mathbb{R}$. Additionally, we consider free final time τ_f limited by $\tau_{\max} > 0$, control bound V and state bound $\mathcal{W}$ such that $\{w \in \mathbb{R}^{n_w} : \exists x \in \mathcal{K} : \Omega(x) = w\} \subseteq \mathcal{W}$ holds. While the second constraint accounts for the initial value, the third and forth constraint represent the original collision avoidance and goal constraints in low dimensional space.

Remark: In practice, a reasonable choice of the mapping Ω is crucial for the approach and highly dependent on the application. A prominent example is the forward kinematics mapping from high dimensional joint to low dimensional end effector space in robotic manipulator tasks.

To be able to apply the method of dynamic programming as described in Chapter 3, an additional transformation using the penalty method [63] is required such that we end up with

$$
\begin{aligned}
\text{Minimize } & \rho_1 \mathcal{M}(w(\tau_f)) + \int_0^{\tau_f} \bar{c}(w(\tau), v(\tau)) + \rho_2 \mathcal{P}(w(\tau))\mathrm{d}\tau & \quad (4.2) \\
\text{subject to } & w'(\tau) = \bar{f}(w(\tau), v(\tau)) & \quad a.e. \text{ in } [0, \tau_f] \\
& v(\tau) \in V & \quad \forall \tau \in [0, \tau_f] \\
& w(0) = \Omega(z),\ \tau_f \in [0, \tau_{\max}] &
\end{aligned}
$$



using scaling parameters $\rho_1, \rho_2 > 0$ as well as penalty function

$$\mathcal{P}(w) \coloneqq \min_{x \in \mathcal{K}} \{\|\Omega(x) - w\| + \|\min(0, g(x))\|\} \tag{4.3}$$

accounting for the collision avoidance constraint and Mayer term

$$\mathcal{M}(w) \coloneqq \min_{x \in \mathcal{K}} \{\|b(x)\|, \Omega(x) = w\}$$

incorporating the terminal condition. Subsequently, we can follow the steps in Chapter 3 and discretize the arising value function equidistantly in time, control and state space using grids $\mathbb{G}_\tau, \mathbb{G}_v, \mathbb{G}_w$. Recapitulating the overall idea of our method (see Figure 4.1), including additional information from NLP into the DP problem is crucial. We therefore consider an iterative penalty adaptation scheme, updating the penalty function individually, for every discrete time point $\tau_i \in \mathbb{G}_\tau$ and state $w \in \mathbb{G}_w$ in every algorithm iteration. The point-wise penalty function evaluation is used as an initial value, resulting in the first iteration of the algorithm in $P_i(w) \coloneqq \mathcal{P}(w)$. In later iterations, we replace the original penalty function by its updated counterpart in the definition of the fully approximated value function. Finally, we can evaluate the discrete dynamic programming algorithm for optimal stopping as described in Chapter 3 and denote the arising discrete solution trajectory as a sequence of points by $\{\tau_i, w_i, v_i\}_{i=0}^{i=M}$.

Remark: Regarding convergence of the method of dynamic programming, we can consider $\mathcal{W}$ compact and follow Lipschitz continuity in w as well as boundedness of penalty function $\mathcal{P}$. Regarding the Mayer term, this is not possible in general, but defining the original goal region directly in low dimensional space i.e. as a sphere of radius r around goal point w_g with $b(x) \coloneqq \max(0, \|\Omega(x) - w_g\| - r)$ we can follow the Lipschitz property and boundedness. Letting additionally functions $\bar{f}$ and $\bar{c}$ as well as control and state spaces fulfill assumptions (A2) (A5),(A4),(A7), (A1) and (A9), we can apply the convergence properties derived in Chapter 3 towards the continuous value function.

Remark: In contrast to the variable time step size model proposed in the original publication [22] we use optimal stopping to represent the free final time. We motivate the exchange by an increased efficiency in dynamic programming evaluation due to the reduction of control space dimensionality.



4.2 Inverse Dimensional Mapping

Given a low dimensional sequence of points arising from DP, we want to map them back into the original dimension for trajectory generation. Unfortunately, the inverse mapping Ω^{-1} can be ill-posed, resulting for any low dimensional state in either one, multiple or no high dimensional representations. Here, considering a low dimensional state w_i as well as the high dimensional representation of its predecessor in the DP trajectory $x_{i-1} = \Omega^{-1}(w_{i-1})$, we initialize optimization problem

$$\begin{aligned} \text{Minimize} \quad & \varrho_1 \|x - x_{i-1}\|^2 + \varrho_2 \langle 1, \sigma \rangle \\ \text{subject to} \quad & \Omega(x) = w_i, \quad g(x) \geq \sigma \geq 0, \quad x \in X \end{aligned} \tag{4.4}$$

using weighting parameters $\varrho_1, \varrho_2 > 0$, to filter among possible high dimensional representations. Regarding the objective function, we select from feasible solutions with respect to inverse mapping and collision avoidance, according to the following criteria: On the one hand, we prefer high dimensional representations with large margins σ to the collision avoidance constraint. The reason for that is simply to reduce the chance for obstacle avoidance violations when connecting the way-points during trajectory generation. Additionally, we encourage the high dimensional representation x_i to stay close to its predecessor. By lowering changes in the original systems state, we again aim to decrease the chance of collisions arising within the NLP trajectory, but additionally try to generate way-points to be easily tracked by the system dynamics. In order to obtain a full sequence of high dimensional points, we evaluate Problem (4.4) for every point in the DP sequence $\{\tau_i, w_i, v_i\}_{i=0}^{i=M}$ recursively, starting from $x_1 = \Omega(w_1) = z$. As can be obtained from Figure 4.1 we directly move on to the penalty adaptation step, if we encounter an infeasible problem. Otherwise, we hand the resulting high dimensional trajectory $\{\tau_i, x_i\}_{i=0}^{i=M}$ over to the trajectory generation block for way-point tracking.

Remark: While the filtering criteria, try to encourage useful high dimensional representations in general, their weighting terms ϱ_1, ϱ_2 require careful tuning with respect to specific applications in practice.

Remark: To illustrate the abstract concept of the inverse dimensional mapping on a practical example, we can choose a robotic arm consisting of multiple joints, where we defined the dimensional mapping Ω to track its forward dynamics. In this case, we obtain three dimensional way-points representing positions of the robotic endeffector from DP. Now, within the inverse dimensional mapping, we are searching for corresponding joint configurations. Solving Problem (4.4), we filter among the possibly multiple configurations, for those with high proximity between robotic links and environment objects, as well as for small changes with respect to the previous configuration point.



4.3 Trajectory Generation

Given collision free way-points from DP and inverse dimensional mapping, our goal in this section is to find states x connecting them with respect to a maximum final time T. We therefore initialize optimal control problem

$$\begin{aligned} \text{Minimize} \quad & \int_0^{t_f} c(x(t), u(t))dt & & \\ \text{subject to} \quad & x'(t) = f(x(t), u(t)) & & a.e. \text{ in } [0, t_f] \\ & x(t) \in \mathcal{K},\ u(t) \in \mathbb{U} & & \forall\ t \in [0, t_f] \\ & p_i(x(t_f \tau_i / \tau_M)) = 0 & & \forall\ i \in \{0, \dots, M\} \\ & x(0) = z & & \\ & t_f \in [0, T], & & \end{aligned} \tag{4.5}$$

where state and control bounds, initial state z, as well as the dynamics function f, are directly taken from Problem (4.1). The function $c : \mathcal{K} \times \mathbb{U} \to \mathbb{R}$ allows for an application specific running cost term, defined with respect to state and control information. It can be used to filter resulting trajectories i.e. for minimum time or minimum control. While the collision avoidance constraint from the original problem is missing, we consider tracking constraints $p_i(x)$ instead. Their explicit definition can vary with respect to the application, but we want to point out the prominent example $p_i(x) = \|x - x_i\|$, forcing x to exactly traverse the way-point.
Searching for a solution of non-linear Problem (4.5) there exist a huge number of algorithms and strategies in literature [62], ranging from direct approaches using direct (multiple) shooting or collocation to those based on necessary optimality conditions arising from Pontryagin's maximum principle. Here, we have chosen to follow the *'first discretize, then optimize'* approach and introduce time grid $\mathbb{G}_t$ to sample the interval $[0, T]$ with equidistant intervals of size h_t. To allow for accurate way-point tracking, we additionally include the time points $\{T\tau_i/\tau_M\}_{i=0}^{i=M}$ into the grid such that we end up with N intervals where $N >> M$ holds. We denote the arising discrete states by $\{x_j\}_{j=0}^{N}$ as well as controls by $\{u_j\}_{j=0}^{N-1}$ and approximate the dynamics function with respect to some discretisation scheme, i.e. Euler or Runge-Kutta methods. As a result, we can formulate a finite dimensional optimal control problem suitable to off the shelf solvers, i.e. based on the SQP or interior point strategies. The arising discrete solution trajectory $\{t_j, x_j\}_{j=0}^{N}$ is not guaranteed to avoid collision and therefore subject to obstacle avoidance validation and penalty adaptation subsequently.

Remark: In general, the optimal control problems arising from trajectory generation can be non-linear. A notable exception is the case of linear dynamics and tracking restrictions combined with convex state and control bounds as well as a quadratic cost function. Here, additional assumption of a fixed final time allows to derive a convex quadratic program (QP) by full discretisation, well suited for plenty of numerical solvers i.e. [42, 75].



Remark: Exploiting the free final time included into the DP using $\tau_{max} = T$ and consequently $t_f = \tau_f$, we could possibly restrict above trajectory generation task to a fixed end time. Nevertheless, despite the increased effort arising from the solving process, we use a variable end time model to provide an additional degree of freedom and decrease the number of infeasible problems.

4.4 Penalty adaptation

In order to close the loop, we include information on infeasible states into the next iterations DP problem using an adaptive penalty approach. The intention here arises from the highly nonlinear behavior of the penalty function within a cell of the DP grid. Basically, instead of representing only the penalty function $\mathcal{P}$ evaluated at a specific grid point $w \in \mathbb{G}_w$, we want the penalty value associated with every point w to provide also information about the functions behavior in its cell neighborhoods. Consequently, by including information from infeasible states in every iteration of the algorithm, we iteratively improve our approximation of the full penalty landscape.

Regarding the algorithms structure, two different types of "infeasible inputs" can be distinguished: On the one hand, the sequence of points arising from DP can generate an infeasible optimization problem for one or multiple of its members during the inverse mapping. In this case, the input of the penalty adaptation Θ matches the sequence of candidate points arising from DP $\Theta \coloneqq \{\tau_i, w_i\}_{i=0}^{i=M}$. On the other hand, the trajectory arising from NLP can suffer from collision and is therefore given to the penalty adaptation. Here, the goal is an adaptation of the penalty at the corresponding low dimensional states, such that the input set reads as $\Theta \coloneqq \{t_i \cdot \frac{\tau_f}{t_f}, \Omega(x_j)\}_{j=0}^{N}$. For the design of the penalty adaptation scheme, we keep in mind the usually ill-posed structure of the inverse mapping, resulting in multiple high dimensional states mapping to the same low dimensional point. We consider it therefore beneficial for trajectory planning, to avoid a certain low level region only at the time interval where infeasibility has been observed, thus allowing to pass the region at another time interval with possibly different high dimensional representations. Additionally associating an infeasibility with its respective DP time-step allows to preserve the Markov property essential for the principle of dynamic programming. As a result, we consider individual updates for any DP time and state grid point $\{\tau_i, w\} \in \mathbb{G}_\tau \times \mathbb{G}_w$ and collect neighboring points with respect to time into sets $J_{\tau_i} \coloneqq \{\{s, w\} \in \Theta, s \in [\tau_{i-1}, \tau_{i+1}]\}$. Using function $\widehat{P} : \mathbb{G}_w \to \mathbb{R}_+$ we evaluate the penalty update as a weighted average of surrounding points by

$$\widehat{P}(\tau_i, w; \Theta) \coloneqq \frac{\sum_{\{s,\tilde{w}\} \in \mathcal{J}_{\tau_i}} \alpha(\tilde{w}, w) \mathcal{P}(\tilde{w})}{\sum_{\{s,\tilde{w}\} \in \mathcal{J}_{\tau_i}} \alpha(\tilde{w}, w)} \tag{4.6}$$

using function $\alpha : \mathcal{W} \times \mathbb{G}_w \to \mathbb{R}$ as a weighting term to scale for spatial relevance of point $\tilde{w}$ with respect to grid point w. An intuitive definition would i.e result in large values for high proximity of the respective points. Furthermore, we restrict the penalty update function $\widehat{P}$ to zero on zero



denominator observed. Finally, we can update the overall penalty values used within DP as

$$\tilde{P}_i(w) = P_i(w) + \widehat{P}(\tau_i, w; \Theta)$$

in every iteration of our algorithm.

Remark: Above penalty adaptation scheme basically results in a penalization term associated with every DP time and state grid point. To never the less allow evaluation of penalty values in non-grid points, as required within the DP forward pass, an interpolation scheme can be used. Within the examples presented in Sections 4.7 and 4.8 we choose linear interpolation.

Remark: Regarding the definition of penalty function $\mathcal{P}$ in Equation (4.3) the minimum constraint violation associated with a low dimensional point is returned. Consequently, given a state x_i from NLP suffering from collision, the corresponding low level state w_i can nevertheless be feasible, resulting in zero penalty. As this prevents on the one hand an excessive restriction of the DP state space, it can also result in a sequence of idle penalty updates. We therefore extend our practical implementation to check for idle updates and adapt the penalty according to the measured collision avoidance $g(x_j)$ instead of $\mathcal{P}(\Omega(x_j))$, if required.



4.5 Iterative Dynamic and Nonlinear Programming (IDNP) Algorithm

Having discussed all four core components of our strategy, we can finally summarize them, following the idea from Figure 4.1, into the numerical Algorithm 5. Recapitulating the overall idea of splitting an initially complex problem into less difficult parts, we propose a combination of the methods of DP and NLP. While, without DP, the NLP solver struggles with the highly non-convex collision avoidance constraints, a pure DP solution is considered impractical due to the curse of dimensionality. For their cooperation described in Algorithm 5, the individual solvers are connected building upon the ideas of a dimensional mapping in Step 6 as well as an adaptive penalty scheme in Step 10. In practice, careful application specific tuning of both components is crucial for the algorithms performance. On termination, the trajectory returned by Algorithm 5 does successfully avoid collision due to Step 8 while also satisfying the constraints included into NLP problem from Step 7 with respect to the time discretisation. It is therefore a valid solution to our overall problem (4.1).

Algorithm 5: Iterative scheme combining DP and NLP

1 Construct DP time, state and control grids $\mathbb{G}_\tau$, $\mathbb{G}_w$, $\mathbb{G}_v$
2 Initialize penalty using $P_i^0(w) := \mathcal{P}(w)$ for all $\{\tau_i, w\} \in \mathbb{G}_\tau \times \mathbb{G}_w$
3 **for** $k \leftarrow 1, 2, \ldots$ **do**
Low dimensional Level: DP
4 $\{\tau_i, w_i, v_i\}_{i=0}^M \leftarrow$ (4.2) with penalty grids P^k
Inverse dimensional Mapping
5 **try** $\{x_i\}_{i=0}^M \leftarrow \{w_i\}_{i=0}^M$ by (4.4)
6 **catch** $\Theta \leftarrow \{\tau_i, w_i\}_{i=0}^M$ and **go to** Step 10
Trajectory generation: NLP
7 $\{t_j, x_j, u_j\}_{i=0}^N \leftarrow$ (4.5) with $\{p_i\}_{i=0}^M$ tracking $\{\tau_i, x_i\}_{i=0}^M$.
Validate Collision Avoidance
8 **if** $\forall j \in \{1, \ldots, N\}\ g(x_j^k) \geq 0$ **then return**
Penalty Adaptation
9 $\Theta \leftarrow \{t_j \cdot \frac{\tau_f}{t_f}, \Omega(x_j)\}_{j=0}^N$
10 $\tilde{P}^{k+1} \leftarrow P^k + \widehat{P}(\cdot, \cdot; \Theta)$ by (4.6)



4.6 Extension: Non-autonomous Systems

Due to the free final time considered within NLP trajectory generation, the way-points arising from the lower dimensional level are affected by a shift in time during tracking. Thus, our current approach is limited to autonomous problems. As time dependent collision avoidance and goal constraints nevertheless can arise in practical applications, i.e. when chasing a tumbling piece of space debris, this section proposes an extension of our algorithm to handle them. Let us initially equip the original overall problem (4.1) with time dependent goal and collision avoidance constraints such that we end up with

$$
\begin{aligned}
& x'(t) = f(x(t), u(t)) && a.e \text{ in } [0, t_f] \\
& g(t, x(t)) \geq 0 && \forall\ t \in [0, t_f], \\
& x(t) \in \mathcal{K},\ u(t) \in \mathbb{U} && \forall\ t \in [0, t_f], \\
& x(0) = z, \quad b(t_f, x(t_f)) = 0, && \\
& t_f \in [0, T]. &&
\end{aligned}
\tag{4.7}
$$

In order to solve that problem, we propose two essential changes, compared to our original iterative dynamic and nonlinear programming strategy: On the one hand, in order to avoid a remapping of the way-points in time, the NLP problem is limited to a fixed final time. Consequently, it becomes more likely to encounter an infeasible NLP problem. Addressing this issue, the second core component of this extension provides an adaptation of the penalty scheme introduced in Section 4.4 to not only recover from collision avoidance violations, but also handle infeasibilities arising from the NLP solver. As a result, the extension of our algorithm proposed in the following does not only allow to handle moving environments and goals but is also considered more robust with respect to infeasible NLP problems. Figure 4.2 provides an overview, with the changes compared to our original algorithm highlighted in red. In the following, we walk through the extended strategy and describe the components affected by the changes in more detail.

Starting from the low dimensional level, the penalized DP problem (4.2) modified with respect to time dependent goal and collision avoidance constraints reads as

$$
\begin{aligned}
& \text{Minimize } \rho_1 \mathcal{M}(\tau_f, w(\tau_f)) + \int_0^{\tau_f} \bar{c}(w(\tau), v(\tau)) + \rho_2 \mathcal{P}(\tau, w(\tau)) \mathrm{d}\tau && \\
& \text{subject to } w'(\tau) = \bar{f}(w(\tau), v(\tau)) && a.e. \text{ in } [0, \tau_f] \\
& \qquad v(\tau) \in V && \forall \tau \in [0, \tau_f] \\
& \qquad w(0) = \Omega(x_0),\ \tau_f \in [0, T] &&
\end{aligned}
\tag{4.8}
$$



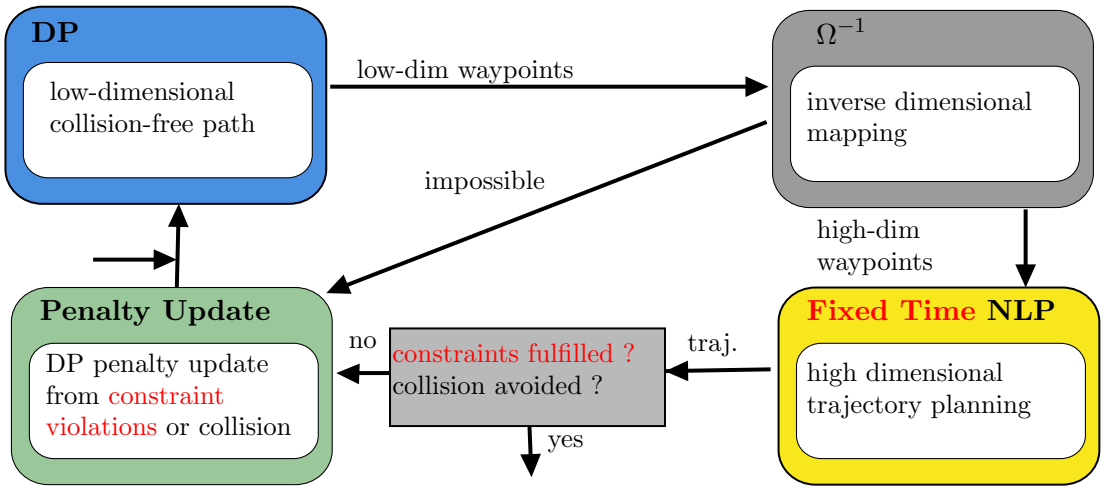


Figure 4.2: Iterative DP and NLP Strategy extended to non-autonomous trajectory planning problems subject to collision avoidance

using time dependent penalty function

$$\mathcal{P}(\tau, w) := \min_{x \in \mathcal{K}} \{\|\Omega(x) - w\| + \|\min(0, g(\tau, x))\|\} \tag{4.9}$$

as well as Mayer term

$$\mathcal{M}(\tau, w) := \min_{x \in \mathcal{K}} \{\|b(\tau, x)\|, \Omega(x) = w\}.$$

Keeping in mind that we aim to track the waypoints using a fixed final time in NLP, we choose, instead of an artificial time horizon τ_{max}, the original horizon T here. Subsequently, we follow the original approach on Section 4.1 by discretisation in time, space and control as well as application of the dynamic programming algorithm for optimal stopping problems to generate a low dimensional trajectory $\{\tau_i, w_i\}_{i=0}^{i=M}$.

Remark: Due to the non-autonomous problem, the convergence properties derived in Chapter 3 can no longer be applied. For further information on DP convergence in time dependent scenarios, we refer the interested reader to [9, 51].

Given the low dimensional way-points, we want to apply an inverse dimensional mapping. Adding the current time point τ_i as an additional constant into the collision avoidance constraint of optimization problem (4.4) we can directly adapt the recursive solution process described in Section 4.2. On successful generation of a high dimensional way-point sequence, we move on to the trajectory generation.
As already pointed out before, time dependent collision avoidance and goal constraints do no longer allow a rescaling of the way-points with respect to time on the high dimensional level.

As a result, we characterize the fixed final time NLP problem as

$$\begin{aligned} \text{Minimize} \quad & \int_0^{\tau_f} c(x(t), u(t))dt & & \\ \text{subject to} \quad & x'(t) = f(x(t), u(t)) & & a.e. \text{ in } [0, \tau_f] \\ & x(t) \in \mathcal{K},\ u(t) \in \mathbb{U} & & \forall\ t \in [0, \tau_f] \\ & p_i(x(\tau_i)) = 0 & & \forall\ i \in \{0, \dots, M\} \\ & x(0) = z & & \end{aligned} \tag{4.10}$$

where τ_f represents the final time arising from the DP level. As described in detail in Section 4.3 we introduce a discretisation with respect to time grid $\mathbb{G}_t$ and exploit off the shelf NLP solvers to strive for a high dimensional solution trajectory $\{t_j, x_j, u_j\}_{j=0}^{N}$.
Please note that there is in general no guarantee for the solver to find a feasible trajectory. While the original approach failed in these cases, the limitation to fixed final time makes their appearance more likely, thus requiring as strategy to handle them. In the following, we modify the penalty adaptation process, to account not only for collision avoidance violations, but also for these infeasible NLP problem formulations. We assume the NLP solver to return on infeasible problems a candidate trajectory violating one or multiple constraints of Problem (4.10). Following the arguments in Section 4.4 we let Θ represent the low dimensional sequence given as an adaptive penalty input, as well as $J_{\tau_i} \coloneqq \{\{s, w\} \in \Theta, s \in [\tau_{i-1}, \tau_{i+1}]\}$ collect points based on DP time intervals. Subsequently, the penalty adaptation proposed in Section 4.4 evaluates the collision avoidance constraint in every trajectory point and adds the weighted average of the violations to the DP grid points $\{\tau_i, w\}$ using update function

$$\widehat{P}(\tau_i, w; \Theta) \coloneqq \frac{\sum_{\{s,\tilde{w}\} \in \mathcal{J}_{\tau_i}} \alpha(\tilde{w}, w) \mathcal{P}(s, \tilde{w})}{\sum_{\{s,\tilde{w}\} \in \mathcal{J}_{\tau_i}} \alpha(\tilde{w}, w)}. \tag{4.11}$$

In case of infeasible NLP problems, we can basically follow the same strategy. Given an infeasible candidate solution from the NLP solver, we introduce function $\mathcal{F} : \mathbb{G}_t \times \mathcal{K} \to \mathbb{R}_+$ to return the overall constraint violation for a high dimensional trajectory point. Subsequently, we evaluate the weighted average at every DP time and state grid point $\{\tau_i, w\}$ using function $\widehat{\mathcal{F}}$ with

$$\widehat{\mathcal{F}}(\tau_i, w; \Theta) \coloneqq \frac{\sum_{\{s,\Omega(x)\} \in \mathcal{J}_{\tau_i}} \alpha(\Omega(x), w) \mathcal{F}(s, x)}{\sum_{\{s,\Omega(x)\} \in \mathcal{J}_{\tau_i}} \alpha(\Omega(x), w)}. \tag{4.12}$$

Summarizing both violation variants into the update function, we characterize the penalty update scheme of our extended algorithm as

$$\tilde{P}(\tau_i, w) \coloneqq P(\tau_i, w) + \begin{cases} \widehat{F}(\tau_i, w; \Theta) & (4.10) \text{ infeasible} \\ \widehat{P}(\tau_i, w; \Theta) & \text{else} \end{cases}$$

starting on the very first iteration from penalty function $\mathcal{P}$ evaluated at DP grid points.



Remark: While compensation for infeasible NLP solutions, reads quite simple in theory, finding a reasonable explicit definition of function $\mathcal{F}$ is highly dependent on the application and non-trivial in practice. Among other things, it has to account for violations obtained from various constraints, requiring a careful orchestration of the associated weighting terms.

Finally, we can summarize our iterative dynamic and nonlinear programming procedure extended for moving environments into Algorithm 6.

Algorithm 6: Iterative scheme combining DP and NLP for moving environments

1 Construct DP time, state and control grids $\mathbb{G}_\tau$, $\mathbb{G}_w$, $\mathbb{G}_v$
2 Initialize penalty using $P^0(\tau_i, w) \coloneqq \mathcal{P}(\tau_i, w)$ for all $\{\tau_i, w\} \in \mathbb{G}_\tau \times \mathbb{G}_w$
3 **for** $k \leftarrow 1, 2, \ldots$ **do**
Low dimensional Level: DP
4 $\{\tau_i, w_i, v_i\}_{i=0}^{M} \leftarrow$ (4.8) with penalty grids P^k
Inverse dimensional Mapping
5 **try** $\{\tau_i, x_i\}_{i=0}^{M} \leftarrow \{\tau_i, w_i\}_{i=0}^{M}$ by (4.4)
6 **catch** $\Theta \leftarrow \{\tau_i, w_i\}_{i=0}^{M}$ and **go to** Step 10
Trajectory generation: NLP
7 $\{t_j, x_j, u_j\}_{i=0}^{N} \leftarrow$ (4.10) with $\{p_i\}_{i=0}^{M}$ tracking $\{\tau_i, x_i\}_{i=0}^{M}$.
Validate Collision Avoidance
8 **if** $\forall i \in \{1, \ldots, N\}\ g(x_i^k) \geq 0$ **then return**
Penalty Adaptation
9 $\Theta \leftarrow \{t_j, \Omega(x_j)\}_{j=0}^{N}$
10 $\tilde{P}^{k+1} \leftarrow P^k + \begin{cases} \widehat{F}(\tau_i, w; \Theta) \text{from} (4.12) & (4.10) \text{ infeasible} \\ \widehat{P}(\tau_i, w; \Theta) \text{from} (4.11) & \text{else} \end{cases}$



4.7 Experiment: Kinova Mico Trajectory Planning

To verify the methods combining dynamic and nonlinear programming for trajectory planning, we plan a trajectory for a pick and place task, executed by the Kinova MICO™ robotic arm with six degrees of freedom within an industrial environment. In addition to a comparison of the two different variants of our strategy, we verify their performance against the CHOMP [126] and RTTConnect [92] algorithms from the field of robotic motion planning as well as the general purpose NLP solver Ipopt [147]. We describe the scene as well as our model of the robotic arm in detail in Section 4.7.1. Subsequently, Section 4.7.2 provides details on the algorithms practical implementation, including a specification of all the parameters and functions required for the algorithm variants combining Dynamic and nonlinear Programming. Finally, the results analyzed in Section 4.7.3 characterize our algorithms to outperform the other solvers regarding the number of solvable scenarios.

4.7.1 Experiment Setup

In this experiment, we model an industrial pick and place task for the robotic arm manipulator Kinova MICO™, illustrated in Figure 4.3. Starting from an initial joint configuration, we aim to bring the robotic arm's tool center point (TCP) to a three dimensional goal region, while preventing collisions with robotic parts as well as environment objects. As we are not interested in the grasping process itself, but only focus on bringing the arm to the goal, it is sufficient to model the six joints of the robotic arm, ignoring the fingers of the hand. The resulting robotic model illustrated in Figure 4.3 is therefore missing the fingers as kinematic parts, but allows for a closed static hand motion due to an increased collision envelope around its last link. From the dynamic perspective, we consider states $x \coloneqq (q, \nu)^\top$ composed of joint configurations q and joint velocities ν, as well as controls u representing accelerations in joint space, and assume a linear function

$$f(x(t), u(t)) \coloneqq (\nu(t), u(t))^\top.$$

To allow execution of the resulting trajectories on the real hardware, we adapt the limitations from the robots user guide to form the state and control spaces $\mathcal{K}$ and $\mathbb{U}$ respectively. They are summarized in Table 4.1. Setting up a planning problem of structure (4.1), we choose the goal

Table 4.1: State and Control boundaries Kinova MICO™.

$q_{\min}$	$q_{\max}$	$\nu_{\min}$	$\nu_{\max}$	$u_{\min}$	$u_{\max}$
[rad]	[rad]	[rad/s]	[rad/s]	[rad/s^2]	[rad/s^2]
-∞	∞	-$\pi/9$	$\pi/9$	-1	1
$5\pi/18$	$31\pi/18$	-$\pi/9$	$\pi/9$	-1	1
$7\pi/36$	$65\pi/36$	-$\pi/9$	$\pi/9$	-1	1
-∞	∞	-$\pi/9$	$\pi/9$	-1	1
-∞	∞	-$\pi/9$	$\pi/9$	-1	1
-∞	∞	-$\pi/9$	$\pi/9$	-1	1



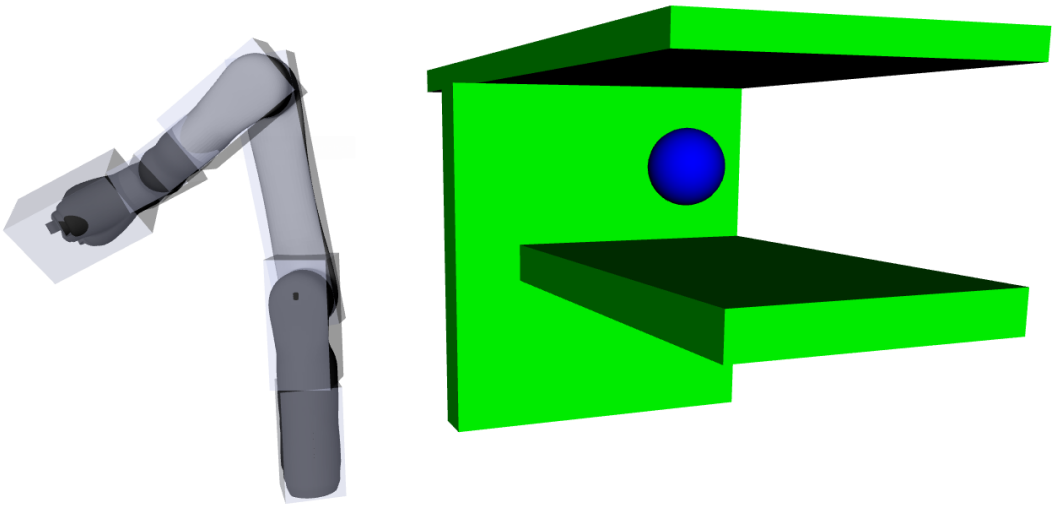


Figure 4.3: The Kinova MICO™ robotic arm in configuration $q \coloneqq (0.37, 3.29, 1.05, 0.14, -0.23, 0)^\top$ equipped with rectangular collision geometry (light gray), the obstacles in the scene approximated by convex polytopes (green), as well as the desired goal region for the robots TCP(blue).

function

$$b_0(x(t_f)) \coloneqq \max(0, \|\kappa(q(t_f)) - p_g\| - r)$$

to describe a sphere of radius $r = 0.06\,\mathrm{m}$ in the robots TCP space around point $p_g \coloneqq (0.59, 0, 0.45)^\top \mathrm{m}$, with function κ representing the robots forward kinematics. Furthermore, we incorporate zero goal velocity constraints using

$$b_i(x(t_f)) = \nu(t_f) \quad i = 1, \ldots, 6.$$

To model collision avoidance, we initialize a total signed distance function based on the signed distance function d from Chapter 2, resulting in

$$g(x(t)) \coloneqq \min_{a \in A(q(t)), b \in B \cup A(q(t))} d(a, b) - \epsilon$$

with safety distance $\epsilon = 0.01\,\mathrm{m}$, Polygon set $A(q)$ representing the objects from the robotic collision geometry in configuration q as well as Polygon set B collecting the obstacles from the environment. Due to the enhanced complexity of the robotic structure, we consider, in contrast to the example in Section 3.6, also self collisions here. Finally, we limit the maximum trajectory end time to $T = 300\,\mathrm{s}$.



4.7.2 Algorithm Setup

Given the overall problem structure, we want to initialize our algorithms combining DP and NLP methods. Initially, we choose the forward kinematics of the robot as dimensional mapping $\Omega(x(t)) \coloneqq \kappa(q(t))$, allowing to map the originally high dimensional states of dimension 12 to the three dimensional TCP space. Regarding the low dimensional problem, we choose

$$\bar{f}(w(\tau), v(\tau)) \coloneqq v(\tau),$$

$\tau_{max} = 20\,\mathrm{s}$, penalty scaling parameters $\rho_1 = \rho_2 = 1e3$, as well as state bounds

$$\mathcal{W} \coloneqq \{w \in \mathbb{R}^3 | (-0.7, -0.7, 0.0)^\top \mathrm{m} \leq w \leq (0.7, 0.7, 1.0)^\top \mathrm{m}\}$$

and control limitations

$$V \coloneqq \{v \in \mathbb{R}^3 | -0.11\,\mathrm{m/s} \leq v_i \leq 0.11\,\mathrm{m/s}, \quad i = 1, 2, 3\}.$$

Furthermore, we filter feasible results according to objective function

$$\bar{c}(w(\tau), v(\tau)) \coloneqq \|v(\tau)\|_2^2$$

within the original algorithm and

$$\bar{c}(w(\tau), v(\tau)) \coloneqq \|v(\tau)\|_1$$

when using the variant for non-autonomous systems, with $\|\cdot\|_1$ representing L1 norm. To apply the DP algorithm, we discretise the arising problem in time with $M \coloneqq 20$, in state space with $\{20, 20, 20\}$ as well as in control space with $\{8, 8, 8\}$ grid points. Subsequently, the inverse dimensional mapping uses scaling parameters $\varrho_1 = 1$ and $\varrho_2 = 10.0$. Regarding the NLP problem we choose to filter resulting trajectories with respect to time and control using

$$c(x, u, t_f) \coloneqq \int_0^{t_f} 1 + 0.5\|u(t)\|^2 dt$$

in the original method, as well as with

$$c(x, u, t_f) \coloneqq \int_0^{t_f} 0.5\|u(t)\|^2 dt$$

when using the variant for non-autonomous problems. Furthermore, we initialize tracking functions

$$p_i(x(t)) = x(t) - x_i$$

for every waypoint x_i with $i = 0, \ldots, M$. Subsequently, we equidistantly sample the time space with 500 points and apply the trapezoidal rule as well as finite differences to discretise the



dynamics and cost function respectively. On penalty adaptation, we initialize $\alpha(\cdot, w)$ for grid point $\{\tau_i, w\} \in \mathbb{G}_\tau \times \mathbb{G}_w$ as a triangular function with stencil $[\tau_{i-1}, \tau_{i+1}]$. In the non-autonomous problem variant, we additionally choose constraint penalization function $\mathcal{F}$ in (4.12) as

$$\begin{aligned}\mathcal{F}(t_j, x_j) = 100[&\mathcal{F}_{\text{dyn}}(t_j, x_j) + \mathcal{F}_{\text{dyn}}(t_{j-1}, x_{j-1}) \\ &+ \mathcal{F}_{\text{wp}}(t_j, x_j) + \mathcal{F}_{\text{b}}(t_j, x_j)],\end{aligned}$$

composed of violations on the waypoint tracking constraints

$$\mathcal{F}_{\text{wp}}(t_j, x_j) = \begin{cases} \|p_i(x_j)\|_1 & \text{if } t_j = \tau_i \\ 0 & \text{else} \end{cases},$$

the discretised dynamics,

$$\mathcal{F}_{\text{dyn}}(t_j, x_j) = \begin{cases} \|x_j + h_t \frac{(f(x_j, u_j)) + f(x_{j+1}, u_j)}{2} - x_{j+1}\|_1 & \text{if } t_j \neq \tau_m \\ 0 & \text{else} \end{cases}$$

and the boundary constraint

$$\mathcal{F}_{\text{b}}(t_j, x_j) = \begin{cases} \|x(t_j) - z\|_1 & \text{if } t_j = 0 \\ 0 & \text{else} \end{cases}.$$

Finally, we allow the variants of our algorithm to execute at maximum $k = 20$ iterations.
All algorithm implementations used in the following are written in C++ and executed on an AMD Ryzen Threadripper 3390X system with 256GB RAM. We apply the algorithm combining SAT with GJK from Chapter 2 to evaluate the signed distance function d, as well as the Discrete Dynamic Programming Algorithm for optimal stopping from Chapter 3 using CPU parallelisation as suggested in Algorithm 4 to solve the low dimensional problem. Furthermore, the general purpose solver Ipopt version 3.14.12 [147] is accessed through the Ifopt interface version 2.1.3 [152] and combined with HSL linear solvers [78] to solve the high dimensional problem (4.5) as well as the inverse dimensional optimization problems (4.4) and the initial penalty function (4.3).
For comparison, we also try to solve the overall problem (4.1) as is using Ipopt, using a finite difference approach to approximate the collision avoidance constraints gradient. Furthermore, to verify against state of the art planning algorithms from the robotics community, we initialize our scene in the MoveIt Motion planning interface shipped with ROS2 version Humble [144]. We use the RRTConnect[92] implementation from the Open Motion Planning Library (OMPL) [141] integrated into MoveIt, to determine a kinematically feasible path in joint space and make use of MoveIt's default time parametrization algorithm to generate a trajectory. Furthermore, we apply the CHOMP algorithm's [126] implementation integrated into the MoveIt motion planning environment. As it does not allow for a goal sphere defined in TCP space, but instead



requires a target joint configuration, we use valid goal configurations obtained from our algorithms instead. Finally, we forward kinematic paths in joint space obtained from RRTConnect as an initial guess to the CHOMP algorithm to enhance its performance. Please note the all the algorithms accessed through MoveIt are simply executed with their default parameters without any further tuning.

4.7.3 Results

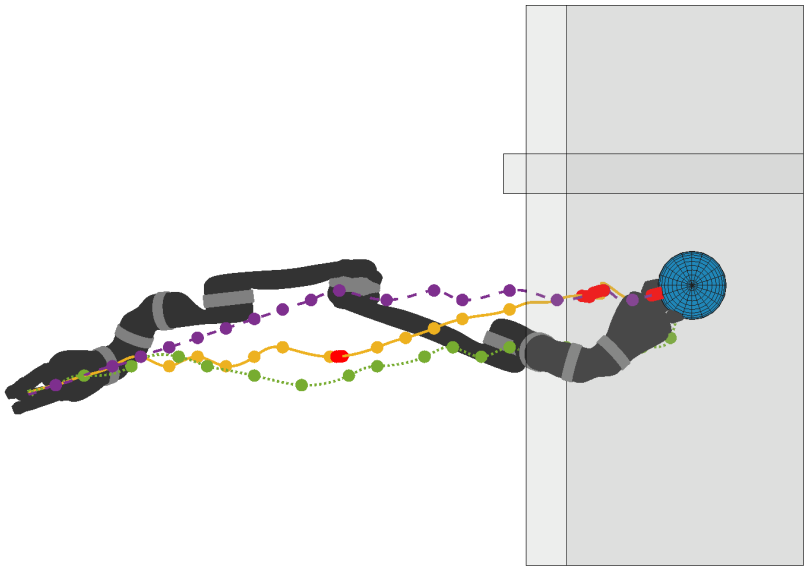
In order to validate our algorithms in a more generalized setting, we sample the robots TCP space envelope $\mathcal{W} \subseteq \mathbb{R}^3$ with 10 equidistant intervals in every dimension and use the inverse dimensional mapping to determine a corresponding feasible joint configuration for every grid point. The arising 477 feasible states are given, one after another, as an initial constraint to our trajectory planning problem.
We pick initial configuration

$$z \coloneqq (-0.11, 5.4, 3.42, 3.14, 1.05, -2.29, 0, 0, 0, 0, 0, 0)^\top$$

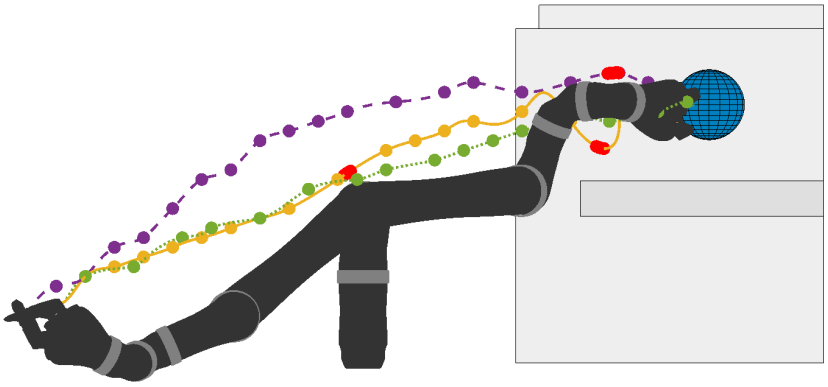
to prove the concept of our algorithms valid, starting from our original algorithm. As can be obtained from Figure 4.4, its first iteration suggests a trajectory suffering from collision. Subsequently, exploiting the adaptation of the penalty term, the way-points from dynamic programming, and consequently also the NLP trajectories try to navigate around problematic regions, leading to a feasible trajectory in the third iteration. To describe the arising robotic motion, Figure 4.5 illustrates four intermediate configurations of the robotic arm, while the trajectories states and controls are plotted over time in Figure 4.6.
Handing the problem to the algorithm variant designed for non-autonomous problems, we obtain a different trajectory from the first iteration due to the unique time scale and the different cost functions used within DP and NLP evaluation. As can be obtained from Figure 4.7, it nevertheless violates the collision avoidance approach and therefore the penalty update strategy also used in the original algorithm is applied. In the subsequent iteration our extended strategy is already able to provide a feasible trajectory. Again, we illustrate some intermediate robotic configurations in Figure 4.8, as well as states and controls of the resulting trajectory in Figure 4.9, to visualize the robotic motion. From the detailed result analysis of this single scenario, we were able to demonstrate the ability of our algorithms to generate dynamically feasible trajectories and recover from collision avoidance violation using penalty adaptation on the DP layer. We have therefore completed our proof of concept.

Moving on to a more general evaluation, we consider the overall collection of 477 different scenarios. Here, the original algorithm was able to provide feasible results in 383 ($\approx 80\,\%$) of them, while the variant designed for non-autonomous problems was still able to find 382 ($\approx 80\,\%$) feasible solutions. Considering only the solvable cases, the original algorithm converged on average on four to five iterations, while it took the variant on average three to four iterations.





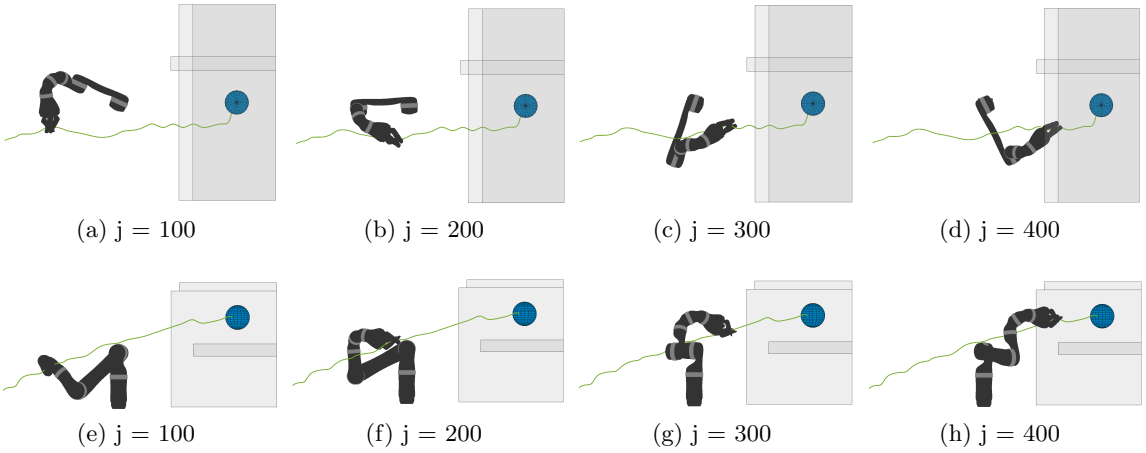
(a) XY Plane View



(b) XZ Plane View

Figure 4.4: Trajectories arising from our algorithm combining DP and NLP after the first (orange, solid), second (purple, dashed) and third (green, dotted) iteration, with points indicating the respective waypoints from DP. Parts suffering from collision are marked in red, as well as static obstacles (gray) and the target region (blue). The Mico arm is illustrated in the initial configuration as well as in the goal configuration arising from the third algorithm iteration.

In contrast, handing our set of scenarios to the pure Ipopt algorithm with a maximum number of 3000 iterations and zero initial guess, we were only able to find a solution in 38 cases ($\approx$8 %), highlighting the struggle of gradient based optimization strategies when it comes to collision avoidance based on the signed distance function. Regarding the solvers from the robotics community, pure CHOMP was able to provide 77 feasible trajectories ($\approx$ 16 %), while combined with an initial guess from RRTConnect, a feasible solution was found for 316 ($\approx$ 66 %) of the scenarios. Using pure RTTConnect in combination with the standard time parametrization we were



(a) j = 100 (b) j = 200 (c) j = 300 (d) j = 400

(e) j = 100 (f) j = 200 (g) j = 300 (h) j = 400

Figure 4.5: Trajectory arising from our algorithm combining DP and NLP after the third iteration (green), with robotic configurations illustrated at discrete time-steps 100,200,300 and 400 from XY plane view (a)-(d) and XZ plane view (e)-(h).

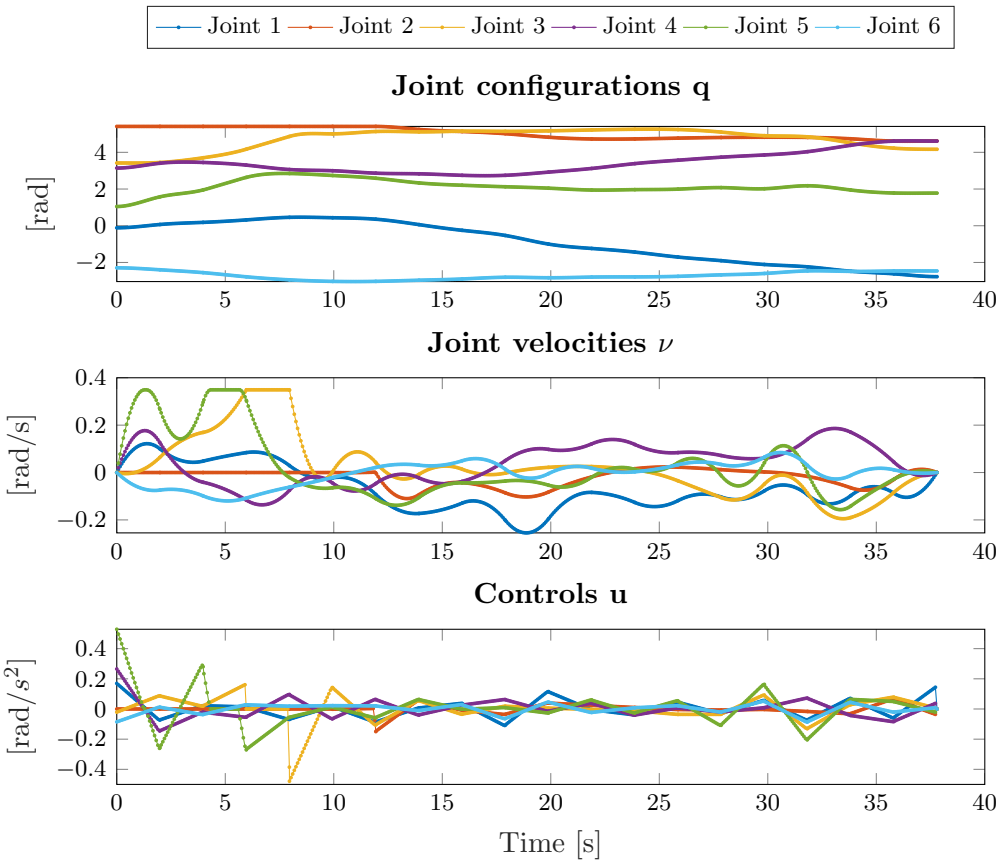
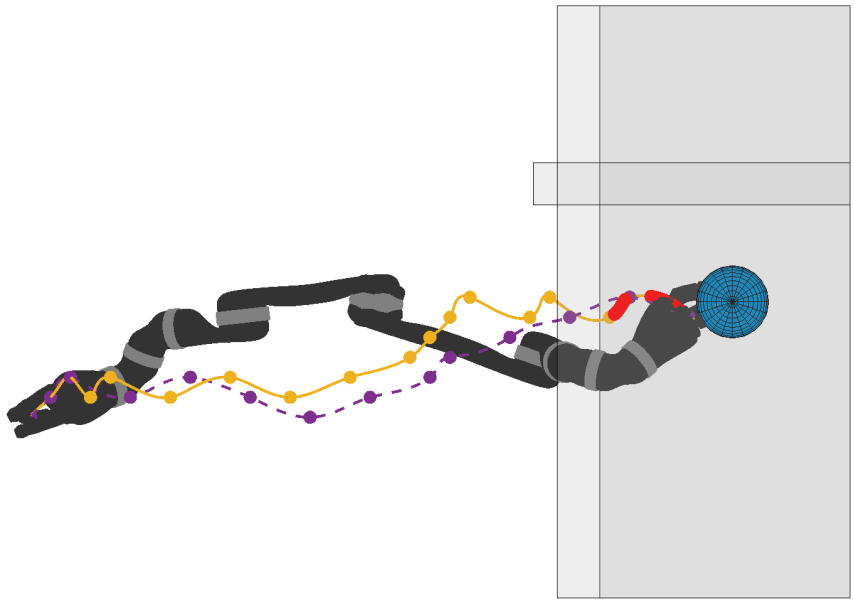
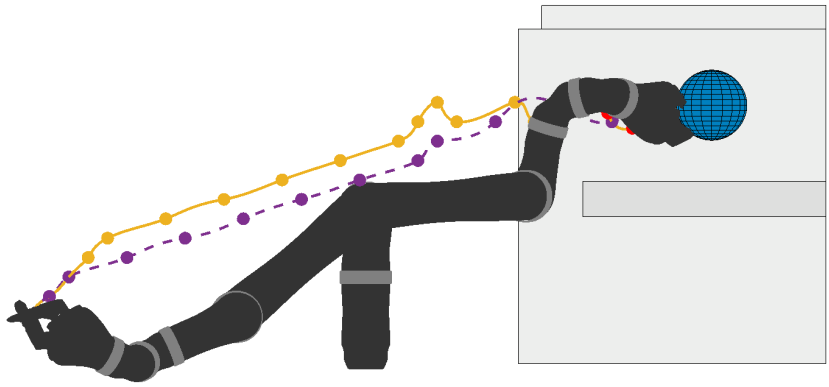


Figure 4.6: Joint configurations, joint velocities and controls arising from the third iteration of our algorithm combining DP and NLP, illustrated over time.

able to find 318 feasible trajectories which is also around 66 %. Please note that the strategy combining RTTConnect with Chomp uses a joint state goal, while this solver was able incorpo-



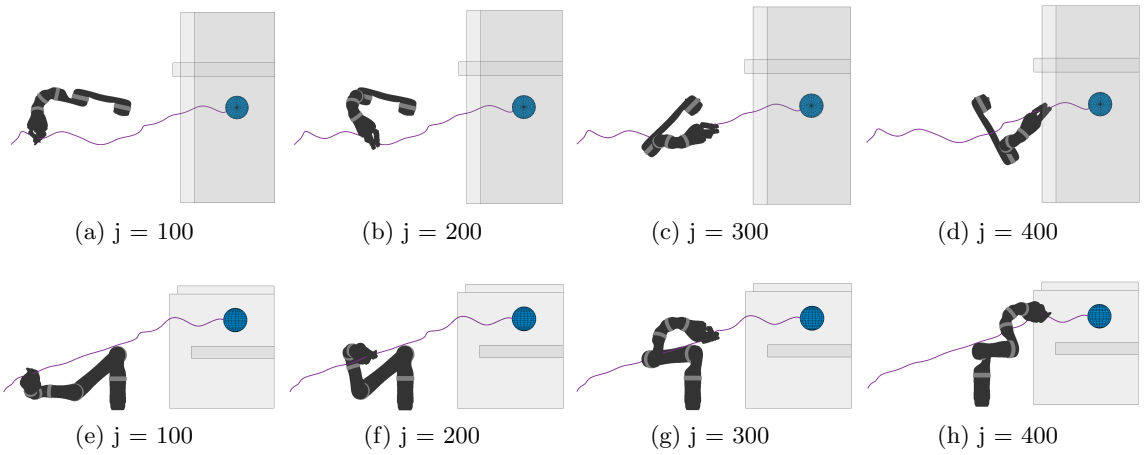
(a) XY Plane View



(b) XZ Plane View

Figure 4.7: Trajectories arising from the variant designed for non-autonomous problems of our algorithm after the first (orange, solid), and second (purple, dashed) iteration, with points indicating the respective waypoints from DP. Parts suffering from collision are marked in red, as well as static obstacles (gray) and the target region (blue). The Mico arm is illustrated in the initial configuration as well as in the goal configuration arising from the second algorithm iteration.

rate the original TCP goal region. Consequently, the different initializations of the RTTConnect solver result in varying explicit scenarios considered feasible. For the sake of completeness, we would like to point out here, that we detected 21 scenarios that were not solved by any strategy and therefore are possibly infeasible by construction. In total, we can characterize our original algorithm, as well as its variant designed for non-autonomous problems to outperform the other



(a) j = 100 (b) j = 200 (c) j = 300 (d) j = 400

(e) j = 100 (f) j = 200 (g) j = 300 (h) j = 400

Figure 4.8: Trajectory arising from the variant designed for non-autonomous problems of our algorithm combining DP and NLP after the second iteration (purple), with robotic configurations illustrated at discrete time-steps 100,200,300 and 400 from XY plane view (a)-(d) and XZ plane view (e)-(h).

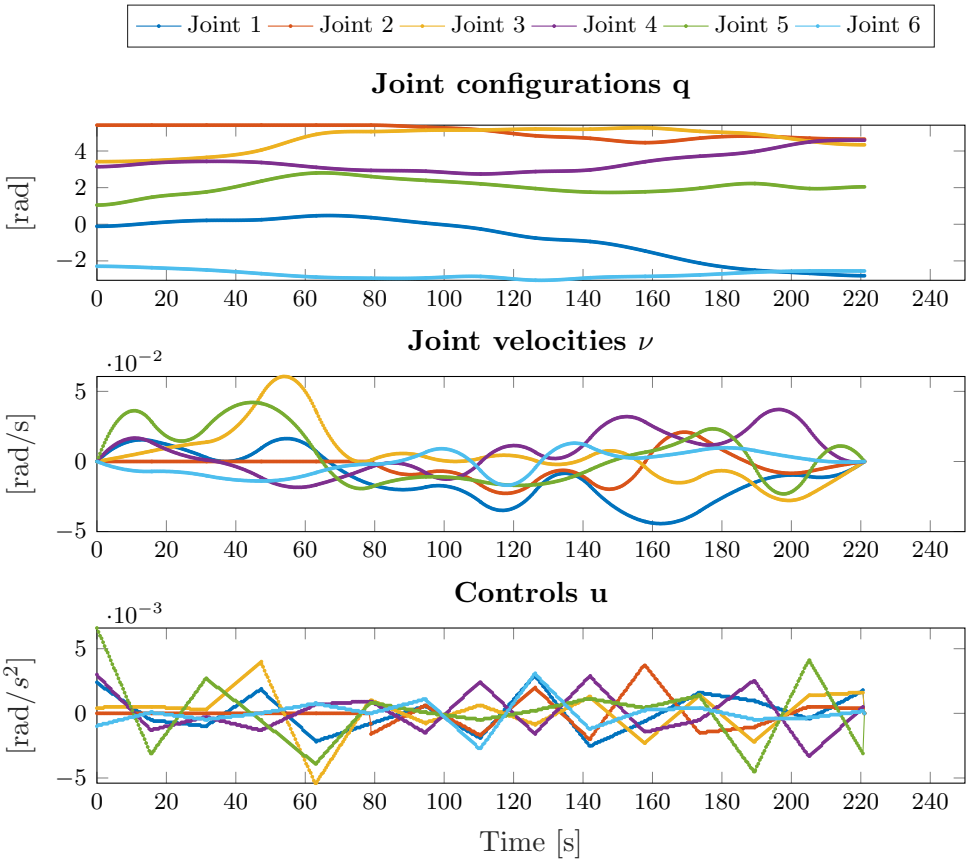


Figure 4.9: Joint configurations, joint velocities and controls arising from the second iteration of the variant designed for non-autonomous problems of our algorithm combining DP and NLP, illustrated over time.

solvers when searching for a feasible trajectory in this experiment.
Regarding the execution times, our algorithms take around 15 s for the initial penalty evaluation and another 10 to 20 seconds for every iteration, resulting in runtimes ranging from 25 seconds up to 6 minutes for this experiment. The strategy and the corresponding implementation are therefore, in contrast to RTTConnect and Chomp, not yet applicable in real time scenarios.

Remark: Due to the linear dynamics and the usage of a huge upper trajectory time bound, the NLP problems arising from this scenario within our algorithm where in almost every case solvable. As a result, the only practical difference between the original Algorithm and the variant designed for non-autonomous problems here was their different scaling with respect to time as well as the different cost function in the DP and NLP problems. It is therefore not surprising that we obtained an approximately equal performance of the two algorithm variants on this planning task.



4.8 Experiment: Satellite Trajectory Planning

As a part of the project SeRANIS [90], we are dealing with the issue of space debris removal. Given i.e a tumbling wreck of a satellite, we want to chase this target using a satellite manipulator with a robotic arm attached and bring it to a garbage orbit, where it can not cause any damage on other functional objects. From the motion planning perspective, we split this task into three phases: Starting from a position far away, we initially aim to maneuver the chaser within a certain range closer to the target. In this phase, it is sufficient to consider motion of the satellite body, with the robotic arm kept static for simplicity. Subsequently, a precise maneuvering of the robotic arm to a grasping point on the target surface is required, involving a detailed dynamic model of the full multi body system. Additionally, we have to consider collision avoidance between geometric parts of target and chaser. Finally, docking, stabilization and motion to a garbage orbit of the arising coupled system is requested. In this experiment, we focus on the second phase, bringing the chaser from an initial pose close to the target, to a valid configuration for grasping.
Considering the high dimensional multi-body system of the chaser, reaching for a tumbling piece of space debris with respect to geometric collision avoidance, this results in a task perfectly designed for the variant of our method combining DP and NLP extended to time dependent scenarios. We describe the full problem as well as our chaser model in detail in Section 4.8.1, while the functions and parameters required for our algorithm's variant are introduced in Section 4.8.2. The results presented in Section 4.8.3 do not only illustrate our algorithms ability to handle time dependent goals and collision avoidance constraints, but also demonstrate the recover procedure from infeasible NLP problems using the adaptive penalty approach. Finally we characterize our algorithm to be, in contrast to the general purpose solver Ipopt, able to provide a feasible solution in most of the scenarios considered.

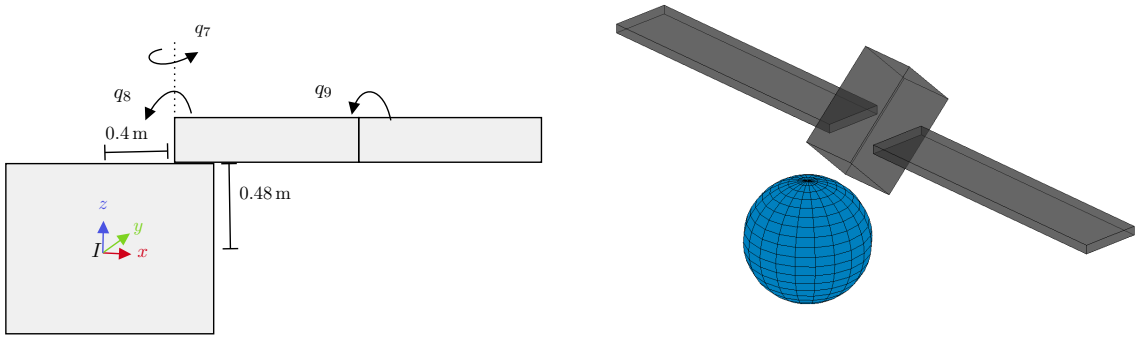
4.8.1 Experiment Setup

In order to chase parts of debris in space, we equip a rectangular satellite body, with a three joint robotic arm, as illustrated in Figure 4.10a. Relating to the real satellite build within the project of SeRANIS[90], we choose our modeled satellite body to approximately track the originals extend of $(0.93, 0.96, 0.78)^\top$m as well as its weight of 200 kg. The additional robotic links are fitted accordingly with extends of $(1, 0.1, 0.1)^\top$m and weight of 10 kg each. Considering a six dimensional pose to describe the satellite's position and orientation in space, we end up with a vector $q \in \mathbb{R}^9$ to fully represent the systems configuration.

We assume the chaser to be floating freely in space and equipped with thrusters that allow application of a force on each of the six satellite axis of motion. Furthermore, we assume a control on each of the joint axis, such that we end up with a control vector $u \in \mathbb{R}^9$ to manipulate the chaser. While we directly control the acceleration on the robotic arm joints using double integrators

$$\ddot{q}_i \coloneqq u_i \quad i = 6, 7, 8,$$





(a) Space manipulator (b) Tumbling target

Figure 4.10: Geometric models of the space manipulator with arm joint axis highlighted in configuration zero and the tumbling target object with goal sphere (blue).

we derive the equation of motion

$$M(q, \dot{q})\ddot{q}_s = \tilde{f}(q, \dot{q}, u_s)$$

with $u_s \coloneqq (u_0, \ldots, u_5)^\top$, $q_s \coloneqq (q_0, \ldots, q_5)^\top$ and mass matrix $M \in \mathbb{R}^{6\times 6}$ arising from Lagrangian mechanics for a free floating multi-body system in [61] based on [91], to describe the control forces acting on the satellite body. In order to include above second order model into a first order dynamics function, such as required in Problem (4.1), we consider states $x \coloneqq (q, \dot{q})^\top$ combining current configuration q and velocity $\dot{q}$ and derive the 18 dimensional dynamics function

$$f(x, u) \coloneqq \begin{pmatrix} \dot{q} \\ M^{-1}(x)\tilde{f}(x, u_s) \\ u_a \end{pmatrix}$$

with robotic arm controls $u_a \coloneqq (u_5, u_6, u_7)^\top$. As we model the second phase of the motion planning process, we can assume chaser and target to stay close together, such that we can assume a bounded position space for our manipulator. Additionally, we choose limitations on the rotational pose entries to ensure invertible, non-singular mass matrices and limit the robotic arm's joint positions by their geometric capabilities. Leaving the velocity space unrestricted, we end up with state space

$$\mathcal{K} \coloneqq \{q \in \mathbb{R}^9 | q_{min} \leq q \leq q_{max}\}$$

limited by

$$q_{min} \coloneqq (-1, -5, -2, -\infty, -\pi/2 + 0.01, -\infty, -\pi/2, -\pi, -\pi/2)^\top$$
$$q_{max} \coloneqq (15, 5, 5, \infty, \pi/2 - 0.01, \infty, \pi/2, 0, \pi/2)^\top$$

with position limits given in meters and rotational limits represented in radians. Please recapitulate that we aim to bring the chaser from a configuration close to the target to a valid grasping

state. Consequently, it is sufficient to choose narrow position limits covering the target here. In control space, we limit the force on the translational axes of the satellite body by $\pm 0.5\,\mathrm{N}$, on the rotational degrees of freedom by $\pm 0.1\,\mathrm{Nm}$ and the arm joint acceleration with $\pm 0.1\,\mathrm{rad/s^2}$. For the target object, we consider a wreck composed of a satellite body with solar panels attached on both sides as illustrated in Figure 4.10b. As we aim to grasp the target at the lower surface of the rectangular satellite body, we attach a spherical region of radius $r = 0.5\,\mathrm{m}$ as illustrated in Figure 4.10b, as a goal for the chaser's TCP during motion planning. To model the behavior of an uncooperative target in space, we consider a simplified tumbling motion as illustrated in Figure 4.11 with parameters $m = 5\,\mathrm{m}$ and $\gamma_m = \pi/4\mathrm{rad}$ as well as rotational velocities $\frac{\pi}{600}\mathrm{rad/s}$ and $\frac{4\pi}{600}\mathrm{rad/s}$ executed in directions of r1 and r2 respectively. Here, we place the tumbling motion's origin is at $(11.5, 0, -2)^\top \mathrm{m}$. Using function $p_{ref} : [0, T] \to \mathbb{R}^3$ to track the tumbling goal sphere center point, we define the arising goal constraint of our planning problem as

$$b_0(t, x) = \beta \max\{|\kappa(q) - p_{ref}(t)\| - r, 0\}$$

with scaling $\beta = 10^3$, as well as forward kinematics $\kappa : \mathcal{K} \to \mathbb{R}^3$. To ease the docking process, we additionally request the chaser satellite body's translational and rotational velocity to match the respective velocities of the targets docking point. As a result, we introduce additional constraints to restrict the velocity at the chaser's goal configuration with

$$b_i(t, x) = x_{i+7} - \nu_{ref,i}(t) \quad i = 1, \ldots, 6$$

using function $\nu_{ref} : [0, T] \to \mathbb{R}^6$ to describe the velocity of the goal sphere's center point. To

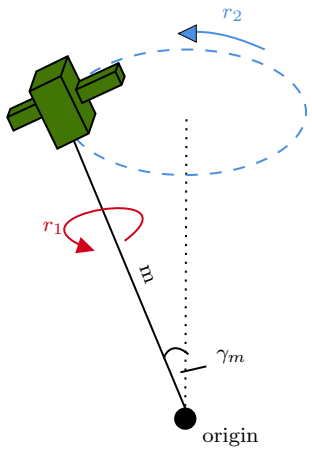


Figure 4.11: Tumbling motion of the space debris, including spinning around it self r1 (red) as well as rotation r2 (blue) of axis m around global z direction (dashed)

model collision avoidance, we let polyhedral set $A(q)$ collect the objects from the chasers geometry in configuration q, while set $B(t)$ represents the tumbling target's geometric components

at time point t. Based on that, we can introduce a total signed distance function

$$g(t, x) = \min_{a \in A(q), b \in B(t)} d(a, b) - \epsilon$$

using the signed distance function d from Chapter 2 as well as safety distance $\epsilon = 0.01\,\mathrm{m}$, to ensure non-overlapping target and chaser components. Finally, we restrict the maximum trajectory time to $T = 600\,\mathrm{s}$.

4.8.2 Algorithm Setup

As already done within the previous experiment, we let the dimensional mapping Ω characterize the forward kinematics of the space manipulator, mapping from a 18 dimensional state space to the three dimensional TCP space. Choosing again a linear dynamics function

$$\bar{f}(w(t), v(t)) \coloneqq v(t)$$

on the lower dimensional space, we let the corresponding state space $\mathcal{W}$ be bounded by the space manipulator's position bounds, while we limit the control bounds $\mathbb{V}$ with $\pm 0.05\,\mathrm{m/s}$ on every dimension respectively. Additionally, using penalty scaling $\rho_1 = \rho_2 = 10^3$, we discretise the DP problem with $M \coloneqq 20$ equidistant points in time, $(20, 11, 15)^\top$ equidistant grid points in space as well as with 14 points on every control dimension. The arising problem is solved using the DP Algorithm from Chapter 3 with optimal stopping as well as CPU parallelization applied. For the inverse mapping, we choose parameters $\varrho_1 = 50$ and $\varrho_2 = 1$. Within the NLP problem, we filter feasible results according to cost function

$$c(x, u) \coloneqq \sum_{j=0}^{N} \|q(t_j) - q^{init}(t_j)\|^2$$

with initial guess $\{q^{init}(t_j)\}_{j=0}^{N}$ obtained from linear interpolation of the waypoints. Additionally, we ensure waypoint tracking, by adding functions

$$p_i(x) \coloneqq x - x_i \quad i = 1, \ldots, M$$

for every waypoint. Finally, we reuse the penalty adaptation's parametrisation introduced within the robotic arm Experiment in the previous Section 4.7 and allow a maximum number of $k = 15$ algorithm iterations.

As in the previous experiment, we use a C++ implementation of our algorithm, executed on an AMD Ryzen Threadripper 3090X system with 256GB RAM. Again, we exploit the general purpose solver Ipopt version 3.14.12 [147] combined with linear solvers from HSL [78] and accessed through the Ifopt interface [152], to solve the NLP problem (4.10), the inverse dimensional mapping (4.4) as well as the initial penalty (4.9).

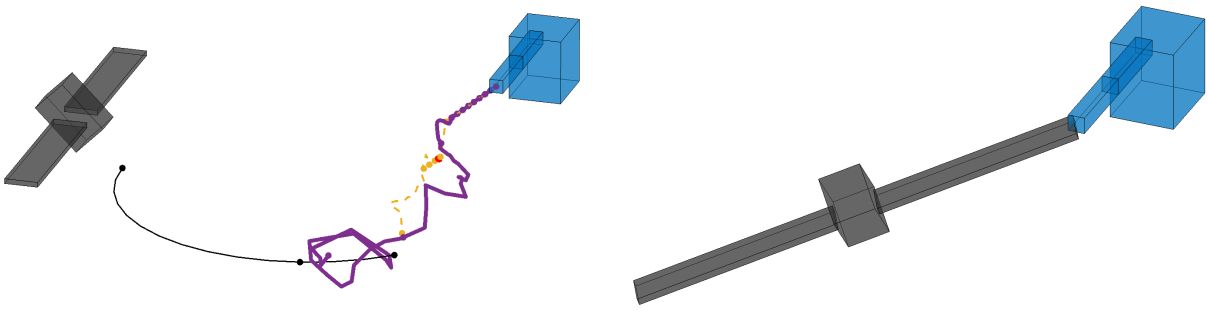


4.8.3 Results

To allow a more generalized evaluation of our algorithm, we discretize the three dimensional space $\mathcal{W}$ with 10 equidistant points in every dimension and use the 1000 different initial configurations $z \in \mathcal{K}$ arising from inverse dimensional mapping in every grid point, to generate a set of 1000 scenarios. To analyze the behavior of our algorithm in detail, we initially pick configuration

$$z_1 = (12.6, 3.9, 3.7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)^\top$$

and illustrate the trajectories arising from our algorithm in Figure 4.12. Starting from the first



(a) Trajectories (b) Configuration in Collision

Figure 4.12: Trajectories arising from initial configuration z_1 with the variant designed for non-autonomous problems of our algorithm after the first (orange, dashed), and second (purple, solid) iteration, with points indicating the respective way-points from DP. Parts suffering from collision are marked in red. The space manipulator and the tumbling target are illustrated in the initial configuration in Figure 4.12a, as well as in a configuration causing collision from the first trajectory in Figure 4.12b

iteration, we obtain a trajectory suffering from collision, due to a solar panel of the target hitting the arm of the space manipulator (see Figure 4.12b), when connecting the way-points. Subsequently, the penalty adaptation strategy results in different low dimensional way-points, as well as a feasible overall trajectory, verifying our algorithms ability to handle collisions observed in NLP trajectories. Regarding the arising configurations (Figure 4.13), velocities (Figure 4.14) and controls (Figure 4.15), we would like to point out that we did not optimize with respect to minimum control when generating the trajectory, but instead penalized configuration deviations with respect to reference trajectory q^{init}, arising from way-point interpolation. It is therefore not surprising that we obtain a kind of bang-bang control. Additionally, as we were mainly searching for a feasible proposal for our overall problem, we have chosen a rather large convergence tolerance of 0.01 during trajectory generation for a more efficient execution. As a consequence, we can especially in the rotational joints of the satellite obtain configurations that allow to fulfill all the constraints of the problem, but in some parts show a large deviation from the reference trajectory that remains on any rotational joint within interval $[-0003, 0.003]$rad on our fixed time scale. Increasing the solvers convergence tolerance could possible lower this gap.

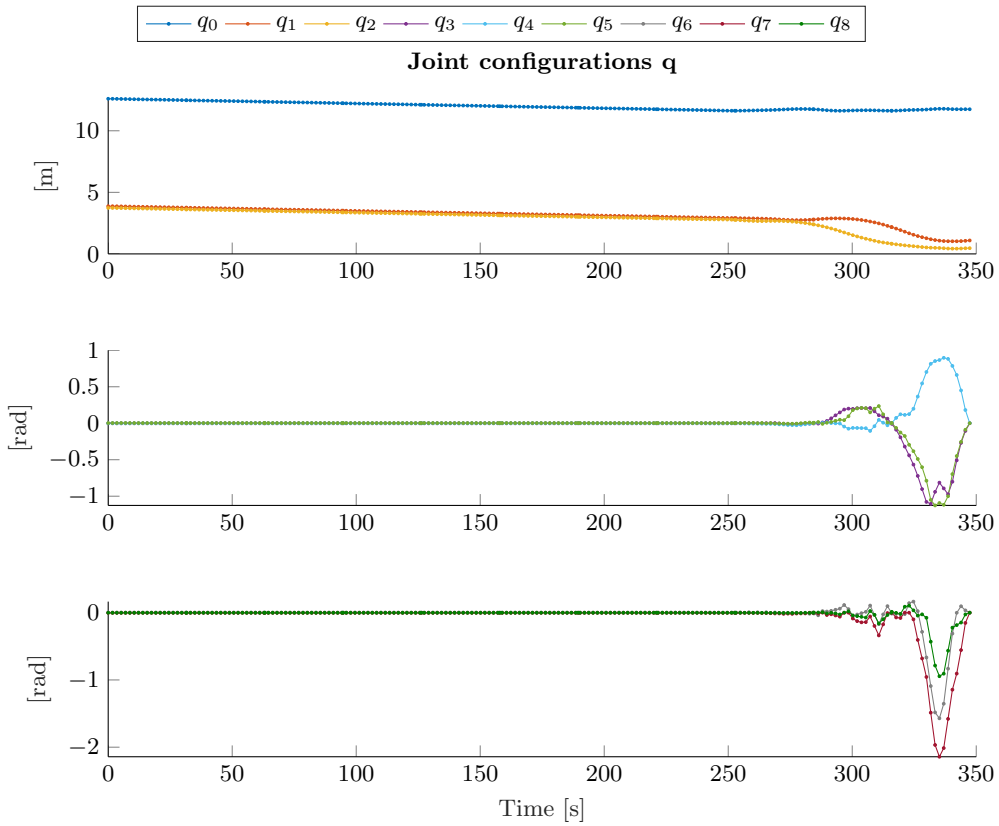


Figure 4.13: Joint configurations, arising from the second iteration of the variant designed for non-autonomous problems of our algorithm combining DP and NLP using initial z_1, illustrated over time.

As a second example, we analyze the results arising from configuration

$$z_2 = (9.04, -0.55, -1.7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)^\top$$

in Figure 4.16 to illustrate recovery from infeasible NLP problems. Here, we start from a small number of five way-points, associated with a final time of approximately 126 s. On trajectory generation, the NLP solver is not able to solve this problem, returning a suggestion where the goal velocity constraint is violated. As a result, the penalty is increased at the goal configuration, when associated with time point 126 s, leading to a trajectory using six waypoints with final time 158 s in the subsequent iteration. Again, the arising NLP problem is not considered feasible, and thus a penalty update according to the constraint violations, leads to an increase of the number of tracking points associated with a larger final time. The resulting trajectory using eight DP points and final time 221 s is not only a valid solution to the NLP problem (4.10), but also solves the overall task from (4.7). Hence, our algorithm terminates.

Regarding now the full set of 1000 scenarios, our algorithm was able to provide as feasible solution in 886 of them (88.6 %), with on average two to three iterations executed. For comparison, we

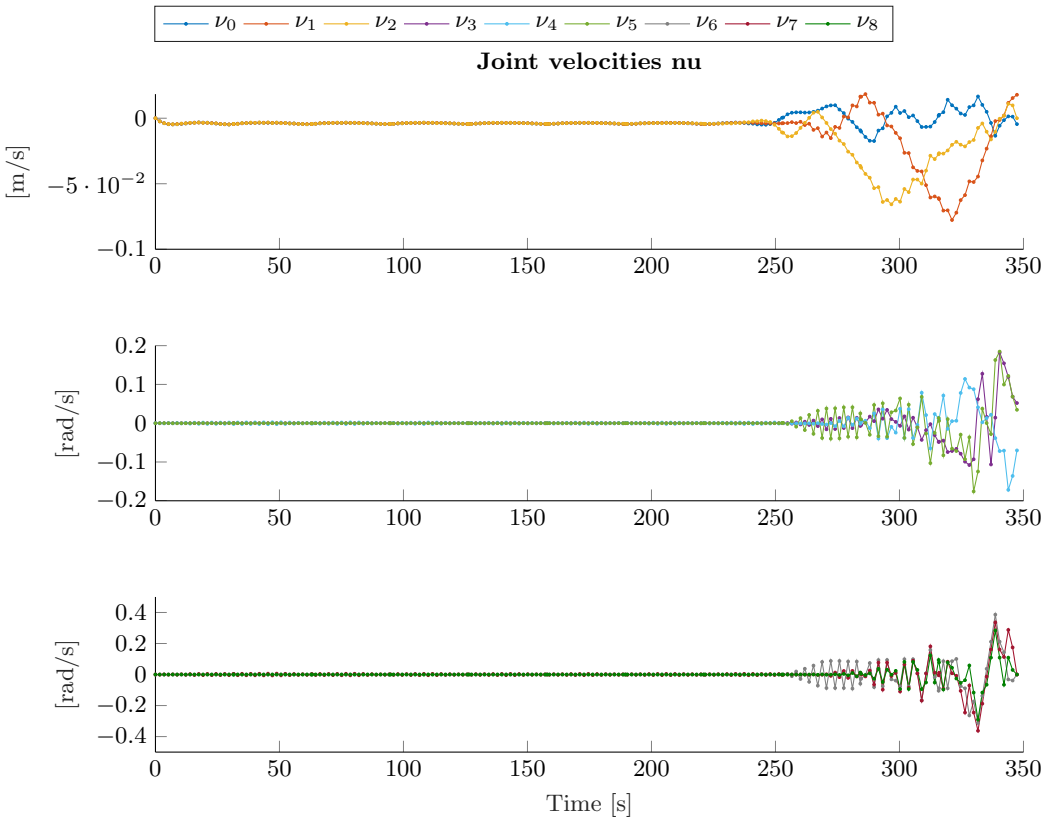


Figure 4.14: Joint velocities, arising from the second iteration of the variant designed for non-autonomous problems of our algorithm combining DP and NLP using initial z_1, illustrated over time.

handed 125 of our scenarios equidistantly distributed in TCP space to a pure Ipopt solver with zero initial guess. Executing at maximum 3000 iterations, the solver was only able to provide a feasible solution in two cases (1.6 %). Out of curiosity, we tried to solve the same 125 problems with Ipopt again, this time ignoring the collision avoidance constraint. Here, all of the scenarios were successfully solved, with a subsequent collision avoidance check validating 120 of the solution trajectories to also successfully avoid collision. While on the one hand this points out the struggle of gradient based NLP solvers with our collision avoidance constraint, it also emphasized the need for alternative algorithms as proposed here. In total, we can conclude from this experiment, that our algorithm is able to recover from suggested trajectories suffering from collision, as well as from infeasible trajectory generation problems, as intended by design of the penalty adaptation strategy. Furthermore, in contrast to a pure NLP solver, it was able to provide a feasible trajectory in most of the test cases. Overall, we have proven the extended algorithm combining DP and NLP to be able to deal with time dependent obstacles as well as moving targets successfully.

Regarding execution times, it took around 35 s to evaluate the two algorithm iterations required for scenario z_1, while in total 63 s where necessary to provide a solution to scenario z_2 within three iterations. As already pointed out in the previous experiment, our algorithm is therefore

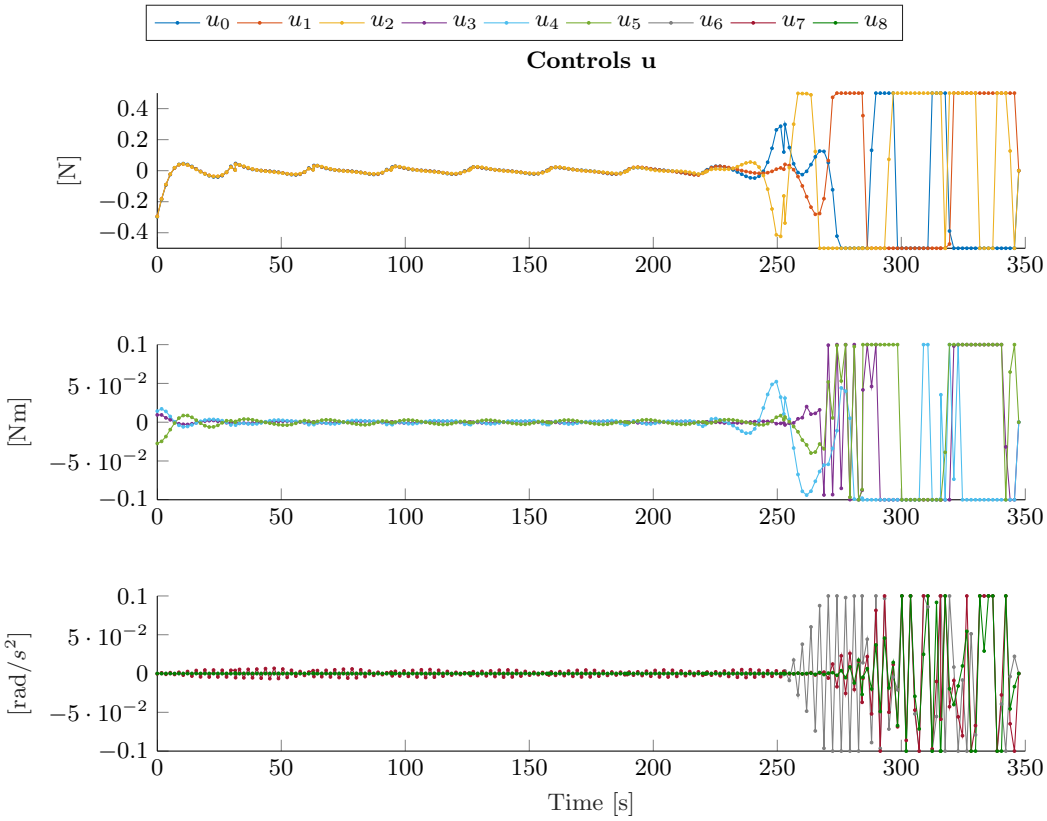


Figure 4.15: Controls, arising from the second iteration of the variant designed for non-autonomous problems of our algorithm combining DP and NLP using initial z_1, illustrated over time.

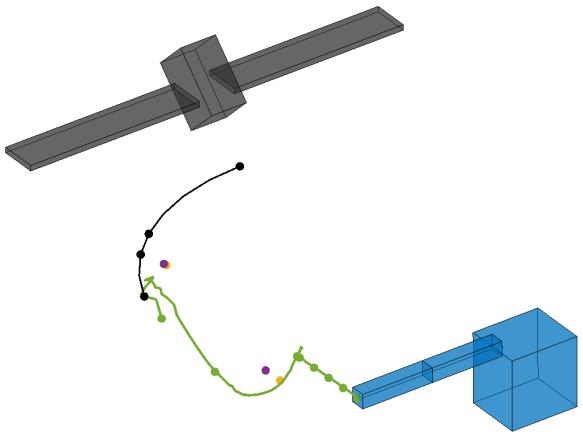


Figure 4.16: Trajectories arising from initial configuration z_2 with the variant designed for non-autonomous problems of our algorithm after the first (orange), second (purple) and third (green) iteration. DP way-points are marked with points, connected on feasible NLP results by the corresponding paths.

not in general applicable in real time, but might still be beneficial for the large time scales with slowly controllable motions concerned in space applications.

Remark: Regarding the resulting trajectories from Figures 4.12 and 4.16, we can identify circular motions in the feasible trajectories when connecting the last two way-points. A possible explanation for that arises from the goal velocity constraint requesting a match of the 6 dimensional velocities obtained at the docking point from the tumbling motion, with the goal velocities on the space manipulator's satellite body. Trying to fulfill this constraint within the last part of the trajectory, it might be necessary to execute some additional motion even though increasing the trajectories cost.



Chapter 5

Summary and Future Work

Initially started from the analysis of current strategies to model and evaluate collision avoidance with respect to convex geometric obstacles, we have not only proven correctness of the SAT method for collision checking in Chapter 2, but additionally provided an extension to evaluate the penetration distance. Based on a small numerical example, we have seen this algorithm to outperform current state of the art strategies. As the efficiency of SAT based algorithms is directly related to the number of features arising from the considered objects, future work could focus on numerically estimating the threshold in object features limiting the improved efficiency of SAT compared to incremental strategies.

Afterwards, we dedicated Chapter 3 to the method of dynamic programming, following the backward Semi-Lagrangian discretisation scheme proposed in [21] based on [49]. Therein, slight variations in the approximations convergence proof allowed us to skip highly restricitive assumptions. Searching for strategies to improve the arising discrete dynamic programming algorithms efficiency on a high dimensional motion planning problem, we adapted the common strategies of optimal stopping, adaptive discretisations as well as parallelisation to our robotic motion planning task. While the approximations convergence guarantees remained untouched in splitting and parallelisation extensions, the optimal stopping problem required a reformulation of the DPP and HJB equations and consequently an extensive derivation to verify convergence properties. Regarding numerical evaluation on an industrial task using a velocity controlled robotic manipulator with four degrees of freedom, especially the reformulation with respect to optimal stopping as well as the GPU parallelisation strategy have proven their efficiency. In a real world autonomous driving scenario using MPC, we were able to run the accelerated DP algorithm in real time. Within our numerical experiments, the concept of adaptive grid refinement has not been proven very efficient, possibly due to the splitting operation performed simultaneously in every state space dimension. An alternative could for example provide sparse grids [18, 31] or dimension specific splitting criteria [111], as well as global splitting criteria to refine regions with respect to their relevance for the overall value function environment [111, 133]. Also an investigation into more efficient algorithms to provide a solution to the HJB such as single step pass methods [26] or optimistic planning [17] can possibly be beneficial in future work.

In Chapter 4 we proposed a novel strategy, combining dynamic and nonlinear programming for



high dimensional trajectory planning with respect to geometric obstacle avoidance. Exploiting DP to handle collision avoidance as well as NLP capabilites to handle dynamics constraints in an high dimensional scenario, our method was designed to combine best of both worlds. To allow also application in moving environments, we additionally proposed an extension to non-stationary problems. Given a motion planning scenario for an acceleration controlled robotic arm with six degrees of freedom, we have demonstrated our algorithm as well as the extension to outperform current state of the art solvers. Furthermore, effectivity of the variant designed for non-autonomous problems was demonstrated on the task of capturing a tumbling target arising from space debris removal. Future work on our algorithm could i.e. include the addition of a cost function into the overall problem, as well as convexification of the trajectory generation problem to improve evaluation times. Also addition of a differential collision avoidance formulation such as [54] into the NLP problem, might lower the number of algorithm iterations required and therefore increase efficiency.



Bibliography

[1] YouBot 3D Model - youBot wiki.

[2] Yves Achdou and Italo Capuzzo-Dolcetta. Approximation of solutions of hamilton-jacobi equationson the heisenberg group. *ESAIM: Mathematical Modelling and Numerical Analysis*, 42(4):565–591, 2008.

[3] Evan Ackerman. A robot for the worst job in the warehouse: Boston dynamics' stretch can move 800 heavy boxes per hour. *Ieee Spectrum*, 59(1):50–51, 2022.

[4] Haider AF Almurib, Haidar Fadhil Al-Qrimli, and Nandha Kumar. A review of application industrial robotic design. In *2011 Ninth International Conference on ICT and Knowledge Engineering*, pages 105–112. IEEE, 2012.

[5] Albert Altarovici, Olivier Bokanowski, and Hasnaa Zidani. A general hamilton-jacobi framework for non-linear state-constrained control problems. *ESAIM: Control, Optimisation and Calculus of Variations*, 19(2):337–357, 2013.

[6] Mohamed Assellaou. *Hamilton Jacobi Bellman Approach for some applied optimal control problems.* PhD thesis, Ensta ParisTech, 2015.

[7] I Babuvška and Werner C Rheinboldt. Error estimates for adaptive finite element computations. *SIAM Journal on Numerical Analysis*, 15(4):736–754, 1978.

[8] Antoine Bambade, Sarah El-Kazdadi, Adrien Taylor, and Justin Carpentier. Prox-qp: Yet another quadratic programming solver for robotics and beyond. In *RSS 2022-Robotics: Science and Systems*, 2022.

[9] Martino Bardi, Italo Capuzzo Dolcetta, et al. *Optimal control and viscosity solutions of Hamilton-Jacobi-Bellman equations*, volume 12. Springer, 1997.

[10] Richard Bellman. *Dynamic Programming.* Princeton University Press, Princeton, New Jersey, 1957.

[11] Mehdi Benallegue, Adrien Escande, Sylvain Miossec, and Abderrahmane Kheddar. Fast c 1 proximity queries using support mapping of sphere-torus-patches bounding volumes. In *2009 IEEE International Conference on Robotics and Automation*, pages 483–488. IEEE, 2009.



[12] Gino van den Bergen. A fast and robust gjk implementation for collision detection of convex objects. *Journal of graphics tools*, 4(2):7–25, 1999.

[13] Karl-Eduard Berger and Francois Galea. An efficient parallelization strategy for dynamic programming on gpu. In *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum*, pages 1797–1806. IEEE, 2013.

[14] Dimitri Bertsekas. *Dynamic Programming and Optimal Control: Volume I*. Athena Scientific, 2012.

[15] James E Bobrow. A direct minimization approach for obtaining the distance between convex polyhedra. *The International Journal of Robotics Research*, 8(3):65–76, 1989.

[16] Olivier Bokanowski, Nicolas Forcadel, and Hasnaa Zidani. Deterministic state-constrained optimal control problems without controllability assumptions. *ESAIM: Control, Optimisation and Calculus of Variations*, 17(4):995–1015, 2011.

[17] Olivier Bokanowski, Nidhal Gammoudi, and Hasnaa Zidani. Optimistic planning algorithms for state-constrained optimal control problems. *Computers & Mathematics with Applications*, 109:158–179, 2022.

[18] Olivier Bokanowski, Jochen Garcke, Michael Griebel, and Irene Klompmaker. An adaptive sparse grid semi-lagrangian scheme for first order hamilton-jacobi bellman equations. *Journal of Scientific Computing*, 55:575–605, 2013.

[19] Riccardo Bonalli, Abhishek Cauligi, Andrew Bylard, and Marco Pavone. Gusto: Guaranteed sequential trajectory optimization via sequential convex programming. In *2019 International conference on robotics and automation (ICRA)*, pages 6741–6747. IEEE, 2019.

[20] Vincent Boyer, Didier El Baz, and Moussa Elkihel. Solving knapsack problems on gpu. *Computers & Operations Research*, 39(1):42–47, 2012.

[21] Andreas Britzelmeier. *Decomposition and Hamilton-Jacobi-Bellman methods for nonconvex generalized Nash equilibrium problems*. PhD thesis, Dissertation, Neubiberg, Universität der Bundeswehr München, 2021.

[22] Andreas Britzelmeier, Alberto De Marchi, and Rebecca Richter. Dynamic and nonlinear programming for trajectory planning. *IEEE Control Systems Letters*, 7:2569–2574, 2023.

[23] Hans-Joachim Bungartz and Michael Griebel. Sparse grids. *Acta numerica*, 13:147–269, 2004.

[24] Christof Büskens and Dennis Wassel. The esa nlp solver worhp. In *Modeling and optimization in space engineering*, pages 85–110. Springer, 2012.



[25] Lucian Buşoniu, Előd Páll, and Rémi Munos. Discounted near-optimal control of general continuous-action nonlinear systems using optimistic planning. In *2016 American Control Conference (ACC)*, pages 203–208. IEEE, 2016.

[26] Simone Cacace, Emiliano Cristiani, and Maurizio Falcone. Two semi-lagrangian fast methods for hamilton-jacobi-bellman equations. In *System Modeling and Optimization: 26th IFIP TC 7 Conference, CSMO 2013, Klagenfurt, Austria, September 9-13, 2013, Revised Selected Papers 26*, pages 74–84. Springer, 2014.

[27] S. Cameron and R. Culley. Determining the minimum translational distance between two convex polyhedra. In *Proceedings. 1986 IEEE International Conference on Robotics and Automation*, volume 3, pages 591–596, 1986.

[28] Stephen Cameron. Enhancing gjk: Computing minimum and penetration distances between convex polyhedra. In *Proceedings of international conference on robotics and automation*, volume 4, pages 3112–3117. IEEE, 1997.

[29] F Camilli, M Falcone, P Lanucara, and A Seghini. A domain decomposition method for bellman equations. *Contemporary Mathematics*, 180:477–477, 1994.

[30] Alexandre Campeau-Lecours, Hugo Lamontagne, Simon Latour, Philippe Fauteux, Véronique Maheu, François Boucher, Charles Deguire, and Louis-Joseph Caron L'Ecuyer. Kinova modular robot arms for service robotics applications. In *Rapid Automation: Concepts, Methodologies, Tools, and Applications*, pages 693–719. IGI global, 2019.

[31] Elisabetta Carlini, Maurizio Falcone, and Roberto Ferretti. An efficient algorithm for hamilton–jacobi equations in high dimension. *Computing and Visualization in Science*, 7:15–29, 2004.

[32] Elisabetta Carlini, Adriano Festa, and Nicolas Forcadel. A semi-lagrangian scheme for hamilton–jacobi–bellman equations on networks. *SIAM Journal on Numerical Analysis*, 58(6):3165–3196, 2020.

[33] Cristiano Cervellera, Mauro Gaggero, Danilo Macciò, and Roberto Marcialis. Quasi-random sampling for approximate dynamic programming. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2013.

[34] Cristiano Cervellera and Marco Muselli. Efficient sampling in approximate dynamic programming algorithms. *Computational Optimization and Applications*, 38:417–443, 2007.

[35] Nilanjan Chakraborty, Jufeng Peng, Srinivas Akella, and John E Mitchell. Proximity queries between convex objects: An interior point approach for implicit surfaces. *IEEE Transactions on Robotics*, 24(1):211–220, 2008.

[36] Sachin Chitta, Ioan Sucan, and Steve Cousins. Moveit![ros topics]. *IEEE Robotics & Automation Magazine*, 19(1):18–19, 2012.



[37] Jia-Ruei Chiu, Jean-Pierre Sleiman, Mayank Mittal, Farbod Farshidian, and Marco Hutter. A collision-free mpc for whole-body dynamic locomotion and manipulation. In *2022 international conference on robotics and automation (ICRA)*, pages 4686–4693. IEEE, 2022.

[38] Richard Courant, Eugene Isaacson, and Mina Rees. On the solution of nonlinear hyperbolic differential equations by finite differences. *Communications on pure and applied mathematics*, 5(3):243–255, 1952.

[39] Wesley Cox, Lyndon While, and Mark Reynolds. A review of methods to compute minkowski operations for geometric overlap detection. *IEEE Transactions on Visualization and Computer Graphics*, 27(8):3377–3396, 2020.

[40] Michael G Crandall, Lawrence C Evans, and P-L Lions. Some properties of viscosity solutions of hamilton-jacobi equations. *Transactions of the American Mathematical Society*, 282(2):487–502, 1984.

[41] Michael G Crandall and P-L Lions. Two approximations of solutions of hamilton-jacobi equations. *Mathematics of Computation*, 43(167):1–19, 1984.

[42] Alberto De Marchi. On a primal-dual Newton proximal method for convex quadratic programs. *Computational Optimization and Applications*, 81(2):369–395, 2022.

[43] Berend Denkena and Thomas Lepper. Enabling an industrial robot for metal cutting operations. *Procedia CIRP*, 35:79–84, 2015.

[44] Christian Dietz, Sebastian Albrecht, Armin Nurkanović, and Moritz Diehl. Efficient collision modelling for numerical optimal control. In *2023 European Control Conference (ECC)*, pages 1–7. IEEE, 2023.

[45] I Capuzzo Dolcetta. On a discrete approximation of the hamilton-jacobi equation of dynamic programming. *Applied Mathematics and Optimization*, 10(1):367–377, 1983.

[46] I Capuzzo Dolcetta and Hitoshi Ishii. Approximate solutions of the bellman equation of deterministic control theory. *Applied Mathematics and Optimization*, 11(1):161–181, 1984.

[47] Adrien Escande, Sylvain Miossec, Mehdi Benallegue, and Abderrahmane Kheddar. A strictly convex hull for computing proximity distances with continuous gradients. *IEEE Transactions on Robotics*, 30(3):666–678, 2014.

[48] Maurizio Falcone. A numerical approach to the infinite horizon problem of deterministic control theory. *Applied Mathematics and Optimization*, 15(1):1–13, 1987.

[49] Maurizio Falcone and Roberto Ferretti. Semi-lagrangian schemes for hamilton–jacobi equations, discrete representation formulae and godunov methods. *Journal of computational physics*, 175(2):559–575, 2002.



[50] Maurizio Falcone and Roberto Ferretti. *Semi-Lagrangian approximation schemes for linear and Hamilton—Jacobi equations*. SIAM, 2013.

[51] Maurizio Falcone and Tiziana Giorgi. *An Approximation Scheme for Evolutive Hamilton-Jacobi Equations*, pages 289–303. Birkhäuser Boston, Boston, MA, 1999.

[52] Maurizio Falcone, Piero Lanucara, and Alessandra Seghini. A splitting algorithm for hamilton-jacobi-bellman equations. *Applied Numerical Mathematics*, 15(2):207–218, 1994.

[53] Jiayu Fan, Nikolce Murgovski, and Jun Liang. Efficient optimization-based trajectory planning. *arXiv preprint arXiv:2312.17440*, 2023.

[54] Jiayu Fan, Nikolce Murgovski, and Jun Liang. Efficient optimization-based trajectory planning for unmanned systems in confined environments. *IEEE Transactions on Intelligent Transportation Systems*, 2024.

[55] Farbod Farshidian, Michael Neunert, Alexander W Winkler, Gonzalo Rey, and Jonas Buchli. An efficient optimal planning and control framework for quadrupedal locomotion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 93–100. IEEE, 2017.

[56] Pedro López-Adeva Fernández-Layos and Luis FS Merchante. Convex body collision detection using the signed distance function. *Computer-Aided Design*, 170:103685, 2024.

[57] Daniel Fiser. Libccd github. `https://github.com/danfis/libccd`. Accessed: 2024-07-15.

[58] Hélene Frankowska and Sławomir Plaskacz. Semicontinuous solutions of hamilton–jacobi–bellman equations with degenerate state constraints. *Journal of mathematical analysis and applications*, 251(2):818–838, 2000.

[59] Hélene Frankowska and Richard B Vinter. Existence of neighboring feasible trajectories: applications to dynamic programming for state-constrained optimal control problems. *Journal of Optimization Theory and Applications*, 104:20–40, 2000.

[60] Matthias Gerdts. *Optimal control of ODEs and DAEs*. Walter de Gruyter GmbH & Co KG, 2023.

[61] Matthias Gerdts. A mechanical multibody model for a satellite with a mounted robotic arm. Technical report, Engineering Mathematics Group, University of the Bundeswehr Munich, 2024.

[62] Matthias Gerdts, René Henrion, Dietmar Hömberg, and Chantal Landry. Path planning and collision avoidance for robots. *Numerical Algebra, Control and Optimization*, 2(3):437–463, 2012.



[63] Matthias Gerdts and Frank Lempio. *Mathematische Optimierungsverfahren des Operations Research*. Walter de Gruyter, 2011.

[64] 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*, 1(1):21–30, 1985.

[65] Elmer G Gilbert, Daniel W Johnson, and S Sathiya Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal on Robotics and Automation*, 4(2):193–203, 1988.

[66] Philip E Gill, Walter Murray, and Michael A Saunders. Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM review*, 47(1):99–131, 2005.

[67] Stefan Gottschalk, Ming C Lin, and Dinesh Manocha. Obbtree: A hierarchical structure for rapid interference detection. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 171–180, 1996.

[68] Lars Grüne. An adaptive grid scheme for the discrete hamilton-jacobi-bellman equation. *Numerische Mathematik*, 75:319–337, 1997.

[69] Yan Gu, Yong He, Kayvon Fatahalian, and Guy Blelloch. Efficient bvh construction via approximate agglomerative clustering. In *Proceedings of the 5th High-Performance Graphics Conference*, pages 81–88, 2013.

[70] James Guthrie. A differentiable signed distance representation for continuous collision avoidance in optimization-based motion planning. In *2022 IEEE 61st Conference on Decision and Control (CDC)*, pages 7214–7221. IEEE, 2022.

[71] James Guthrie, Marin Kobilarov, and Enrique Mallada. Closed-form minkowski sum approximations for efficient optimization-based collision avoidance. In *2022 American Control Conference (ACC)*, pages 3857–3864. IEEE, 2022.

[72] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[73] Simon Helling and Thomas Meurer. Dual collision detection in model predictive control including culling techniques. *IEEE Transactions on Control Systems Technology*, 31(6):2449–2464, 2023.

[74] Dominik Henrich, Christian Wurll, and Heinz Worn. Online path planning with optimal c-space discretization. In *Proceedings. 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems. Innovations in Theory, Practice and Applications (Cat. No. 98CH36190)*, volume 3, pages 1479–1484. IEEE, 1998.



[75] Ben Hermans, Andreas Themelis, and Panagiotis Patrinos. QPALM: a proximal augmented lagrangian method for nonconvex quadratic programs. *Mathematical Programming Computation*, 14(3):497–541, 2022.

[76] Taylor A Howell, Brian E Jackson, and Zachary Manchester. Altro: A fast solver for constrained trajectory optimization. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7674–7679. IEEE, 2019.

[77] Taylor A Howell, Kevin Tracy, Simon Le Cleac'h, and Zachary Manchester. Calipso: A differentiable solver for trajectory optimization with conic and complementarity constraints. In *The International Symposium of Robotics Research*, pages 504–521. Springer, 2022.

[78] HSL. A collection of fortran codes for large scale scientific computation, 2013.

[79] Andreas Karl-Phillip Huber. *Methoden zur berechnung optimaler rennlinien im dynamischen grenzbereich.* PhD thesis, Dissertation, Neubiberg, Universität der Bundeswehr München, 2020, 2020.

[80] Thomas C Hudson, Ming C Lin, Jonathan Cohen, Stefan Gottschalk, and Dinesh Manocha. V-collide: Accelerated collision detection for vrml. In *Proceedings of the second symposium on Virtual reality modeling language*, pages 117–ff, 1997.

[81] Phillip Hyatt and Marc D Killpack. Real-time nonlinear model predictive control of robots using a graphics processing unit. *IEEE Robotics and Automation Letters*, 5(2):1468–1475, 2020.

[82] Brian Ichter, James Harrison, and Marco Pavone. Learning sampling distributions for robot motion planning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7087–7094. IEEE, 2018.

[83] Hitoshi Ishii and Shigeaki Koike. A new formulation of state constraint problems for first-order pdes. *SIAM Journal on Control and Optimization*, 34(2):554–571, 1996.

[84] Guang-Shan Jiang and Danping Peng. Weighted eno schemes for hamilton–jacobi equations. *SIAM Journal on Scientific computing*, 21(6):2126–2143, 2000.

[85] Pablo Jiménez, Federico Thomas, and Carme Torras. 3d collision detection: a survey. *Computers & Graphics*, 25(2):269–285, 2001.

[86] Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal. Stomp: Stochastic trajectory optimization for motion planning. In *2011 IEEE international conference on robotics and automation*, pages 4569–4574. IEEE, 2011.

[87] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996.



[88] Ali Khajeh-Saeed, Stephen Poole, and J Blair Perot. Acceleration of the smith–waterman algorithm using single and multiple graphics processors. *Journal of Computational Physics*, 229(11):4247–4258, 2010.

[89] Young J. Kim, Ming C. Lin, and Dinesh Manocha. Fast penetration depth estimation using rasterization hardware and hierarchical refinement. *Springer Tracts in Advanced Robotics*, 7 STAR:505 – 521, 2004. Cited by: 3.

[90] Artur Kinzel, J Bachmann, R Jaiswal, M Karnal, ER Novo, F Porcelli, A Schmidt, R Förstner, C Hofmann, A Knopp, et al. Seamless radio access network for internet of space (SeRANIS): New space mission for research, development, and in-orbit demonstration of cutting-edge technologies. In *Proceeding of the 73rd International Astronautical Congress (IAC) Paris, France*, 2022.

[91] Willi Kortüm and Peter Lugner. *Systemdynamik und Regelung von Fahrzeugen: Einführung und Beispiele*. Springer-verlag, 2013.

[92] James J Kuffner and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium conference. IEEE international conference on robotics and automation. Symposia proceedings (Cat. No. 00CH37065)*, volume 2, pages 995–1001. IEEE, 2000.

[93] Benjamin Kuipers, Edward A Feigenbaum, Peter E Hart, and Nils J Nilsson. Shakey: from conception to history. *Ai Magazine*, 38(1):88–103, 2017.

[94] KUKA Laboratories GmbH. KUKA youBot Operating and Assembly Instructions, January 2012.

[95] Jean-Claude Latombe. *Robot motion planning*, volume 124. Springer Science & Business Media, 2012.

[96] Steven LaValle. Rapidly-exploring random trees: A new tool for path planning. *Research Report 9811*, 1998.

[97] Bai Li and Zhijiang Shao. A unified motion planning method for parking an autonomous vehicle in the presence of irregularly placed obstacles. *Knowledge-Based Systems*, 86:11–20, 2015.

[98] Ming C Lin and John F Canny. A fast algorithm for incremental distance calculation. In *ICRA*, volume 91, pages 9–12, 1991.

[99] Daniel S Lopes, Miguel T Silva, Jorge A Ambrósio, and Paulo Flores. A mathematical framework for rigid contact detection between quadric and superquadric surfaces. *Multibody System Dynamics*, 24:255–280, 2010.

[100] Tomas Lozano-Perez. *Spatial planning: A configuration space approach*. Springer, 1990.



[101] Tomás Lozano-Pérez and Michael A Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, 1979.

[102] Thorsten Lüttel. *Konzeption und Aufbau von Autonomen Versuchsfahrzeugen für die Forschung*. PhD thesis, Universität der Bundeswehr München, 2023.

[103] Danylo Malyuta, Taylor P Reynolds, Michael Szmuk, Thomas Lew, Riccardo Bonalli, Marco Pavone, and Behçet Açıkmeşe. Convex optimization for trajectory generation: A tutorial on generating dynamically feasible trajectories reliably and efficiently. *IEEE Control Systems Magazine*, 42(5):40–113, 2022.

[104] Svetlin A Manavski and Giorgio Valle. Cuda compatible gpu cards as efficient hardware accelerators for smith-waterman sequence alignment. *BMC bioinformatics*, 9:1–9, 2008.

[105] Tobia Marcucci, Mark Petersen, David von Wrangel, and Russ Tedrake. Motion planning around obstacles with convex optimization. *Science robotics*, 8(84):eadf7843, 2023.

[106] David Mayne. A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems. *International Journal of Control*, 3(1):85–95, 1966.

[107] Helen Mi, Robert E MacLaren, and Jasmina Cehajic-Kapetanovic. Robotising vitreoretinal surgeries. *Eye*, pages 1–10, 2024.

[108] Brian Mirtich. V-clip: Fast and robust polyhedral collision detection. *ACM Transactions On Graphics (TOG)*, 17(3):177–208, 1998.

[109] Mattia Montanari, Nik Petrinic, and Ettore Barbieri. Improving the gjk algorithm for faster and more reliable distance queries between convex objects. *ACM Transactions on Graphics (TOG)*, 36(3):1–17, 2017.

[110] Louis Montaut, Quentin Le Lidec, Vladimir Petrik, Josef Sivic, and Justin Carpentier. Collision detection accelerated: An optimization perspective. *arXiv preprint arXiv:2205.09663*, 2022.

[111] Remi Munos and Andrew Moore. Variable resolution discretization in optimal control. *Machine learning*, 49:291–323, 2002.

[112] Ramkumar Natarajan, Howie Choset, and Maxim Likhachev. Interleaving graph search and trajectory optimization for aggressive quadrotor flight. *IEEE Robotics and Automation Letters*, 6(3):5357–5364, 2021.

[113] Chong Jin Ong and Elmer G Gilbert. Growth distances: New measures for object separation and penetration. *IEEE Transactions on Robotics and Automation*, 12(6):888–903, 1996.



[114] Stanley Osher and James A Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on hamilton-jacobi formulations. *Journal of computational physics*, 79(1):12–49, 1988.

[115] Stanley Osher and Chi-Wang Shu. High-order essentially nonoscillatory schemes for hamilton–jacobi equations. *SIAM Journal on numerical analysis*, 28(4):907–922, 1991.

[116] Jia Pan, Sachin Chitta, Dinesh Manocha, Florent Lamiraux, Joseph Mirabel, Justin Carpentier, Louis Montaut, et al. Hpp-fcl: an extension of the flexible collision library. https://github.com/humanoid-path-planner/hpp-fcl, 2015–2023.

[117] Suresh K Pandey and Vidushi Sharma. Robotics and ophthalmology: Are we there yet? *Indian Journal of Ophthalmology*, 67(7):988–994, 2019.

[118] Evangelos Papadopoulos, Farhad Aghili, Ou Ma, and Roberto Lampariello. Robotic manipulation and capture in space: A survey. *Frontiers in Robotics and AI*, 8:686723, 2021.

[119] Mattia Piccinini, Simon Gottschalk, Matthias Gerdts, and Francesco Biral. Computationally efficient minimum-time motion primitives for vehicle trajectory planning. *IEEE Open Journal of Intelligent Transportation Systems*, 2024.

[120] Mihail Pivtoraiko and Alonzo Kelly. Kinodynamic motion planning with state lattice motion primitives. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2172–2179. IEEE, 2011.

[121] Brian Plancher and Scott Kuindersma. A performance analysis of parallel differential dynamic programming on a gpu. In *International Workshop on the Algorithmic Foundations of Robotics*, pages 656–672. Springer, 2018.

[122] Marc Quincampoix and Oana-Silvia Serea. A viability approach for optimal control with infimum cost. *Annals. Stiint. Univ. Al. I. Cuza Iasi, sI a, Mat*, 48:113–132, 2002.

[123] Carlos Quintero-Pena, Anastasios Kyrillidis, and Lydia E Kavraki. Robust optimization-based motion planning for high-dof robots under sensing uncertainty. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9724–9730. IEEE, 2021.

[124] Alain E Rapaport. Characterization of barriers of differential games. *Journal of optimization theory and applications*, 97:151–179, 1998.

[125] Karthik Murali Madhavan Rathai, Olivier Sename, and Mazen Alamir. Gpu-based parameterized nmpc scheme for control of half car vehicle with semi-active suspension system. *IEEE Control Systems Letters*, 3(3):631–636, 2019.

[126] Nathan Ratliff, Matt Zucker, J Andrew Bagnell, and Siddhartha Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *2009 IEEE international conference on robotics and automation*, pages 489–494. IEEE, 2009.



[127] Rebecca Richter. Dynamic and nonlinear programming for trajectory planning in moving environments. *IFAC-PapersOnLine*, 59(1):265–270, 2025.

[128] Rebecca Richter, Andreas Britzelmeier, and Matthias Gerdts. An adaptive mesh dynamic programming algorithm for robotic manipulator trajectory planning. In *2023 European Control Conference (ECC)*, pages 1–8. IEEE, 2023.

[129] Martin Roche. The mako robotic-arm knee arthroplasty system. *Archives of orthopaedic and trauma surgery*, pages 1–5, 2021.

[130] Sipu Ruan and Gregory S Chirikjian. Closed-form minkowski sums of convex bodies with smooth positively curved boundaries. *Computer-Aided Design*, 143:103133, 2022.

[131] Sipu Ruan, Karen L Poblete, Yingke Li, Qian Lin, Qianli Ma, and Gregory S Chirikjian. Efficient exact collision detection between ellipsoids and superquadrics via closed-form minkowski sums. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 1765–1771. IEEE, 2019.

[132] Basak Sakcak, Luca Bascetta, Gianni Ferretti, and Maria Prandini. Sampling-based optimal kinodynamic planning with motion primitives. *Autonomous Robots*, 43(7):1715–1732, 2019.

[133] Antonio Sala and Leopoldo Armesto. Adaptive polyhedral meshing for approximate dynamic programming in control. *Engineering Applications of Artificial Intelligence*, 107:104515, 2022.

[134] Philip Schneider and David H Eberly. *Geometric tools for computer graphics*. Elsevier, 2002.

[135] Alex Joachim Ernst Schreiber. *Dynamic programming with radial basis functions and Shepard's method*. PhD thesis, Technische Universität München, 2016.

[136] John Schulman, Yan Duan, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, Jia Pan, Sachin Patil, Ken Goldberg, and Pieter Abbeel. Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research*, 33(9):1251–1270, 2014.

[137] Xu Shen, Edward L Zhu, Yvonne R Stürz, and Francesco Borrelli. Collision avoidance in tightly-constrained environments without coordination: a hierarchical control approach. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2674–2680. IEEE, 2021.

[138] Gary Snethen. Xenocollide: Complex collision made simple. In *Game Programming Gems 7*, pages 165–178. Course Technology, 2008.

[139] Halil Mete Soner. Optimal control with state-space constraint I. *SIAM Journal on Control and Optimization*, 24(3):552–561, 1986.



[140] Panagiotis E Souganidis. Approximation schemes for viscosity solutions of hamilton-jacobi equations. *Journal of differential equations*, 59(1):1–43, 1985.

[141] Ioan A Sucan, Mark Moll, and Lydia E Kavraki. The open motion planning library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, 2012.

[142] Yuval Tassa, Tom Erez, and Emanuel Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4906–4913. IEEE, 2012.

[143] Yuval Tassa, Nicolas Mansard, and Emo Todorov. Control-limited differential dynamic programming. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1168–1175. IEEE, 2014.

[144] MoveIt Team. Moveit 2 documentation: Humble, Year.

[145] Gino Van Den Bergen. Proximity queries and penetration depth computation on 3d game objects. In *Game developers conference*, volume 170, page 209, 2001.

[146] Jörn Vogel, Annette Hagengruber, Maged Iskandar, Gabriel Quere, Ulrike Leipscher, Samuel Bustamante, Alexander Dietrich, Hannes Höppner, Daniel Leidner, and Alin Albu-Schäffer. Edan: An emg-controlled daily assistant to help people with physical disabilities. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4183–4190. IEEE, 2020.

[147] Andreas Wächter and Lorenz T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106:25–57, 2006.

[148] Baicun Wang, S Jack Hu, Lei Sun, and Theodor Freiheit. Intelligent welding system technologies: State-of-the-art review and perspectives. *Journal of Manufacturing Systems*, 56:373–391, 2020.

[149] Wenping Wang, Jiaye Wang, and Myung-Soo Kim. An algebraic condition for the separation of two ellipsoids. *Computer aided geometric design*, 18(6):531–539, 2001.

[150] Christophe Weibel. *Minkowski sums of polytopes: combinatorics and computation*. PhD thesis, EPFL, 2007.

[151] N-E Wiberg and XD Li. A postprocessed error estimate and an adaptive procedure for the semidiscrete finite element method in dynamic analysis. *International Journal for numerical methods in engineering*, 37(21):3585–3603, 1994.

[152] Alexander W. Winkler. Ifopt - a modern, light-weight, Eigen-based C++ interface to nonlinear programming solvers IPOPT and SNOPT., 2018.



[153] Chongkun Xia, Yunzhou Zhang, and I-Ming Chen. Learning sampling distribution for motion planning with local reconstruction-based self-organizing incremental neural network. *Neural Computing and Applications*, 31:9185–9205, 2019.

[154] Zhaoming Xie, C Karen Liu, and Kris Hauser. Differential dynamic programming with nonlinear constraints. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 695–702. IEEE, 2017.

[155] Saïd Zeghloul, Patrick Rambeaud, and Jean-Paul Lallemand. A fast distance calculation between convex objects by optimization approach. In *Proceedings 1992 IEEE International Conference on Robotics and Automation*, pages 2520–2521. IEEE Computer Society, 1992.

[156] Jun Zeng, Prasanth Kotaru, Mark W Mueller, and Koushil Sreenath. Differential flatness based path planning with direct collocation on hybrid modes for a quadrotor with a cable-suspended payload. *IEEE Robotics and Automation Letters*, 5(2):3074–3081, 2020.

[157] Xiaojing Zhang, Alexander Liniger, and Francesco Borrelli. Optimization-based collision avoidance. *IEEE Transactions on Control Systems Technology*, 29(3):972–983, 2020.

[158] Yong-Tao Zhang and Chi-Wang Shu. High-order weno schemes for hamilton–jacobi equations on triangular meshes. *SIAM Journal on Scientific Computing*, 24(3):1005–1030, 2003.

[159] Yu Zheng and Katsu Yamane. Ray-shooting algorithms for robotics. *IEEE Transactions on Automation Science and Engineering*, 10(4):862–874, 2013.

