AUTOMATION & CONTROL INSTITUTE
INSTITUT FÜR AUTOMATISIERUNGS- & REGELUNGSTECHNIK

Optimal Local Path-Planning and Control for Mobile Robotics

MASTER THESIS

Conducted in partial fulfillment of the requirements for the degree of a

Diplom-Ingenieur (Dipl.-Ing.)

supervised by

Dr. Markus Bader
Assoc. Prof. Dr. Wolfgang Kemmetmüller

submitted at the

TU Wien
Faculty of Electrical Engineering and Information Technology
Automation and Control Institute

by
Horatiu George Todoran
Matriculation number 1128205
Firmiangasse 34/8
1130 Vienna
Austria

Vienna, February 2018

Abstract

Autonomous navigation of mobile robots represents a challenging task in the field of robotics. This is especially the case when accounting for generic, non-trivial robot dynamics, unstructured, possibly dynamic environments, realistic sensoric assumptions as well as the real-time computation requirements.

The thesis addresses this navigation problem by means of Moving Horizon Trajectory Planning (MHTP), an approach that based on dynamic optimization, lying at the boundary between the topics of path planning and control.

In such methods, the declarative formulation of the optimization problem is of great importance, as such formulations are succinct, expressive, posses theoretically provable characteristics and result in powerful behaviours. Moreover, they allow a safety analysis of the navigation approach, resulting in formulations of the navigation strategy that can guarantee that the agent will never collide with the environment. The discussed assumptions regarding the environment characteristics range from the simple known, static environment model up to environment models that assume non-trivial sensoric capabilities (allow sensing only in non-occluded surfaces that are within FOV and maximum sensing distances) as well as possessing arbitrary dynamic models with bounded uncertainty.

Special care and focus is given also in the practical implementation of the discussed approaches. To this end, an efficient representation of the relaxed optimization problem is being presented, the minimal parametric representation. Non-trivial, PDE-induced metrics used to encode the cost-function of the optimization problem are analysed, aiming for a convexification of the formulation, with the practical benefits of removing local minima and thus guaranteeing that the agent is always reaching its navigation goal. Moreover, the implementation of such methods is addressed assuming an asynchronous system, presenting synchronization approaches and coupling strategies with the other modules within the system such as state observers and lower-level controllers.

Simulated and real-world experimental results are presented on three different autonomous platforms (differential drive, independent wheel steering drive as well as a race car) operating in various environment types, illustrating the quality of the proposed approaches.

Kurzzusammenfassung

Das autonome Navigieren von mobilen Roboter ist eine komplexe und herausfordernde Aufgabe in der Robotik. Dies gilt insbesondere, wenn man einen Roboter berücksichtigt, der eine allgemeine, nicht triviale Dynamik aufweist, der mit realer, fehlerbehafteter Sensorik ausgestattet ist, der in einer unstrukturierten, möglicherweise sogar dynamischen Umgebung betrieben wird, und der zusätzliche Echtzeit-Systemanforderungen erfüllen soll.

Diese Probleme werden in dieser Arbeit mittels einer Moving Horizon Trajektorienplanung (MHTP) behandelt. Dies ist eine Methode, die auf dynamischer Optimierung basiert und mit Themen der Pfadplanung und -regelung verwandt ist.

Für solche Methoden ist eine deklarative Formulierung des Optimierungsproblems von grösster Bedeutung, da dies eine prägnante, aussagekräftige und theoretisch beweisbare Beschreibung des Problems ermöglicht. Zugleich stellt dieser Navigationsansatz sicher, dass jegliche mögliche Kollision des Roboters mit seiner Umgebung vermieden wird. Der vorgestellte Ansatz begünstigt hierbei ein breites Einsatzgebiet - von einfachen statischen Umgebungen bis zu komplexen Umgebungsmodellen mit nicht-trivialen Sensoreigenschaften (ermöglichen nur die Erfassung von nicht okkludierten Oberflächen, die sich innerhalb des FOV und der maximalen Messabständen befinden) und Modellen, die beliebige Dynamik und beschränkte Unsicherheit aufweisen.

Spezielle Aufmerksamkeit wurde in dieser Arbeit zudem der praktischen Umsetzung gewidmet. Es wird eine effiziente Repräsentation des Optimierungsproblems vorgestellt, die Minimale Parametrische Repräsentation. Nicht-triviale, PDE-induzierte Metriken werden verwendet um eine Kostenfunktion des Optimierungsproblems zu analysieren. Diese zielen auf eine konvexe Formulierung des Problems ab, die es ermöglicht, lokale Minima zu beseitigen und somit zu garantieren, dass der Roboter stets sein Navigationsziel erreicht. Unter Annahme eines asynchronen Systems werden die Methoden mittels Synchronisierungsansätzen und Kopplungsstrategien mit anderen Modulen des Systems wie Zustandsbeobachter und Low-Level Regler implementiert.

Die vorgestellten Methoden werden sowohl in simulierten als auch in realen Experimenten unter verschiedenen Umgebungsbedingungen und auf drei verschiedenen autonomen Plattformen (Differentialantrieb, unabhängiger Lenkantrieb, und Rennauto) evaluiert und ausführlich diskutiert.

Contents

List of Figures V

List of Examples VII

1 Introduction 1
1.1 State of Art . . . 3

2 General Definitions and Models 6
2.1 Classes of Dynamic Models . . . 6
2.1.1 Non-linear Models . . . 6
2.1.2 Affine-Input Models . . . 14
2.1.3 Linear Time-Variant Models . . . 15
2.2 Parametric Functions . . . 15
2.2.1 Piece-wise Constant . . . 16
2.2.2 Piece-wise Linear . . . 18
2.2.3 Polynomials . . . 21
2.2.4 Splines . . . 22
2.3 Solving ODEs . . . 24
2.3.1 Single-step Methods . . . 25
2.3.2 Multi-Step Methods . . . 30
2.4 Computing Sensitivities of ODEs . . . 31
2.4.1 Numeric Differences . . . 31
2.4.2 Analytic . . . 32

3 Optimal Planning and Control 37
3.1 Definitions and Models . . . 37
3.1.1 Dynamic Models . . . 37
3.1.2 Constraints . . . 38
3.1.3 Cost Function . . . 39
3.2 Optimization-based Control . . . 40
3.2.1 Infinite Optimization Horizon, Perfect Environment Model . . . 40
3.2.2 Finite Optimization Horizon, Perfect Environment Model . . . 41
3.2.3 Classification of Imperfect Environment Models . . . 42
3.3 Constraints for Ensuring Safety . . . 44
3.3.1 Notation and Related Concepts . . . 44
3.3.2 Collision-free Navigation Constraints . . . 46

4 Implementation of MHTP **50**
4.1 Discretizing the Optimization Problem . . . 51
4.1.1 Fully Discretized Representation . . . 51
4.1.2 Minimal Parametric Representation . . . 54
4.1.3 Evaluation Lattices . . . 55
4.1.4 Solving the Discretized Optimization Problem . . . 58
4.2 Gradient-based Non-Linear Programming . . . 60
4.2.1 Solver Algorithms . . . 60
4.2.2 Generic Constraint Modelling . . . 61
4.2.3 Preconditioning . . . 69
4.3 Initial Solutions . . . 70
4.4 Temporal Synchronization . . . 72

5 A Suitable Cost Function **75**
5.1 Reaching a Goal . . . 75
5.1.1 Metrics and Norms . . . 76
5.1.2 PDE Candidates . . . 78
5.1.3 Sensor-processing: Layered Local Maps . . . 83
5.2 Other Navigation Objectives . . . 85

6 Experimental Results **87**
6.1 General Considerations . . . 87
6.2 Differential-Drive: Navigation in Human-Shared Environments . . . 89
6.2.1 Navigation in Partially-Mapped Static Environments . . . 91
6.2.2 Navigation using Safety Constraints . . . 92
6.2.3 MPC vs Stabilized MHTP Comparison . . . 95
6.2.4 Using State Observers . . . 101
6.2.5 Real-Robot Testing . . . 107
6.3 Ackerman-Drive: Navigation of the TU Autonomous Race-Car . . . 109
6.4 IWS-Drive: Navigation with Controlled Torso-Orientation . . . 111

7 Conclusions **115**
7.1 Future Work . . . 116

List of Figures

2.1 Differential-drive geometric model 8
2.2 Ackerman-drive geometric model 9
2.3 I4WS geometric model 10
2.4 I4WS global frame considerations 11
2.5 The single-track dynamic model 12
2.6 A 1-dimensional piece-wise constant function $\omega_b(t)$ 17
2.7 Visualization of a pixel-map 18
2.8 Piece-wise linear encoding of kinematic inputs for a differential-drive . . . 19
2.9 Visualization of the bi-linear interpolation 20
2.10 Visualization of the bi-linear interpolation 20
2.11 Unconstrained pose-to-pose trajectories using polynomial fitting 21
2.12 Illustration of the gradient computation using the mid-point method . . . 27
2.13 Comparison of various discretization schemes used for numerical integration 28

3.1 Illustration of the emerging trajectories of MPC and Stabilised MHTP . . 42
3.2 Nominal versus guaranteed free space for several future simulated time-steps 45
3.3 Nominal and emergency trajectories when enforcing the safety constraints 49
3.4 Agent and env. simulations timeline when enforcing the safety constraints 49

4.1 Bounds of the discretized distance constraint 53
4.2 Trajectory parametrization with different lattice points 57
4.3 Nominal and guaranteed free space for Random-Walk Environment Models 67
4.4 Precomputed trajectories towards a feasible-invariant state 68
4.5 Control-space vs. state-space trajectory sampling 71

5.1 Local-minima when minimizing the Euclidean Distance 77
5.2 Different Distance Fields 81
5.3 Solutions of the Eikonal equation with velocity damping 82

6.1 Robotic Platform *Pioneer-3DX* 89
6.2 State-Machine for the Differential-Drive Navigation Module 90
6.3 Resulting optimized trajectory takes a shortcut in the environment 92
6.4 Temporal sequence of MPC Controller when avoiding unmapped obstacle 93
6.5 Evaluation of different safety constraints during a cornering manoeuvre . 94
6.6 Navigation under uncertainty 95
6.7 Testing scenario and computed global path 97
6.8 Trajectories of the agent using MPC and SMHTP with a Lyapunov Controller 98

6.9 Distance to obstacles, linear and angular velocities along the MPC trajectory 98
6.10 Distance to obstacles, linear and angular velocities along the SMHTP trajectory . . . 99
6.11 Trajectory of the MPC under modelling errors . . . 99
6.12 Trajectory of the SMHTP under modelling errors . . . 100
6.13 Linear and angular velocities along the SMHTP trajectory under modelling errors . . . 100
6.14 Diagram illustrating the observers structure . . . 101
6.15 Estimated Pose of the MPC vs Ground-Truth Pose . . . 105
6.16 Ground-truth trajectories of MPC and SMHTP using state estimation under modelling errors . . . 106
6.17 Adaptation of the observed model parameters as well as linear and angular velocities along the SMHTP trajectory . . . 106
6.18 Resulting trajectory of the real platform in partially-mapped static environments . . . 107
6.19 Linear and angular velocity commands along the real-robot trajectory in static environment . . . 108
6.20 Resulting trajectory of the real platform navigating near humans . . . 108
6.21 Formula-Student Race-Car *Edge8* . . . 109
6.22 Resulting trajectory in a closed-circuit . . . 110
6.23 Resulting linear velocity and steering angle in a closed-circuit . . . 110
6.24 Robotic Platform *Blue* . . . 111
6.25 Visualization of an ICC trajectory . . . 112
6.26 Generated trajectories under different dominating weights . . . 113

List of Examples

2.1 Differential-drive kinematic model 8
2.2 Ackerman-drive kinematic model 9
2.3 Independent wheel steering (IWS) kinematic model 9
2.4 Single-track dynamic model 12
2.5 From non-linear to input-affine model 14
2.6 Dense encoding of system input values 17
2.7 Pixel maps 17
2.8 Piece-wise linear representation of kinematic inputs 18
2.9 Continuous pixel maps 20
2.10 Unconstrained trajectory generation between two states 21
2.11 Mid-point method 26
2.12 Comparison of different explicit methods for a differential-drive model 27
2.13 Differential-Drive ODE Sensitivities using piece-wise linear kinematic inputs . 33
4.1 Discretizing the Distance-to-Obstacles Constraint for a circular agent 52
4.2 Evaluation Lattices of kinematic models 56
4.3 Kinematic Constraints: Differential Drive 63
4.4 Collision-avoidance Constraints: Random-Walk Environment Model 64
4.5 Safety Constraints: Static-Known Environment Model 65
4.6 Safety Constraints: Random-Walk Environment Model 66
6.1 State-Machine for the Differential-Drive Navigation Module 90
6.2 Lyapunov-based Trajectory-Following Control 95
6.3 Input-Output Linearization Trajectory-Following Control 96
6.4 Kalman Filter for State and Parameter Estimation 102

1 Introduction

Autonomous Navigation is a field of research that has recently started to be applied in a variety of domains, ranging from industrial applications where fleets of agents perform transportation tasks, the field of assistive robotics in which fetch-and-carry tasks require autonomous navigation, as well as the field of autonomous vehicle driving.

Navigation in unstructured environments represents a fundamental task for autonomous agents. A request stated as simple as "going from A to B" has proven to be very difficult to solve when non-trivial agent (and possibly environment) dynamics as well as constraints are taken into account. Due to the complexity of this task, the approaches are typically split into the research sub-topics of of Routing, Global Path-Planning, Localization, Local Path-Planning and/or Low-Level Control. The above-mentioned topics can be introduced by utilizing an intuitive analogy: a trip that an insurance employee would perform with a vehicle fulfilling a job-required task.

Routing Globally, suppose that the company has to coordinate a considerable number of employees trips. Because of this, the coordination happens at high-level, such that the company only tells every employee a required *route*, i. e. key locations (*checkpoints*) where they have to be present in certain time intervals. Of course, at this level, the coordination is desired to optimize the trips requested to the employees, such that e.g. only one employee visits a location, not more employees require the same vehicle at the same time. In the context of autonomous navigation, this relates to Routing problems, which are in general considered as a separate (high-level) module that typically coordinates *multiple* autonomous agents. Its typical frequencies are in the order of 0.1 Hz.

Global Path-Planning Given the route of the day, one of the high-level tasks of each employee is to come up with the *path* of the day, such that the checkpoints of the route required by the company are achieved. At this point, the employee can perform improvements by optimizing the order in which the checkpoints are reached, the taken roads and high-ways, potentially accounting for some coarse live-traffic data. Returning from the analogy, the Global Path-Planning module typically accounts for computing global paths of each agent. Such paths are usually computed using discrete algorithms (graph search methods such as Dijkstra or A*) and do not account for the dynamics of the platform and environment. Different approaches expand this module with more or less functionality from other modules. For example, higher-level functionality is achieved by computing global paths for multiple agents in the same algorithm, optimizing thus for "traffic jams" within the coordinated fleet. Approaches that provide such functionality

relate to the topic of Multi-Agent-Planning, including algorithms such as reactive planning [1] or prioritized planning methods [2]. Typical frequencies of this modules are on the order of 1 Hz.

Localization Nowadays, it is wide-spread for drivers to make use of a GPS module. This allows the driver to *localize* itself in a global reference frame and more importantly make use of this information to be able to plan global paths for its trip. However, prior to the appearance of the GPS technology, drivers had the more complicated task of identifying their location by perception of their local neighbourhood and search on a static *map*. In the context of Navigation, Localization represents a key capability that the system has to possess. As in the old driving-days, in many practical (indoor) applications, GPS signals are not available or not sufficiently accurate. Thus, environment maps are generated using Simultaneous Localization and Mapping (SLAM) algorithms, by means of particle-filtering [3] or graph-optimization techniques [4]. Afterwards, the generated maps and local environment perception are used in Localization algorithms. The current preferred underlying concept for solving the Localization problem are Particle-Filters [5]. Typical frequencies of this modules are in the order of 10 Hz.

Local Path-Planning Moving towards actually driving the car, the task of the driver would be to control the vehicle locally, making decisions in dependence with the topology and dynamics of its immediate neighbourhood, such as changing lanes, stopping at traffic lights, avoiding collisions with other cars etc. Note that here, the dynamics of the car is taken into account (e.g. the driver knows that the car requires some distance to stop or that it cannot take a sharp turn at high velocities). Moreover, the driver has to account for the dynamics of the environment, i. e. driving between other cars or stopping when a pedestrian crosses the street. Moreover, the planned *trajectory* is relatively short (*local*), i. e. the driver does not plan how to turn the steering wheel after the next turn, or it does not consider that maybe in 2 km distance a pedestrian might cross the road. In Autonomous Navigation, the above-motivated capabilities are typically bundled in Local Path-Planning. Typical frequencies such modules are in the order of $10 - 50$ Hz.

Low-Level Control Even though the driver coordinates the motion of the vehicle, the vehicle itself is a controlled system, mapping the driver inputs (steering wheel angle and pedals) to actuator inputs that physically put the mechanics in motion. This relates to Low-Level Control, approaches that control sub-parts of the physical system, running typically at frequencies ≥ 100 Hz.

The work presented in this thesis addresses methodology and algorithms that mostly relate to the Local Path-Planning module in the Navigation ecosystem. The initial motivation of starting at this layer is the fact that it is still sufficiently low-level to account for challenging dynamics of the platform and(or) environment, but sufficiently high-level to allow non-trivial (and thus relatively computationally-intensive) algorithms

to be applied. Given those considerations, the fundamental aims of this work can be summarized as:

- obtaining *optimal* agent trajectories while systematically accounting for various constraints
- *guaranteed* collision-avoidance by design through generic formal analysis
- *generic* (dynamic) environment modelling
- *asynchronous* interfacing with Low-Level Control modules
- *reduced requirements* on the capabilities and functionality of the higher-level Navigation Modules
- *robustness*
- *mathematical generality* (scalability)
- *encapsulation* from higher-level robotics tasks (such as human-robot interaction, grasping, etc.)
- *fast and modular libraries* implementing the above

1.1 State of Art

Offline computational methods for finding feasible navigation paths in dynamic environments have been proposed. In such approaches, the trajectory is calculated before the motion begins [6]. Even though such approaches scale well with respect to the length of the planned trajectories, we believe they are not specifically suited for agents that are expected to navigate fast, efficiently and in environments that do not have accurate models (unstructured). This motivates us to approach the problem from a receding horizon perspective, where system dynamics is explicitly taken into account. This relates to the generic term of Moving Horizon Trajectory Planning (MHTP).

The paradigm behind MHTP is that by having knowledge about the agent model, one can predict sufficiently accurate outcomes of different commands over a larger *horizon* into the future. This results in the capability of the controller to maintain the satisfiability of the imposed constraints as well as to induce an *optimized* trajectory with respect to some user-definable cost. This process is performed periodically and at each iteration, the agent is applying only the initial controls of the planned trajectory.

A closely related approach to MHTP is the Model Predictive Control (MPC), a thorough introduction towards MPC in the context of autonomous navigation being given in [7]. The main difference between the two approaches is that MPC provides the feed-back character through its repeated iteration, while MHTP is usually used in open-loop. Here, we propose an extended approach that provides the feedback-character of MPC, the Stabilized MHTP (SMHTP).

Even though sampling techniques such as the Dynamic Window Approach [8] exist for searching for a solution in the context of MHTP, we focus ourselves on the approach in which the planned trajectory is optimized. Similar with fundamental approaches used in autonomous-driving contexts [9], this motivates the theoretical formulation of the system as a dynamic optimization problem [10], in practice being discretized and solved using a non-linear optimization solver. Here, the presented work approaches the concepts related to dynamic optimization problems in-depth, proposing various improvements and formulations ranging from Ordinary Differential Equations (ODE) solvers, sensitivity computation as well as dynamic optimization problem parametrizations and discretization. Moreover, the convexity of the resulting static optimization problem is addressed, proposing the usage of non-trivial metrics such as the solution of the Eikonal equation [11] that can lead to *local-minima-free* optimization programs.

Even though a lot of research has been done towards autonomous driving and mobile robotics navigation [7, 12], the industrial market such as the logistic industry still prefers Automated Guided Vehicles (AGV's) over autonomous vehicles due to their simplicity and more importantly due to their certified safe behaviour [13]. However, AGV systems are only economical in predictable, structured environments – work-spaces where only trained humans are present. The certified safety issue is resolved by using devices such as a special laser range scanner which is directly connected to the wheel encoders and the braking/motor control system. If an obstacle appears within a predefined safety area of such a safety device, the vehicle will slow down or stop on the currently planned trajectory. Of course, an intelligent safety sensor is able to scale and transform the safety areas according to the current vehicle speed and angular velocity; nevertheless, in a case of a safety violation, the system will slow down the vehicle on the given trajectory or trigger a stop/hold. The proposed approach towards safe navigation differs: the notion of safety relates to the vehicle capability to deviate from its planned trajectory towards a safe location. Therefore, our system is able to navigate at higher speeds through narrow passages, as it enforces the existence of emergency trajectories candidates along the entire planned trajectory.

It is well known that for moving-horizon controllers, even though a solution exists for iteration k, it is in general not guaranteed for a solution to exist at iteration $k+1$ [14]. This statement is valid even in the simple case in which the models of the system are exact. However, to ensure safety, one has to account additionally for the uncertainties of the system models and numerical methods. In some approaches uncertainty is being dealt with by not assuming hard constraints but rather solving a stochastic optimization problem [15]. However, such methods prove to be relatively empirical to mathematically describe (no explicit constraint modelling), computationally expensive to solve ([15] requires a high-end GPU for online computation) and difficult to formally analyse when interested in safety-proofs. Alternatively, Robust MPC [16, 17] is known in the literature as an extension to classical MPC by assuming a "worst-case" evolution of the system given bound uncertainties and constraining the allowed motions accordingly. This motivates the search of a formulation that when applied as an inequality constraint in the optimization problem would always guarantee non-collision. In his work, Schouwenaars made use of the concept of feasible invariant states: states of the agent that are guaranteed by

assumption to be collision free indefinitely. For example, a state in which the robot stops can be considered such a state. However, some platforms (such as fixed-wing UAVs) cannot maintain such a state. In a similar fashion, [18] makes use of the dual concept of feasible invariant states: inevitable collision states. In [19], Schouwenaars has extended the concept of feasible invariant state to allow periodic sequences of states (such as loiter manoeuvres of a fixed wing flying vehicle). Other approaches propose to compute the feasible invariant set of agent states explicitly. While computationally inexpensive during navigation, such approaches have, however, the strong limitation that they require an accurate a priori knowledge of the environment (requiring most of the cases for it to be static). In contrast to them, our safe-navigation formulation does not assume that the environment is static. Also, most works assume that the environment is known, or the known region is simply a circular space centred on the agent. We directly approach this by taking into account the intrinsic method of visual sensing: the line of sight. Moreover, we propose a formulation that addresses the typical problem encountered in Robust MPC formulations when accounting for environment dynamics: the uncertainty of the environment state increases quickly during forward-prediction, constraining the planning horizon of the trajectory generator.

This thesis is organized as follows: Chapter 2 introduces generic concepts related to dynamic models, parametrizations as well as solving ODEs and their sensitivities. Chapter 3 addresses (continuous) dynamic optimizations, their usage in developing feed-back controllers as well as the constraint formulations that guarantee navigation safety. Considerations regarding the practical implementation of dynamic optimization-based feed-back controllers are given in Chapter 4. Chapter 5 analyses the discretized optimization problem in terms of convexity and proposes a suitable cost-function that encodes the fundamental task of reaching a goal. Chapter 6 presents simulated and real-platform results, validating the proposed methods using three different autonomous platforms. Conclusions are drawn in Chapter 7.

Remark: The main content of this thesis has been written having *generality* in mind. To this end, the application of the presented content to the evaluated platforms is spread throughout all the thesis chapters, structured as examples.

2 General Definitions and Models

2.1 Classes of Dynamic Models

The dynamics of a large class of systems can be described and modelled through the use of (ordinary) differential equations. Such models are typically the representation of choice in the context of control theory. Here, one can distinguish between the analysis of such systems in the Laplace domain (applicable when dealing with linear systems) or in the time domain using the state-space model. As the state-space model is considerably more general and provides better tools of analysis and design for MIMO as well as non-linear systems, it will be the representation of choice throughout this work.

Remark: Any higher-order differential equation can be represented in a state space model. For example, the ODE:

$$a\ddot{x} + b\dot{x} + cx + d = 0, \quad x(0) = x_0, \dot{x}(0) = \dot{x}_0 \tag{2.1}$$

can be written in a state-space representation as:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ -\frac{bx_2+cx_1+d}{a} \end{bmatrix}, \quad x_1(0) = x_0, x_2(0) = \dot{x}_0 \tag{2.2}$$

In the following, a general continuous-time non-linear state-space model will be presented, along with several structure-exploiting simplifications of it.

2.1.1 Non-linear Models

In a very general form, the evolution of a dynamic system with an n-dimensional state $\mathbf{x} \in \mathbb{R}^n$ can be described as

$$\dot{\mathbf{x}}(t) = \mathbf{f}_{\mathbf{x}}(\mathbf{x}(t), t) \ , \quad \mathbf{x}(t_0) = \mathbf{x}_0 \tag{2.3}$$

with $\mathbf{x}(t)$ a vector-valued function representing the state $\mathbf{x}$ at time t as well as $\mathbf{f}_{\mathbf{x}}(\mathbf{x}(t), t)$ a vector-valued function describing the temporal derivative of the state $\mathbf{x}$. The explicit temporal dependency of $\mathbf{f}_{\mathbf{x}}$ is a very general formulation as it can be used to describe a large class of structures. For example, an ODE that depends on a vector-valued input function (denoted as $\mathbf{u}(t)$) falls under such a notation, i. e. $\mathbf{f}_{\mathbf{x}}(\mathbf{x}(t), t) = \mathbf{f}'_{\mathbf{x}}(\mathbf{x}(t), \mathbf{u}(t))$.

Note that systems modelled by (2.3) have a unique solution given the initial condition t_0 and $\mathbf{x}(t_0) = \mathbf{x_0}$, as long as the function $\mathbf{f_x}(\cdot)$ is reasonable[1].

In the case in which the initial condition $\mathbf{x}(t_0)$ as well as the state transition function $\mathbf{f_x}$ are exact, the solution of (2.3) represents exactly the dynamic evolution of the system. However, in real-world applications, $\mathbf{f_x}$ as well as $\mathbf{x}_0$ are not exactly known, but rather only an approximation of them ($\hat{\mathbf{f}}_\mathbf{x}$ and $\hat{\mathbf{x}}_0$, respectively). In order to describe the system (2.3) with those approximations, a correcting initial condition and time-varying function are needed:

$$\dot{\mathbf{x}}(t) = \hat{\mathbf{f}}_\mathbf{x}(\mathbf{x}(t), t) + \mathbf{w}_\mathbf{x}(t) \ , \quad \mathbf{x}(t_0) = \hat{\mathbf{x}}_0 + \mathbf{w}_{\mathbf{x}_0}. \tag{2.4}$$

The term $\mathbf{w}_{\mathbf{x}_0}$ can be interpreted as an observation error from the exact initial condition, while $\mathbf{w}_\mathbf{x}(t)$ could account for all the uncertainties and errors that occur in the system. In many approaches aiming to account for the non-explicitly modelled part of the system, a generic assumption regarding $\mathbf{w}_\mathbf{x}(t)$ relates to it being a stochastic variable subject to Brownian motion. However, the analysis and calculus required for exploiting the structure of such an assumption becomes considerably more difficult, as classical calculus cannot be directly used[2].

A more restricting assumption that leads to considerably simpler calculus is to assume that the $\mathbf{w}$ terms in (2.4) are stochastic variables with normal distribution. This results in the case of linear (or locally linearised) differential equations to a closed-form manipulation of the stochastic part of the system. This has been widely exploited in the literature especially on the topic of state observers; however, designing controllers that account for such assumptions is still a topic of research.

However, the easiest method to deal with the uncertainties is to neglect them. As long as the uncertainties are sufficiently small (i. e. the system model is sufficiently accurate), such a simple approach has shown to provide a good trade-off between accuracy, design and analysis complexity as well as algorithmic (runtime) complexity of control.

The nominal model of the system (2.3) is

$$\dot{\hat{\mathbf{x}}}(t) = \hat{\mathbf{f}}_\mathbf{x}(\hat{\mathbf{x}}(t), \mathbf{u}(t), t) \ , \quad \hat{\mathbf{x}}(t_0) = \hat{\mathbf{x}}_0. \tag{2.5}$$

Most of the attention of this work will be towards the nominal systems, addressing however the model uncertainties as a bound of the system ODE solutions when discussing about navigation safety.

[1] The (local) Lipschitz condition is a sufficient condition for the (local) uniqueness of the solution of an ODE.

[2] Due to the quadratic variation of the Brownian motion, certain 2nd order terms of the Taylor expansion have to be accounted for in differential calculus. For more details, see Ito's Lemma [20].

Example 2.1 (Differential-drive kinematic model). One of the most common mechanical designs of mobile platforms is the differential drive. The platform possesses two parallel wheels whose revolution can be independently actuated, as presented in Figure 2.1. Here, the platform pose is defined by x, y, θ, the wheels displacement by d, the wheels radius is denoted by r_w, as well as the linear and angular velocities in the local frame v and ω, respectively.

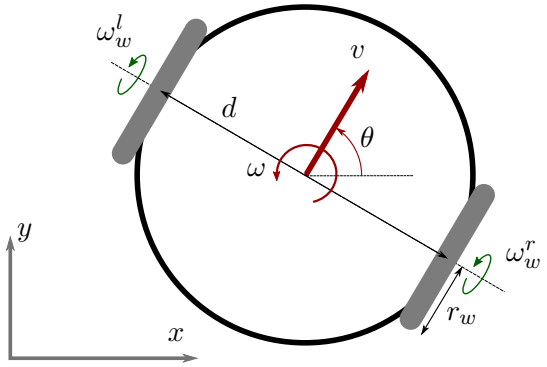


Figure 2.1: Differential-drive geometric model

Assuming a perfect-rolling motion, from geometrical considerations we have

$$v = r_w \frac{\omega_w^r + \omega_w^l}{2}, \quad \omega = r_w \frac{\omega_w^r - \omega_w^l}{d}. \tag{2.6}$$

Note that the relationship between the wheels angular velocities (ω_w^r and ω_w^l, respectively) and the chassis linear and angular velocity (v and ω) is linear invertible and unique. This motivates to parametrize our system kinematic state by the linear and angular velocities of the chassis, resulting in the so-called *mono-cycle model*.

Depending on the type of motor-controllers in use, one can distinguish between wheel velocity controllers and wheel acceleration controllers. In many situations, low-level motor controllers provide the wheel velocities as an input. Thus, the system model is

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} v \\ \omega \end{bmatrix}, \quad \dot{\mathbf{x}} = \begin{bmatrix} v\cos(\theta) \\ v\sin(\theta) \\ \omega \end{bmatrix}, \quad \mathbf{x}(t_0) = \mathbf{x}_0. \tag{2.7}$$

Let us assume for the following that we have a platform whose motors can be only acceleration controlled. To that end, due to the relationship in (2.6), one can simply extend the system model (2.7) with one more integrator from the acceleration inputs. Thus, our acceleration-input platform model is

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ \theta \\ v \\ \omega \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} a \\ \alpha \end{bmatrix}, \quad \dot{\mathbf{x}} = \begin{bmatrix} v\cos(\theta) \\ v\sin(\theta) \\ \omega \\ a \\ \alpha \end{bmatrix}, \quad \mathbf{x}(t_0) = \mathbf{x}_0. \tag{2.8}$$

Example 2.2 (Ackerman-drive kinematic model). The kinematic motion of an automobile can be modelled as the so-called *bicycle model*, illustrated in Figure 2.2.

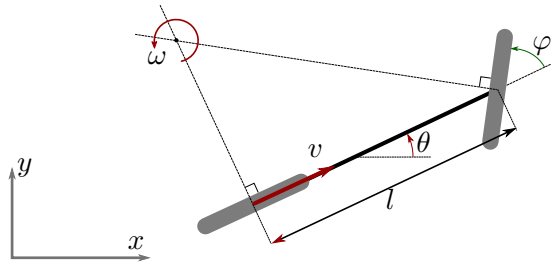


Figure 2.2: Ackerman-drive geometric model

Out of trivial geometrical considerations, the state and dynamics of the model when actuating the velocity of the rear wheel are

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} v \\ \varphi \end{bmatrix}, \quad \dot{\mathbf{x}} = \begin{bmatrix} v\cos(\theta) \\ v\sin(\theta) \\ \frac{v\tan(\varphi)}{l} \end{bmatrix}, \quad \mathbf{x}(t_0) = \mathbf{x}_0. \tag{2.9}$$

Example 2.3 (Independent wheel steering (IWS) kinematic model). Many fields of application of mobile robotics benefit greatly from holonomic motion capabilities. True holonomicity represents the capability of the system to undergo an arbitrary trajectory (position and orientation) from any starting configuration. This property allows wheeled mobile agents to navigate effectively in cluttered environments such as work-spaces shared with humans or work-spaces restricted with respect to the agent size (e.g. forklifts operating in indoor warehouses). As true holonomic platforms using omni or mecanum wheels [21, 22] tend to be avoided for operation in unstructured environments (due to their generally required wheel maintenance), the typical design used for holonomic motions makes use of independent steering systems.

In such a configuration, the platform possesses a certain number of wheels (at least two) that are actuated both in their revolution as well as in their orientation. This allows the platform to perform (up to certain mechanical design constraints) arbitrary (unconstrained) motions in the Spherical Euclidean Group [23] SE(2) configuration space.

A useful concept when analysing the kinematics of such a platform is the Instantaneous Center of Curvature (ICC), defined as the point around which the platform is currently performing a circular motion. Figure 2.3 illustrates the local frame of an Independent four Wheel Steering (I4WS) with its ICC at an arbitrary location. It is worth noting that for control and wheel synchronization, the ICC should be considered, in general, to be a local-frame quantity due to present geometrical constraints. In this way, its location is tractable and parametrization is substantially easier as opposed to when

considered in a global frame.

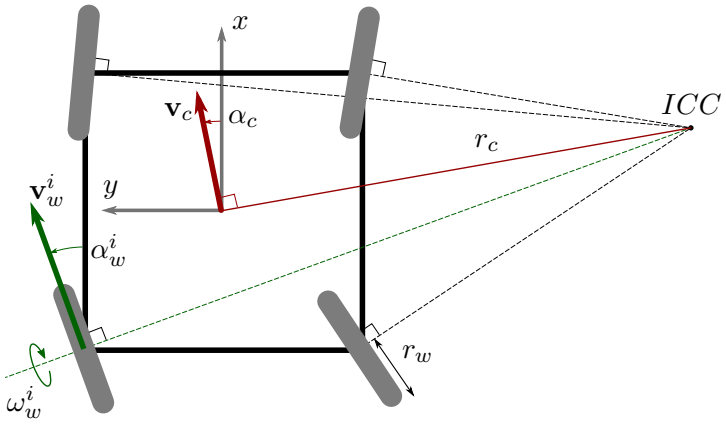


Figure 2.3: I4WS geometric model

Assuming that no drifting motion is desired, wheel orientations are constrained to be perpendicular to the ICC and their revolution has to be consistent with the angular velocity induced by the ICC. Thus, the instantaneous kinematic state of an IWS can be described by three abstract parameters $\mathbf{p} \in \mathbb{R}^3$ for which a (non-linear) algebraic relationship with the ICC exists. For example, such a parametrization can be: v – velocity of the body center and the position of the ICC represented in polar coordinates (r and α_c). However, such a parametrization leads to singularity in the case of parallel wheel placement ($r \to \infty$) and therefore r is substituted with $\rho = 1/r$. In order to be able to perform changes to the ICC from one side of the base to the other, ρ is allowed to be negative. In this case, the platform state and dynamics are given by

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} v \\ \rho \\ \alpha_c \end{bmatrix}, \quad \dot{\mathbf{x}} = \begin{bmatrix} v\cos(\theta + \alpha_c) \\ v\sin(\theta + \alpha_c) \\ -v\rho \end{bmatrix}, \quad \mathbf{x}(t_0) = \mathbf{x}_0. \tag{2.10}$$

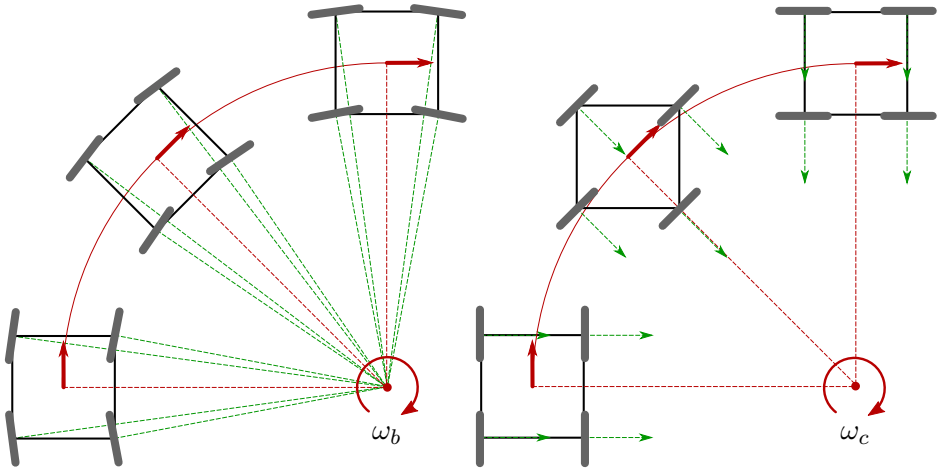
The orientation $\alpha_w^i(t)$ and rotation $\omega_w^i(t)$ of wheel i result from geometrical considerations and are presented in (2.11a) and (2.11b), with x_w^i and y_w^i the wheel position coordinates in the local frame and r_w the wheel radius [24]

$$\alpha_w^i(t) = \arctan\left(\frac{\sin(\alpha_c)/\rho - y_w^i}{\cos(\alpha_c)/\rho - x_w^i}\right) \tag{2.11a}$$

$$\omega_w^i(t) = \frac{-v}{r_w}\sqrt{1 + \rho^2({x_w^i}^2 + {y_w^i}^2) - 2\rho(x_w^i\cos(\alpha_c) + y_w^i\sin(\alpha_c))} \tag{2.11b}$$

Note that these equations hold for other kinematic parametrizations as well as for other mobile platforms (such as differential drives or Ackerman drives) through trivial variable substitution. Thus, given the fixed sub-state describing the configuration space $\begin{bmatrix} x & y & \theta \end{bmatrix}^{\mathrm{T}} \in \mathrm{SE}(2)$ and the mappings (2.11a) and (2.11b), we are interested in parametrizing the rest of the system state and(or) system inputs such that the resulting model is satisfactory for a given application.

Due to the non-linear nature of the geometrical configuration of an IWS, several parametrizations of its kinematic state are possible and advisable depending on further application. For example, the above-mentioned parametrization $\mathbf{p}^{\mathrm{T}} = \begin{bmatrix} v & \rho & \alpha_c \end{bmatrix}$ has the advantage of remaining well defined when the base velocity tends towards 0, simplifying initial conditions and parametrization switches. However, the trade-off lies on the fact that a singularity is present when performing pure rotation ($\rho \to \infty$). An alternative would be to use $\mathbf{p}^{\mathrm{T}} = \begin{bmatrix} v & \omega_b & \alpha_c \end{bmatrix}$, ($\omega_b = -v/r$) and avoid the previous singularity. However, this parametrization is not defined at $v \to 0$. This can be exploited by the fact that when the platform stands still, the wheels do not have to be synchronized in order to achieve a different initial state and thus can rotate at full speed around the z–axis, which generally complicates the control scheme. One could also use a linear kinematic state (with respect to the configuration space), $\mathbf{p}^{\mathrm{T}} = \begin{bmatrix} \dot{x} & \dot{y} & \dot{\theta} \end{bmatrix}$. This simplifies some platform path-following controller designs. However, it is not well defined when the platform stands still. Another drawback of such a parametrization is the fact that it is not straight-forward to encode the direction of the chassis linear velocity. This can lead to problems when designing path-following controllers as 180° jumps in the ICC angle (α_c) can occur.



wheel geometry ICC induced ICC

Figure 2.4: I4WS global frame considerations; the vehicles body can rotate as well

Other parametrizations can be used by exploiting global frame characteristics of the kinematics. More intuitively, consider the two situations presented in Figure 2.4. On the left, the global frame motion of the platform with a fixed wheel geometry ICC is shown. The same trajectory can be achieved by modifying α_c while the wheels stay parallel. This creates an induced ICC and the total trajectory motion curvature can be interpreted as a superposition of the two $\omega_{traj} = \omega_b + \omega_c$, where the orientation of the platform is influenced only by ω_b. This motivates the parametrization $\mathbf{p}^{\mathrm{T}} = \begin{bmatrix} v & \omega_{traj} & \omega_b \end{bmatrix}$ due to the parameter decoupling between trajectory and orientation of the platform, a fact that can be further exploited when solving optimization problems.

Example 2.4 (Single-track dynamic model). We would like to address the cases in which we do not assume perfect rolling motion of a vehicle. In this case, a simple dynamic model of an auto-mobile is the single-track model, illustrated in Figure 2.5 [25].

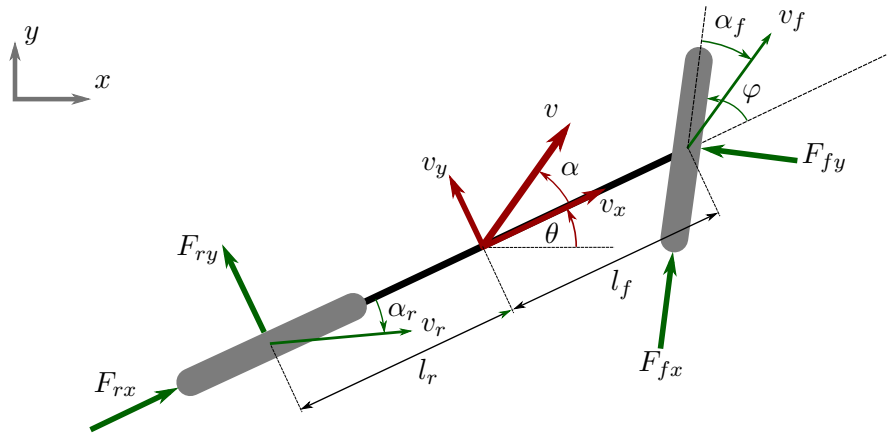


Figure 2.5: The single-track dynamic model

A useful measure of the lateral velocity of the vehicle is the side-slip angle at the center of mass, defined as

$$\alpha = \arctan\left(\frac{v_y}{v_x}\right). \tag{2.12}$$

Similarly, the side-slip angles at the front and back wheels are

$$\alpha_f = \varphi - \arctan\left(\frac{v_y + \omega l_f}{v_x}\right), \quad \alpha_r = \arctan\left(\frac{v_y - \omega l_r}{v_x}\right). \tag{2.13}$$

The forces that act on the vehicle are assumed to be planar and for brevity of notation, $F_\star(\mathbf{x})$ denotes the sum of all the forces that act on a specific wheel, in a specific direction. Having this in mind, the dynamics of the single-track model is given by

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} x \\ y \\ \theta \\ v_x \\ v_y \\ \omega \end{bmatrix} = \begin{bmatrix} v_x\cos(\theta) - v_y\sin(\theta) \\ v_x\sin(\theta) + v_y\cos(\theta) \\ \omega \\ \frac{1}{m}(F_{fx}(\mathbf{x})\cos(\varphi) - F_{fy}(\mathbf{x})\sin(\varphi) + F_{rx}(\mathbf{x})) + v_y\omega \\ \frac{1}{m}(F_{fx}(\mathbf{x})\sin(\varphi) + F_{fy}(\mathbf{x})\cos(\varphi) + F_{ry}(\mathbf{x})) - v_x\omega \\ \frac{1}{I_{zz}}(F_{fx}(\mathbf{x})\sin(\varphi) + F_{fy}(\mathbf{x})\cos(\varphi))l_f - F_{ry}(\mathbf{x})l_r) \end{bmatrix}. \tag{2.14}$$

However, depending on the application of the model, it is sometimes desired to have a parametrization that encodes the chassis velocity as $\begin{bmatrix} v & \alpha \end{bmatrix}$. In order to derive such a model, we begin by expressing $v(t)$ and its derivative as a function of the previous state variables

$$v = \sqrt{v_x^2 + v_y^2}, \quad \dot{v} = \frac{v_x\dot{v}_x + v_y\dot{v}_y}{v}. \tag{2.15}$$

Using the solution for $\dot{v}_x$ and $\dot{v}_y$ from (2.14) and dropping temporal dependency yields

$$\dot{v} = \frac{F_{fx}(\mathbf{x})(v_x \cos(\varphi) + v_y \sin(\varphi)) - F_{fy}(\mathbf{x})(v_x \sin(\varphi) - v_y \cos(\varphi)) + F_{rx}(\mathbf{x})v_x - F_{ry}(\mathbf{x})v_y}{mv}. \tag{2.16}$$

Note that

$$\frac{v_x \cos(\varphi) + v_y \sin(\varphi)}{v} = \cos(\varphi)\cos(\alpha) + \sin(\varphi)\sin(\alpha) = \cos(\varphi - \alpha)$$
$$\frac{v_x \sin(\varphi) - v_y \cos(\varphi)}{v} = \sin(\varphi)\cos(\alpha) - \cos(\varphi)\sin(\alpha) = \sin(\varphi - \alpha).$$

Thus, we have

$$\dot{v} = \frac{F_{fx}(\mathbf{x})\cos(\varphi - \alpha) - F_{fy}(\mathbf{x})\sin(\varphi - \alpha) + F_{rx}(\mathbf{x})\cos(\alpha) - F_{ry}(\mathbf{x})\sin(\alpha)}{m}. \tag{2.17}$$

Analogously, starting from $\alpha = \arctan(\frac{v_y}{v_x})$ one can compute $\dot{\alpha}$. Finally, the $\begin{bmatrix} v & \alpha \end{bmatrix}$ parametrized model is

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} x \\ y \\ \theta \\ v \\ \alpha \\ \omega \end{bmatrix} = \begin{bmatrix} v\cos(\theta + \alpha) \\ v\sin(\theta + \alpha) \\ \omega \\ \frac{1}{m}\left(F_{fx}(\mathbf{x})\cos(\varphi - \alpha) - F_{fy}(\mathbf{x})\sin(\varphi - \alpha) + F_{rx}(\mathbf{x})\cos(\alpha) - F_{ry}(\mathbf{x})\sin(\alpha)\right) \\ \frac{1}{mv}\left(F_{fx}(\mathbf{x})\sin(\varphi - \alpha) + F_{fy}(\mathbf{x})\cos(\varphi - \alpha) - F_{rx}(\mathbf{x})\sin(\alpha) - F_{ry}(\mathbf{x})\cos(\alpha)\right) - \omega \\ \frac{1}{I_{zz}}(F_{fx}(\mathbf{x})\sin(\varphi) + F_{fy}(\mathbf{x})\cos(\varphi))l_f - F_{ry}(\mathbf{x})l_r) \end{bmatrix} \tag{2.18}$$

Regarding the equations that model the tire friction forces, the literature proposes various approaches. One naive model is to assume that the tire friction is proportional to the side-slip and longitudinal slip for lateral and longitudinal friction forces respectively:

$$F_y(\mathbf{x}) = F_y(\alpha_w) = c\alpha_w \tag{2.19}$$

A more reasonable model that proves to be quite accurate in low-traction conditions such as icy roads, is the arctangent model [26]

$$F_y(\mathbf{x}) = F_y(\alpha_w) = c_1 \arctan(c_2\alpha_w). \tag{2.20}$$

A semi-empirical model that closely describes the pure lateral (longitudinal) tire-friction forces is the Pajeka-model [26]

$$F_y(\mathbf{x}) = F_y(\alpha_w) = d\sin(c\arctan(b\alpha_w - e(b\alpha_w - \arctan(b\alpha_w)))) \tag{2.21}$$

Note that in this model, the linear region is included by $\frac{\partial}{\partial\alpha_w}F_y(\alpha_w)\Big|_{\alpha_w=0} = bcd$.

Even in the above presented Pajeka model, several simplifications are performed. Namely, the normal tire force is assumed to be constant and the tire is assumed to have 0 camber. Moreover, detailed models take into account the so-called combined slip forces as well as induced over-turning moments (moments around the z-axis) that the tire-surface contact generate [26]. As noted previously, additional forces (torques) on the vehicle such as wheel rolling resistance torque, air-drag, suspension stiffness as well as tire load-shifts have to be taken into account if an accurate model is desired.

2.1.2 Affine-Input Models

A sub-class of non-linear systems is represented by the affine-input models

$$\dot{\mathbf{x}}(t) = \mathbf{f}_{\mathbf{x}}(\mathbf{x}(t), t) + \mathbf{g}_{\mathbf{x}}(\mathbf{x}(t), t)\mathbf{u}(t) \ , \quad \mathbf{x}(t_0) = \mathbf{x}_0. \tag{2.22}$$

Such a representation is typically appealing in the context of differential-geometric methods, where successive derivation of a scalar function along the system dynamics vector-field (Lie-derivative) simplifies considerably. This allows in certain cases to compute the function inverse $\mathbf{u}(\mathbf{y})$ by means of a simple matrix-inversion.

It is here worth noting that a system can be typically transformed in an input-affine system by creating a fictional input as the derivative of the actual system input [27].

Example 2.5 (From non-linear to input-affine model). Consider the IWS model from Example 2.3. For the inputs $\mathbf{u} = \begin{bmatrix} v & \rho & \alpha_c \end{bmatrix}^{\mathrm{T}}$ the system is clearly not input-affine, as:

$$\exists u_i \quad \text{such that} \quad \frac{\partial}{\partial u_i}\left(\frac{\partial \dot{\mathbf{x}}}{\partial \mathbf{u}}\right) = \frac{\partial}{\partial u_i}\left(\begin{bmatrix} \cos(\theta+\alpha_c) & 0 & -v\sin(\theta+\alpha_c) \\ \sin(\theta+\alpha_c) & 0 & v\cos(\theta+\alpha_c) \\ -\rho & -v & 0 \end{bmatrix}\right) \neq \mathbf{0} \tag{2.23}$$

However, by adding the inputs to the state and defining the new fictional inputs $\mathbf{u}_a = \dot{\mathbf{u}}$, we obtain the affine-input system

$$\mathbf{x}_a = \begin{bmatrix} x \\ y \\ \theta \\ v \\ \rho \\ \alpha_c \end{bmatrix}, \quad \mathbf{u}_a = \begin{bmatrix} \dot{v} \\ \dot{\rho} \\ \dot{\alpha}_c \end{bmatrix}, \quad \dot{\mathbf{x}}_a = \underbrace{\begin{bmatrix} v\cos(\theta+\alpha_c) \\ v\sin(\theta+\alpha_c) \\ -v\rho \\ 0 \\ 0 \\ 0 \end{bmatrix}}_{\mathbf{f}_{\mathbf{x}}(\mathbf{x})} + \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{g}_{\mathbf{x}}(\mathbf{x})} \mathbf{u}_a. \tag{2.24}$$

Of course, such a model extension will lead to the fictional input $\mathbf{u}_a$ being a (higher order) derivative of the physical input of the system. In such situations in the context of control, a so called dynamic controller can be used, which (numerically) integrates the fictional input to obtain the physical input [28]. In our example, suppose we can physically control the angular velocity of the wheels revolution and their steering angle, i.e. α_w^i and ω_w^i. Thus, a dynamic controller can be applied on the extended model inputs

$$\mathbf{u} = \begin{bmatrix} v & \rho & \alpha_c \end{bmatrix}^{\mathrm{T}}, \quad \dot{\mathbf{u}} = \mathbf{u}_a. \tag{2.25}$$

Note that such an approach is not advised to be used in open-loop, as the open-loop numerical integration will in practice drift from the exact solution.

2.1.3 Linear Time-Variant Models

A further simplification of the affine-input model is to assume that $\mathbf{f}_\mathbf{x}$ is linear in $\mathbf{x}$ and $\mathbf{g}_\mathbf{x}$ does not depend on $\mathbf{x}$. This results in the Linear Time-Variant (LTV) model

$$\dot{\mathbf{x}}(t) = \mathbf{A}(t)\mathbf{x}(t) + \mathbf{B}(t)\mathbf{u}(t) \ , \quad \mathbf{x}(t_0) = \mathbf{x}_0 \tag{2.26}$$

It is worth noting that compared to Linear Time-Invariant (LTI) models, the LTV model is considerably more difficult to manipulate in the context of control. Nevertheless, the solution of the ODE can be in certain cases computed in closed form. We will later-on make use of such models in order to evaluate a part of the system ODE (and its related sensitivities ODE) in closed form.

Theorem 2.1 (State-transition Matrix of Time-Variant System)**.** *Given is a system of the form (2.26). If the matrix* $\mathbf{A}(t)$ *can be decomposed in*

$$\mathbf{A}(t) = \sum_{i=1}^{n} \mathbf{M}_i f_i(t) \tag{2.27}$$

with $\mathbf{M}_i$ *– a constant matrix such that* $\forall i, j \in [1, n], \mathbf{M}_i\mathbf{M}_j = \mathbf{M}_j\mathbf{M}_i$*, then the system state transition matrix* $\mathbf{\Phi}(t_0, t)$ *is*

$$\mathbf{\Phi}(t_0, t) = \prod_{i=1}^{n} \mathrm{e}^{\mathbf{M}_i \int_{t_0}^{t} f_i(\tau)\mathrm{d}\tau}. \tag{2.28}$$

Thus, the solution for $\mathbf{x}(t)$ *is*

$$\mathbf{x}(t) = \mathbf{\Phi}(t_0, t)\mathbf{x}(0) + \int_{t_0}^{t} \mathbf{\Phi}(\tau, t)\mathbf{B}(\tau)\mathrm{d}\tau. \tag{2.29}$$

A proof of the Theorem can be found in [29].

2.2 Parametric Functions

Representing information in a mathematical sound way strongly relates with the concept of functions. Moreover, in many cases, it is convenient to encode the entire search-space of a given problem into a set of parameters $\mathbf{p}$. However, in many problems, the entire search-space is so large that it becomes intractable. An approach to circumvent this is to consider a relaxed problem in which $\mathbf{p}$ has a considerably smaller dimensionality and thus searching in the problem-space becomes feasible. However, the function shapes that can be encoded through the (reduced dimensionality) parameters $\mathbf{p}$ has to be sufficiently *expressive*, i. e. it has to approximate a wide range of desired function shapes with sufficient accuracy.

For the beginning, let us define a general form of a parametric function:

Definition 2.1 (Parametric function)**.** Given the input space of the problem $\mathcal{X} \subseteq \mathbb{R}^n$, the output space $\mathcal{Y} \subseteq \mathbb{R}^m$ as well as the search space $\mathcal{P} \subseteq \mathbb{R}^p$, we define the vector valued function (parametric function)

$$\mathbf{f}(\mathbf{p}, \boldsymbol{\xi}), \quad \boldsymbol{\xi} \in \mathcal{X}, \mathbf{f} \in \mathcal{Y}, \mathbf{p} \in \mathcal{P}. \tag{2.30}$$

Definition 2.1 allows us to formulate a search problem as follows: finding an appropriate parametric encoding $\mathbf{p}$ of the function family $\mathbf{f}$ such that the function provides a desired mapping from certain domains of the input space $\mathcal{X}' \subseteq \mathcal{X}$ to certain domains of the output space $\mathcal{Y}' \subseteq \mathcal{Y}$. Typical examples of such problems are function fitting or function minimization.

An important property of such functions is their continuity and differentiability. We denote that a function is continuous by C^0. Moreover, we denote a function is continuous and n-times differentiable by C^n. When a function is only piece-wise n-times differentiable, we represent it with $\hat{C}^n$.

In the following, several parametric function families along with problem representation examples will be presented, which are beneficial in the context of control.

2.2.1 Piece-wise Constant

Probably one of the simplest yet useful function family is the piece-wise constant function. As its structure within one interval is very simple (constant), it is appealing to use due to the ease of manipulation (for example for sensitivities computation). For functions that depend on a one-dimensional arc parametrization, any piece-wise constant function $\mathbf{f} : \mathbb{R}^p \times [\xi_0, \xi_{n-1}) \to \mathbb{R}^m$ can be expressed as

$$\mathbf{f}(\mathbf{p},\ \xi) = \mathbf{f}_i \quad \text{for} \quad \xi_i \leq \xi < \xi_{i+1}. \tag{2.31}$$

As the function is constant on every interval, it follows that it is continuously differentiable on each interval. The total number of parameters to describe any function from such a function family with n intervals is

$$\mathbf{p} = \begin{bmatrix} \xi_0 & \xi_1 & \dots & \xi_{n-1} & \mathbf{f}_1^{\mathrm{T}} & \mathbf{f}_2^{\mathrm{T}} & \dots & \mathbf{f}_{n-1}^{\mathrm{T}} \end{bmatrix}^{\mathrm{T}} \tag{2.32}$$

having the cardinality $\dim(\mathbf{p}) = n(1 + m)$.

However, in many applications, using only a few of such constant intervals does not provide sufficient degrees of freedom. Because of this, the duration of the piece-wise constant intervals is typically small in relation to the definition domain (i. e. many pieces are required to express the mapping we are interested in). For such a dense representation, one can argue that being able to vary the bounds of each constant interval would not

yield a considerable expressibility gain. This fact motivates reducing the dimensionality of the function parameter vector by fixing the length of the constant-value intervals

$$\mathbf{f}(\mathbf{p},\ \xi) = \mathbf{f}_i(\mathbf{p}) \quad \text{for} \quad i\Delta\xi \leq \xi < (i+1)\Delta\xi \tag{2.33a}$$

$$\mathbf{p} = \begin{bmatrix} \Delta\xi & \mathbf{f}_1^{\mathrm{T}} & \mathbf{f}_2^{\mathrm{T}} & \dots & \mathbf{f}_{n-1}^{\mathrm{T}} \end{bmatrix}^{\mathrm{T}}. \tag{2.33b}$$

In many cases in which ξ represents the time, $\Delta\xi$ is fixed and chosen to be equal to the sampling time of the controller. This enables easier manipulation of time-shifts of such a function as the system time evolves. In other cases, however, one might want to be able to vary the interval in which the function is defined.

The variation of $\Delta\xi$ can be interpreted as a scaling parameter of the function definition domain. To that end, note that this implies that when using a fixed dimensionality of the parameter vector $\mathbf{p}$, larger definition domains imply lower expressibility (resolution) of the function, while as $\Delta\xi \to 0$, the piece-wise constant function can express any function shape. Even though a function where $\Delta\xi \to 0$ is not typically useful, this property will provide arguments regarding the expressibility increase of a parametrised trajectory whose duration is well defined but its arc length approaches zero.

Example 2.6 (Dense encoding of system input values). In certain (simplified) models, actuators can be approximated by an algebraic model that maps their physical input to the force (torque) they produce, i. e. $\boldsymbol{\tau}_{act} = \mathbf{f}(\mathbf{u}_{act})$. Provided that this equation is invertible, one can abstract away the actuators and model the higher-level problem with the input $\mathbf{u}_{sys} = \boldsymbol{\tau}_{act}$. Note that in some cases where the actuator dynamics are non-negligible, their dynamics can be accounted for by a lower-level controller running at a higher frequency, allowing the above-mentioned simplification.

As under the laws of classical mechanics, the forces (torques) of a system are not required to be differentiable nor continuous, a good candidate for encoding the system inputs is the piece-wise constant function

$$\mathbf{u}_{sys}(t) = \sum_{i=0}^{n-2} \mathbf{u}_{sys_i} \sigma(t - i\Delta t)\sigma((i+1)\Delta t - t), t \in [0, (n-1)\Delta t]. \tag{2.34}$$

An illustration of such a 1-dimensional function is presented in Figure 2.6.

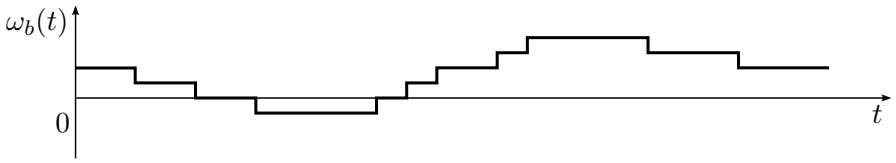


Figure 2.6: A 1-dimensional piece-wise constant function $\omega_b(t)$

Example 2.7 (Pixel maps). Of-course, we can use the concept of piece-wise constant functions in higher input dimensions as well. A helping simplification in this case is to assume again that for each dimension, the constant-value intervals are equal. In 2-dimensional case, this yields to the function family

$$f(\mathbf{p},\ \boldsymbol{\xi}) = f_{ij}(\mathbf{p}) \quad \text{s.t.} \quad \begin{bmatrix} i\ \Delta\xi_0 \\ j\ \Delta\xi_1 \end{bmatrix} \leq \boldsymbol{\xi} < \begin{bmatrix} (i+1)\Delta\xi_0 \\ (j+1)\Delta\xi_1 \end{bmatrix} \tag{2.35a}$$

$$\mathbf{p} = \begin{bmatrix} \Delta\boldsymbol{\xi}^{\mathrm{T}} & \begin{bmatrix} f_{ij} \mid 0 \leq i < n,\ 0 \leq j < n \end{bmatrix} \end{bmatrix}^{\mathrm{T}}. \tag{2.35b}$$

In the case where $\boldsymbol{\xi} = \begin{bmatrix} x & y \end{bmatrix}^{\mathrm{T}}$, we can interpret the function (2.35) as a pixel-map with variable resolution on x and y dimension. An example of such a function is illustrated in Figure 2.7.

Figure 2.7: Visualization of a pixel-map

2.2.2 Piece-wise Linear

A natural extension of the piece-wise constant function is the piece-wise linear function. Its structure is still simple enough such that many useful quantities can be computed in closed-form, but its increased expressibility allows for sparser representations. Moreover, as it possesses a piece-wise constant derivative, it can be used to encode quantities that are one integrator higher than the acceleration space for Newtonian mechanics, resulting in piece-wise constant accelerations of the system.

Example 2.8 (Piece-wise linear representation of kinematic inputs). The equations that describe the dynamics of a system can be typically derived as a white box model from physical considerations. However, in many applications, high-level controllers do not take into account a very detailed model of the system dynamics, as the structure and computational complexity of such models are more detrimental than the performance gain that they provide.

Thus, mechanical dynamic systems are often treated under kinematic laws. Following the kinematic modelling example of the differential drive (2.1), the system kinematic

input is the velocities of the wheels. Previously it has been showed that there is an affine relationship between this velocities (accelerations) and the input parametrization

$$\mathbf{u}^{\mathrm{T}} = \begin{bmatrix} v & \omega \end{bmatrix}. \tag{2.36}$$

Thus, for such a motion model, this parametrization proves to be beneficial. Moreover, the orientation of the robot θ equals to the integral of the angular velocity ω

$$\theta = \theta_0 + \int_{t_0}^{t_1} \omega(t)\,\mathrm{d}t. \tag{2.37}$$

Also, the travelled distance of the agent equals to

$$s = \int_{t_0}^{t_1} |v(t)|\,\mathrm{d}t. \tag{2.38}$$

It would be thus convenient to have a parametric representation of the system input (2.36) that can evaluate the vehicle orientation and travelled distance in closed form. However, modelling and taking into account the accelerations of the system is desired for improved models and controllers. Thus, we want our kinematic input to be at least C^1 (differentiable).

A simple but yet powerful parametrization that partially fulfils the above-mentioned is the piece-wise linear function (differentiable on every interval). Even though higher-order polynomials and splines provide more expressive functions than the piece-wise linear, piece-wise linear can be evaluated very efficiently, together with various integrals (2.37) and (2.38) as well as derivatives.

For the given example, such functions can be expressed mathematically as

$$v(t) = \sum_{i=0}^{n-2} \sigma(t - t_i)\sigma(t_{i+1} - t) \cdot \left(v_i + (v_{i+1} - v_i)\frac{t - t_i}{t_{i+1} - t_i} \right) \tag{2.39a}$$

$$\omega(t) = \sum_{i=0}^{n-2} \sigma(t - t_i)\sigma(t_{i+1} - t) \cdot \left(\omega_i + (\omega_{i+1} - \omega_i)\frac{t - t_i}{t_{i+1} - t_i} \right), \tag{2.39b}$$

with t_i the arc parametrization of the function, and v_i and ω_i, respectively, the values of the function at the inflection points. An illustration of such encodings of the kinematic input of the system is presented in Figure 2.8. Note that for such parametrization, pre-computation can be performed such that afterwards, evaluations at an arbitrary $t \in [t_0, t_{n-1})$ of the function itself or various derivatives and integrals can be computed very efficiently.

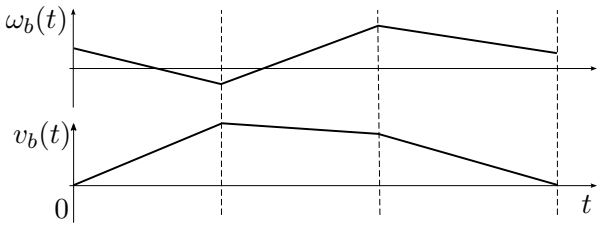


Figure 2.8: Piece-wise linear encoding of kinematic inputs for a differential-drive

Example 2.9 (Continuous pixel maps). The concept of piece-wise linear functions can be as well used in higher input space dimensions. An example is the use of bi-linear interpolation for pixel-maps. This allows the pixel-maps to become continuous functions, requiring the evaluation of 4 pixels to obtain one function value [30].

For simplification of notation, let us assume that the coordinates x, y of the point we are willing to evaluate fulfil $x, y \in [0, 1]$. This results in the bi-linearly interpolated function value

$$\begin{aligned} f(x,y) &= a + bx + cy + dxy \qquad (2.40)\\ \text{where: } a &= f(0,0)\\ b &= f(1,0) - f(0,0)\\ c &= f(0,1) - f(0,0)\\ d &= f(1,1) + f(0,0) - (f(1,0) + f(0,1)). \end{aligned}$$

A graphical interpretation of the bi-linear interpolation is presented below:

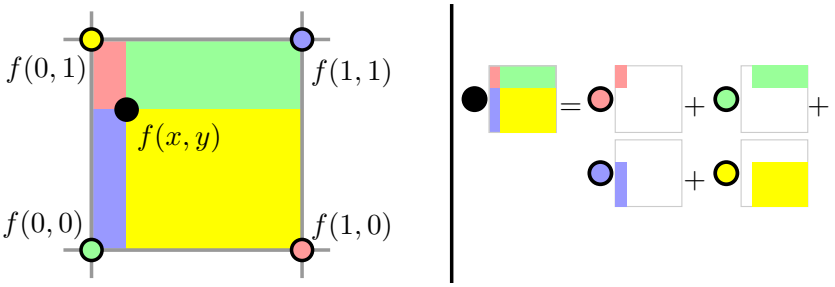


Figure 2.9: Visualization of the bi-linear interpolation. In this geometric visualisation, the value at the black spot is the sum of the value at each coloured spot multiplied by the area of the rectangle of the same colour, divided by the total area of all four rectangles. (*source: Wikipedia*)

For illustration purposes, the pixel-map from Example 2.7 is presented when over-sampled 20 times using bilinear interpolation:

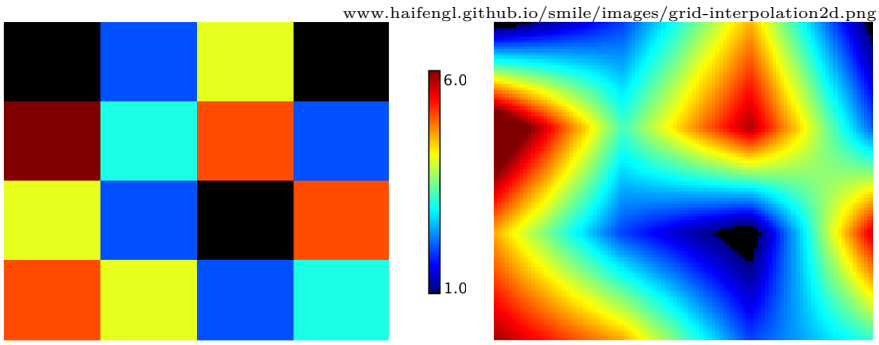


Figure 2.10: Pixel-map (*left*) and its bilinear interpolation (*right*)

2.2.3 Polynomials

Assuming that the input-space $\subseteq \mathbb{R}^p$, the most general form of a scalar-valued polynomial of order r can be given by

$$f(\mathbf{p}, \boldsymbol{\xi}) = \sum_{i_1=0}^{r} \sum_{i_2=0}^{r} \cdots \sum_{i_n=0}^{r} p_{i_1,i_2,\ldots,i_n} \xi_1^{i_1} \xi_2^{i_2} \ldots \xi_n^{i_n}. \tag{2.41}$$

An appealing property of polynomials is that they provide a sparse encoding of considerably expressive function shapes. Moreover, polynomials are a relatively simple algebraic structure that can be mathematically easily manipulated. However, a drawback of polynomials is that it is somewhat cumbersome to encode vector-valued functions (one method could be to use matrix-polynomials). Moreover, they typically possess very large sensitivities for the higher order terms [31].

Example 2.10 (Unconstrained trajectory generation between two states). We would like to find a trajectory for a differential-drive robot for a given start state $\mathbf{x}_{d0}^{\mathrm{T}} = \begin{bmatrix} x_{d0} & y_{d0} & \theta_{d0} \end{bmatrix}$ and end state $\mathbf{x}_{d1}^{\mathrm{T}} = \begin{bmatrix} x_{d1} & y_{d1} & \theta_{d1} \end{bmatrix}$. From the fact that the instantaneous velocity v_d is constrained to be along the orientation of the robot, it follows that $\theta_d = \arctan(\dot{y}_d / \dot{x}_d)$, i. e. $\dot{x}_d = v_d \cos(\theta_d)$, $\dot{y}_d = v_d \sin(\theta_d)$. In this case, excluding the case where $v_d = 0$, finding suitable functions for the temporal evolution of x_d and y_d will suffice to describe a feasible motion of the robot model. Moreover, from the functions derivatives, we will be able to extract the input of the robot motion-model along the generated trajectories. By counting the number of boundary-conditions for the functions $x_d(t)$ and $y_d(t)$, we get

$$x_d(t_0){=}x_{d0},\ x_d(t_1){=}x_{d1},\ \dot{x}_d(t_0){=}v_0\cos(\theta_0),\ \dot{x}_d(t_1){=}v_1\cos(\theta_1) \tag{2.42a}$$
$$y_d(t_0){=}y_{d0},\ y_d(t_1){=}y_{d1},\ \dot{y}_d(t_0){=}v_0\sin(\theta_0),\ \dot{y}_d(t_1){=}v_1\sin(\theta_1). \tag{2.42b}$$

We trivially note that a function would require at least 4 parameters to exactly satisfy the constraints. Thus, one could consider two univariate third order polynomials

$$x_d(t){=}a_x t^3 + b_x t^2 + c_x t + d_x, \quad y_d(t) = a_y t^3 + b_y t^2 + c_y t + d_y. \tag{2.43}$$

Combining the above two equations, the polynomial coefficients of the functions can be obtained. Figure 2.11 illustrates computed trajectories from the origin to three different end-points with various final orientations for different time intervals.

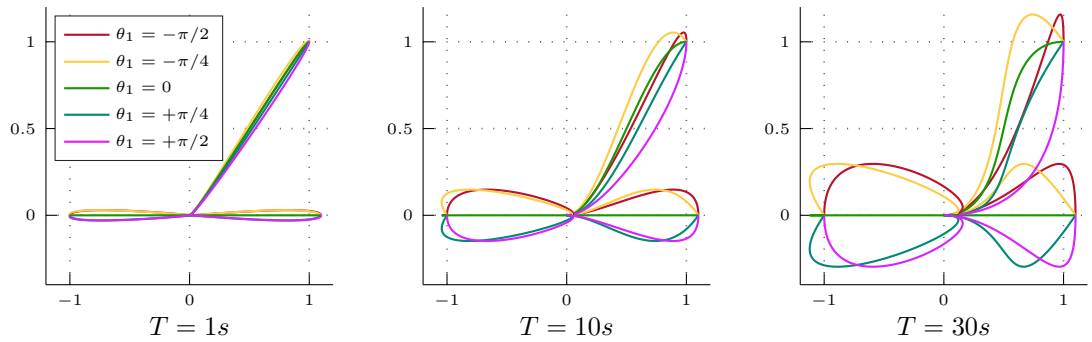


Figure 2.11: Unconstrained pose-to-pose trajectories using polynomial fitting from origin to various coordinates for different temporal durations T.

2.2.4 Splines

A popular mathematical structure that provides a parametric representation of a function is the Spline. In this subsection, we will take a closer look at the definition, structure and properties of the B-Spline and its generalization, the Non-Uniform Rational B-Spline (NURBS) [32].

We have already seen that Subsections 2.2.1 and 2.2.2 make use of increasingly more expressive functions (piece-wise constant and then piecewise-linear). Subsection 2.2.3 presented the polynomial function, a function that is easy to evaluate (together with its derivatives and integrals) and can be made arbitrarily expressive by increasing its order.

Combining the two concepts, the general idea of B-Splines is to make use of polynomials as Basis Functions (hence the name) of a certain degree d that are well connected with each-other (continuous up to order $n-1$). This results in a function family that generalizes well and can provide certain benefits in practical applications.

Knots As in the case of the piece-wise constant and piece-wise linear functions, the knots of the Spline define the locations at which certain basis-functions become inactive while certain basis-functions become active. They are required to be defined in a non-decreasing order

$$\xi_i \leq \xi_{i+1}, \quad i = 0, 1, \ldots, n + d - 1 \tag{2.44}$$

with the number n of control points and the degree d of the resulting Spline. Note that in the case of piece-wise constant function (a 0-Order B-Spline), we require exactly n knots. For the piece-wise linear case, we require $n + 1$ knots (an additional knot that ends the last interval).

Basis Functions B-Splines make use of Basis Functions of degree d that are active (non-zero) only on a sub-interval of the function. For the case of 0-order Basis Functions, they are defined as

$$b_{i,0}(\xi) = \begin{cases} 1, & \text{if} \quad \xi_i \leq \xi < \xi_{i+1} \\ 0, & \text{otherwise} \end{cases} \tag{2.45}$$

Higher order basis functions are defined recursively (Cox-de Boor formula) [33]

$$b_{i,d}(\xi) = \frac{\xi - \xi_i}{\xi_{i+d} - \xi_i} b_{i,d-1}(\xi) + \frac{\xi_{i+d+1} - \xi}{\xi_{i+d+1} - \xi_{i+1}} b_{i+1,d-1}(\xi). \tag{2.46}$$

Control Points Having such basis functions, we can now construct a Spline as a linear combination of Basis Functions using control points

$$\mathbf{s}_d(\xi) = \sum_{i=0}^{n-1} \mathbf{c}_i b_{i,d}(\xi), \tag{2.47}$$

with $\mathbf{c}_i$ being the ith Control Point (vector). Note that given the recursive definition (2.46), (2.47) reduces to [33]

$$\mathbf{s}_d(\xi) = \sum_{i=k-d}^{d} \mathbf{c}_i b_{i,d}(\xi), \quad \forall \xi \in [\xi_k, \xi_{k+1}). \tag{2.48}$$

We note here the *Local Support* property of the B-Splines, that is, every basis function (and control-point) has influence only on the local shape of the curve.

An additional degree of freedom is obtained in the more generalized variant of B-Splines, the NURBS function

$$\mathbf{s}_d(\xi) = \sum_{i=0}^{n-1} \frac{w_i b_{i,d}(\xi)}{\sum_{j=0}^{n-1} w_j b_{j,d}(\xi)} \mathbf{c}_i. \tag{2.49}$$

Note that here every control-point is weighted with a weighting factor w_i and the result is accordingly normalized.

Finally, we would like to briefly mention typical operations that can be applied on B-Spline curves and their practical applicability [32].

- Curve Fitting: This is one of the most used operation that involves a spline. The typical algorithm (Cox-de Boor algorithm) performs the fit relatively fast, as it evaluates a fitted point recursively using linear interpolation of depth d.
- Evaluation of Derivatives / Integrals: Given their basis-function structure, B-Splines derivatives and integrals are again B-Splines (of one degree lower or higher respectively). This allows relatively easy evaluation of those properties even for higher orders.
- Knot Insertion: Typically used for increasing the expressibility of the curve in a certain region. Note that the insertion of a knot does not alter the function shape.
- Knot Removal: Typically used in optimization for reducing the number of control-points the function possesses. Note that knot removal in general alters the function shape.
- Degree Elevation: Useful when initialization of a lower order spline is simple, but finally a higher order spline is desired. For example, if one parametrizes the motion of a mobile agent with a spline, one might want to initialize it according to a 2D discrete path (resulting in a 1-order spline) but would want to achieve motions of higher orders.

In conclusion, this section presented different parametric function families (of increasing order), presenting their strengths and drawbacks. The main motivation of this discussion lies on the usage of such a parametric representations in order to encode (at least) the system inputs in a temporal interval. With this, system trajectories, typically represented by differential equations, can be analysed.

2.3 Solving ODEs

Section 2.2 discussed the concept of encoding the search-space of a given problem by means of parametric functions. Such encodings can be used to uniquely define the trajectory of a dynamic system (i. e. the solution of an ODE) by parametrizing its input. That is, the differential equation of the system can be expressed as

$$\mathbf{x}(\mathbf{p}, t_1) = \mathbf{x}_0(\mathbf{p}, t_0(\mathbf{p})) + \int_{t_0(\mathbf{p})}^{t_1(\mathbf{p})} \mathbf{f}(\mathbf{x}(t), \mathbf{p}, t)\mathrm{d}t \quad . \tag{2.50}$$

With the exception of few well-known cases, the solutions of an ODE cannot be found in closed-form. Rather, one has to resort to numerical methods for solving such initial-value problems, more specifically for the integral term in (2.50). Solving the ODE of a dynamical system efficiently and with sufficient accuracy is of great importance when solving dynamic optimization problems. Thus, this section is focusing on varius approaches that solve ODEs and their sensitivities, discussing their differences with efficiency of numerical evaluation in mind. Analysing (2.50) in its differential form

$$\dot{\mathbf{x}}(\mathbf{p}, t) = \mathbf{f}(\mathbf{x}(t), \mathbf{p}, t), \quad \mathbf{x}(\mathbf{p}, t_0(\mathbf{p})) = \mathbf{x}_0(\mathbf{p}) \quad , \tag{2.51}$$

let us initially have a look at the simplest way of numerically solving an ODE, using the explicit Euler method

$$\mathbf{x}_{k+1} = \mathbf{x}_k + h\dot{\mathbf{x}}_k, \quad \mathbf{x}_0 = \mathbf{x}_0(\mathbf{p}) \tag{2.52}$$

with the step size h, the numerical solution of the ODE $\mathbf{x}_k$ at time $t = kh$. It can be shown that as $h \to 0$, the numerical solution of the ODE using (2.52) will converge to the correct solution of the ODE. However, in many situations, willing to reduce required computational effort, one is interested in finding a sufficiently accurate solution of an ODE with as large of a h as possible (i. e. least number of steps). This fact motivates the usage of better integration methods.

Informally speaking, solving an initial-value problem of an ODE can be seen as follows: we are given a function for which we know the initial value. Moreover, we can access the value of the function (temporal) derivative given any state of the ODE, but we can never access the function itself. The Euler integration scheme can be then seen as an approximation in which one assumes that for every interval of length h, the value of the function derivative is *constant*. Of-course, even for the case of linear ODEs, this is generally not true. Nevertheless, one could argue that a perfect integration step can still be of the form:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + h\dot{\bar{\mathbf{x}}}_k \tag{2.53}$$

with a "magical" step slope $\dot{\bar{\mathbf{x}}}_k$ that corresponds to such a perfect step. From here, the question is how can we improve our approximate step-slope such that for as large as possible steps, it is still sufficiently close to the ideal step slope. Abstractly speaking, one can use additional information from the vicinity of the function to smartly improve the step-slope estimate, by:

- probing function derivative on the interval $[kh, (k+1)h]$ at multiple locations and weighting them to a step slope
- making use of the local history of few previous steps in order to have an expected slope direction
- creating an implicit dependency between the step slope and the function value *at the next step*

All those generic concepts have been widely studied and exploited and as a consequence, various advanced ODE solvers have emerged [34, 35]. In the following sub-sections, some major classes of solvers are being presented, mentioning their typical use-cases, strengths and weaknesses.

2.3.1 Single-step Methods

Single-step Methods are represented by the fact that for the computation of the new step $\mathbf{x}_{k+1}$, the values of $\mathbf{x}_j,\ j = 0, \ldots, k-1$ are not explicitly needed. Rather, the method performs an averaging of various function slopes that are evaluated on the interval. Probably the most popular single-step methods are of the Runge-Kutta methods family [34]. In the following, its explicit, adaptive as well as implicit variants are presented and discussed.

Explicit We refer to an integration method as explicit when we have no implicit equation that binds the evaluated function slopes values with future function values. The general form of an Explicit Runge-Kutta method is given by [34]

$$\dot{\tilde{\mathbf{x}}}_k = \sum_{i=1}^{s} b_i \mathbf{k}_i, \quad \mathbf{k}_i = \mathbf{f}(\mathbf{x}_k + h \sum_{j=1}^{i-1} a_{ij} \mathbf{k}_j, t_k + h c_i) \tag{2.54}$$

where s denotes the number of stages of the method (the number of function derivative evaluations required for a single step). A convenient way of summarising such a method's coefficients is using the Runge-Kutta matrix (also called as Butcher tableau):

0					
c_2	a_{21}				
c_3	a_{31}	a_{32}			
$\vdots$	$\vdots$		$\ddots$		
c_s	a_{s1}	a_{s2}	$\ldots$	$a_{s,s-1}$	
	b_1	b_2	$\ldots$	b_{s-1}	b_s

Table 2.1: Butcher tableau of an explicit Runge-Kutta integration scheme [34]

For a method to be consistent, it is required that [34]

$$\sum_{j=1}^{i-1} a_{ij} = c_i, \quad \forall i = 2 \dots s \tag{2.55}$$

The analysis of such methods shows that for a given order p of a method, the local truncation error is of the order $\mathcal{O}(h^p)$ while the total accumulated error is of the order $\mathcal{O}(h^{p-1})$. The minimum number of stages s to achieve a certain order of a method is an open question. Nevertheless, for most practical applications, an order smaller than 8 suffices. To that end, the minimum number of stages required for achieving an order up to eight is summarised in the table below:

p	1	2	3	4	5	6	7	8
min s	1	2	3	4	6	7	9	11

Table 2.2: Minimum stages required for achieving a certain truncation error order in Runge-Kutta integration schemes [34]

One of the most popular Runge-Kutta methods is the 4th Order Runge-Kutta (RK4). In many practical applications, it provides a good trade-off between computational complexity and achieved accuracy. Its Butcher tableau is:

0				
1/2	1/2			
1/2	0	1/2		
1	0	0	1	
	1/6	1/3	1/3	1/6

Table 2.3: Butcher tableau of the 4th order Runge-Kutta method [34]

Example 2.11 (Mid-point method). A family of 2nd order methods is given by the Butcher tableau:

0		
α	α	
	$(1 - \frac{1}{2\alpha})$	$\frac{1}{2\alpha}$

Table 2.4: Butcher tableau of 2nd order Runge-Kutta methods [34]

For $\alpha = 1/2$, we obtain the so-called mid-point method. Assuming a one dimensional state and evaluating (2.54) yields

$$k_0 = f(x_k,\ t_k) \tag{2.56a}$$

$$k_1 = f(x_k + \frac{h}{2}k_0,\ t_k + \frac{h}{2}) \tag{2.56b}$$

$$x_{k+1} = x_k + hk_1 \tag{2.56c}$$

For the sake of intuition-creation, Figure 2.12 illustrates geometrically how a mid-point method integration-step is being performed. Note that the error e_1 of the mid-point method is substantially smaller than of an Euler-step e_0 with the same step-size. However, one might argue that a 2-stage method requires two gradient evaluations, and thus it should be compared with the error e_0' relating to an Euler-step method with half the step-size. Note that even in this case, the second order method step still results in a smaller error.

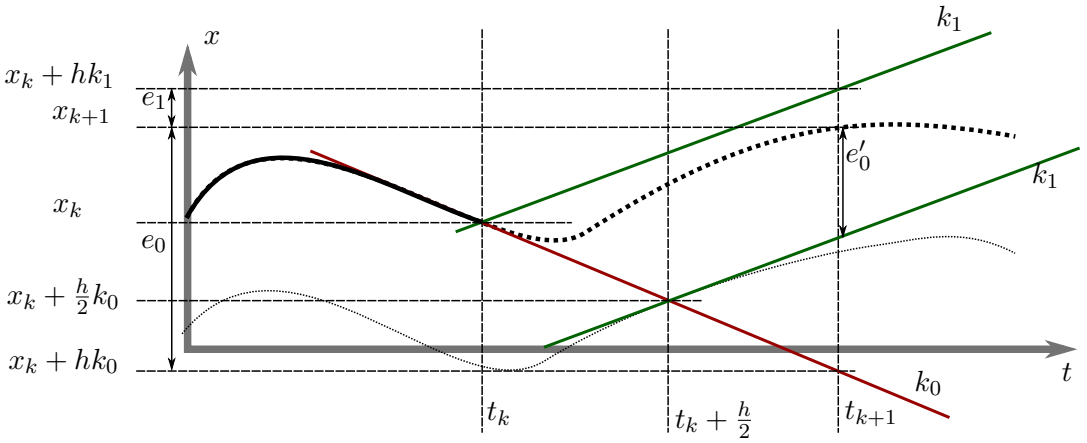


Figure 2.12: Illustration of the gradient computation using the mid-point method

Example 2.12 (Comparison of different explicit methods for a differential-drive model). An alternative to improve the accuracy of simulating the trajectory of a platform is to directly model it in discrete time. For example, the literature [36] proposes a discretized model of the differential drive, based on geometrical considerations and assuming that no accelerations occur in the system for the duration of a step

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} v \\ \omega \end{bmatrix}, \quad \mathbf{x}_{k+1} = \mathbf{x}_k + \begin{bmatrix} \frac{v_k}{\omega_k}(-\sin(\theta_k) + \sin(\theta_k + h\omega_k)) \\ \frac{v_k}{\omega_k}(+\cos(\theta_k) - \cos(\theta_k + h\omega_k)) \\ h\omega_k \end{bmatrix} . \tag{2.57}$$

One can note that a drawback of such a model is the singularity of this representation as $\omega \to 0$. Such models are typically designed to improve the accuracy of numerical integration. In order to evaluate the quality of this model compared to various Runge-Kutta discretization schemes of the continuous differential-drive model (presented in Example 2.1), the models are integrated for the same initial state and input-functions. Moreover, the number of steps used in the integration scheme have been scaled such that the computational expense is similar for all evaluated methods.

Figure 2.13 illustrates the integration of the different motion-models for the discrete model as well as for the continuous model using Euler, Mid-point as well as RK4 discretizations, respectively. As it can be seen, the discrete motion-model is more accurate only when compared to the Euler method. The second and fourth order RK4 methods clearly possess a higher accuracy.

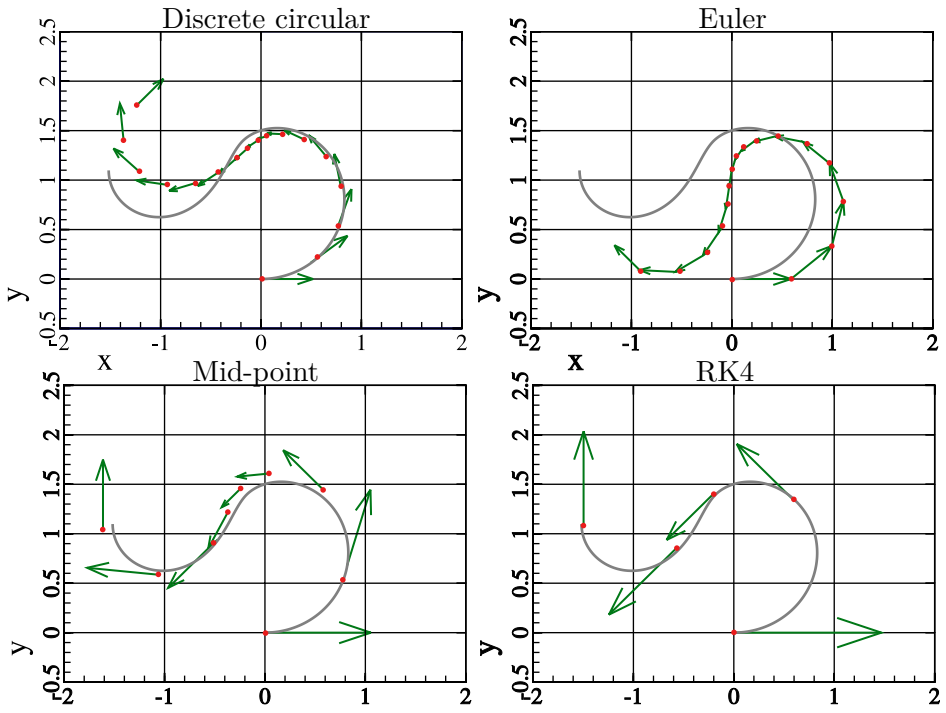


Figure 2.13: Comparison of various discretization schemes used for numerical integration. The gray trajectory represents the ground-truth solution of the ODE.

Adaptive The Runge-Kutta methods presented so far assume a fixed step-size h. The reasoning behind adaptive methods is to choose automatically a suited step-size such that the error (with respect to some error metric) of the scheme is kept within some given bound ϵ. The general idea is to use two methods of different order and to evaluate the error depending on their differences [35]. In order to save computation, the methods are designed such that the evaluation of the function slopes (i. e. the computation of the $\mathbf{k}$ terms) is identical for both. However, a different set of parameters $\mathbf{b}^\star$ is chosen which degrades an order p method into an order $p-1$ method. Thus, the Butcher tableau is:

$$
\begin{array}{c|ccccc}
0 & & & & & \\
c_2 & a_{21} & & & & \\
c_3 & a_{31} & a_{32} & & & \\
\vdots & \vdots & & \ddots & & \\
c_s & a_{s1} & a_{s2} & \dots & a_{s,s-1} & \\
\hline
 & b_1 & b_2 & \dots & b_{s-1} & b_s \\
 & b_1^\star & b_2^\star & \dots & b_{s-1}^\star & b_s^\star
\end{array}
$$

Table 2.5: Butcher tableau of an explicit, adaptive Runge-Kutta integration scheme [35]

With this, the error can be defined as

$$\mathbf{e}_n = h \sum_{i=1}^{s} (b_i - b_i^\star)\mathbf{k}_i \tag{2.58}$$

Note that the truncation error is of the order $\mathcal{O}(h^p)$. Next, we have to decide how to suggest a new step-size given some desired tolerances as well as the previously defined error. We distinguish two types of tolerances [35]:

- constant absolute tolerances: In this case, we are interested in not exceeding a certain constant absolute value of the error for each state dimension $e_i, i = 1, \ldots, \dim(\mathbf{e})$. Thus, given the desired tolerances ε_i, our new step-size can be computed as [35]

$$h' = \min \left\{ h \left| \frac{\bar{\varepsilon}_i}{e_i} \right|^{\frac{1}{p}} \;\middle|\; i = 1, \ldots, \dim(\mathbf{e}),\ \bar{\varepsilon}_i = \varepsilon_i \right\} \quad . \tag{2.59}$$

  It can be shown that when for the next step the step size h' is used, for $h' < h$, it is expected that the tolerances will be maintained [35].

- fractional errors: In some applications, one is interested to maintain the fractional (i. e. relative) errors within some given tolerances. In this case, one can use (2.59) with the modification

$$\bar{\varepsilon}_i = \varepsilon_i x_i \quad . \tag{2.60}$$

  Independent of the type of tolerances that are kept, the scheme advances as follows:

  - $h' < h$: the step is rejected and the re-iterated with $h \leftarrow h'$
  - $h' \geq h$: the (higher-order) step is accepted and for the new iterate, $h \leftarrow h'$

Implicit All the methods presented so far are, as mentioned, explicit methods. Even though more accurate that the naive Euler method, they still possess a small region of absolute stability. In many practical applications this is not of concern. However, if the differential equation is stiff[3], explicit methods typically perform poorly [34]. A concept that greatly increases the stability of ODE solvers is given by implicit methods. For this, consider the simplest implicit method (Backwards Euler)

$$\mathbf{x}_{k+1} = \mathbf{x}_k + h\dot{\mathbf{x}}_{k+1} \quad . \tag{2.61}$$

Note that the equation contains the slope *at the next step*, thus being an implicit equation. Even though such an equation can be sometimes solved in closed-form for $\mathbf{x}_{k+1}$, in general one has to resort to numerical optimization for finding a solution, thus making such methods comparatively slow.

[3] Stiffness of differential equations is a vague term, being quite difficult to formally define. However, an intuitive definition would be that the manifold of the differential equation have drastically different scaling(speeds) in different dimensions.

Going back to the Runge-Kutta methods, the implicit relations that are now present can be encoded again in the Butcher tableau, for which the a terms no longer form a lower-triangular matrix but a full matrix.

c_1	a_{11}	a_{12}	$\dots$	a_{1s}
c_2	a_{21}	a_{22}	$\dots$	a_{2s}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
c_s	a_{s1}	a_{s2}	$\dots$	a_{ss}
	b_1	b_2	$\dots$	b_s
	$b_1^\star$	$b_2^\star$	$\dots$	$b_s^\star$

Table 2.6: Butcher tableau of an explicit, adaptive Runge-Kutta integration scheme [35]

2.3.2 Multi-Step Methods

In single-step methods, there is no usage of the information regarding the previously taken steps, but rather on different evaluations in the evaluated interval. However, in many situations, such multiple slope evaluations for every step can be too computationally expensive (for example in situations where evaluating the function gradient has a quadratic time-complexity). This motivates an alternative approach towards improving the integration scheme, namely to make use of the previously-computed steps.

An informal way of interpreting such methods is to perform a function fit on the last few steps, and to take this function value into account at the evaluated step.

Generally, multi-step methods have the following form [34]

$$\mathbf{x}_k = \mathbf{X}_k\mathbf{a} + h\mathbf{F}_k\mathbf{b}, \tag{2.62}$$

with

$$\mathbf{X}_k = \begin{bmatrix}\mathbf{x}_{k-1} & \mathbf{x}_{k-2} & \dots & \mathbf{x}_{k-s}\end{bmatrix}, \quad \mathbf{F}_k = \begin{bmatrix}\mathbf{f}(\mathbf{x}_k, t_k) & \mathbf{f}(\mathbf{x}_{k-1}, t_{k-1}) & \dots & \mathbf{f}(\mathbf{x}_{k-s}, t_{k-s})\end{bmatrix} \tag{2.63a}$$

$$\mathbf{a} = \begin{bmatrix}a_{s-1} & a_{s-2} & \dots & a_0\end{bmatrix}^{\mathrm{T}}, \quad \mathbf{b} = \begin{bmatrix}b_s & b_{s-1} & \dots & b_0\end{bmatrix}^{\mathrm{T}} \tag{2.63b}$$

Note that for such methods, invariant of the number of stages, the number of function gradient evaluations for every new step is exactly 1. To this end, one might ask why they are not superior to Runge-Kutta methods, given their reduced computational requirements. The answer lies in the convergence proofs and analysis of such methods, which conclude that Multi-step methods have in general a smaller region of stability when compared to Runge-Kutta methods.

Many popular methods (Adams-Bashforth and Adams-Moulton) impose the restriction $a_{s-1} = 1$, $a_{s-2} = a_{s-3} = \dots = a_0 = 0$ [34]. As Adams-Bashforth methods are explicit, additionally they require $b_s = 0$. That is, the simplified law has the following form

$$\mathbf{x}_k = \mathbf{x}_{k-1} + h\mathbf{F}_k\mathbf{b} \tag{2.64}$$

Note that the contents of $\mathbf{F}_k$ are exactly the values of the function derivative of the previous steps. Regarding initialization, the first steps are typically performed using an appropriate singe-step method.

2.4 Computing Sensitivities of ODEs

In many optimization problems, the methods used for finding a solution in this parametric search-space require the sensitivities of the system with respect to the optimization variables. Intuitively, we are interested in finding, how small variations of the optimization variables (parameters) $\mathbf{p}$ influence the shape of the trajectory of the system.

In the following, several methods for computing such sensitivities are presented, along with their strengths and limitations. Moreover, for brevity of notation, we will make use of the Nabla operator to denote the total derivative with respect to the system parameters $\nabla(\cdot) = \frac{\mathrm{d}}{\mathrm{d}\mathbf{p}}(\cdot)$.

2.4.1 Numeric Differences

Probably the most straight-forward and least time-consuming way of computing sensitivities is using numerical differences. Here, we can distinguish between different variants of their computation. For example, the least computationally expensive method is through forward differences

$$\frac{\partial}{\partial p_i}\mathbf{x}(\mathbf{p},t) = \lim_{\epsilon\to 0}\frac{\mathbf{x}(\mathbf{p}+\epsilon\mathbf{e}_i,t)-\mathbf{x}(\mathbf{p},t)}{\epsilon}, \quad i=1,\ldots,\dim(\mathbf{p}), \tag{2.65}$$

followed by central differences

$$\frac{\partial}{\partial p_i}\mathbf{x}(\mathbf{p},t) = \lim_{\epsilon\to 0}\frac{\mathbf{x}(\mathbf{p}+\frac{\epsilon}{2}\mathbf{e}_i,t)-\mathbf{x}(\mathbf{p}-\frac{\epsilon}{2}\mathbf{e}_i,t)}{\epsilon}, \quad i=1,\ldots,\dim(\mathbf{p}) \tag{2.66}$$

where $\mathbf{e}_i$ represents the unit vector.

Remark (Function differentiability when using numerical differentiation)**:** Even though formally, for gradient-based algorithms, we require the evaluated functions to be continuous and at least once (continuously) differentiable, the differentiability property can be omitted when evaluating gradients (sensitivities) using numerical differences. The reason is that even in a region where the true derivative of the function is not defined, a finite-difference computed gradient will "smooth" the derivative with respect to the function neighbourhood, leading to a well-defined approximation of the function derivative.

However, this approach possesses two considerable drawbacks: firstly, their evaluation tends to be time-consuming. Note that the ODE (along with all its associated functions evaluation and complexity) has to be evaluated $\dim(\mathbf{p})$ times. Secondly, it has an accuracy

bound due to finite numerical precision. Especially in nested systems and implementations in which many intrinsic functions occur, this can lead to a considerable loss of accuracy of the resulting sensitivities. These considerations motivate alternative methods for computing the parameter sensitivities.

2.4.2 Analytic

The alternative to numerical differences is to make use of the fact that the mathematical structure of the ODE is known. Thus, calculus provides tools for computing at least analytic partial derivatives. We can approach this analysis by considering two mathematical structures present in an ODE:

- a mathematical relationship that defines the ODE derivative
- a mathematically defined numerical integration method of the ODE

In the following, these different approaches are presented.

Differentiating the ODE integration method An approach towards computing the ODE sensitivities is to differentiate the ODE integrator method. For example, in the case of an Euler integrator (note the general case in which the step-size depends on $\mathbf{p}$)

$$\mathbf{x}_{k+1}(\mathbf{p}) = \mathbf{x}_k(\mathbf{p}) + h(\mathbf{p})\frac{\mathrm{d}\mathbf{x}_k}{\mathrm{d}t}(\mathbf{p}, t) \tag{2.67}$$

we obtain

$$\nabla\mathbf{x}_{k+1}(\mathbf{p}) = \nabla\mathbf{x}_k(\mathbf{p}) + \left(\frac{\partial h}{\partial \mathbf{p}}\right)(\mathbf{p})\frac{\mathrm{d}\mathbf{x}_k}{\mathrm{d}t}(\mathbf{p}, t) + h(\mathbf{p})\left(\frac{\partial}{\partial \mathbf{p}}\frac{\mathrm{d}\mathbf{x}_k}{\mathrm{d}t}\right)(\mathbf{p}, t) \tag{2.68}$$

It means that by having computed $\nabla\mathbf{x}_0(\mathbf{p})$ as well as $\left(\frac{\partial h}{\partial \mathbf{p}}\right)(\mathbf{p})$ and $\left(\frac{\partial}{\partial \mathbf{p}}\frac{\mathrm{d}\mathbf{x}_k}{\mathrm{d}t}\right)(\mathbf{p}, t)$, we can recursively construct the derivative of the system state $\mathbf{x}_n$ with respect to $\mathbf{p}$.

However, such an approach has some limitations. Even though it is relatively straightforward to compute the sensitivities for an Euler discretization method, it is quite obvious that the equations quickly become very complicated and cumbersome when using higher-order integration methods. Perhaps even more discouraging, the designer is required to design new sets of equations for the sensitivities for every different ODE solver method that might be used. The situation is even more discouraging when considering an adaptive-step method, as the adaptation algorithm and the second method used for error control have to be differentiated as well.

Differentiating the ODE Taking the total derivative of (2.50) and applying Leibnitz integral rule, we obtain

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}\mathbf{p}}\mathbf{x}(\mathbf{p},t_1(\mathbf{p})) =&\nabla\mathbf{x}_0(\mathbf{p},t_0(\mathbf{p})) + \int_{t_0(\mathbf{p})}^{t_1(\mathbf{p})} \frac{\partial}{\partial\mathbf{p}}\mathbf{f}(\mathbf{x}(\mathbf{p},t),\mathbf{p},t) + \frac{\partial}{\partial\mathbf{x}}\mathbf{f}(\mathbf{x}(\mathbf{p},t),\mathbf{p},t)\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}\mathbf{p}}(\mathbf{p},t)\mathrm{d}t \\ &+\mathbf{f}(\mathbf{x}(\mathbf{p},t_1(\mathbf{p})),\mathbf{p},t_1(\mathbf{p}))\frac{\mathrm{d}}{\mathrm{d}\mathbf{p}}t_1(\mathbf{p}) \\ &-\mathbf{f}(\mathbf{x}(\mathbf{p},t_0(\mathbf{p})),\mathbf{p},t_0(\mathbf{p}))\frac{\mathrm{d}}{\mathrm{d}\mathbf{p}}t_0(\mathbf{p}) \end{aligned} \tag{2.69}$$

That is, we have to evaluate the coupled (matrix) integral:

$$\begin{aligned}\begin{bmatrix}\mathbf{x}^{\mathrm{T}}\\ \nabla\mathbf{x}^{\mathrm{T}}\end{bmatrix}(\mathbf{p},t) =&\begin{bmatrix}\mathbf{x}_0^{\mathrm{T}}\\ \nabla\mathbf{x}_0^{\mathrm{T}}\end{bmatrix}(\mathbf{p},t_0(\mathbf{p})) + \int_{t_0(\mathbf{p})}^{t_1(\mathbf{p})}\begin{bmatrix}\mathbf{f}(\mathbf{x}(t),\mathbf{p},t)^{\mathrm{T}}\\ \left(\frac{\partial}{\partial\mathbf{p}}\mathbf{f}(\mathbf{x}(t),\mathbf{p},t) + \frac{\partial}{\partial\mathbf{x}}\mathbf{f}(\mathbf{x}(t),\mathbf{p},t)\nabla\mathbf{x}(t)\right)^{\mathrm{T}}\end{bmatrix}\mathrm{d}t \\ &+\begin{bmatrix}\mathbf{0}^{\mathrm{T}}\\ (\mathbf{f}(\mathbf{x}(t_1),\mathbf{p},t_1)\nabla t_1(\mathbf{p}) - \mathbf{f}(\mathbf{x}(t_0),\mathbf{p},t_0)\nabla t_0(\mathbf{p}))^{\mathrm{T}}\end{bmatrix}\end{aligned} \tag{2.70}$$

with $\nabla(\cdot) = \frac{\mathrm{d}}{\mathrm{d}\mathbf{p}}(\cdot)$ This result can be interpreted as follows: by differentiating the ODE, we obtain a higher dimension $\dim(\mathbf{x})(1+\dim(\mathbf{p}))$ ODE, also refered to in the literature as the sensitivity differential equation. When solved, it will contain the state as well as the state sensitivities of the system. The main benefit of using such a method is that we can now easily modify or use any ODE solver method available: more accurate methods will implicitly increase the accuracy of the sensitivities as well.

Example 2.13 (Differential-Drive ODE Sensitivities using piece-wise linear kinematic inputs). In this example, we are interested in computing the sensitivities of the differential-drive kinematic model with state

$$\mathbf{x}^{\mathrm{T}} = \begin{bmatrix} x & y & \theta & v & \omega & s \end{bmatrix} \tag{2.71}$$

and state transition function

$$\dot{\mathbf{x}}^{\mathrm{T}} = \begin{bmatrix} v\cos(\theta) & v\sin(\theta) & \omega & \dot{v} & \dot{\omega} & |v| \end{bmatrix}. \tag{2.72}$$

For simplicity of the emerging equations, we will consider the case in which the agent is allowed to move only forwards, i. e. $v(t) \geq 0, \forall t$.

We assume that the initial state of the system is fixed and its trajectory is defined similar to Example 2.8 by piece-wise linear functions describing the linear and angular velocity of the agent. For simplicity, we choose the functions such that their inflection-points are at the same temporal intervals, which however are still parameters. Thus, the parameter vector can be summarized as

$$\mathbf{p}^{\mathrm{T}} = \begin{bmatrix} v_1 & v_2 & \dots & v_n & \omega_1 & \omega_2 & \dots & \omega_n & t_1 & t_2 & \dots & t_n \end{bmatrix}. \tag{2.73}$$

It is worth noting that the inflection points $v_1,\dots,v_n$ and $\omega_1,\dots,\omega_n$ are not to be mistaken with the system velocities state v and ω. A point that is omitted in many

approaches is to ask whether parts of the system state could be computed in closed-form. This not only would reduce the computational time for solving the ODE (especially for higher order ODE solvers) but also would increase the accuracy of the entire state solution. Given the simple structure of piece-wise linear functions, it follows that in this case, a large portion of the system state can be computed in closed-form. We thus split the state into a part that has to be solved numerically ($\mathbf{x}_n$) and a part that can be computed in closed-form ($\mathbf{x}_c$)

$$\mathbf{x}_n^{\mathrm{T}} = \begin{bmatrix} x & y \end{bmatrix}, \quad \mathbf{x}_c^{\mathrm{T}} = \begin{bmatrix} \theta & v & \omega & s \end{bmatrix}. \tag{2.74}$$

In the following, we will go through the process of computing the sensitivities of the closed-form state and numerical state respectively.

Closed-form State Sensitivities We are interested in obtaining a closed-form solution of $\nabla \mathbf{x}_c (= \frac{\mathrm{d}\mathbf{x}_c}{\mathrm{d}\mathbf{p}})$. Given the considerations from the discussion on differentiating the ODE, if the (matrix) integral could be evaluated in closed-form, a first step would be to compute $\nabla \dot{\mathbf{x}}_c (= \frac{\mathrm{d}\dot{\mathbf{x}}_c}{\mathrm{d}\mathbf{p}})$. As the state linear and angular velocities are uniquely defined by the parametric functions, it follows that the state transition function $\dot{\mathbf{x}}_c$ is of the form $\dot{\mathbf{x}}_c = \mathbf{f}_c(\mathbf{x}_c)$, specific for a system with autonomous dynamics. Thus, for a simpler approach towards computing $\nabla \dot{\mathbf{x}}_c$, we could consider a chaining such as:

$$\nabla \dot{\mathbf{x}}_c = \frac{\partial \dot{\mathbf{x}}_c}{\partial \mathbf{x}_c} \frac{\partial \mathbf{x}_c}{\partial \mathbf{u}_c} \frac{\mathrm{d}\mathbf{u}_c}{\mathrm{d}\mathbf{p}} \tag{2.75}$$

with $\mathbf{u}_c$ – a valid (fictitious) input of the system. A simple choice for such a fictitious input is $\mathbf{u}_c^{\mathrm{T}} = \begin{bmatrix} \dot{v} & \dot{\omega} \end{bmatrix}$. With this, we have:

$$\frac{\partial \dot{\mathbf{x}}_c}{\partial \mathbf{x}_c} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \frac{\mathrm{d}|v|}{\mathrm{d}v} & 0 & 0 \end{bmatrix} \tag{2.76}$$

Choosing two constant matrices:

$$\mathbf{M}_1 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \quad \mathbf{M}_2 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.77}$$

according to Theorem 2.1, we can compute the dynamic matrix $\mathbf{\Phi_c}$ as:

$$\mathbf{\Phi}_c(t_0, t) = \begin{bmatrix} 1 & 0 & t - t_0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & \int_{t_0}^{t} \frac{\mathrm{d}|v(t)|}{\mathrm{d}v} \mathrm{d}t & 0 & 1 \end{bmatrix} \tag{2.78}$$

Using the chaining discussed above, we obtain the closed-form state sensitivities:

$$\nabla \mathbf{x}_c = \mathbf{\Phi}(t_0, t)\nabla \mathbf{x}_{c_0} + \int_{t_0}^{t} \mathbf{\Phi}(\tau, t)\frac{\partial \dot{\mathbf{x}}_c}{\partial \mathbf{u}_c}\frac{\mathrm{d}\mathbf{u}_c}{\mathrm{d}\mathbf{p}}\mathrm{d}\tau \tag{2.79}$$

As in this approach, $\nabla \mathbf{x}_{c_0} = \mathbf{0}$, we are interested in solving the integral term of (2.79). Algebraic evaluation yields:

$$\frac{\mathrm{d}\mathbf{x}_c}{\mathrm{d}v_i} = \begin{bmatrix} 0 \\ \int_{t_0}^{t} \frac{\mathrm{d}\dot{v}(\tau)}{\mathrm{d}v_i}\mathrm{d}\tau \\ 0 \\ \int_{t_0}^{t} (t-\tau)\frac{\mathrm{d}\dot{v}(\tau)}{\mathrm{d}v_i}\mathrm{d}\tau \end{bmatrix}, \frac{\mathrm{d}\mathbf{x}_c}{\mathrm{d}\omega_i} = \begin{bmatrix} \int_{t_0}^{t} (t-\tau)\frac{\mathrm{d}\dot{\omega}(\tau)}{\mathrm{d}v_i}\mathrm{d}\tau \\ 0 \\ \int_{t_0}^{t} \frac{\mathrm{d}\dot{\omega}(\tau)}{\mathrm{d}v_i}\mathrm{d}\tau \\ 0 \end{bmatrix}, \frac{\mathrm{d}\mathbf{x}_c}{\mathrm{d}t_i} = \begin{bmatrix} \int_{t_0}^{t} (t-\tau)\frac{\mathrm{d}\dot{\omega}(\tau)}{\mathrm{d}t_i}\mathrm{d}\tau \\ \int_{t_0}^{t} \frac{\mathrm{d}\dot{v}(\tau)}{\mathrm{d}t_i}\mathrm{d}\tau \\ \int_{t_0}^{t} \frac{\mathrm{d}\dot{\omega}(\tau)}{\mathrm{d}t_i}\mathrm{d}\tau \\ \int_{t_0}^{t} (t-\tau)\frac{\mathrm{d}\dot{v}(\tau)}{\mathrm{d}t_i}\mathrm{d}\tau \end{bmatrix} \tag{2.80}$$

where $\frac{\mathrm{d}\mathbf{x}_c}{\mathrm{d}v_i}$ represents the sensitivities of the closed-form state with respect to the ith control-point representing the linear velocity value v and analogously for ω_i and t_i. It is worth mentioning here that the system still behaves in an autonomous fashion (its motion is uniquely determined by the parametric function shape). The introduction of the fictitious input is solely an artefact that allows the computation of the closed-form sensitivities in a systematic fashion, by making use of Theorem 2.1.

After some (tedious) evaluation, the explicit expressions for the sensitivities of the closed form state variable v, its derivative $\dot{v}$ and its integral s are:

$$\frac{\mathrm{d}}{\mathrm{d}v_i}v(t) = +\sigma(t-t_i)\frac{\min(t, t_{i+1}) - t_i}{t_{i+1} - t_i} - \sigma(t-t_{i+1})\frac{\min(t, t_{i+2}) - t_{i+1}}{t_{i+2} - t_{i+1}} \tag{2.81}$$

$$\frac{\mathrm{d}}{\mathrm{d}v_i}\dot{v}(t) = \begin{cases} +\frac{1}{t_{i+1}-t_i}, & t_i < t \leq t_{i+1} \\ -\frac{1}{t_{i+2}-t_{i+1}}, & t_{i+1} < t \leq t_{i+2} \\ 0, & \text{otherwise} \end{cases} \tag{2.82}$$

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}v_i}\left(\int_{t_0}^{t} v(t)\mathrm{d}t\right) =& -\sigma(t-t_i)\frac{(\min(t, t_{i+1}) - t_i)(t_i - 2t + \min(t, t_{i+1}))}{2(t_{i+1} - t_i)} \\ &+\sigma(t-t_{i+1})\frac{(\min(t, t_{i+2}) - t_{i+1})(t_{i+1} - 2t + \min(t, t_{i+2}))}{2(t_{i+2} - t_{i+1})} \end{aligned} \tag{2.83}$$

$$\frac{\mathrm{d}}{\mathrm{d}t_i}v(t) = \begin{cases} -(v_{i+1} - v_i)\frac{\min(t,t_{i+1})-t_i}{(t_{i+1}-t_i)^2}, & t_i < t \leq t_{i+1} \\ -(v_{i+2} - v_{i+1})\frac{t_{i+2}-\min(t,t_{i+2})}{(t_{i+2}-t_{i+1})^2}, & t_{i+1} < t \leq t_{i+2} \\ 0, & \text{otherwise} \end{cases} \tag{2.84}$$

$$\frac{\mathrm{d}}{\mathrm{d}t_i}\dot{v}(t) = \begin{cases} -\frac{v_{i+1}-v_i}{(t_{i+1}-t_i)^2}, & t_i < t \leq t_{i+1} \\ +\frac{v_{i+2}-v_{i+1}}{(t_{i+2}-t_{i+1})^2}, & t_{i+1} < t \leq t_{i+2} \\ 0, & \text{otherwise} \end{cases} \tag{2.85}$$

$$\frac{\mathrm{d}}{\mathrm{d}t_i}\left(\int_{t_0}^{t} v(t)\mathrm{d}t\right) = \begin{cases} -\frac{(v_{i+1}-v_i)(\min(t,t_{i+1})-t_i)(t_i-2t+\min(t,t_{i+1}))}{2(t_{i+1}-t_i)}, & t_i < t \leq t_{i+1} \\ -\dfrac{\left(\begin{array}{l}(v_i - v_{i-1})t_{i+1}^2 + \\ (v_{i+1} - v_{i-1})(t_i^2 - 2t_i t_{i+1}) + \\ (v_{i+1} - v_i)\left(\begin{array}{l}\min(t, t_{i+1})^2 + \\ 2t(t_{i+1} - \min(t, t_{i+1}))\end{array}\right)\end{array}\right)}{2(t_{i+2}-t_{i+1})}, & t_{i+1} < t \leq t_{i+2} \\ -\frac{v_{i+1}-v_i}{2}, & t > t_{i+2} \\ 0, & \text{otherwise} \end{cases} \tag{2.86}$$

The sensitivities with respect to the angular velocity values control points are not present, as they equate to 0. Note that the above equations hold using variable substitution for the sensitivities of the angular velocity ω, its derivative $\dot{\omega}$ and its integral θ.

Numerical State Sensitivities Having had those sensitivities computed, let us look at the numerical state $\mathbf{x}_n$. In this case, we can consider $\mathbf{u}_n^{\mathrm{T}} = \begin{bmatrix} v & \theta \end{bmatrix}$ as the new fictitious input of the sub-system, as $\dot{\mathbf{x}}_n$ depends only on this closed-form state for which we obtained also closed-form sensitivities. Thus, we have:

$$\frac{\partial \dot{\mathbf{x}}_n}{\partial \mathbf{u}_n} = \begin{bmatrix} \cos(\theta) & -v\sin(\theta) \\ \sin(\theta) & v\cos(\theta) \end{bmatrix} \tag{2.87}$$

Finally, using the chain rule, we obtain the sensitivities of the numerical state derivative:

$$\nabla \dot{\mathbf{x}}_n = \frac{\mathrm{d}\dot{\mathbf{x}}_n}{\mathrm{d}\mathbf{p}} = \frac{\partial \dot{\mathbf{x}}_n}{\partial \mathbf{u}_n} \begin{bmatrix} \nabla v \\ \nabla \theta \end{bmatrix} \tag{2.88}$$

which can be integrated together with the numerical state, according to (2.70).

3 Optimal Planning and Control

In the context of control-theory, the typical goal of controlling a system relates to means by which the inputs of the system are actuated such that the system behaves in a desired way. One generic approach for computing such system inputs relates to finding a *feedback-law*. As the name suggests, the task of the controller designer is to find a (mathematical) relationship between measurable outputs and inputs of the system such that the system behaves as desired. Typical families of methods for obtaining such control laws include Linear Control [37], Lyapunov-Based Control [38] as well as Exact Input-Output Linearisation [27]. All these methods possess the mathematical apparatus of design and analysis such that the resulting control-law behaves sufficiently well, i.e. they possess the characteristic of (asymptotically) (exponentially) stabilising the system. However, virtually any system that is desired to be controlled possesses constraints, at least on the magnitude of the applicable system inputs. However, due to the nature of such methods, systematically accounting for such constraints is in general impossible.

An alternative approach towards controlling a system is to make use of optimization. In the following, the general constructs related to optimization as well as the methods through which it is used in the context of control are presented.

3.1 Definitions and Models

3.1.1 Dynamic Models

For the purpose of this work, let us consider trajectory-planning for systems that are represented by a dynamic model (related to an ODE). Moreover, without loss of generality, navigation is a task strongly connected with the concept of an environment, more specifically free space, i. e. locations in the pose space of the agent that are collision-free.

In general, we possess modelling techniques that allow us to describe, with a relatively high accuracy, the behaviour of the agent. On the contrary, modelling unstructured dynamic environments with respect to dynamics is a lot more challenging. Thus, even though in general the agent and the environment model are coupled into a system model $\mathbf{x}_{sys}$, we choose to split the system into the agent $\mathbf{x}$ and environment $\mathbf{z}$, $\mathbf{x}_{sys}^{\mathrm{T}} = \begin{bmatrix}\mathbf{x}^{\mathrm{T}} & \mathbf{z}^{\mathrm{T}}\end{bmatrix}$, in order to be able to focus on the environment sub-system later-on.

For the beginning, let us define the dynamics of the system in a continuous time formulation. However, keep in mind that the following analysis can be easily transformed to a discrete

time system.

$$\dot{\mathbf{x}}(t) = \hat{\mathbf{f}}_{\mathbf{x}}(\mathbf{x}(t), \mathbf{u}(t)) + \mathbf{w}_{\mathbf{f}_{\mathbf{x}}}(t) \ , \ \mathbf{x}(0) = \hat{\mathbf{x}}_0 + \mathbf{w}_{\mathbf{x_0}} \tag{3.1a}$$

$$\dot{\mathbf{z}}(t) = \hat{\mathbf{f}}_{\mathbf{z}}(\mathbf{z}(t), \mathbf{x}(t), t) + \mathbf{w}_{\mathbf{f}_{\mathbf{z}}}(t) \ , \ \mathbf{z}(0) = \hat{\mathbf{z}}_0 + \mathbf{w}_{\mathbf{z_0}} \tag{3.1b}$$

with the modelled (nominal) state transition function $\hat{\mathbf{f}}$, agent control input $\mathbf{u}$ and process noise $\mathbf{w}$.

Moving horizon trajectory generators (controllers) relate to optimization or sampling *into the future*. Thus, it is convenient to define a short-hand notation for the solutions of a system ordinary differential equations (ODE).

Definition 3.1 (System Trajectory)**.** Given a system with the state $\mathbf{x} \in \mathbb{R}^n$, control input $\mathbf{u} \in \mathbb{R}^m$ satisfying the ODE $\dot{\mathbf{x}} = \mathbf{f}_{\mathbf{x}}(\mathbf{x}, \mathbf{u}, t)$, we define the trajectory of the system $\mathbf{x}$ as the vector-valued function $traj_{\mathbf{x}}$ as

$$\begin{aligned} &traj_{\mathbf{x}}(\mathbf{x}_0, t_0, \mathbf{w}_{\mathbf{x_0}}, \mathbf{u}, t_1, \mathbf{w}_{\mathbf{f}_{\mathbf{x}}}) \\ &\quad = \left\{ \mathbf{x} \mid \mathbf{x}(t_0) = \hat{\mathbf{x}}_0 + \mathbf{w}_{\mathbf{x_0}} \text{ and } \dot{\mathbf{x}}(t) = \hat{\mathbf{f}}_{\mathbf{x}}(\mathbf{x}(t), \mathbf{u}(t), t) + \mathbf{w}_{\mathbf{f}_{\mathbf{x}}}(t), \ \forall t \in [t_0, t_1] \right\}. \end{aligned} \tag{3.2}$$

Moreover, we define the *nominal* trajectory of the system as

$$traj^{\mathcal{N}}_{\mathbf{x}}(\mathbf{x}_0, t_0, \mathbf{u}, t_1) = traj_{\mathbf{x}}(\mathbf{x}_0, t_0, \mathbf{0}, \mathbf{u}, t_1, \mathbf{0}). \tag{3.3}$$

In practice, $traj_{\mathbf{x}}$ and $traj_{\mathbf{z}}$ are computed using various types of ordinary differential equation solvers. A more in-depth discussion about the numerical algorithms is given in Section 2.3.

3.1.2 Constraints

Every system is constrained (actuator limits, ranges of operation in which the model is accurate enough, constraints induced by desired mode of operation etc.). Being able to account for arbitrary non-linear constraints in an explicit manner is one of the key benefits of using dynamic (non-linear) optimization. Thus, the system with the state $\mathbf{x}$, additional to the state transition constraint $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}, t)$, will possess a general set of equality constraints $\boldsymbol{\psi}$ and inequality constraints $\mathbf{h}$ in the form

Definition 3.2 (Dynamic System (Nominal) Constraints)**.** Given a system of the form (3.1a), as well as the vector-valued equality constraints $\boldsymbol{\psi}$ and inequality constraints $\mathbf{h}$, we define the constraints $C_{\mathbf{x}}$ of the dynamic system with state $\mathbf{x}$ as

$$C_{\mathbf{x}}(\mathbf{x}_0, t_0, \mathbf{u}, t_1) = \begin{cases} \textbf{true}, & \text{if} \quad \begin{aligned} &\mathbf{x} = traj^{\mathcal{N}}_{\mathbf{x}}(\mathbf{x}_0, t_0, \mathbf{u}, t_1), \\ &\boldsymbol{\psi}(\mathbf{x}(t_1), t_1) = \mathbf{0}, \\ &\mathbf{h}(\mathbf{x}(t), \mathbf{u}(t), t) \geq \mathbf{0}, \quad \forall t \in [t_0, t_1] \end{aligned} \\ \textbf{false}, & \text{otherwise} \end{cases} \tag{3.4}$$

Moreover, we define the set of all feasible solutions of a constraint satisfying program as

$$\mathcal{SOL}_{\mathbf{x}}(\mathbf{x}_0, t_0) = \left\{ \begin{bmatrix} \mathbf{u} & t_1 \end{bmatrix} \;\middle|\; C_{\mathbf{x}}\left(\mathbf{x}_0, t_0, \mathbf{u}, t_1\right) = \mathbf{true} \right\}. \tag{3.5}$$

Note that this formulation allows equality constraints evaluated only on the end state $\mathbf{x}(t_1)$. However, this is not a restrictive formulation, as any equality constraint $\mathbf{g}$ that is active at every state along the trajectory can be included through the inequality constraints $\mathbf{h} \supseteq \mathbf{h}_{eq}$, as $\mathbf{h}_{eq} = \{\mathbf{g} \geq \mathbf{0}\} \cup \{-\mathbf{g} \geq \mathbf{0}\}$.

3.1.3 Cost Function

The goal of a large set of problems, including vehicle navigation, is to behave (navigate) optimally with respect to some criteria. In its most general form, the criteria for the trajectories of the system are multiple. For example, in the case of navigation, it might be desired to navigate time-optimally to the goal while reducing accelerations in the system. The concept of optimization and finding optimal trajectories with respect to multiple criteria is denoted in the literature as multi-objective optimization. However, the search, evaluation and selection of trajectories with respect to multiple cost-functions becomes considerably harder and more expensive to evaluate, as one has to find multiple optimal solutions that relate to the multi-objective Pareto front.

Because of this, this work addresses the relaxed problem of finding optimal solutions for the single-objective optimization problem, which in practice can be achieved by combining the (vectorial) multi-objectives e.g. through the use of a generic inner product. Thus, the single objective cost function of a dynamic system can be formulated as

$$J(\mathbf{u}, t_1) = \varphi(\mathbf{x}(t_1), t_1) + \int_{t_0}^{t_1} l(\mathbf{x}(t), \mathbf{u}(t), t)\, \mathrm{d}t, \tag{3.6}$$

with the cost function J, the terminal cost φ and the Lagrange density function l.

Having the above definitions and assumptions, we can now define the optimization problem of the system

Definition 3.3 (Dynamic Optimization Problem)**.** Given is a system of the form (3.1a), together with system constraints and cost-function as defined in (3.4) and (3.6) respectively. The solution $\mathbf{u}^*, t_1^*$ of the optimization problem:

$$\min_{\mathbf{u}, t_1} \quad J(\mathbf{u}, t_1) \quad \text{s.t.} \quad C_{\mathbf{x}}(\mathbf{x}_0, t_0, \mathbf{u}, t_1) = \mathbf{true} \tag{3.7}$$

when applied to the ideal system with state $\mathbf{x}$, will result in the optimal trajectory of the system.

Note that in (3.7), the system control input function $\mathbf{u}$ as well as the duration of the trajectory (through t_1) are optimization variables. Thus, in theory, with such a formulation, any arbitrary navigation task can be solved, the solution of the problem representing the optimal trajectory of the system and its control input function.

3.2 Optimization-based Control

The previous section presented the general concept and notations used when dealing with optimization problems. In this section, we will take a closer look at how optimization can be used to control a system. To that end, we will constructively analyse different assumptions regarding the quality of our system models and possible methods of optimal-control.

3.2.1 Infinite Optimization Horizon, Perfect Env. Model ($t_1 \to \infty,\ \mathbf{w_z} = \mathbf{0}$)

We will first take a look at the situation in which we possess sufficient expressibility and computational power to be able to find optimal solutions of arbitrary length. Moreover, we will assume that we precisely know how the environment state is at all times $t > t_0$. From here, we differentiate:

Perfect agent model ($\mathbf{w_x} = \mathbf{0}$): With this, we would in theory be able to solve any navigation task by solving an optimization problem once. As all the models are exact, we could trivially feed-forward the computed agent inputs and the optimal trajectory would be executed.

Of-course, such a setting is highly unreasonable. Thus, we advance to:

Imperfect agent model ($\mathbf{w_x} \neq \mathbf{0}$): This setting becomes reasonable when for example an agent is navigating in an environment without obstacles and has a navigation task that is relatively short. To this end, we could solve the optimization problem once using the nominal model of the agent. This would then become the reference trajectory of the agent. In order to account for the agent model uncertainties, a classical (static) controller can be used to stabilise it and perform trajectory tracking control.

Keep in mind, however, that the underlying feed-back controller will in general actuate the system inputs differently than how their nominal value was computed in the solution of the optimization problem. This implies that certain constraints which the optimisation-problem satisfies, might be violated. A simple (empirical) remedy to this problem is to enforce conservative bounds on the constraints of the optimization problem. A more careful approach would be to take into account bounds of the agent model uncertainties and modify the enforced constraints accordingly.

Nevertheless, even this setting is often unsatisfactory, as in practice we are interested in autonomous navigation of arbitrary length, resulting in the input function of the agent

ODE requiring an arbitrary expressibility. However, this is typically inapplicable as the computational complexity of the optimization-problem becomes intractable.

3.2.2 Finite Optimization Horizon, Perfect Env. Model ($t_1 < t_1^{max},\ \mathbf{w_z} = \mathbf{0}$)

An approach to circumvent this problem is to iteratively solve the optimization problem while taking into account a maximum temporal horizon into the future. This concept is known in the literature as Moving Horizon Trajectory Planning (MHTP) [39]. Of course, this leads to globally sub-optimal trajectories. However, the resulting trajectories quality increases with the increase of the optimization horizon, while such a horizon bounds the computational complexity of the optimization problem. This has led to many applications making use of such an approach with great success. Moreover, iteratively re-solving the optimization problem is already giving a hint that such an approach could take into account partial knowledge of the environment as well as reactive (adaptive) behaviour.

A popular method in which MHTP is used in the context of control is represented by Model Predictive Control (MPC) [40, 41]. In this approach, the optimization problem is solved iteratively at discrete time intervals t_k while changing the initial state of the agent according to measurements and(or) observers of its state. At the end of each iteration, the initial interval of the computed control command is applied. It has been proven that, under certain assumptions, this leads to an (exponentially) asymptotically stable behaviour of the system [42]. More informally, the iterative solution of the optimization problem with updated initial state creates an implicit feed-back loop that leads under certain assumptions to (exponential) asymptotic stability.

An alternative to MPC is to perform MHTP without updating the initial state of the agent. This relies on the assumption of the MHTP that the agent follows exactly the computed trajectory in the interval between two iterative solves. In order to stabilise the agent along such a trajectory, a static trajectory-following feed-back controller is used.

Theorem 3.1 (Stabilised MHTP)**.** *Given is a system of the form (3.1a) with initial condition $\hat{\mathbf{x}}_{0_0}$ ($\hat{\mathbf{x}}_0$ at iterate 0) at time t_{0_0}. Moreover, a MHTP solving an optimization of the form (3.7) is run at discrete time intervals t_{0_k} ($k \geq 0, t_{0_{k+1}} > t_{0_k}$) with the solution $\hat{\mathbf{x}}_k^*(\cdot)$ such that*

$$\hat{\mathbf{x}}_{0_{k+1}} = \hat{\mathbf{x}}_k^*(t_{0_{k+1}}). \tag{3.8}$$

If the system is controlled by an (exponentially) asymptotically trajectory tracking controller of the form

$$\mathbf{u}(t) = \mathbf{k}(\mathbf{x}(\tau), \mathbf{x}_d(\tau), \tau), \quad \tau \leq t, \tag{3.9}$$

with a static desired system trajectory $\mathbf{x}_d(t)$, then the feed-back law

$$\mathbf{u}(t) = \mathbf{k}(\mathbf{x}(\tau), \hat{\mathbf{x}}_k^*(\tau), \tau), \quad \forall k \geq 0,\ t \in [t_{0_k}, t_{0_{k+1}}],\ \tau \leq t \tag{3.10}$$

(exponentially) asymptotically stabilises the system along the MHTP-computed trajectory.

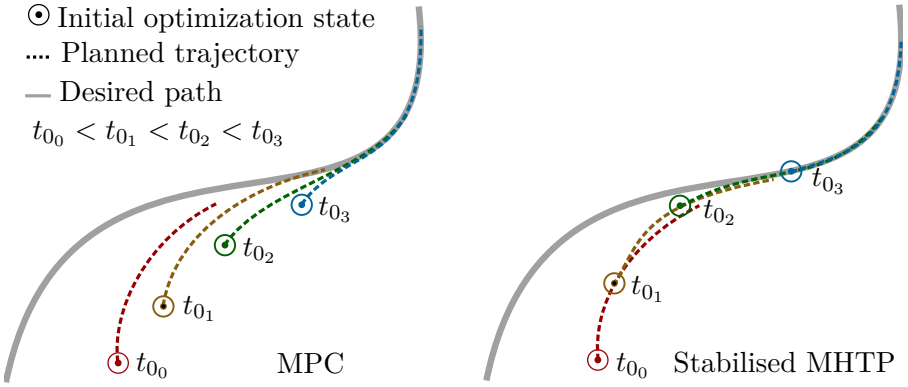


Figure 3.1: Measured states, iteratively-planned and resulting trajectories for MPC and Stabilised MHTP using an inexact motion model

Proof. (sketch) The main argument of the proof lies in the fact that the chosen controller does not require evaluation of the desired trajectory $\mathbf{x}_d(t)$ at future times. Moreover, the MHTP is iterated such that at a new iterate $k+1$, the initial condition is precisely the previously computed optimal solution $\hat{\mathbf{x}}_k^*$ at time $t_{0_{k+1}}$. This implies that from the perspective of the feed-back controller (3.9), the function $\hat{\mathbf{x}}_k^*(t),\ \forall k \geq 0,\ t \in [t_{0_k}, t_{0_{k+1}}]$ is static. An intuitive interpretation can be given by the fact that the MHTP computes new parts of the trajectory just in time, its value between iterates k and $k+1$ on the interval $[t_{0_k}, t_{0_{k+1}}]$ remaining constant and consistent with the initial conditions of the $k+1$ iterate. □

Figure 3.1 illustrates the MPC and the stabilised MHTP approaches that lead to asymptotic stability. As it can be seen in the image, one of the benefits of Stabilized MHTP is the fact that model inaccuracies are compensated by the lower-level control loop. Moreover, this allows the feedback frequency to be considerably higher, as the controller (3.9) is in general a lot less computationally intensive. However, the drawback of such an approach lies in the fact that input constraints are not guaranteed to be satisfied any more.

3.2.3 Classification of Imperfect Environment Models

In all previous scenarios, we have assumed that the environment model is exact. Even in this setting, it is not necessarily trivial to guarantee that the navigation algorithm will always be collision free (safe). However, the assumption of a perfect environment model is in practice far from reasonable. Targeting unstructured environments, it is intuitively clear that the models of the environment will possess inaccuracies of large magnitude in comparison with the inaccuracies of the agent model. It is thus expected that guaranteeing safety of navigation becomes considerably more complicated. Nevertheless, one of the main interests of this work is to devise a generic approach that is *proven* to be safe, a characteristic that is required in most industrial applications as well as autonomous driving before being accepted and put into practice.

In the following, we will analyse different environment assumptions and the imperfect models that emerge.

Static known environment This assumption is probably the most restrictive and easiest to deal with.

$$\dot{\mathbf{z}} = \mathbf{0}, \quad \mathbf{z}(t) = \mathbf{z}, \quad \mathbf{z}(0) = \mathbf{z}_0 \tag{3.11a}$$

$$\mathbf{y}_\mathbf{z} = \mathbf{h}_\mathbf{z}(\mathbf{z}) + \mathbf{v}_\mathbf{z} \tag{3.11b}$$

In this case, all the obstacles states $\mathbf{z}$ are static and known, their observation noise closely relating to the quality of the provided map.

Static observable environment As in the previous model, the dynamics of the environment are non-existent. However, the environment (or parts of it) are only partially observable, the observation function being dependent on the state of the agent sensors.

$$\dot{\mathbf{z}} = \mathbf{0}, \quad \mathbf{z}(t) = \mathbf{z} \tag{3.12a}$$

$$\mathbf{y}_\mathbf{z}(t) = \mathbf{h}_\mathbf{z}(\mathbf{z}, \mathbf{x}(t)) + \mathbf{v}_\mathbf{z}(t) \tag{3.12b}$$

Even though still naive with respect to the dynamics, such a model becomes feasible in many applications, where the environment dynamics is slow with respect to the agent dynamics. Moreover, by assuming that the agent sensors are reliable, such a model will allow to perform agent-centric navigation guaranteeing collision avoidance.

Dynamic autonomous observable environment In this model, we allow the environment to change over time. However, we assume that the evolution of the environment is not influenced by the trajectory of the agent.

$$\dot{\mathbf{z}}(t) = \mathbf{f}_\mathbf{z}(\mathbf{z}(t), t) + \mathbf{w}_\mathbf{z}(t) \tag{3.13a}$$

$$\mathbf{y}_\mathbf{z}(t) = \mathbf{h}_\mathbf{z}(\mathbf{z}(t), \mathbf{x}(t)) + \mathbf{v}_\mathbf{z}(t) \tag{3.13b}$$

Such a model is desired to reduce computational complexity in finding a solution to the navigation task as well as make use of environment dynamic models that are decoupled with respect to the agent. Note that without additional assumptions, this model cannot guarantee safe navigation of the agent. An intuitive example would be the situation in which an agent is stationary in the corner of a room. Such a model will allow dynamic parts of the environment (e.g. other vehicles) to move towards the agent and ultimately collide, situation which in general cannot be avoided by the agent.

Dynamic non-intrusive observable environment In indoors navigation, a reasonable assumption of long-term safety of an agent is to consider that a stationary agent is in a safe state (i. e. the environment will not collide with it when it does not move). In the

automotive domain, a similar safe state would be a stationary vehicle on the emergency lane of a high-way. In the topic of fixed-wing Unmanned Aerial Vehicles (UAV), such safe states could be periodic circular motions (loiter manoeuvres) in certain locations.

All those considerations require the environment to posses certain characteristics, namely to be *non-intrusive* when such safe states are reached by the agent. In the following, we are interested to analyse this concept more formally and ultimately provide a generic constraint programming formulation that guarantees navigation safety assuming the dynamic non-intrusive observable environment model.

3.3 Constraints for Ensuring Safety

In a nutshell, the general constraint that we would like to enforce is that the agent is always in a valid state, i. e. its state is collision-free. We define $\mathcal{F}(\mathbf{z}(t))$ as the collision-free set of agent states with respect to the environment $\mathbf{z}(t)$ at time t. With this definition in mind, the constraint that we want to enforce is

$$\mathbf{x}(t) \in \mathcal{F}(\mathbf{z}(t)), \quad \forall t \in [t_0, \infty). \tag{3.14}$$

Even though succinct and general, this formulation has some major drawbacks: It requires the knowledge of the true environment state $\mathbf{z}(t),\ \forall t \in [t_0, \infty)$. In practice, this is rarely the case even for current time $t \to t_0$. Also, without any additional assumptions regarding the environment, there is no guarantee that there exists a solution to (3.14).

3.3.1 Notation and Related Concepts

In the following, we will define several concepts that will allow to realistically and feasibly evaluate (3.14). They will also introduce the main additional assumptions that have to be taken regarding the nature of the environment $\mathbf{z}(t)$.

Assuming noise in the environment model

Depending on the applicability domain, environments can be arbitrarily complex; the design of qualitative environment models is a rich field of research of its own. However, all such models will possess inaccuracies, especially when predicting their state into the future. Nevertheless, we want to make use of their nominal prediction quality. Thus, we define the *nominal environment state* as follows $traj_{\mathbf{z}}^{\mathcal{N}}$

$$traj_{\mathbf{z}}^{\mathcal{N}}(\mathbf{z}_0, t_0, \mathbf{x}(\cdot), t_1) = traj_{\mathbf{z}}(\mathbf{z}_0, t_0, \mathbf{0}, \mathbf{x}(\cdot), t_1, \mathbf{0}). \tag{3.15}$$

Note that this set would coincide with the true free-space set if our environment model is perfect $\mathbf{w}_{\mathbf{z}} = \mathbf{0}$. However, in all other cases, generally $\mathcal{F}(traj_{\mathbf{z}}^{\mathcal{N}}(\mathbf{z}_0, t_0, \mathbf{x}(\cdot), t)) \not\subseteq \mathcal{F}(\mathbf{z}(t))$. In order to circumvent this problem, we require that our environment simulator also provides a function $C_{\mathbf{w}_{\mathbf{z}}}$ that enforces the bounds of the environment model initial state

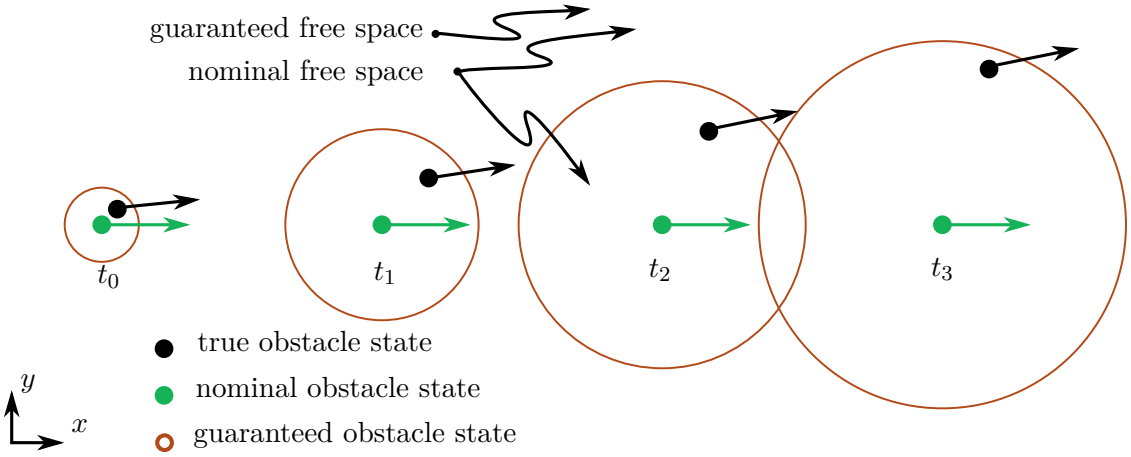


Figure 3.2: Nominal versus guaranteed free space for several future simulated time-steps

error and process noise. With this, we can define the *guaranteed free space environment state* as

$$\begin{aligned} & traj^{\mathcal{G}}_{\mathbf{z}}(\hat{\mathbf{z}}_0, t_0, \mathbf{x}(\cdot), t_1) = \mathbf{z} \\ \text{s.t.} \quad & \mathcal{F}(\mathbf{z}) = \bigcap_{\mathbf{z}_0, \mathbf{w}_{\mathbf{z}}(\cdot)} \mathcal{F}(traj_{\mathbf{z}}(\mathbf{z}_0, t_0, \mathbf{w}_{\mathbf{z}0}, \mathbf{x}(\cdot), t_1, \mathbf{w}_{\mathbf{z}}(\cdot))) \\ & C_{\mathbf{w}_{\mathbf{z}}}(\hat{\mathbf{z}}_0, \mathbf{z}_0, t_0, \mathbf{w}_{\mathbf{z}}(t), t) \leq \mathbf{0}, \ \forall t \in [t_0, t_1] \end{aligned} \tag{3.16}$$

(3.16) can be interpreted as follows: given all allowed variations of the initial state and process noise, it returns the environment state having the free space set valid in all variations, i. e. the free space that is guaranteed. Of course, in practice, implementations of computing $traj^{\mathcal{G}}_{\mathbf{z}}$ would make use of simple noise models and stochastics for efficient computation. However, this could lead to relatively conservative bounds on the environment model error, which would result in $\mathcal{F}(traj^{\mathcal{G}}_{\mathbf{z}})$ strongly shrinking even after short periods of simulation. Figure 3.2 illustrates the dual of the free space set (namely the obstacle set) for a nominal vs. guaranteed simulation.

Reducing the constraint evaluation for infinite time

The general idea is to assume that there exists a set of agent safe states that are free and time-invariant with respect to the environment. This implies that if the agent enters in such a state at an arbitrary time, it can always remain in this set and be guaranteed free of collisions. Similarly to [14], we refer to such a set as *feasible invariant* $\mathcal{FI}$

$$\begin{aligned} \mathcal{FI}(\mathbf{z}_0, t_0) = \{ \ \mathbf{x}_s \mid \exists \mathbf{u}(\cdot) \text{ s.t. } & \mathbf{x}(t) = traj^{\mathcal{N}}_{\mathbf{x}}(\mathbf{x}_s, t_0, \mathbf{u}(\cdot), t), \\ & \mathbf{z}(t) = traj^{\mathcal{G}}_{\mathbf{z}}(\mathbf{z}_0, t_0, \mathbf{x}(\cdot), t), \\ & C_{\mathbf{x}}(\bar{\mathbf{x}}, t_0, \mathbf{u}(\cdot), t) = \textbf{true}, \\ & \mathbf{x}(t) \in \mathcal{F}(\mathbf{z}(t)), \ \forall t \geq t_0 \}. \end{aligned} \tag{3.17}$$

This formulation comes in handy, as for many environment models, computing a subset of the true feasible invariant set is reasonably straight-forward. One such example is to assume the stationary state of the agent (for example zero velocity for a differential drive) for every free pose point with respect to a static map. This would apply, however, some constraints on the dynamics of the environment, namely no obstacle should collide in the future with the agent stopping at an arbitrary free location in the static map.

We invite the reader to consider this fact the other way around: by defining a feasible invariant set of the environment, one constrains the type of the environment in an arbitrary way; however, as we will see in the next section, it will imply the existence of a solution for the moving horizon trajectory planning.

Having introduced the above notations, we can now formally define the dynamic non-intrusive observable environment

$$\dot{\mathbf{z}}(t) = \mathbf{f}_{\mathbf{z}}(\mathbf{z}(t), \mathbf{x}(t), t) + \mathbf{w}_{\mathbf{z}}(t) \tag{3.18a}$$

$$\mathbf{y}_{\mathbf{z}}(t) = \mathbf{h}_{\mathbf{z}}(\mathbf{z}(t), \mathbf{x}(t)) + \mathbf{v}_{\mathbf{z}}(t) \tag{3.18b}$$

$$\mathcal{FI}(\mathbf{z}(\tau), \tau) \neq \emptyset \tag{3.18c}$$

$$\mathcal{FI}(\mathbf{z}(\tau), \tau) \subseteq \mathcal{F}(\mathbf{z}(\tau)), \quad \forall \tau \in [t_0, \infty). \tag{3.18d}$$

An additional notation that we would like to consider is the trajectories bundle that can move the agent from a state $\mathbf{x}_0$ to a feasible invariant state $\mathcal{TFI}$

$$\begin{aligned}
&\mathcal{TFI}(\mathbf{x}_0, \mathbf{z}_0, t_0) = \\
&\{\ \mathbf{x}(\cdot) \mid \exists \mathbf{u}(\cdot), t_1 \text{ s.t. } \mathbf{x}(t) = traj_{\mathbf{x}}(\mathbf{x}_0, t_0, \mathbf{u}(\cdot), t), \\
&\qquad\qquad \mathbf{z}(t) = traj_{\mathbf{z}}^{\mathcal{G}}(\mathbf{z}_0, t_0, \mathbf{x}(\cdot), t), \\
&\qquad\qquad C_{\mathbf{x}}(\mathbf{x}_0, t_0, \mathbf{u}(\cdot), t) = true, \\
&\qquad\qquad \mathbf{x}(t) \in \mathcal{F}(\mathbf{z}(t)), \\
&\qquad\qquad \mathbf{x}(t_1) \in \mathcal{FI}(\mathbf{z}(t_1), t_1),\ \forall t \in [t_0, t_1]\ \}.
\end{aligned} \tag{3.19}$$

3.3.2 Collision-free Navigation Constraints

In this section, we will make use of the definitions and assumptions from the previous section in order to guarantee safety of a control law devised by a MHTP. Similar approaches for ensuring solutions of a MHTP are referred in literature as Robust MPC [17]. The concept behind such approaches is to provide conservative bounds of the models uncertainty and to constrain the solutions such that given all allowed uncertainties, it is guaranteed that the constraints will not be violated. However, in the following, we will also provide a formulation which possesses a tighter bound of the emerging motions, allowing to plan trajectories of virtually arbitrary length. We will make in the following assumptions regarding the MHTP:

- A1: the search for a control input $\mathbf{u}(\cdot)$ is performed at discrete time-steps at temporal intervals T_s

- A2: MHTP computes a solution in zero time
- A3: the agent model is exact
- A4: the low-level controller executes exactly the command $\mathbf{u}_k(t)$ at time t, with $\mathbf{u}_k(\cdot)$ representing the input commands computed at time t_{0_k} that result in a valid trajectory

The assumption A3 is in most of the cases reasonable, as the uncertainty of the agent model is negligible in comparison with the environment uncertainty ($\mathbf{w_x} \ll \mathbf{w_z}$). Note that by assuming A3, it is easy to drop the assumption A2 and assume that MHTP computes a solution in at most T_s time. However, we require A2 for the sake of a cleaner formulation. A more in-depth discussion regarding non-negligible computation-time is given in Section 4.4.

Under these assumptions, we would like to define a set of constraints such that if there exists a control input $\mathbf{u}_0(\cdot)$ that satisfies them at the initial iteration of the MHTP (t_{0_0}), there will exist a control input $\mathbf{u}_k(\cdot)$ that satisfies them $\forall k > 0$. This allows us to provide a safety guaranteeing constraint for moving horizon trajectory planning for finite horizon ($t_1 < \infty$) and to account for environment process noise ($\mathbf{w_z}(\cdot) \neq \mathbf{0}$) as follows:

Theorem 3.2. *Worst-case Safety Constraints*

Given are an agent and environment system of the form (3.1a),(3.1b) as well as guaranteed bounds of the environment initial state error and process noise, enforced through the constraint $C_{\mathbf{w_z}}$. Moreover, the agent is controlled using a MHTP that satisfies A1-A4. If

$$\mathcal{TFI}(\mathbf{x}_{0_k}, \hat{\mathbf{z}}_{0_k}, t_{0_k}) \neq \emptyset \tag{3.20}$$

at the initial MHTP iteration ($k = 0$) for the initial conditions $\mathbf{x}_{0_0}$, $\hat{\mathbf{z}}_{0_0}$, t_{0_0}, there will exist a solution $\mathbf{u}_{\mathbf{x}_k}(\cdot)$, t_{1_k} that satisfies (3.20) $\forall k \geq 0$.

Proof. By construction, $\mathcal{TFI}$ represents the set of agent trajectories starting from a state consistent with the initial agent state $\hat{\mathbf{x}}_0$ that are guaranteed to be collision-free and ending in a feasible invariant set. It implies that for any following iteration, there exists at least the valid trajectory computed in the initial iteration that is valid and collision-free. □

Even though this formulation is safe, it possesses one drawback, namely it requires guaranteed simulation of the environment until the terminal state at time t_1. As discussed previously, due to the conservative bounds in the environment model uncertainty, this will lead to a considerable shrinkage of the free space with respect to the guaranteed simulation in comparison with the free space of the real environment state. This implies that for satisfying non-collision, t_1 will be constrained by the "inflation" of the predicted obstacles state. Conversely, this formulation makes no use of the quality of our nominal environment simulator, but rather of the guaranteed simulator. This motivates an alternative formulation of the safety constraints:

Theorem 3.3. *Nominal-case Safety Constraints*

Given are an agent and environment system of the form (3.1a),(3.1b) as well as guaranteed bounds of the environment initial state error and process noise, enforced through the constraint $C_{\mathbf{w_z}}$. Moreover, the agent is controlled using a MHTP that satisfies A1-A4. If there exists a solution $\mathbf{u}_{\mathbf{x}_0}(\cdot),\ t_{1_0}$ that satisfies:

$$\begin{aligned}
&\mathbf{x}(t) = traj_{\mathbf{x}}(\mathbf{x}_{0_k}, t_{0_k} \mathbf{u}_{\mathbf{x}_k}(\cdot), t),\\
&\mathbf{z}_{t-T_s} = traj_{\mathbf{z}}^{\mathcal{N}}(\hat{\mathbf{z}}_{0_k}, t_{0_k}, \mathbf{x}(\cdot), t - T_s)\\
&\bar{\mathbf{z}}(t) = traj_{\mathbf{z}}^{\mathcal{G}}(\mathbf{z}_{t-T_s}, t - T_s, \mathbf{x}(\cdot), t)\\
&\mathcal{TFI}(\mathbf{x}(t), \bar{\mathbf{z}}(t), t) \neq \emptyset,\ \forall t \in [t_{0_k} + T_s, t_{1_k}) \qquad (3.21)
\end{aligned}$$

at the initial MHTP iteration ($k = 0$) for the initial conditions $\mathbf{x}_{0_0},\ \hat{\mathbf{z}}_{0_0},\ t_{0_0}$, there will exist a $\mathbf{u}_k(\cdot)$ that satisfies (3.21) $\forall k > 0$.

Proof. (sketch) We assume that (3.21) holds at an iteration $k-1$ starting at time $t_{0_{k-1}} = t_{0_k} - T_s$. This implies that there exists a trajectory induced by $\mathbf{u}_{\mathbf{x}_{k-1}}(t),\ t \in [t_{0_k} - T_s, t_{0_k})$ and $\mathbf{u}'_{\mathbf{x}_{k-1}}(t),\ t \in [t_{0_k}, t'_1]$ such that the agent is guaranteed safe and enters in a feasible invariant state at t'_1. The reason this is always the case is the fact that when evaluating (3.21) at iteration $k-1$, for $t = t_{0_{k-1}} + T_s = t_{0_k}$, the nominal simulation is performed for $\Delta t = 0$, i. e. $\mathbf{z}_{t-T_s} = \hat{\mathbf{z}}_0$. This implies that the initial environment state evaluated in $\mathcal{TFI}$ is guaranteed. Thus, the MHTP at iteration k will find at least $\mathbf{u}_{\mathbf{x}_k}(t) = \mathbf{u}'_{\mathbf{x}_{k-1}}(t)$ that continues to satisfy (3.21). □

This formulation is interpretable as follows: we assume that our nominal simulator is perfect and constrain our motion accordingly. However, in case our nominal simulator will drift from the real environment in future iterations, we will always have a (guaranteed) safe trajectory that we can follow. An illustration of the nominal and emergency trajectories is presented in Figure 3.3, while the performed simulations in (3.21) on a temporal scale is visualized in Figure 3.4.

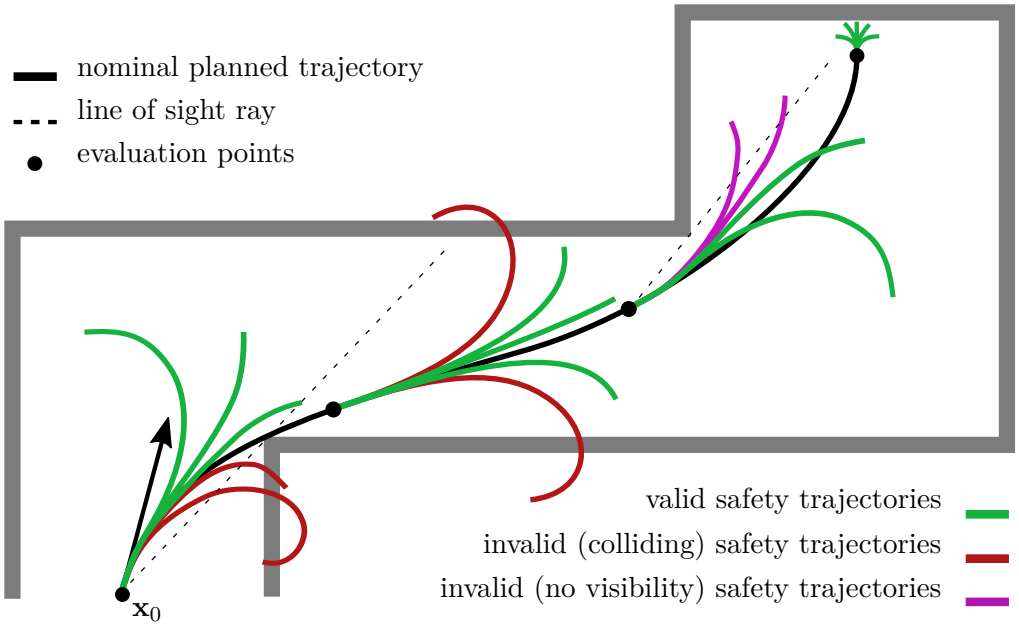


Figure 3.3: A nominal planned trajectory starting at state x_0 and emergency trajectories at each evaluation point. An emergency trajectory becomes valid if does not collide with an obstacle and if it is visible from the evaluation point.

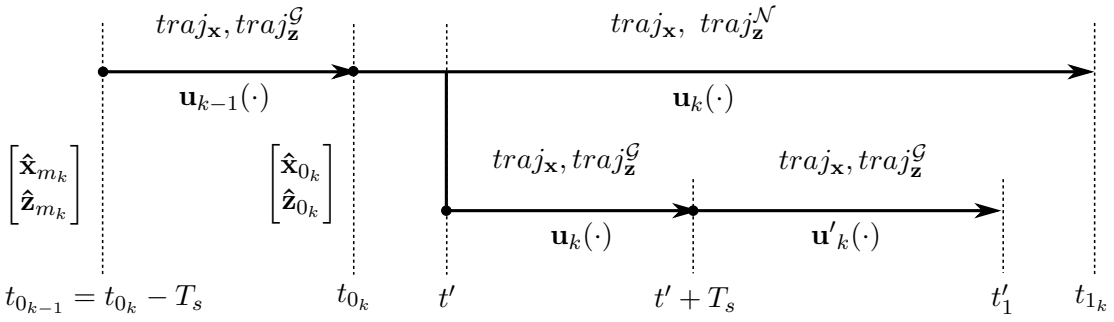


Figure 3.4: Timeline of the agent and environment simulations when enforcing the constraint (3.21). Note that when using the assumption A2, the computation cycle starts and ends at t_{0_k} with measurements $\begin{bmatrix}\hat{\mathbf{x}}_{0_k} & \hat{\mathbf{z}}_{0_k}\end{bmatrix}^{\mathrm{T}}$. The top timeline represents the nominal trajectory of agent and environment (black in Figure 3.3). The bottom timeline represents an emergency trajectory that starts at t'. Note that for the initial duration T_s, of the emergency trajectory, it evolves as the nominal trajectory using $\mathbf{u}_k(\cdot)$, however with guaranteed environment simulation. Beyond this temporal point, the trajectory alters according to $\mathbf{u'}_k(\cdot)$ towards a feasible invariant state. As mentioned, A2 can be easily dropped by performing a guaranteed simulation with the control inputs from the previous cycle. In this case, the beginning of the computation cycle is at $t_{0_{k-1}}$ with measurements $\begin{bmatrix}\hat{\mathbf{x}}_{m_k} & \hat{\mathbf{z}}_{m_k}\end{bmatrix}^{\mathrm{T}}$.

4 Implementation of MHTP

Chapter 3 discussed methods in which dynamic optimization can be used in the context of control. The introduced concepts and their analysis has been conducted in a continuous formulation. However, the actual methods through which the presented concepts can be computed (solved) are far from trivial. As presented in Section 2.3, finding solutions of ODEs requires in general numerical methods. Even in the case where the optimization problem is static, only numerical algorithms that typically guarantee only the local optimality of a solution can be used. The difficulty increases further when one is interested to solve dynamic optimization problems of the form (3.7). In general there exist two main methods to solve a dynamic optimization of the form (3.7):

Indirect Methods The main idea of such methods is to reformulate the dynamic optimization problem into a (n-point) boundary value problem which is then solved numerically [43]. This can be achieved by exploiting the mathematical properties of the (piece-wise) continuously differentiable structure of the problem by making use of Variational Calculus [44] or the Minimality Principle of Pontryagin [45]. Besides their elegance, the main strength of such approaches is that they provide solutions that are optimal with respect to the original dynamic optimization problem. However, they possess a fundamental drawback in the context of navigation: it is very difficult to systematically take into account constraints that involve the state of the system. As shown in Section 3.3, when non-trivial environment models are taken into account, constraints that depend on the agent state (namely its pose) are crucial. For this reason, indirect methods will not be considered in this thesis for the practical implementation of Optimal Navigation.

Direct Methods An alternative approach is to discretize the dynamic optimization problem with respect to time, leading to an optimization problem that no longer searches for a function (infinite dimensional vector-space) but for a finite set of variables (parameters). This results in the optimization problem to become static. However, this entails in the possibility of not satisfying the constraints on the entire desired interval. Moreover, only a suboptimal solution with respect to the original optimization problem might be found (due to the limited expressibility when searching in a finite vector-space). On the other hand, various efficient numerical solvers that can take into account arbitrary constraints exist.

Having considered the above, this chapter focuses on the practical implementation of MHTP by means of direct methods, illustrating efficient approaches and design-choices that lead to a robust, sufficiently accurate and efficient implementation.

4.1 Discretizing the Optimization Problem

We are now interested in transforming the dynamic optimization problem (3.7) into a problem of the form

$$\begin{aligned} \min_{\mathbf{p}} \quad & L(\mathbf{p}) \\ \text{subject to:} \quad & \mathbf{g}(\mathbf{p}) = \mathbf{0} \\ & \mathbf{h}(\mathbf{p}) \geq \mathbf{0}, \end{aligned} \tag{4.1}$$

where $\mathbf{p} \in \mathbb{R}^p$ are the optimization variables over a finite vector-space. In order to perform this transformation, several methods can be applied.

4.1.1 Fully Discretized Representation

The general idea behind a fully discretized representation is to consider all system state variables as well as state inputs as optimization variable. For the beginning, let us discretize the optimization time-interval $[t_0, t_1]$ into a temporal lattice with N lattice points

$$t_0 = t^0 < t^1 < \cdots < t^N - 1 = t_1. \tag{4.2}$$

Based on this lattice, we proceed by discretizing the system state and control-input at all the temporal-lattice locations

$$\mathbf{X}^{\mathrm{T}} = \begin{bmatrix} (\mathbf{x}^0)^{\mathrm{T}} & (\mathbf{x}^1)^{\mathrm{T}} & \dots & (\mathbf{x}^{N-1})^{\mathrm{T}} \end{bmatrix}, \quad \mathbf{U}^{\mathrm{T}} = \begin{bmatrix} (\mathbf{u}^0)^{\mathrm{T}} & (\mathbf{u}^1)^{\mathrm{T}} & \dots & (\mathbf{u}^{N-1})^{\mathrm{T}} \end{bmatrix}. \tag{4.3}$$

Summarized, the most general form of optimizing such a discretized system is to optimize for

$$\mathbf{p}^{\mathrm{T}} = \begin{bmatrix} \mathbf{X}^{\mathrm{T}} & \mathbf{U}^{\mathrm{T}} & t^0 & t^1 & \dots & t^{N-1} \end{bmatrix} \tag{4.4}$$

Note that an addition of $N-1$ inequality constraints is necessary to ensure that the temporal lattice is consistent

$$t^{i+1} - t^i \geq 0, \qquad i = 0, 1, \dots, N-2. \tag{4.5}$$

An alternative approach that leads to a reduced dimensionality of the search-space is to assume equal temporal lattice intervals. In this approach, one would not have to optimize for all temporal lattice points but rather for their interval Δt.

Having defined our optimization state, we can now discretize the remaining of the optimization problem. Note that when using full discretization, one has to choose the ODE discretization method and encode it explicitly in the optimization problem as the involved ODEs will be implicitly solved during optimization. For example, the cost-functional (3.6) can be discretized using the Trapeze Rule in the form

$$L(\mathbf{p}) = \varphi(t^{N-1}, \mathbf{x}^{N-1}) + \frac{1}{2} \sum_{i=0}^{N-2} (t^{i+1} - t^i) \left(l(\mathbf{x}^i, \mathbf{u}^i, t^i) + l(\mathbf{x}^{i+1}, \mathbf{u}^{i+1}, t^{i+1}) \right). \tag{4.6}$$

Similarly, the state transition function can be discretized and included in the optimization problem by the equality constraints

$$\mathbf{x}^{i+1} - \mathbf{x}^i - \frac{t^{i+1} - t^i}{2}\left(\mathbf{f}(\mathbf{x}^i, \mathbf{u}^i, t^i) + \mathbf{f}(\mathbf{x}^{i+1}, \mathbf{u}^{i+1}, t^{i+1})\right) = \mathbf{0}, \quad i = 0, 1, \ldots, N-2. \tag{4.7}$$

Regarding the end-constraint $\boldsymbol{\psi}$, it is simply evaluated on the end-state of the trajectory

$$\boldsymbol{\psi}(\mathbf{x}^{N-1}, \mathbf{u}^{N-1}, t^{N-1}) = \mathbf{0}. \tag{4.8}$$

The last part of the original optimization problem that has to be discretized are the inequality constraints

$$\mathbf{h}(\mathbf{x}(t), \mathbf{u}(t), t) \geq \mathbf{0}, \quad \forall t \in [t_0, t_1]. \tag{4.9}$$

The most straight-forward way of discretization is to enforce the inequality constraints at the lattice points

$$\mathbf{h}(\mathbf{x}^i, \mathbf{u}^i, t^i) \geq \mathbf{0}, \quad i = 0, 1, \ldots, N-1. \tag{4.10}$$

In practice, such an approach will not guarantee that the inequality constraints are satisfied on the entire interval $[t_0, t_1]$, but only at the discrete evaluation points. Thus, one would tighten the bounds of such a constraint such that its discrete evaluation would result in the initial constraint to be guaranteed satisfied on the continuous interval. The following example provides a more in-depth discussion on the concept.

Example 4.1 (Discretizing the Distance-to-Obstacles Constraint for a circular agent). Suppose we are considering the navigation problem of an agent with circular footprint with radius r and a single obstacle represented by a point. With the agent position $\mathbf{x}_r^{\mathrm{T}} = \begin{bmatrix} x_r & y_r \end{bmatrix}$ and the obstacle position $\mathbf{x}_o^{\mathrm{T}} = \begin{bmatrix} x_o & y_o \end{bmatrix}$, the continuous constraint that enforces non-collision of the agent with the obstacle *along a planned trajectory* is

$$||\mathbf{x}_r(t) - \mathbf{x}_o(t)||_2 - r \geq 0, \quad \forall t \in [t_0, t_1]. \tag{4.11}$$

As presented in (4.10), the discrete constraint is

$$||\mathbf{x}_r(t^i) - \mathbf{x}_o(t^i)||_2 - r' \geq 0, \quad i = 0, 1, \ldots, N-1. \tag{4.12}$$

Now, we are interested in computing a value for r' as well as a bound for the maximum distance between the evaluation points t^i such that the original, continuous constraint is always satisfied, as illustrated in Figure 4.1.

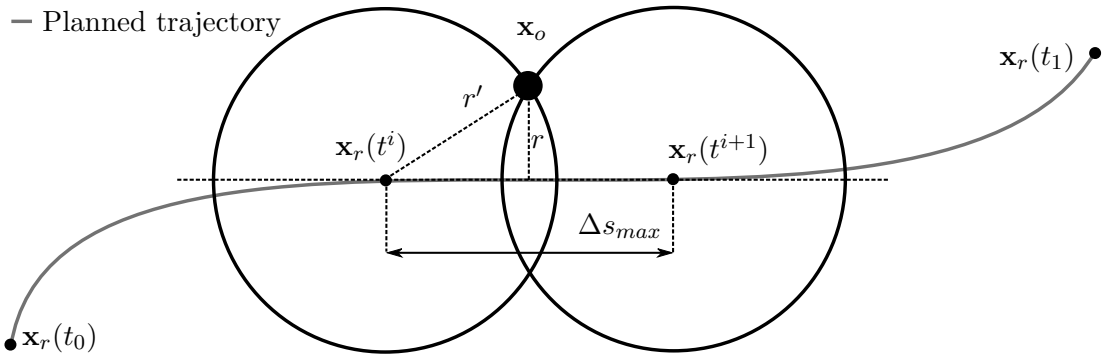


Figure 4.1: Bounds of the discretized distance constraint

For the beginning, let us suppose that the maximum distance between two evaluation points is Δs_{max}. As the minimum distance between the obstacle and the robot is present at the point where the two circles of consecutive agent footprints of radius r' meet, it follows that the required discrete radius $r' \geq \sqrt{\left(\frac{\Delta s_{max}}{2}\right)^2 + r^2}$. Note that as $\Delta s_{max} \to 0$, we approach the continuous formulation $r' \geq r$. Thus, the governing parameter on the tightness of the discrete constraint is governed by the distance between consecutive lattice points Δs_{max}. We can conclude that the continuous obstacle-distance constraint is of *spatial nature*, even though the so-far considered lattices are of temporal nature.

Next, let us analyse how we can find a bound on Δs_{max}. For the case in which the temporal lattice points t^i are assumed to be equally spaced ($t^{i+1} = t^i + \Delta t$), we could find such a bound by taking into account the maximum velocity of the agent v_{max}:

$$\Delta s_{max} = \frac{v_{max}}{\Delta t} \tag{4.13}$$

However, this is a quite conservative bound. Note that even if the planned trajectory has a very short distance, still the parameter that governs the bound of the constraint relaxation is v_{max}. An improvement could be thus achieved if we would have a lattice that is spaced at *equal distance*. This would result on r' to become independent on v_{max} and depend on the maximum planned travelled distance s_{max}.

As a final note, one might ask how can one evaluate the distance-to-obstacles constraint in the non-trivial setting where more than one obstacle point is involved. An efficient method for this evaluation when considering static unstructured obstacles is to pre-compute on a pixel-map the Euclidean Distance Transform and evaluate it using bilinear interpolation, as presented in Example 2.9. This approach which will be discussed more in-depth in Chapter 5.

We would like now to summarise several characteristics of the full-discretization approach. A noteworthy property of the full-discretization formulation of the optimization problem relies in the fact that the differential (now difference) equations are explicitly solved as

equality constraints. This implies that we no longer require an external ODE solver for any involved ODE. The strengths of this approach are that we can easily make use of implicit ODE solver methods, as for their solution, numerical algorithms are typically required. However, due to the fixed defined (temporal) lattice points, it is quite difficult to make use of ODE solvers that require intermediate steps, as this would require an increase of the lattice points. Thus, we are limited to simple RK methods, such as the Trapeze Method or to Multi-step methods. Another draw-back of solving the ODEs implicitly in the optimization problem relates to the solution accuracy. In many implementations of non-linear solvers, the exact satisfaction of the equality constraints is not guaranteed. Rather, the optimization process will evolve and converge to a solution in which a norm of the equality constraints is minimized. Thus, the evaluated system trajectory can possess relatively high inaccuracies. Lastly, the ODE solver used in the optimization problem has to be explicitly coded. This results in a design restriction, such that switching between different ODE solvers becomes a tedious task.

As the ODEs are solved implicitly, one benefit of the full-discretization approach is that we can evaluate first (and second) order sensitivities in closed form. However, this comes at the cost of a large optimization space (generally $N(n+m+1)$ optimization variables, with the state dimension n, control dimension m and number of discretized states N)as well as a large number of additional constraints (for the general case $n(N-1)$ additional equality constraints and $N-1$ additional inequality constraints).

An alternative to address the reduced numerical accuracy of the ODEs solutions is to solve the system-dynamics related equality constraints external to the optimization problem. This is performed by *iteratively* solving the system ODE using an ODE solver, thus guaranteeing a system trajectory numerical error of the order of the used ODE solver type, invariant to the optimization problem convergence.

However, with such an approach, as discussed in Section 2.4, computing (iteratively) the sensitivities of the system states with respect to the input sequence $\mathbf{u}_k$ can induce considerable computational overhead. This motivates using a discretization approach at the other end of the spectrum, namely by using a minimal parametric representation.

4.1.2 Minimal Parametric Representation

The general concept of such a discretization approach is to formulate the problem such that no additional equality constraints (related to the ODE solve) are required. This implies that an external ODE solver is used to iteratively evaluate information required by the optimization algorithm (such as cost function, equality and inequality constraints as well as their sensitivities). We will analyse this approach in a constructive manner, arriving finally at the general formulation of the discretized optimization problem.

The main difference to the full-discretization approach is that we want to use an external solver for all the ODEs present in our optimization problem. As presented in Example 2.13, it is in general convenient to consider that parts of the system state (and their sensitivities) can be computed in closed form. This choice not only reduces the computational

requirements for solving the system but also improves the accuracy of the solution. Thus, we choose to split the state of the system

$$\mathbf{x}_{sys}^{\mathrm{T}} = \begin{bmatrix} \mathbf{x}_n^{\mathrm{T}} & \mathbf{x}_c^{\mathrm{T}} \end{bmatrix} \tag{4.14}$$

into a part that requires numerical solutions $\mathbf{x}_n$ and a part that can be evaluated in closed-form $\mathbf{x}_c$. Additionally, recall that we have to solve an additional ODE related to the integral term of the optimization cost-function (3.6). Thus, the state that we have to solve numerically is

$$\mathbf{x}_{ode}^{\mathrm{T}} = \begin{bmatrix} L & \mathbf{x}_{Nm}^{\mathrm{T}} \end{bmatrix} \tag{4.15}$$

with the integral (Lagrangian) term L of the cost-function. This results in the (matrix) state for the sensitivities computation to

$$\begin{bmatrix} \mathbf{x}_{ode}^{\mathrm{T}} \\ \nabla \mathbf{x}_{ode}^{\mathrm{T}} \end{bmatrix} = \begin{bmatrix} L & \mathbf{x}_{Nm}^{\mathrm{T}} \\ \nabla L^{\mathrm{T}} & \nabla \mathbf{x}_{Nm}^{\mathrm{T}} \end{bmatrix} \tag{4.16}$$

Note that in general, the total derivatives of the numerical state with respect to the optimization parameters can be obtained using the chain-rule given the closed-form states sensitivities.

For the simple case of unconstrained optimization, one might ask if some of the numerical states can be omitted from the ODE state. This would be the case when the Lagrangian L would not depend on those states. However, in constrained optimization, such states might be required for evaluating certain equality or inequality constraints. For a very efficient implementation, one might want to define 3 different ODE states, used for the evaluation of the cost-function, equality constraints and inequality constraints respectively.

4.1.3 Evaluation Lattices

As discussed in Example 4.1, continuous inequality constraints can be of different natures. Even though in the full-discretization approach, the inclusion of such different lattices is very difficult[1], we will see that they can be feasibly evaluated using a minimal parametric representation.

We will begin by defining the time as the consistent arc parametrization that we will use to encode and relate all the evaluation lattices. However, it must not necessarily be time but rather a function that is continuous and increasing. Now, let us consider a generic lattice $\boldsymbol{\xi}$ that can be evaluated at different arc parametrizations (times). For such a lattice

[1] The difficulty lies in the fact that on the course of optimization, when e.g. an equal distance lattice point coincides with an e.g. equal time lattice point, the sensitivities of the related optimization constraints are typically undefined or complicated to evaluate.

to be consistent, we require

$$\boldsymbol{\xi}^{\mathrm{T}} = \begin{bmatrix} \xi^0 & \xi^1 & \dots & \xi^{N_{\boldsymbol{\xi}}-1} \end{bmatrix} \tag{4.17a}$$

$$\xi^i \leq \xi^{i+1}, \quad i = 0, \dots, N_{\boldsymbol{\xi}} - 2 \tag{4.17b}$$

$$\xi^0 = t_0 \tag{4.17c}$$

$$\xi^{N_{\boldsymbol{\xi}}-1} = t_1 \tag{4.17d}$$

that is, the lattice is non-decreasing.

Provided that such lattices can be computed in closed form (with respect to the optimization parameters), we could solve the ODE only once and by stepping exactly in all lattice points, we could save their states and sensitivities for later use. This results in an efficient algorithm that solves the optimization problem ODE in a single pass, resulting in the numerical state solutions and sensitivities for every lattice point ξ_k^i of every lattice $\boldsymbol{\xi}_k$. Algorithm 1 presents the pseudo-code of the discussed approach.

Algorithm 1 *solve_multilattice_sensitivity_ode*$(\mathbf{p})$
parameters: lattices structure

// initialization:
$\mathbf{x}_{ode} \leftarrow \mathbf{x}_0(\mathbf{p})$
$\nabla\mathbf{x}_{ode} \leftarrow \nabla\mathbf{x}_0(\mathbf{p})$
$\xi_k^i \leftarrow computeLatticeArc(\mathbf{p}, \boldsymbol{\xi}_k, i, t_0), \quad \forall k, \forall i = 0, \dots, N_{\boldsymbol{\xi}_k} - 1$

// sorting and storage of all arc parametrization points:
$\boldsymbol{\xi}_{all} \leftarrow sort\left(\left\{\xi_k^i \middle| \forall k, i = 0, \dots, N_{\boldsymbol{\xi}_k} - 1\right\}\right)$

// one-pass solve of the ODE:
for $i = 0, \dots, \dim(\boldsymbol{\xi}_{all}) - 1$ **do**
  $\xi_k^j \leftarrow getLatticePoint(\boldsymbol{\xi}_{all}, i)$
  $\begin{bmatrix} \mathbf{x}_{ode}^{\mathrm{T}} \\ \nabla\mathbf{x}_{ode}^{\mathrm{T}} \end{bmatrix} \leftarrow advanceODE(\mathbf{p}, \mathbf{x}_{ode}, \nabla\mathbf{x}_{ode}, \xi_k^j)$
  $\mathbf{x}_{\boldsymbol{\xi}_k}^j \leftarrow \mathbf{x}_{ode}$
  $\nabla\mathbf{x}_{\boldsymbol{\xi}_k}^j \leftarrow \nabla\mathbf{x}_{ode}$

  // patching the sensitivities of the ODE:
  $\nabla\mathbf{x}_{\boldsymbol{\xi}_k}^j \leftarrow \nabla\mathbf{x}_{\boldsymbol{\xi}_k}^j + \mathbf{f}_{\mathbf{x}}(\mathbf{x}_{\boldsymbol{\xi}_k}^j, \mathbf{p}, \xi_k^j)\nabla\xi_k^j(\mathbf{p}) - \mathbf{f}_{\mathbf{x}}(\mathbf{x}_0(\mathbf{p}), \mathbf{p}, t_0)\nabla t_0(\mathbf{p})$
end for

Note that when one is not interested to evaluate the sensitivities of the system state, the algorithm simplifies by omitting the sensitivities-related variables and algorithmic steps. As a note, we require that all lattice points are computed in closed-form as otherwise, evaluation of the lattice arcs as well as the correction of the gradients would be unreasonably expensive to compute.

Example 4.2 (Evaluation Lattices of kinematic models). Let us consider the kinematic model of a platform whose model parametrization includes the chassis velocity v using a piece-wise linear function.

The first lattice that we want to possess is the lattice that relates to the end-point of the trajectory. This is required when an end-cost term φ is present in the cost-function or when end-state constraints ψ are present. Thus, we have for the lattice $\boldsymbol{\xi}_{end}$,

$$\xi_{end}(\mathbf{p}) = t_1(\mathbf{p}). \tag{4.18}$$

The next lattice that we want to possess is (as usual) the temporal lattice $\boldsymbol{\xi}_{\Delta t}$, spaced at equal (but variable) time intervals

$$\xi^i_{\Delta t}(\mathbf{p}) = \frac{i}{N_{\boldsymbol{\xi}_{\Delta t}} - 1} t_1(\mathbf{p}), \quad \forall i = 0, \dots, N_{\boldsymbol{\xi}_{\Delta t}} - 1. \tag{4.19}$$

As discussed in Example 4.1, navigation under obstacles would benefit of a spatial lattice $\boldsymbol{\xi}_{\Delta s}$. As the travelled distance is defined by

$$s(\mathbf{p}, t_1(\mathbf{p})) = s_0(\mathbf{p}, t_0(\mathbf{p})) + \int_{t_0(\mathbf{p})}^{t_1(\mathbf{p})} |v(\mathbf{p}, \tau)| \mathrm{d}\tau, \tag{4.20}$$

we require a function parametrization of $v(t)$ such that $s(t)$ can be computed in closed-form. For piece-wise linear functions, this is simply the case. Moreover, if we consider only positive velocities $v(t) \geq 0$, any parametric function whose integral can be evaluated in closed-form is a good candidate for obtaining a spatial-lattice. Assuming (variable) equal distances between the lattice points, we obtain $\xi^i_{\Delta s}(\mathbf{p})$ by solving

$$s(\mathbf{p}, \xi^i_{\Delta s}(\mathbf{p})) = \frac{i}{N_{\boldsymbol{\xi}_{\Delta s}} - 1} s_1(\mathbf{p}, t_1(\mathbf{p})), \quad \forall i = 0, \dots, N_{\boldsymbol{\xi}_{\Delta s}} - 1. \tag{4.21}$$

We would like here to highlight that other types of lattices can be beneficial. In the case of the velocity parametrization using piece-wise linear functions, the velocity derivative is a convex function on every linear interval. If our optimization problem constrains the linear acceleration of the system, it is sufficient to evaluate the inequality constraint only at the inflection-points of the piece-wise linear function. Thus, the equal control-points lattice $\boldsymbol{\xi}_{\Delta knt}$ is

$$\xi^i_{\Delta knt}(\mathbf{p}) = t_i(\mathbf{p}), \quad \forall i = 0, \dots, N_{\boldsymbol{\xi}_{\Delta knt}} - 1. \tag{4.22}$$

An illustration of the above-presented evaluation lattices for a differential drive is presented in Figure 4.2.

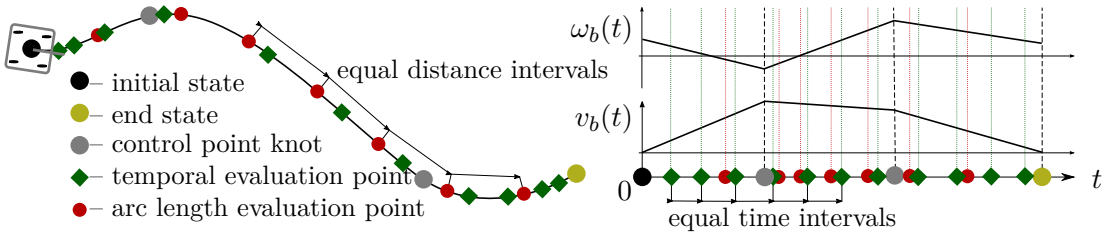


Figure 4.2: Trajectory parametrization with different lattice points

Using an analysis similar to the one presented in Example 4.1, one might conclude that certain constraints suffice to be evaluated at fewer points along the optimized trajectory. In this case, one could simply define an additional lattice (with a different number of evaluation points), providing a flexible and general method for improving the quality and(or) run-time of the solved optimization problem.

So far, we have discussed how to solve the optimization problem ODEs for the system state and the Lagrangian at lattice points of different nature. Note, however, that from here, any additional terms in the cost-function and constraints can be easily modelled by making use of the chain-rule. This results into an additional benefit of the minimal parametric representation, namely that the designer can split the work in modelling the state of the system, and the cost-function and constraints of the optimization problem.

Given all the previous considerations, we can now formulate the general form of the discretized dynamic optimization problem using a minimal parametric representation:

$$\begin{aligned} \min_{\mathbf{p}} \quad & L(\mathbf{p}) = \varphi(\mathbf{x}_{t_1}(\mathbf{p}), t_1(\mathbf{p})) + \int_{t_0(\mathbf{p})}^{t_1(\mathbf{p})} l(\mathbf{x}(\mathbf{p}, t), t)\mathrm{d}t \\ \text{subject to:} \quad & \mathbf{g}_{\boldsymbol{\xi}_k}(\mathbf{x}^i_{\boldsymbol{\xi}_k}(\mathbf{p})) = \mathbf{0} \\ & \mathbf{h}_{\boldsymbol{\xi}_k}(\mathbf{x}^i_{\boldsymbol{\xi}_k}(\mathbf{p})) \geq \mathbf{0}, \quad \forall k, \forall i = 0, \ldots, N_{\boldsymbol{\xi}_k} - 1 \end{aligned} \tag{4.23}$$

with the equality constraints $\mathbf{g}_{\boldsymbol{\xi}_k}$ and the inequality constraints $\mathbf{h}_{\boldsymbol{\xi}_k}$ evaluated along the lattice $\boldsymbol{\xi}_k$.

4.1.4 Solving the Discretized Optimization Problem

So far, we have addressed methods to represent, model and encode the optimization problem. In the following, we would like to mention different methods by which such an optimization problem can be solved.

Best sampled trajectory Probably one of the simplest ways of solving such an optimization problem is through sampling. The general idea is to sample the optimization space and compute the cost-function and constraints for every sample. At the end, the best-scored solution that satisfies the constraints is selected as the solution of the optimization. In the literature, such an approach was among the first to be used in the context of MPC for vehicle navigation, under the popular name of Dynamic Window Approach (DWA) [8]. However, such an approach has certain drawbacks. Being inspired from discrete optimization, it relies on exhaustively evaluating a neighbourhood of an initial solution in the optimization space. This results in either a huge search-space or in very inexpressive solutions, being a feasible approach only for small forward-simulation times.

Weighted average of high-scored trajectories A variant of the best sampled trajectory approach is to compute the optimal solution as a weighted average of high-scored solutions. Even though this might seem not very elegant and unfounded for the feasibility of the optimal solution, more complex methods can be used to obtain very good results. A successful approach using such a conceptual technique was achieved in [15] where an MPC algorithm is devised by solving a stochastic optimization problem for the purpose of aggressive driving, i. e. driving under large side-slip angles.

Mixed Integer Linear Programming Another technique to solve such an optimization problem is to make use of simplified non-linear models. Their reduced complexity allows a more complex optimization problem to be solved, that includes continuous as well as discrete variables. [19] has successfully applied such techniques for safe navigation of UAVs.

All the above mentioned methods are typically based on sampling, which guarantees finding the global optimal solution in the searched space, or guarantees that the optimal solution is global. The following methods possess this characteristic only for convex optimization problem. For the generic non-linear case, they only guarantee finding a locally optimal solution.

Derivative-Free Optimization Methods Moving towards continuous non-linear methods, the first step is to require the continuity of our optimization problem. Such optimization problems can be solved by (variations of) the Simplex Algorithm[2] [46]. Even though such an approach requires considerably more iterations than gradient-based methods, it does not require computation of sensitivities. In certain problems, these methods prove to be beneficial [47].

Gradient-based Non-Linear Programming Such algorithms require the knowledge of first (and possibly second) order sensitivities of the optimization problem. The literature possesses a multitude of generic approaches, improvements and efficient implementations of such algorithms [48, 49]. One of their strengths is that the solutions possess relatively high numerical accuracy. Even though originally such algorithms were devised for convex optimization problems, variants that are well adjusted for general optimization problems exist.

In the remaining of this work, we will focus on the details and implementation specifics of solving the optimization problem using Non-Linear Programming algorithms as it has constituted the method of choice for Optimal Vehicle Navigation.

[2] not to be mixed with the Simplex Algorithm used for solving linear optimization problems

4.2 Gradient-based Non-Linear Programming

In the following, we will present certain considerations and implementation specifics that relate to solving the optimization problem using a non-linear solver. A main characteristic of non-linear solvers is that they require not only evaluation of the cost-function and constraints, but also their first and possibly second order sensitivities with respect to the optimization variables, requiring thus that the modelled cost-function and constraints to be at least once (twice) differentiable.

4.2.1 Solver Algorithms

For unconstrained non-linear optimization [50], the typical solver algorithms are based on line-search methods or trust-region methods [49].

In line-search methods, at each iteration, the solver first chooses a search direction in the optimization space and then searches along this direction for a point which minimizes the cost-function. It is worth noting here that line-search approaches formally require that the function is *unimodal* on the line-search interval, i. e. it possesses only one local minimum [49]. If this is not the case, the line-search algorithm might end-up oscillating and convergence to a local-minimum is not guaranteed. In practice, this can be obtained even for generic non-convex problems if the line-search interval is forced to be sufficiently small. However, it is practically impossible to obtain a clean formulation that guarantees the uni-modality of the line-search interval [49].

In trust-region methods, the general idea is to approximate the function at each iteration with a quadratic function and enforce a bound on the maximum step that can be performed at one iterate (a *trust-region*). Then, the optimal point of the iterate is computed assuming that the to-be-minimized function is the quadratic approximate. At the end, the actual improvement of the real cost-function is compared with the expected improvement based on the quadratic estimate. Based on this metric, the trust-region is expanded or shrunk, resulting in the algorithm to perform iterates such that locally, the cost-function is described by its quadratic approximate with sufficient accuracy [49].

Moving forward to constrained optimization, solvers have to be able to enforce the equality and inequality constraints. The equality constraints are taken into account by making use of *Lagrange Multiplicators*, which are additional variables that allow manipulation of the equality constraints together with the cost-function [51]. It is important to note that the equality constraints are always active, i. e. every valid point in the optimization-space has to satisfy them. This results in most approaches to perform steps in the tangential space of the equality constraints, i. e. along the manifold that satisfies them [51]. Most solvers cannot guarantee the exact satisfaction of the equality constraints, but rather they will converge with the iterates while preserving a bounded error of the constraints. We note that this implies a *soft* satisfaction of the equality constraints. A popular method to solve equality-constrained optimization problems is the Sequential Quadratic Program (SQP) method. In this approach, the entire optimization problem is replaced by a local

quadratic problem approximation, which is iteratively solved until convergence *at each global iteration* [52]. More advanced variations of SQP are Inexact Composite-step SQP methods [53], in which a step correcting the equality constraints error is added (thus composite-step). Moreover, they converge to optimal solutions even if the sensitivity information of the program is inexact.

Inequality constraints, however, are in general more complicated to deal with. The main reason lies on the fact that they can be inactive (we are in the interior of the domain) or active (we are at the boundary of the domain). At a point where an inequality constraint is active, it behaves like an equality constraint, while points where an inequality constraint is inactive behave as if the constraint is not present [49]. One method to take into account inequality constraints is the Active-Set Method, an approach in which the set of active inequality constraints is book kept [54]. The main drawback of this approach is the combinatorial nature of the possible active constraints at a next iterate. Another popular approach to deal with the inequality constraints is the Interior-Point Method. In this method, the point at each algorithm iterate is guaranteed to always satisfy all inequality constraints, never exactly touching a constraint boundary. The approach makes use of barrier-functions to obtain initial solutions that are intuitively "far-away" from the boundaries and then allows the solution to increasingly approach the boundaries. Even though the Interior-Point method is relatively fast, it possesses two drawbacks: firstly, the initial solution has to be inside the feasible region, i. e. all inequality constraints have to be satisfied. This can be difficult to ensure in practice for synthetic constraints (for example minimum distance to a moving obstacle). Secondly, most implementations require that the inequality constraints are affine with respect to the optimization variables [55].

Given the available implementations as well as the use case in the scope of this thesis, for the following, an optimization algorithm based on the Interior-Point Method together and an Inexact Composite-step SQP is preferred.

4.2.2 Generic Constraint Modelling

Given the specifics of typical non-linear programming solvers, we will have a look at generic methods to encode inequality and equality constraints.

Inequality Constraints The inequality constraints can be generally expressed as

$$\mathbf{h}(\mathbf{p}) \geq \mathbf{0}. \tag{4.24}$$

Given that certain solvers require that inequality constraints are affine, we differentiate

$$\mathbf{h}_{aff}(\mathbf{p}) \geq \mathbf{0} \tag{4.25a}$$
$$\mathbf{h}_{nl}(\mathbf{p}) \geq \mathbf{0}, \tag{4.25b}$$

with the affine inequality constraints $\mathbf{h}_{aff}$ and the general non-linear inequality constraints $\mathbf{h}_{nl}$. We are now interested in reformulating the non-linear inequality constraints such that

specific solvers requirements are satisfied. This is typically done by reformulating them into equality constraints, which solvers do not typically require to be affine. Moreover, as a result, the inequality constraints become soft, i. e. they are not necessarily exactly satisfied.

In the following, we will constructively start from an apparently correct modelling of such reformulations, identifying certain issues, additional requirements and finally proper reformulations along the way. Let us begin by considering a single non-linear inequality constraint

$$h^i_{nl}(\mathbf{p}) - a \geq 0, \tag{4.26}$$

with the constant a. We note that we can reformulate this constraint as an equality constraint

$$g_h(\mathbf{p}) = \begin{cases} \frac{1}{2}(h^i_{nl}(\mathbf{p}) - a)^2, & h^i_{nl}(\mathbf{p})^2 \leq a \\ 0, & h^i_{nl}(\mathbf{p})^2 > a. \end{cases} \tag{4.27}$$

Note that such a function has a continuous derivative (no discontinuities in either function nor gradient values at the switching point $h^i_{nl}(\mathbf{p}) = a$). Also, note that this function is 0 exactly when (4.26) is satisfied.

With this, we can reformulate the optimization problem (4.1) as

$$\begin{aligned} \min_{\mathbf{p}} \quad & L(\mathbf{p}) \\ \text{subject to:} \quad & \mathbf{g}(\mathbf{p}) = \mathbf{0} \\ & \mathbf{g_h}(\mathbf{p}) = \mathbf{0} \\ & \mathbf{h}_{aff}(\mathbf{p}) \geq \mathbf{0}. \end{aligned} \tag{4.28}$$

We would like to mention here that even though this formulation seems valid, it might be ill-posed for certain problems and solver algorithms. Note that a point $\bar{\mathbf{p}}$ in the search space is defined as regular if

$$\operatorname{rank}(\nabla \mathbf{g})(\bar{\mathbf{p}}) = \operatorname{rank} \begin{bmatrix} (\nabla g_0)(\bar{\mathbf{p}}) \\ (\nabla g_1)(\bar{\mathbf{p}}) \\ \vdots \\ (\nabla g_q)(\bar{\mathbf{p}}) \end{bmatrix} = q, \tag{4.29}$$

where $\dim(\mathbf{g}) = q$ and $\dim(\bar{\mathbf{p}}) = p$ [49]. This implies that $q \leq p$, which is expected, since an optimization problem with more equality constraints than optimization variables is clearly ill-posed. However, by using an arbitrary number of equality constraints (4.27), $q \leq p$ cannot be guaranteed any-more. Another characteristic that (4.29) requires by imposing $\operatorname{rank}(\nabla \mathbf{g})(\bar{\mathbf{p}}) = q$ is the Linear Independence Constraint Qualification (LICQ) condition [49]. This condition is required by many minimization algorithms that are based on the Karush-Kuhn-Tucker optimality conditions. It guarantees that the Lagrange-Multipliers of the optimization problem are unique [49]. To this end, equality constraints such as $g(p) = p^2 = 0$ are not regular points, following that a reformulation such as (4.27) does not

satisfy solver requirements. This motivates an alternative transformation of the inequality constraints $\mathbf{h}_{nl}$, using slack variables $\mathbf{s}$

$$\mathbf{g_h}(\mathbf{p}, \mathbf{s}) = \mathbf{h}_{nl}(\mathbf{p}) - \mathbf{s} \tag{4.30a}$$
$$\mathbf{h_s}(\mathbf{s}) = \mathbf{s}. \tag{4.30b}$$

The resulting optimization problem then reads as

$$\begin{aligned} \min_{\mathbf{p},\mathbf{s}} \quad & L(\mathbf{p}) \\ \text{subject to:} \quad & \mathbf{g}(\mathbf{p}) = \mathbf{0} \\ & \mathbf{g_h}(\mathbf{p}, \mathbf{s}) = \mathbf{0} \\ & \mathbf{h}_{aff}(\mathbf{p}) \geq \mathbf{0} \\ & \mathbf{h_s}(\mathbf{s}) \geq \mathbf{0}. \end{aligned} \tag{4.31}$$

Note that now, the optimization problem possesses additionally $\dim(\mathbf{s})$ optimization variables, namely one additional variable per equality constraint. This guarantees that $q \leq p$. It is worth noting here that one might want to scale the slack variable influence on the newly created equality constraint, with the aim of improving the conditioning number of the cost-function Hessian, i. e. $\mathbf{g_h}(\mathbf{p}, \mathbf{s}) = \mathbf{h}_{nl}(\mathbf{p}) - \mathbf{c}^{\mathrm{T}}\mathbf{s}$, with the vector of scaling constants $\mathbf{c}$.

Equality Constraints So far we have discussed two simple methods through which inequality constraints that are non-affine or are desired to be softly satisfied can be reformulated into equality constraints. We would like here to have a short discussion regarding equality-constraint scaling. In general, algorithms will try to minimize a norm of the equality-constraints error up to a given value ε. When equality constraints of various natures are involved, the designer is interested to maintain different tolerances for constraints of different natures, i. e. $\dim(g^i(\mathbf{p})) < \varepsilon^i$. A simple approach to systematically account for this is to modify every equality constraint as

$$g^i_{sc}(\mathbf{p}) = \frac{1}{\varepsilon^i} g^i(\mathbf{p}) \tag{4.32}$$

Imposing the solver to maintain the tolerance $||\mathbf{g}_{sc}(\mathbf{p})|| < 1$ thus provides a relatively tight bound that guarantees that the desired ε^i are satisfied.

Example 4.3 (Kinematic Constraints: Differential Drive). We formulate the kinematic constraints of a differential-drive platform, modelled and parametrized in accordance with example Example 2.13. Firstly, we would like to enforce that the platform moves only forward, given that most perception-sensors do not have a 360° field-of-view. Moreover, we would like to enforce limits on the wheels angular velocities and accelerations as well as a limit on the lateral acceleration of the platform. Given the piece-wise linear parametrization in v and ω, we identify that a number of constraints are affine with respect to the optimization variables, being thus able to formulate them directly as

(hard) inequality constraints

$$\text{Positive chassis velocity:} \quad v(\mathbf{p}) \geq 0 \tag{4.33a}$$

$$\text{Max. right (left) wheel angular velocity:} \quad \omega^w_{max} - \frac{v(\mathbf{p}) \pm \frac{d}{2}\omega(\mathbf{p})}{r_w} \geq 0 \tag{4.33b}$$

$$\text{Min. right (left) wheel angular velocity:} \quad -\omega^w_{min} + \frac{v(\mathbf{p}) \pm \frac{d}{2}\omega(\mathbf{p})}{r_w} \geq 0 \tag{4.33c}$$

Regarding the angular acceleration of the wheels as well as the chassis lateral acceleration, they are no-longer affine. We formulate them as

$$\text{Max. right (left) wheel angular acceleration:} \quad \dot{\omega}^w_{max} - \frac{\dot{v}(\mathbf{p}) \pm \frac{d}{2}\dot{\omega}(\mathbf{p})}{r_w} \geq 0 \tag{4.33d}$$

$$\text{Min. right (left) wheel angular acceleration:} \quad -\dot{\omega}^w_{min} + \frac{\dot{v}(\mathbf{p}) \pm \frac{d}{2}\dot{\omega}(\mathbf{p})}{r_w} \geq 0 \tag{4.33e}$$

$$\text{Max. (minimum) chassis lateral acceleration:} \quad a^{lat}_{max} \mp v\omega \geq 0 \tag{4.33f}$$

and implement them as soft inequality constraints using slack-variables, as presented in (4.30). Note that (4.33a), (4.33b), (4.33c), (4.33d) and (4.33e) are convex on any interval of the piece-wise linear functions as long as the inflection points of the v and ω parametric functions coincide. This implies that for guaranteed satisfaction of these constraints on the entire optimization interval, it is sufficient to evaluate them only at the inflection points. The constraint (4.33f) is most conveniently evaluated on a temporal lattice.

Example 4.4 (Collision-avoidance Constraints: Random-Walk Environment Model). In this example, we formulate the collision-avoidance constraints of a platform assuming a random-walk environment model. Such an environment model results in its nominal state to be static during the course of solving the optimization problem. For simplicity, we will initially assume that the platform has a circular footprint.

A main point of any function evaluated inside the optimization problem is that it should have a minimal computational complexity. Thus, we would like to have a function $d(x, y)$ which returns the minimum distance to any obstacle at the position (x, y). This can be performed efficiently for nominally static (unstructured) environments by computing the Euclidean Distance Transform and its gradient (with respect to x and y) prior to the optimization, resulting in a constant-time complexity evaluation of the function.

$$d_{edt}(\mathbf{x}) = \min\{\ ||\mathbf{x} - \mathbf{p}||_2,\ \forall \mathbf{p} \in \mathbf{z_0}\} \tag{4.34}$$

The details of this method are beyond the scope of this exercise and will be detailed in Chapter 5. We assume thus that we can access the value of d_{edt}, $\frac{\partial}{\partial x} d_{edt}$ and $\frac{\partial}{\partial y} d_{edt}$ at any desired points (x, y).

Going back to the constraint of interest, we initially desire that the distance to the closest obstacle is always larger than a desired value, i. e. $d_{edt}(x, y) - d_{min} \geq 0$. To

account for the radius of the agent footprint, we have

$$d_{edt}(x(\mathbf{p},t),y(\mathbf{p},t)) - d_{min} - r_{rob} \geq 0. \tag{4.35}$$

So far, the formulation assumes that the environment is static. Given the nature of the random-walk model, we simply have to circularly inflate the environment at $t > t_0$ to obtain a guaranteed bound of its motion. This results in the constraint

$$d_{edt}(x(\mathbf{p},t),y(\mathbf{p},t)) - d_{min} - r_{rob} - v_{\mathbf{z}}^{max}(t(\mathbf{p}) - t_0) \geq 0. \tag{4.36}$$

An empirical improvement of this constraint can be obtained by assuming that the uncertainty of the robot motion scales e.g. with its linear velocity. This results into an additional term $v(\mathbf{p},t)v_{noise}^{motion}$ to be subtracted from the left-hand-side of (4.36).

Looking at the simplified version of this constraint (4.35), it is clear that it would be beneficially evaluated on a spatial lattice. Regarding the version in which the environment random-walk is taken into account (4.36), it is still advised to perform its evaluation on a spatial lattice, as such a model is still applicable when $v_{\mathbf{z}}^{max}$ is relatively small. If this is not the case, more refined environment dynamic models are advised.

Finally, we would like to note that such a constraint can be easily scaled to non-circular footprints, by discretizing the footprint-shape into a number of circular regions. For each region, the constraint is enforced on the afferent lattice.

As a follow-up of Section 3.3, we would like to present the specific formulation of the safety constraints by assuming two simple environment models, the Static-Known environment and the Random-Walk Environment.

Example 4.5 (Safety Constraints: Static-Known Environment Model). In this simplified setting, we note that the nominal state of the simulator coincides with the true state of the simulator (by neglecting the maps error). This implies that enforcing the constraint (4.35) along the planned trajectory suffices to guarantee non-collision *in the optimization horizon*. However, this does not guarantee the existence of a solution of the optimization-problem at the next controller cycle. Consider the case in which the agent is navigating with very high speed with a very short planning horizon. This can result in situations in which the agent cannot physically stop before colliding with the environment.

Typical approaches found in the literature to circumvent this problem is to ensure that the planning horizon is *sufficiently* large, where sufficiently is in general an ambiguous statement. Assuming that any obstacle-free position where the agent has zero velocity is a feasible invariant state, the constraint that guarantees safe navigation in this setting is simply:

$$v(\mathbf{p}, t_1(\mathbf{p})) = 0 \tag{4.37}$$

It is worth noting a conceptual drawback of this constraint, namely its distortion of the resulting trajectories quality. In order to obtain trajectories with increased performance, one might want to increase the optimization horizon as the last portion will always move the agent to a stand-still. Informally speaking, end-point constraints can generate

in certain situations (such as this case) a "moving-horizon bottleneck" of the resulting trajectories.

Example 4.6 (Safety Constraints: Random-Walk Environment Model). This example presents the modelling of the safety constraints in a more realistic environment model. We will observe that even in this relatively simple model, the resulting constraints are relatively complicated.

Informally, we want to enforce the following:

Guaranteed safety constraint: *The entire planned trajectory is visible from the initial state. Moreover, assuming temporal inflation of the environment with the maximum random-walk velocity from time t_0, the planned trajectory is collision-free and ends with a stop.*

Note that this statement coincides with enforcing the constraints (4.36) and (4.37) with the additional requirement that the entire trajectory is visible from the initial point.

Nominal safety constraint: *From every point along the trajectory at time t, at least one emergency trajectory (that leads to a stop) is visible. Moreover, assuming temporal inflation of the environment with the maximum random-walk velocity from time t, the emergency trajectory is collision-free.*

In the following, we would like to define, analyse and formulate the Nominal Safety Constraint.

Regarding the environment, we assume that

- the agent kinematic sub-state $\begin{bmatrix} v & \omega \end{bmatrix}^{\mathrm{T}} = \mathbf{0}$ is a candidate for a feasible invariant state,
- if the agent reaches a free-space state with the candidate feasible invariant state, it is in a feasible invariant state,
- the observed initial state of the environment is error-free for the visible region of the on-board sensors.

Note that the last assumption regarding the visible region of the on-board sensors is typically neglected in other navigation approaches. However, we consider it important especially for navigation in narrow or cluttered environments at moderate to high speeds. Without any additional information, we can formulate the nominal initial state of the environment

$$\begin{aligned} \mathbf{z_0} = \{\mathbf{p}^i \mid &\text{an obstacle is sensed at } \mathbf{p}^i \text{ or} \\ &\mathbf{p}^i \text{ is an obstacle in the static map}\}, \end{aligned} \tag{4.38}$$

i. e. the set of all expected obstacle points $\mathbf{p}^i$ (in 2D space) relating to a non-free location. In the following, we have to define a dynamic model of the environment. For simplicity, we will assume that the obstacles are subject to a bounded random-walk model (with

the maximum random-walk velocity with $v_{\mathbf{z}}^{max}$). Similarly to Example 4.5, this implies that our nominal simulator is

$$\mathbf{z}(t) = \{\mathbf{p}^i \mid \mathbf{p}^i \in \mathbf{z_0}\} \tag{4.39}$$

and the guaranteed simulator is

$$\begin{aligned}\mathbf{z}(t) = \{\mathbf{p}^i \mid \forall i, \exists \mathbf{p}^j \in \mathbf{z_0}, \text{ such that } \Big(||\mathbf{p}^i - \mathbf{p}^j|| \leq v_{\mathbf{z}}^{max}(t-t_0)\Big) \wedge \\ \forall \theta,\ \mathbf{x}(t) \neq \begin{bmatrix}(\mathbf{p}^i)^{\mathrm{T}} & \theta & 0 & 0\end{bmatrix}^{\mathrm{T}} \vee \\ \mathbf{p}^i \text{ not visible from } \mathbf{x}(t_0)\}\end{aligned} \tag{4.40}$$

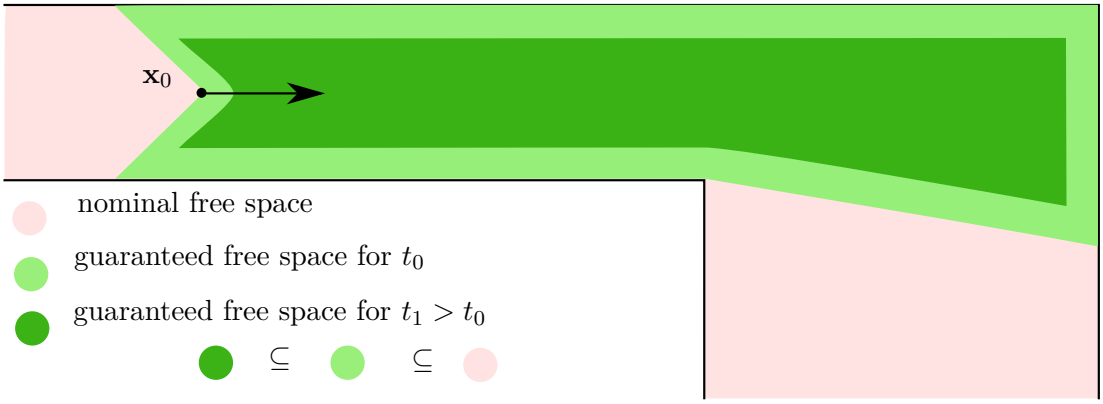


Figure 4.3: Nominal and guaranteed free space with respect to the initial agent state, taking into account sensor visibility and a bounded random-walk environment model

An illustration of the resulting free space from the two simulators is presented in Figure 4.3. For evaluating the distance to the environment, we can again make use of the function d_{edt}. Note, however, that in the random-walk model, the guaranteed minimum distance from a point $\mathbf{p}$ to the closest obstacle point is $d_{obst}^{min}(\mathbf{x}, \Delta t = t - t_0) = \max(0, d_{edt}(\mathbf{x}) - v_{\mathbf{z}}^{max}\Delta t)$.

Additionally, we require a measure to evaluate whether a point $\mathbf{p}$ will be visible from a future agent location $\mathbf{x}$. Thus, we define the function viz as

$$viz(\mathbf{x}, \mathbf{x}_1) = \min\{\ d_{obst}^{min}(\mathbf{x}', t - t_0) \mid \forall \mathbf{x}' \in [\mathbf{x}, \mathbf{x}_1]\} \tag{4.41}$$

with time t of state $\mathbf{x}$ and the line segment $[\mathbf{x}, \mathbf{x}_1]$ between the positions of $\mathbf{x}$ and the end point $\mathbf{x}_1$. Note that this function evaluates to 0 only if the end point $\mathbf{x}_1$ is not visible from location $\mathbf{x}(t)$ at time t. For efficient implementation, one could make use of the *edt* to skip evaluation points when obstacles are far away from the evaluated line segment.

With this definition in mind, the missing constraint for the Guaranteed Safety Constraints is

$$viz(\mathbf{x}_0, \mathbf{x}(t)) > 0, \quad \forall t \in [t_0, t_1], \tag{4.42}$$

which is discretized on a spatial lattice.

Continuing towards the Nominal Safety Constraints, let us have a closer look at the $\mathcal{TFI}$ in order to efficiently evaluate it. Exploiting the fact that a feasible invariant state can be located in any free point, one can argue that it is sufficient to sample a sub-set of $\mathcal{TFI}$ for which the duration of the trajectories is small. Such emergency trajectories are the ones that quickly move the agent from a state $\mathbf{x}_0$ into a feasible invariant state. As the trajectory of the agent depends only on its input, it is motivated to precompute such trajectory candidates for discretized values of initial kinematic state $\begin{bmatrix} v & \omega \end{bmatrix}^{\mathrm{T}}$ in an obstacle-free environment with the pose of the agent coinciding with the origin. Thus, in order to evaluate whether such a trajectory $\in \mathcal{TFI}$, one has to perform just a rigid transformation to the trajectory and then make use of the previously defined d_{obst}^{min} and viz.

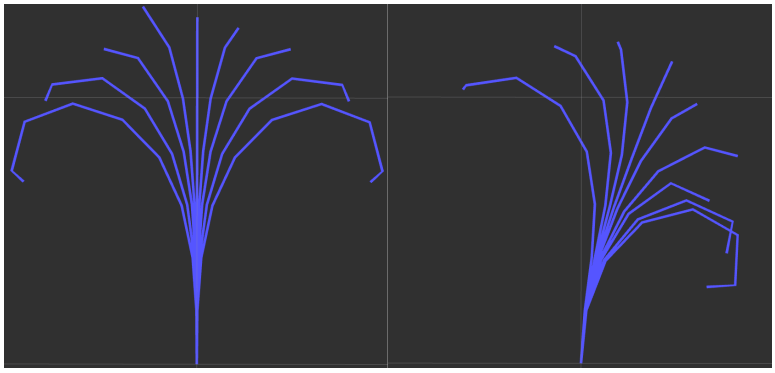


Figure 4.4: Precomputed trajectories towards a feasible-invariant state. Such trajectories are precomputed for initial states $\begin{bmatrix} v_0 & \omega_0 \end{bmatrix}^{\mathrm{T}}$ that span a discrete 2D space.

In the following, we refer with $\mathbf{x_{tf}}_i^j$ as the sampled point $1 \leq j \leq m$ along the precomputed trajectory $1 \leq i \leq n$ after the rigid transformation in the coordinate frame $\mathbf{x}(t_0)$. Finally, our function evaluating whether a trajectory towards a feasible invariant state exists from a state $\mathbf{x}(t)$ is

$$\frac{1}{n} \sum_{i=1}^{n} \sqrt[m]{\prod_{j=1}^{m} d_{obst}^{min}(\mathbf{x_{tf}}_i^j, t) viz(\mathbf{x}(t), \mathbf{x_{tf}}_i^j)} \geq 0 \tag{4.43}$$

Note that this function has the value of zero if at least one point on every trajectory i is inside an obstacle or not visible from $\mathbf{x}(t)$. Moreover, this function is continuous in x_0, y_0 and θ_0. Using bi-linear interpolation in the dimensions v_0 and w_0 of the precomputed trajectories sets, the function is continuous in all the agent state dimensions. Thus, it can

be used for enforcing the safety constraints while solving a gradient-based optimization problem.

Summarising, the constraints required for enforcing the Nominal Safety Constraints are (4.35) (non-collision assuming nominal environment) and (4.43) (guaranteed emergency motion) for every point along the planned trajectory. In practice, a spatial lattice can be used.

4.2.3 Preconditioning

Especially in the case of first-order algorithms (such as gradient-based methods), having a lower conditioning number of the cost-function and the constraints Hessians considerably reduces the number of iterations required for convergence. Thus, instead of solving a system

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{4.44}$$

we are interested to find a left preconditioner $\mathbf{P_l}$ (right preconditioner $\mathbf{P_r}$) such that the condition number of the new system matrix is reduced [56]. This results in a system that is easier to numerically solve

$$\mathbf{A}'\mathbf{x}' = \mathbf{b}, \quad \mathbf{A}' = (\mathbf{P_l})^{-1}\mathbf{A} \text{ or } \mathbf{A}' = \mathbf{A}(\mathbf{P_r})^{-1} \tag{4.45}$$

The simplest preconditioner is the identity matrix, i. e. $\mathbf{P} = \mathbf{P}^{-1} = \mathbf{I}$. This however does not help as the transformed system is exactly the original system. At the other end of the spectrum, $\mathbf{P} = \mathbf{A}^{-1}$ and thus $\mathbf{A}' = \mathbf{P}^{-1}\mathbf{A} = \mathbf{A}\mathbf{P}^{-1} = \mathbf{I}$ has a perfect conditioning number of 1. However, finding the inverse $\mathbf{A}^{-1}$ is as difficult as solving the original problem, thus not gaining any speed-increases.

The key-point of preconditioning is to use a perconditioner that provides a trade-off between accuracy and computation-time with the aim of speeding up the entire solution of the original system.

A simple preconditioner is the Jacobi (diagonal) preconditioner $\mathbf{P} = \text{diag}(\mathbf{A})$ [56]. Clearly, computing its inverse is a trivial operation. More involved methods make use of factorizations, such as LU or QR decompositions [57] to obtain a more accurate preconditioner.

In the following, we will present how to make use of QR decomposition to compute a preconditioner required for solving the equality constraints system by an Inexact Composite-step SQP solver. In this case, we are interested to precondition the matrix $\nabla\mathbf{g}\nabla\mathbf{g}^{\mathrm{T}}$. An efficient (fast) approach is to perform a column-pivoting QR decomposition of the matrix $\nabla\mathbf{g}^{\mathrm{T}}$ [57]. For the following, we denote $\mathbf{A} = \nabla\mathbf{g}^{\mathrm{T}}$.

$$\mathbf{A}\mathbf{P} = \mathbf{Q}\mathbf{R} \tag{4.46a}$$

$$\mathbf{A} = \mathbf{Q}\mathbf{R}\mathbf{P}^{\mathrm{T}} \qquad (\mathbf{P}^{-1} = \mathbf{P}^{\mathrm{T}}) \tag{4.46b}$$

Now we want to solve the system

$$\mathbf{A}^{\mathrm{T}}\mathbf{A}\mathbf{x} = \mathbf{b} \tag{4.47a}$$
$$(\mathbf{Q}\mathbf{R}\mathbf{P}^{\mathrm{T}})^{\mathrm{T}}(\mathbf{Q}\mathbf{R}\mathbf{P}^{\mathrm{T}})\mathbf{x} = \mathbf{b} \qquad \left(\mathbf{A} = \mathbf{Q}\mathbf{R}\mathbf{P}^{\mathrm{T}}\right) \tag{4.47b}$$
$$(\mathbf{P}\mathbf{R}^{\mathrm{T}}\mathbf{Q}^{\mathrm{T}}\mathbf{Q}\mathbf{R}\mathbf{P}^{\mathrm{T}})\mathbf{x} = \mathbf{b} \qquad \text{(expanding first term)} \tag{4.47c}$$
$$(\mathbf{R}^{\mathrm{T}}\mathbf{R})(\mathbf{P}^{\mathrm{T}}\mathbf{x}) = \mathbf{P}^{\mathrm{T}}\mathbf{b} \qquad (\mathbf{Q}^{\mathrm{T}}\mathbf{Q} = \mathbf{I}) \tag{4.47d}$$
$$(\mathbf{P}^{\mathrm{T}}\mathbf{x}) = (\mathbf{R}^{-1}((\mathbf{R}^{-1})^{\mathrm{T}}(\mathbf{P}^{\mathrm{T}}\mathbf{b}))) \qquad \left(\text{solving for } \mathbf{P}^{\mathrm{T}}\mathbf{x}\right) \tag{4.47e}$$

It follows from (4.47e) that we have to perform 2 successive evaluations using the (once-computed) matrix $\mathbf{R}$.

4.3 Initial Solutions

Invariant of the method by which a control-input is chosen, a set of trajectories from which an initial solution of the problem is selected is required. Depending on the computational capabilities, desired expressibility and the problem-solving method, several sets of such initial trajectories can be distinguished:

Null Trajectory This is probably the simplest set of initial trajectories, consisting of a single trajectory having a near-zero duration.

$$\mathcal{ITB} = \{\mathbf{x}(\cdot) \mid \dot{\mathbf{x}}(t) = \mathbf{f}_{\mathbf{x}}(\mathbf{x}(t), \mathbf{u}(t)),\ \mathbf{x}(t_0) = \mathbf{x}_0,\ t \in [0, t_1],\ t_1 \to 0\} \tag{4.48}$$

Starting from such a trajectory has the main benefit that given a valid state $\mathbf{x}_0$, it will satisfy all the hard constraints. However, the optimization problem has to be sufficiently well defined (ideally convex) such that the optimization solver will be able to reach satisfactory solutions from such a poor initial condition.

Previous Trajectory Having in mind that we are interested to solve a similar optimization problems iteratively at every controller cycle, it is expected in many situations that the optimal solution at iterate t_0^{k+1} is relatively close to the optimal solution at iterate t_0^k. Thus, a good initial solution is the optimal solution of the previous cycle. It is worth noting here the importance of using soft constraints for the constraints that can become invalid when initializing the optimization problem at the next iterate. For example, an optimal solution that is very close to an obstacle might become invalidated at the next controller iteration in which the obstacle performed a small motion.

As in the case of the null-trajectory, such an initialization approach works well in practice if the optimization problem is sufficiently well defined (ideally convex). Otherwise, such an initialization will result in the solution to converge (over multiple iterates) to a local minima. A closer look towards the methods of initializing the optimization problem from a previous trajectory will be given in Section 4.4.

courtesy: Thomas M. Howard

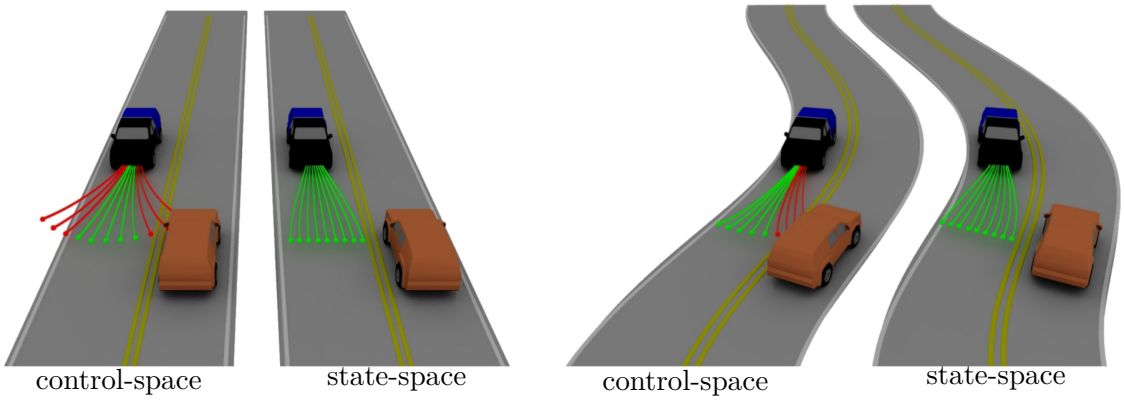


Figure 4.5: Control-space vs. state-space trajectory sampling for a vehicle on a road

The next two presented approaches are based on coarse sampling of the optimization-space to obtain a good initial solution, addressing the problem of converging (over more iterates) to a local minima of the agent trajectory. Note that the evaluation of such sampled trajectories can be computed in parallel.

Control-Space Sampling The robotics literature has proposed a relatively simple local planning algorithm based on control-space sampling [8], an approach mentioned before in this chapter (DWA). Even though crude and sub-optimal, such an approach can be used to search for an initial solution of the optimization problem. We can define the set of trajectories sampled in the control-space as

$$\mathcal{ITB} = \{\mathbf{x}_{\mathbf{u}_{end}}(\cdot) \mid \mathbf{x}_{\mathbf{u}_{end}}(0) = \mathbf{x}_0, \mathbf{u}(\mathbf{x}_{\mathbf{u}_{end}}(t_1)) = \mathbf{u}_{end}\}\,, \tag{4.49}$$

where the sampling space is the control-inputs $\mathbf{u}_{end}$ of the trajectory end-point. Keeping in mind the drawbacks of such an approach, it is advised to perform this sampling for relatively short temporal intervals and to let the optimization improve the trajectory until the desired horizon.

State-Space Sampling An improvement over control-space sampling is to consider sampling trajectories in the state-space of their end-point $\mathbf{x}_{t_1}$ [58].

$$\mathcal{ITB} = \{\mathbf{x}_{\mathbf{x}_{end}}(\cdot) \mid \mathbf{x}_{\mathbf{x}_{end}}(0) = \mathbf{x}_0, \mathbf{x}_{\mathbf{x}_{end}}(t_1) = \mathbf{x}_{end}\} \tag{4.50}$$

This results in a more directed search of the initial solution, sampling trajectories based on desired locations of their end-point. However, in practice such trajectories are typically obtained by means of optimization, thus having to be pre-computed offline. An illustration of control-space versus state-space sampling is given in Figure 4.5.

4.4 Temporal Synchronization

In this section, we would like to have a closer look at means by which the optimization module and the low-level-control modules interact and can be temporally synchronized in an asynchronous setting.

An important operation for being able to re-use the computed optimal solution of the optimization problem from a previous cycle is the temporal-shift of its data, presented in Algorithm 2.

Algorithm 2 $shift\_traj\_data(\mathbf{x}_0, \mathbf{p}, t_0, t_0')$

$\mathbf{x}_0' \leftarrow traj_{\mathbf{x}}(\mathbf{x}_0, \mathbf{p}, t_0, t_0')(t_0')$
$\mathbf{p}'$ such that: $traj_{\mathbf{x}}(\mathbf{x}_0', \mathbf{p}', t_0', t_1)(\tau) = traj_{\mathbf{x}}(\mathbf{x}_0, \mathbf{p}, t_0, t_1)(\tau), \forall \tau \in [t_0', t_1]$
return $\mathbf{x}_0', \mathbf{p}', t_0'$

Given a trajectory of the system starting from time t_0 and encoded by the system initial state $\mathbf{x}_0$ and parameters $\mathbf{p}$, the function $shift\_traj\_data$ returns the system trajectory shifted into the future to time t_0', $t_0' \geq t_0$, through the new initial state $\mathbf{x}_0'$, initial time-point t_0' as well as the parameters $\mathbf{p}'$ such that the initial trajectory and the new trajectory coincide on the interval $[t_0', t_1]$. Having this defined, we will now discuss the temporal synchronization algorithms used in two main controller modes:

MPC Mode Algorithm 3 presents the main loop run by the optimization module while taking into account time synchronization:

Algorithm 3 $opt\_from\_last\_valid$
receives through message from Observers: $\hat{\mathbf{x}}$ at time $t_{\hat{\mathbf{x}}}$

$t_0 \leftarrow global\_time\_now()$
$\mathbf{x}_0^l, \mathbf{p}^l, t_0^l \leftarrow shift\_traj\_data(\mathbf{x}_0^l, \mathbf{p}^l, t_0^l, t_{\hat{\mathbf{x}}})$ *// shift last valid to measurement time*
$\mathbf{x}_0^l \leftarrow update\_from\_measurements(\mathbf{x}_0^l, \bar{\mathbf{x}})$ *// set initial state from measurement*
if *last_traj_valid* **then**
    $\mathbf{x}_0^l, \mathbf{p}^l, t_0^l \leftarrow shift\_traj\_data(\mathbf{x}_0^l, \mathbf{p}^l, t_0^l, t_0)$ *// shift last valid to* t_0
    $\mathbf{p}_l^* \leftarrow \mathbf{p}^*$ *// load trajectory from previous cycle*
    $t_{1_l}^* \leftarrow t_1^*$
end if
$\mathbf{x}_0, \mathbf{p}, \sim \leftarrow shift\_traj\_data(\mathbf{x}_0^l, \mathbf{p}^l, t_0^l, t_0 + \Delta t)$ *//shift to cycle end time* $t_0 + \Delta t$
$\mathbf{x}_0, \mathbf{p} \leftarrow initialize\_opt\_problem(\mathbf{x}_0, \mathbf{p}, t_0 + \Delta t)$
$\mathbf{x}_0^*, \mathbf{p}^*, t_1^* \leftarrow optimize(\mathbf{x}_0, t_0 + \Delta t, \mathbf{p})$
$last\_traj\_valid \leftarrow isValid(\mathbf{x}_0^*, \mathbf{p}^*, t_0 + \Delta t, t_1^*)$
if *last_traj_valid* **then**
    $send\_to\_controller(\mathbf{x}_0^*, \mathbf{p}^*, t_0 + \Delta t, t_1^*)$
end if

The key idea of the algorithm is to keep track of which trajectories have been sent to the low-level controller to execute and predict future states of the system accordingly. At the beginning of the cycle, the last valid optimal trajectory (last trajectory sent to the low-level controller) is time-shifted to the time-instant in which the state measurements $\hat{\mathbf{x}}$ are recorded $t_{\hat{\mathbf{x}}}$.

Remark: In general, the optimization state $\mathbf{x} \in \mathbb{R}^n$ is larger than the state measurements $\hat{\mathbf{x}}$ of the optimization module. For example, a system whose low-level controller stabilizes the velocities of the system might use an optimization module that has constraints in the velocity states as well as the acceleration states of the system. However, as the low-level controller already stabilizes the system velocities, only the states relating to the pose-space have to be fed back to the optimization module. More formally, assuming that the low-level controller (exponentially) asymptotically stabilises a state $\mathbf{x}_l \subseteq \mathbf{x}$, the fed back sub-state $\hat{\mathbf{x}} \subseteq \mathbf{x}$ to the optimization module (step 3 in Algorithm 3) is the state with the highest dimensionality, such that all the scalar relative degrees of the system $\mathbf{x}'$ (with $\mathbf{y}' = \hat{\mathbf{x}} = \mathbf{f}'(\mathbf{x}', \mathbf{u}'), \mathbf{u}' = \mathbf{x}_l$) are positive. Otherwise, generally resonance effects will occur on the optimization problem initial state $\mathbf{x}_0$.

Later-on, if the optimization during the previous iterate was successful, the last trajectory sent to the low-level controller is further advanced to the beginning of the cycle and it is overwritten by the new trajectory, except the propagated measurement states. This ensures that the measurement information is propagated up to the point where the new trajectory is defined. Finally, the trajectory is further time-shifted to the expected temporal end-point of the current cycle and the optimization problem initial solution is computed. This approach deals with the non-negligible computation time of the optimization module, accounting for the fact that the system cannot be physically influenced until the end of the cycle. Note that in the case of initialization by previous trajectory, the initialization step performs no operation.

It follows from the nature of Algorithm 3 that we do not require a constant cycle-time of the module. Rather, we require that for each cycle, Δt is guaranteed to be at least the computational duration needed by the module. This allows the module to vary its frequency, for example running at shorter cycle-times in cases where a previous trajectory is re-optimized.

Such an asynchronous approach requires a synchronization procedure also in the low-level controller, presented in Algorithm 4.

Algorithm 4 *low_level_ctrl_sync*
receives through message from MHTP: $\mathbf{x}^{new}(\cdot), t_0^{new}$
receives through message from Observers: $\hat{\mathbf{x}}$ at time t_0

1: $t_0 \leftarrow global\_time\_now()$
2: **if** $t_0 \geq t_0^{new}$ **then**
3: $\quad \mathbf{x}^a(\cdot) \leftarrow \mathbf{x}^{new}(\cdot)$
4: **end if**
5: $apply\_control\_law(\hat{\mathbf{x}}, \mathbf{x}^a(t_0), t_0)$

The general idea is that the controller should evaluate the last received trajectory at each of its cycles. When a new trajectory it is received, it is stored until the temporal point in which it becomes valid. At this point, the previous trajectory is discarded and the new trajectory is evaluated.

As a remark, note that such a control scheme additionally requires that the optimization module runs at shorter cycles than its planned trajectory duration. Otherwise, the low-level controller as well as the optimization module state-prediction process will reach the undefined domain of the previously valid trajectory.

Stabilized MHTP Mode: Taking into account the algorithms presented for the case of MPC, it follows that Stabilized MHTP relates to a special case. In this setting, the low-level controller (exponentially) asymptotically stabilizes the entire state of the optimization module. Thus, there exists no state that is at a strictly higher integration order. This implies that there is no state that is fed back at step 3 in Algorithm 3.

In such a manner, implementation of an MPC or a Stabilized MHTP no longer assumes the instant solution of the optimization problem. Thus, it is suited for more complex to evaluate problems as well as not real-time, distributed frameworks and implementations.

5 A Suitable Cost Function

Until this point, we have assumed that we are solving an optimization problem with respect to a scalar cost-function. However, we did not address the nature, form or requirements such a function should meet for achieving the desired results. In this chapter, the typical objectives that are to be encoded into the optimization cost-function as well as formulations that lead to robust results are discussed.

5.1 Reaching a Goal

One of the fundamental tasks of navigation is to arrive to a given goal without colliding with obstacles. In this section, a formal definition of obstacles and drivable space will be given. Afterwards, an analysis of cost functions that lead the agent to reach a desired goal are analysed.

We would like to address in the following a simplified sub-part of the optimization problem related to navigation: a point particle converging from an arbitrary location to a goal, given a (non-convex) set representing the free/drivable space, bounded by obstacles.

More formally, let the obstacle state $\mathbf{z}$ span the space $\mathcal{Z} = \text{span}(\mathbf{z})$. With this, we define the differentiable manifold of (drivable) free-space as $\mathcal{X} = \mathcal{F}(\mathbf{z}) = \mathbb{R}^n \backslash \mathcal{Z}$, with its border $(\partial\mathcal{X})_{\mathcal{O}}$ representing the contour of the obstacles. Relaxing the optimization problem by assuming static obstacles, we can formulate a (static) optimization problem

$$\min_{\mathbf{x}} \quad J(\mathbf{x}) = \frac{1}{2} d^2(\mathbf{x}) \tag{5.1}$$

$$\text{subject to} \quad \mathbf{x} \in \mathcal{X} \backslash (\partial\mathcal{X})_{\mathcal{O}}. \tag{5.2}$$

Note that in comparison with the cost-function (3.6), the integral cost term vanishes. Moreover, the constraints modelling the dynamics of the system are not present. One can interpret (5.1) as the simplification of (3.7) by having only an end-cost term, no temporal dependency, no additional constraints (except the free-space boundary) as well as no dynamics of the underlying system.

Thus, we are interested in finding a suitable metric $d(\mathbf{x})$ such that (5.1) becomes a convex optimization problem. For the beginning, let us have a look at the definition of metrics.

5.1.1 Metrics and Norms

Metrics Formally, a metric on a set $\mathbf{X}$ is a function $d : \mathbf{X} \times \mathbf{X} \to \mathbb{R}_+$ such that $\forall \mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbf{X}$, the following conditions are satisfied [59]

$$d(\mathbf{x}, \mathbf{y}) \geq 0, \tag{5.3a}$$
$$d(\mathbf{x}, \mathbf{y}) = 0 \;\Leftrightarrow\; \mathbf{x} = \mathbf{y}, \tag{5.3b}$$
$$d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x}), \tag{5.3c}$$
$$d(\mathbf{x}, \mathbf{z}) \leq d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, \mathbf{z}). \tag{5.3d}$$

In many cases, metrics are induced by norms, a concept strongly connected to vector-spaces.

Norms A norm on a vector space $\mathbf{X}$ is a function $||\mathbf{x}|| : \mathbf{X} \to \mathbb{R}_+$ such that $\forall \mathbf{x} \in \mathbf{X}$, the following conditions are satisfied [60]

$$||\mathbf{x}|| \geq 0, \tag{5.4a}$$
$$||\mathbf{x}|| = 0 \;\Leftrightarrow\; \mathbf{x} = 0, \tag{5.4b}$$
$$||\mathbf{x} + \mathbf{y}|| \leq ||\mathbf{x}|| + ||\mathbf{y}||, \tag{5.4c}$$
$$||\alpha \mathbf{x}|| = |\alpha| \; ||\mathbf{x}||. \tag{5.4d}$$

The typical used norms are the so called p-Norms

$$||\mathbf{x}||_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}, \quad \mathbf{x} \in \mathbb{R}^n, \tag{5.5}$$

where for $p = 2$ we have the typical Euclidean norm.

Note that every norm can induce a metric (by defining $d_p(\mathbf{x}, \mathbf{y}) = ||\mathbf{x} - \mathbf{y}||_p$) but the converse does not hold. As it is known that every norm is a convex function, one might consider a good candidate for our cost function

$$d(\mathbf{x}) = ||\mathbf{x} - \mathbf{x}_{goal}||_2 \tag{5.6}$$

with the (constant) pose $\mathbf{x}_{gloal}$ of the goal location. To formulate the entire optimization problem, we still have to encode the constraint that relates to the particle being on the inside of the domain $\mathcal{X}$. For this, we can make use of the Euclidean Distance Transform.

Euclidian Distance Transform (EDT) Let us define the EDT

$$d_{edt}(\mathbf{x}) = \min\{ \; ||\mathbf{x} - \mathbf{p}||_2 \mid \forall \mathbf{p} \in (\partial \mathcal{X})_{\mathcal{O}} \;, \;\; \mathcal{X} \subseteq \mathbb{R}^n \} \tag{5.7}$$

i. e. the distance to the closest obstacle (with respect to the 2-norm). Figure 2.7 illustrates such a field.

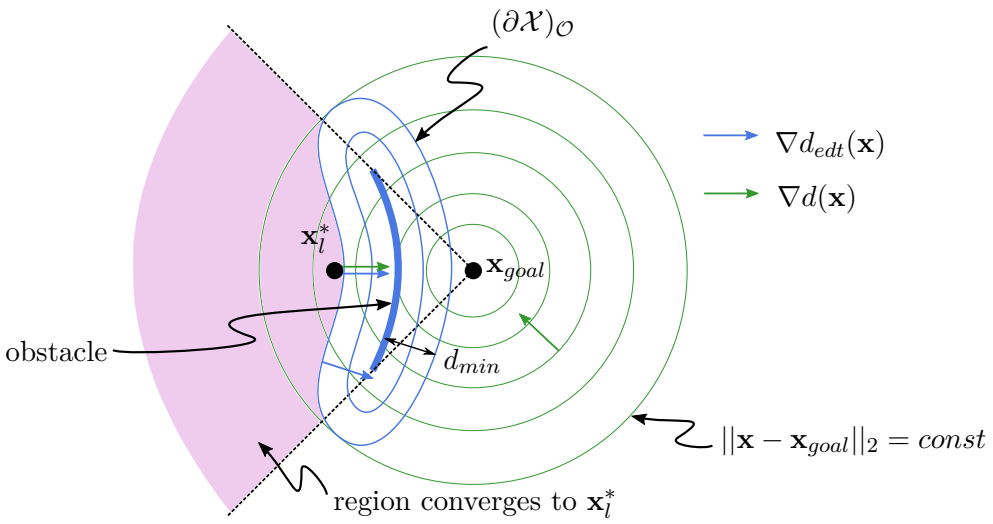


Figure 5.1: Local-minima when minimizing the Euclidean Distance. All the points in the pink region will converge to $\mathbf{x}_l^*$. Note that the obstacle boundary $(\partial\mathcal{X})_\mathcal{O}$ is at distance d_{min} from the actual obstacle. This is simple to evaluate in practice when using an EDT of the region.

As its computation is relatively efficient, it is useful to compute such a field for the purposes of obstacle inflation (to easily account for circular agent footprints). Also, such a field should be sufficient for the purpose of not colliding with obstacles (it is continuous differentiable, its gradient pointing always away from the closest obstacle), which motivates its use at least in emergency cases in which going away from obstacles is the only objective.

Combining (5.6) and (5.7), we obtain the formulation of the optimization problem

$$\min_{\mathbf{x}} \quad J(\mathbf{x}) = \frac{1}{2}||\mathbf{x} - \mathbf{x}_{goal}||_2^2 \tag{5.8}$$

$$\text{subject to} \quad d_{edt}(\mathbf{x}) - d_{min} \geq 0. \tag{5.9}$$

As $J(\mathbf{x})$ is convex, one might ask why such norm-induced metric would not result in a convex optimization problem. However, for our optimization problem to be convex, according to (5.2), it is required additionally that the allowed free space $\mathcal{X}\backslash(\partial\mathcal{X})_\mathcal{O} \subseteq \mathbb{R}^n s$ is a convex set, which is generally not the case. This results in local-minima solutions, as illustrated in Figure 5.1.

Thus, even in the simplified setting of a point-particle, using naive approaches lead to a formulation containing multiple optimal points (and thus local minima). Being interested to obtain a formulation that is convex, let us consider the entire optimization problem (5.1). For this, we will first introduce the Karush-Kuhn-Tucker (KKT) optimality conditions [50]

Theorem 5.1 (Karush-Kuhn-Tucker (KKT) necessary optimality conditions)**.** *Assume that* $\mathbf{x}^*$ *is a Minimum of the optimization problem:*

$$\begin{aligned} \min_{\mathbf{x}} \quad & J(\mathbf{x}) \quad \textit{Cost Function} && \textit{(5.10a)} \\ \textit{subject to} \quad & g_i(\mathbf{x}) = 0, \quad i = 1, ..., p \quad \textit{Equality Constraints} && \textit{(5.10b)} \\ & h_i(\mathbf{x}) \leq 0, \quad i = 1, ..., q \quad \textit{Inequality Constraints} && \textit{(5.10c)} \end{aligned}$$

with $f, g_1, \ldots, g_p, h_1, \ldots, h_q \in C^1$*. Additionally, let* $\mathbf{x}^*$ *be a regular point of the constraints (5.10b), (5.10c). Then there exists a unique Lagrange-Multiplicator* $((\boldsymbol{\lambda}^*)^{\mathrm{T}}, (\boldsymbol{\mu}^*)^{\mathrm{T}})$ *with* $\boldsymbol{\lambda}^* \in \mathbb{R}^p$ *and* $\boldsymbol{\mu}^* \in \mathbb{R}^q$ *so that the following equations hold:*

$$\begin{aligned} (\nabla J)(\mathbf{x}^*) + (\nabla \mathbf{g})(\mathbf{x}^*)\boldsymbol{\lambda}^* + (\nabla \mathbf{h})(\mathbf{x}^*)\boldsymbol{\mu}^* &= \mathbf{0} && \textit{(5.11a)} \\ \boldsymbol{\mu}^* &\geq \mathbf{0} && \textit{(5.11b)} \\ \mathbf{h}^{\mathrm{T}}(\mathbf{x}^*)\boldsymbol{\mu}^* &= \mathbf{0} && \textit{(5.11c)} \\ \mathbf{h}(\mathbf{x}^*) &\leq \mathbf{0} && \textit{(5.11d)} \end{aligned}$$

In order to show that our optimization problem has only one (i. e. global) optimum, it would suffice to show that equations (5.11a)-(5.11d) are satisfied only in the region of our goal. To this end, simplifying these equations for our problem by removing the equality constraints and letting the inequality constraint be scalar we obtain

$$\begin{aligned} (\nabla f)(\mathbf{x}^*) + (\nabla h)(\mathbf{x}^*)\mu^* &= \mathbf{0} && \text{(5.12a)} \\ \mu^* &\geq 0 && \text{(5.12b)} \\ h(\mathbf{x}^*)\mu^* &= 0 && \text{(5.12c)} \\ h(\mathbf{x}^*) &\leq 0. && \text{(5.12d)} \end{aligned}$$

Equation (5.12) can be interpreted as follows: suppose that our inequality constraint (5.2) is defined such that its gradient points away from our free space. Then it would suffice to show that ∇d is non-zero for every interior point of our domain and additionally, along the obstacle domain boundary, ∇d points towards the inside of our domain $\mathcal{X}$.

We are thus interested in a metric that implicitly encodes the topology of the obstacles. To this end, one might notice the restrictive requirement in the characteristic that a norm has to satisfy (5.4d). Given the unstructured nature of environments, we would like to avoid having such a requirement. This will motivate looking for suitable metrics which are not norm-induced.

5.1.2 PDE Candidates

Given their differential formulation, PDEs might be a suitable mathematical structure for the purpose of searching for and analysing a candidate metric that would convexify (5.1).

We will now take a look at two different PDEs that will be proven to provide the desired convex formulation.

Functions solving the Laplace Equation The Laplace equation

$$\nabla^2 d(\mathbf{x}) = 0, \quad \forall \mathbf{x} \in \mathcal{X} \tag{5.13}$$

is a widely used PDE, playing an important role in e.g. electrodynamics. In the simplified case of electrostatics, its solution describes the distribution of the electric field in a domain given the charges of the domain boundaries. Intuitively, if we are following the gradient of the electric field from a positively charged domain boundary, we will always arrive to a negative-charged boundary. This implies that starting from any point from our domain and following the gradient, we will eventually arrive to the negative-charged boundary.

From this consideration, going back to vehicle navigation, we could take an approach to define a boundary value problem (Dirichlet type) in which at the obstacles contour, $d(\mathbf{x})|_{\mathbf{x}\in(\partial\mathcal{X})_\mathcal{O}} = 1$. We have to additionally define the contour of the goals as $(\partial\mathcal{X})_\mathcal{G}$ and define $d(\mathbf{x})|_{\mathbf{x}\in(\partial\mathcal{X})_\mathcal{G}} = 0$ [61]. This leads to the following theorem

Theorem 5.2 (Convex navigation problem using Laplace equation)**.** *Given is a free-space set $\mathcal{X}$ with a closed obstacle boundary $(\partial\mathcal{X})_\mathcal{O}$ and a closed goal boundary $(\partial\mathcal{X})_\mathcal{G}$. The solution of the Laplace equation (5.13) with the boundary conditions*

$$d(\mathbf{x})|_{\mathbf{x}\in(\partial\mathcal{X})_\mathcal{O}} = 1 \tag{5.14a}$$
$$d(\mathbf{x})|_{\mathbf{x}\in(\partial\mathcal{X})_\mathcal{G}} = 0 \tag{5.14b}$$

when applied as cost-function in the optimization problem (5.1) results in a convex optimization problem.

Proof. (sketch) We will spilt the proof by showing that the function gradient is non-vanishing in the interior of the domain and that the gradient points inwards at the obstacles boundary. Based on the KKT conditions, it will thus follow that the only optimal points are those lying at the goal contour.

As $d(\mathbf{x})|_{\mathbf{x}\in(\partial\mathcal{X})_\mathcal{O}} \neq d(\mathbf{x})|_{\mathbf{x}\in(\partial\mathcal{X})_\mathcal{G}}$, it follows that $d(\mathbf{x})$ cannot have a constant value over $\mathcal{X}$. Moreover, according to the maximum principle of harmonic functions [62], a non-constant harmonic function $d(\mathbf{x})$ cannot attain a maximum (or minimum) at an interior point of its domain. This implies that

$$\nabla d(\mathbf{x}) \neq \mathbf{0}\,, \qquad \forall \mathbf{x} \in \mathcal{X}\backslash\partial\mathcal{X} \tag{5.15}$$

Regarding the obstacle boundary, let the tangent hyperplane to the boundary $\partial\mathcal{X}$ at point $\mathbf{x}$ be defined by $\mathbf{w}^\mathrm{T}\mathbf{x} = 0$. As the manifold induced by d behaves locally like a linear vector space, it follows that $\mathbf{w} \times (\nabla d(\mathbf{x})) = 0$ (d has a constant value along the boundary). Using (5.15), it follows that $\nabla d(\mathbf{x})$ is perpendicular (and inwards) to the boundary.

Together with (5.15), the theorem is proven. □

Algorithmic computation In practice, the computation of the field is typically performed numerically by the technique of successive over-relaxation (SOR) [63]. This introduces numerical issues related to the gradient vanishing in the vicinity of d taking values of 1 and is thus not scalable with respect to the resolution. A solution is to map the problem in Log-Space [61], a technique familiar to the field of Machine Learning. However, this drastically increases the computational time and is thus not a valid candidate for online use in Optimal Navigation.

Functions solving the Eikonal Equation The Eikonal equation

$$|\nabla d(\mathbf{x})| = \frac{1}{u(\mathbf{x})}, \quad \forall \mathbf{x} \in \mathcal{X} \backslash (\partial \mathcal{X})_{\mathcal{O}} \tag{5.16}$$

is a PDE with usage in electrodynamics and fluid dynamics.

An intuitive analogy is to consider the water-flood in a maze. We thus start to flood the maze from a broken pipe located at the navigation goal $\mathbf{x}_{goal}$. The solution of the Eikonal at every point $\mathbf{x}$ will contain the time at which the first drop of water reached $\mathbf{x}$. Note that for the simplified case in which $u(\mathbf{x}) = 1, \forall \mathbf{x} \in \mathcal{X} \backslash (\partial \mathcal{X})_{\mathcal{O}}$, the solution is equivalent with the Geodesic Distance (i. e. the minimum distance from point $\mathbf{x}$ to the goal through the maze). For the more general case, an area in which $u(\mathbf{x}) < 1$ translates to our analogy to the fact that the water expands slower in that region. We can interpret this analogy the other way around. If starting from a generic point, we invert time and move upstream the water flow, we will arrive at the source. This consideration motivates the following theorem

Theorem 5.3 (Convex navigation problem using Eikonal equation)**.** *Given is a free-space set $\mathcal{X}$ with a closed obstacle boundary $(\partial \mathcal{X})_{\mathcal{O}}$ and a closed goal boundary $(\partial \mathcal{X})_{\mathcal{G}}$. The solution of the Eikonal equation (5.13) with the conditions*

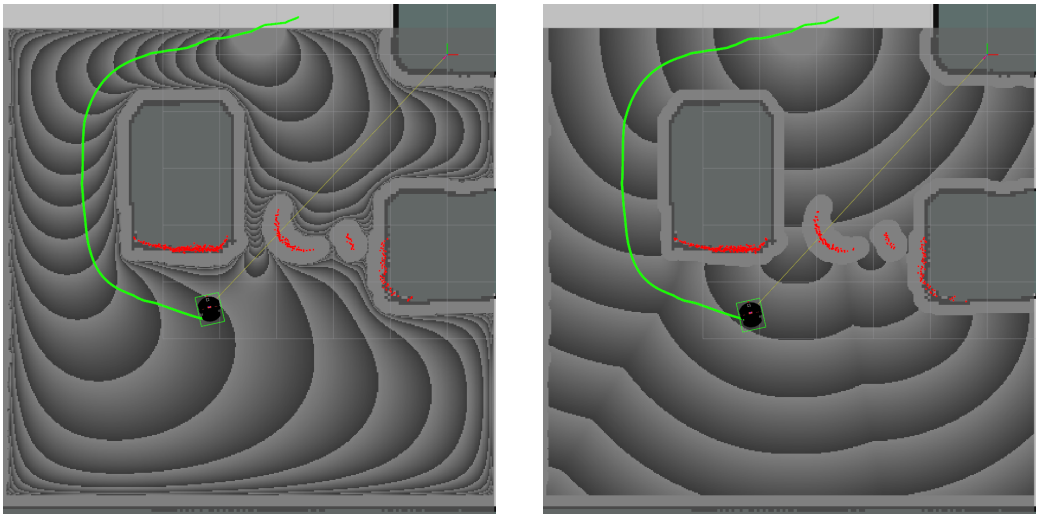
$$u(\mathbf{x}) \neq 0, \quad \forall \mathcal{X} \backslash (\partial \mathcal{X})_{\mathcal{O}} \tag{5.17a}$$
$$u(\mathbf{x}) = 0, \quad \forall (\partial \mathcal{X})_{\mathcal{O}} \tag{5.17b}$$
$$d(\mathbf{x})|_{\mathbf{x} \in (\partial \mathcal{X})_{\mathcal{G}}} = 0, \tag{5.17c}$$

also denoted as a Fast Marching Field (FMF), when applied as cost-function in the optimization problem (5.1) results in a convex optimization problem.

Proof. (sketch) Similarly to the proof for the Laplace equation, we will spilt the proof in two parts. From (5.17a) and the definition of the Eikonal equation, it follows that

$$\nabla d(\mathbf{x}) \neq \mathbf{0}\,, \qquad \forall \mathbf{x} \in \mathcal{X} \backslash (\partial \mathcal{X})_{\mathcal{O}} \tag{5.18}$$

Regarding the obstacles contour, taking into account the condition (5.17b), it follows that the obstacles boundary will have the gradient norm $\to \infty$. From here, the proof is identical with the case of the Laplace equation. □



(a) Solution of Laplace Equation – Harmonic Field

(b) Solution of Eikonal Equation (unity – constant velocity map) – Geodesic Distance Field

Figure 5.2: Different Distance Fields. Note that the goal region is defined as the set of all free-space points within a certain radius from the active global path end-point (with respect to the size of the boundaries of the local map)

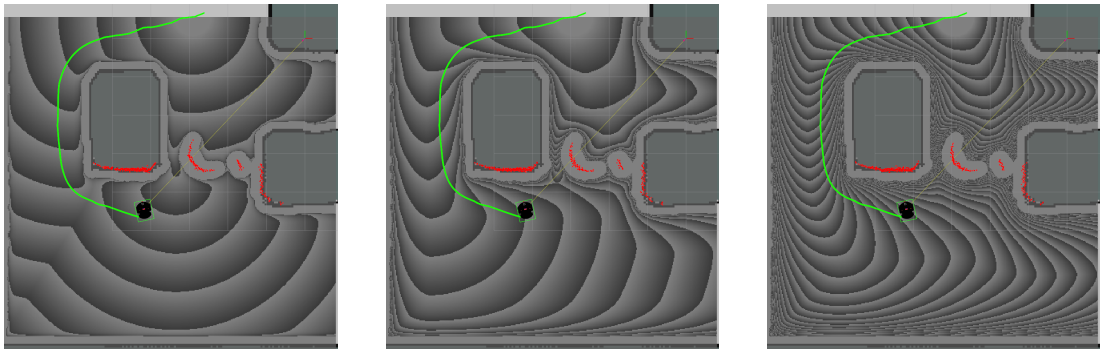
Figure 5.2 illustrates the solutions of the Laplace and Eikonal equations in a generic navigation setting.

Gradients orientation at the boundary – Making use of velocity maps Even though formally, the gradient of the FMF at the boundary is orthogonal to the boundary, in practice this is no longer the case. The reason is that even though at the obstacle boundary $(\partial\mathcal{X})_{\mathcal{O}}$, $u(\mathbf{x}) = 0$, infinitesimally close to the boundary $u(\mathbf{x}) \neq 0$. For example, in the constant velocity-map setting (Geodesic Distance), one can visualize the scenario in which the gradient is parallel to the boundary (when the shortest path curves around an obstacle). Even though this theoretically suffices to show that optimal points are present only on the goal boundary $(\partial\mathcal{X})_{\mathcal{G}}$, due to the discretization and numerical errors, this can lead the optimization process to unwanted local minima close to the boundary when explicitly enforcing the constraint $\mathbf{x} \in \mathcal{X}$.

A simple solution making use of the degree of freedom of specifying the velocity map $u(\mathbf{x})$ is to reduce it when closer to obstacles, i. e.

$$u(\mathbf{x}) = \min\left(\frac{d_{edt}(\mathbf{x})}{d_{obst_{max}}}, 1\right) \tag{5.19}$$

This ensures that sufficiently close to the boundary, $\nabla d(\mathbf{x})$ is not parallel to the boundary. Moreover, such a metric becomes dependent on the narrowness of the corridor, resulting in



(a) velocity map with damped velocities closer to $0.2m$ from boundary

(b) velocity map with damped velocities closer to $1m$ from boundary

(c) velocity map with damped velocities closer to $3m$ from boundary

Figure 5.3: Solutions of the Eikonal equation with velocity damping in vicinity of obstacles. Notice that in a), the shortest distance from the agent position to the goal is through the narrow passage. For b) and c), this is no longer the case.

shorter distances along wider corridors even though with respect to the geodesic distance it would be otherwise, see Fig. 5.3.

Algorithmic computation A historical approach for computing the (constant velocity map) solution to the Eikonal equation is using the Dijkstra algorithm [64]. More modern (and general) approaches are based of the Fast Marching Method (FMM), making use of upwind finite difference [65]. Such algorithms preserve the one-pass characteristic, i. e. they start from the goal region and carefully expand towards unvisited new pixels (each pixel being visited only once).

Comparison Now we would like to summarize the strengths and weaknesses of using each of the two presented PDEs (and their variations) as cost-function for optimal navigation, summarized in Table 5.1.

Evaluation metric	Harmonic	Log-Space Harmonic	FMF	FMF_{damp}
Computation Time	-	- -	+ +	+(+)[1]
Anytime Algorithm	yes	yes	no	no
Query Valid Path	follow gradient	follow gradient	1-cell query	1-cell query
Gradient at Boundary	orthogonal	orthogonal	parallel	quasi-orthogonal

Table 5.1: Qualitative analysis of distance fields

[1] For the purpose of Optimal Navigation, the computation of EDT is required anyway and thus velocity damping induces virtually no overhead.

As it has been noticed previously, the main drawback of the Harmonic function is the required computational time, especially in the scalable case (Log-Space Harmonic). Regarding the Fast Marching Methods, they are comparably very fast (by two orders of magnitude faster than Log-Space Harmonic). For the FMF with damped velocities a computational overhead is added, necessary for computing the EDT.

An interesting property of the Harmonic functions solved by successive over-relaxation is that they result in an any-time algorithm. That is, at every algorithm iteration, the partial solution is a coarse approximation of the Laplace equation. This can be beneficial as one can make use of the resulting map before absolute convergence. However, this does not manage to overcome the large difference in computational requirements when compared with FMM.

An interesting characteristic that we can further evaluate is by which means we can evaluate whether we can reach the goal or not. In the case of the Harmonic functions, not being able to reach the goal would theoretically imply that the gradient of the function at the agent location would vanish. However, in practice, due to numerical errors and(or) the algorithm not yet converged, we would have to follow the gradient until a minima is reached and compare it to the goal location. In the case of FMM, however, we just have to query the function at the location of the agent, a finite value implying that the goal is reachable.

Lastly, we are interested in the (practical) situation of the function gradient at the obstacles boundary. In this case, harmonic functions behave very well, satisfying this requirement implicitly. Regarding the FMM, the variation that performs velocity-damping in the vicinity of obstacles manages to achieve a "quasi-orthogonal" shape of its gradient.

5.1.3 Sensor-processing: Layered Local Maps

Given the previously discussed possible methods, the Fast Marching Method with velocity reduction in the vicinity of obstacles has been chosen for final implementation, due to its speed and degrees of freedom. In the following, we would like to present the general steps required to perform the (efficient) computation of such field given typical inputs of the problem: (static) offline maps and a laser sensor. Algorithm 5 presents the pseudo-code of the pipeline. As the computational complexity would increase with the size of the environment (or resolution would be reduced), computing the metrics online globally is not desired. Thus, an agent-centric map (that at each cycle has the agent in its centre) is proposed.

Given the new pose of the agent, the algorithm initially translates the map data $\mathbf{M}$ to the new location (note that this can be done efficiently by using circular buffers). Afterwards, we can identify 3 conceptual steps of processing:

Computing the Obstacle EDT To also account for laser readings from previous cycles, the map layer containing the obstacles ($\mathbf{M}_o$) from last cycle is multiplied by a

$forgetness\_factor < 1$, thus being discarded when below a threshold $timeout\_thres$. The locations in which obstacles are sensed in the current iteration ($\mathbf{p}_{sns}$) are inserted in this layer. Also, at this stage, static mapped obstacles read from the global map $\mathbf{p}_{gm}$ are introduced. Finally, the obstacle distance field is computed by using values of the map layer $\mathbf{M}_{oi} > 0$ as sources.

Computing the Path EDT In many situations, enforcing a maximum deviation from the global path is desired. To this end, a path EDT is computed, by using the sequence of points relating to the path center $\mathbf{p}_{act\_route}$ as sources. Note that the computation of the two EDTs is independent and thus can be parallelized.

Computing the FMF Computation of the FMF requires as input a map with points labelled as obstacles, points labelled as goals as well as optionally a velocity map. Considering the previously computed EDTs, one can easily define the set of obstacles (including the path boundary as obstacle as well). Regarding the goals, one might want to define not only a goal point but a goal region (the agent might be required to navigate to a neighbouring location from the goal point in case the goal point is obstructed by an obstacle). If the modified velocity map is desired (thus computing FMF_{damp}), no additional processing is required as the obstacle EDT is already computed.

Algorithm 5 $compute\_local\_metrics(\mathbf{x}_{agent}, \mathbf{p}_{sns}, \mathbf{p}_{gm}, \mathbf{p}_{act\_route}, \mathbf{p}_{goal})$
parameters: $forgetness\_factor$, $timeout\_thres$, $obst\_thres$, $path\_thres$, d_vel_sat

$moveMapCenterTo(\mathbf{M}, \mathbf{x}_{agent})$

$\mathbf{M}_o \leftarrow forgetness\_factor\mathbf{M}_o$ *// blur old sensor data*
$\mathbf{M}_o(\mathbf{p}_{sns}) \leftarrow 1$ *// insert new sensor data*
$\mathbf{M}_o(\mathbf{p}_{gm}) \leftarrow 1$ *// insert static map data*
$\mathbf{M}_o(\mathbf{M}_o < timeout\_thres) \leftarrow 0$ *// discard timed-out measurements*
$\mathbf{M}_{o\_edt} \leftarrow computeDistanceField(\mathbf{M}_o)$ *// compute euclidean distance field*

$\mathbf{M}_{path} \leftarrow 0$
$\mathbf{M}_{path}(\mathbf{p}_{act\_route}) \leftarrow 1$ *// set points along active route as sources*
$\mathbf{M}_{path\_edt} \leftarrow computeDistanceField(\mathbf{M}_{path})$ *// compute euclidean distance field*

$\mathbf{M}_{fmm\_inp} \leftarrow 0$ *// initially all is free space*
$\mathbf{M}_{fmm\_inp}(\mathbf{p}_i \mid ||\mathbf{p}_i - \mathbf{p}_{goal}||_2 < goal\_radius) \leftarrow 1$ *// assign goal points*
$\mathbf{M}_{fmm\_inp}(\mathbf{M}_{o\_edt} < obst\_thres) \leftarrow nan$ *// exclude obstacle points*
$\mathbf{M}_{fmm\_inp}(\mathbf{M}_{path\_edt} < path\_thres) \leftarrow nan$ *// exclude paths away from path*
$\mathbf{M}_{fmm\_vel} \leftarrow 1$ *// initialize constant velocity map*
$\mathbf{M}_{fmm\_vel}(\mathbf{M}_{o\_edt} < d\_vel\_sat) \leftarrow \frac{\mathbf{M}_{o\_edt}}{d\_vel\_sat}$ *// reduce velocity close to obstacles*
$\mathbf{M}_{fmm} \leftarrow computeFMM(\mathbf{M}_{fmm\_inp}, \mathbf{M}_{fmm\_vel})$ *// compute metric field*

return $[\mathbf{M}_{o\_edt}, \mathbf{M}_{path\_edt}, \mathbf{M}_{fmm}]$

5.2 Other Navigation Objectives

Section 5.1 presented cost-functions that can be used for reaching a goal, being proven to convexify the problem in the simplified case of the point-particle. However, in most practical applications, the navigation agent to be controlled possesses additional constraints as well as potentially other cost function terms. To this end, it is clear that the goal-reaching cost function is to be formulated as a terminal cost, i. e.

$$\varphi(\mathbf{x}, t) = d_{fmm}(\mathbf{x}). \tag{5.20}$$

We would like to discuss now briefly if in the case of non-trivial agent models, we can still preserve the convexity of the optimization problem. Even though it would be very difficult to be formally proven, an intuitive observation is that such a cost-function does not necessarily result in a convex optimization problem but *the agent never gets stuck* in local minimum given a specific property of the system: the system can perform a sequence of motions such that its reduced state behaves like an unconstrained particle (the concept basically relates to the idea of being able to turn on the spot). This is the case for a differential-drive with a circular foot-print.

The main intuitive argument is by contradiction: suppose that the agent would get stuck into a local minima. Due to the enforced safety constraints, it will thus stand still. However, at stand still, no additional constraints are active (which are typically due to kinematic natures, e.g. maximum angular velocity) and the agent can turn on the spot. As the planned trajectory length can be now arbitrarily small (it can start moving with very low velocity), it follows that any parametrization of its inputs becomes arbitrarily expressive, i. e. any very short-length motion of the agent can be achieved. This results in the optimization problem to have the character of the point-particle, which cannot get stuck.

Having this noted, we would like to discuss now other possible objectives during navigation and methods to model them.

Time Optimality Probably one of the most desired behaviours beside reaching a goal is to reach it fast. To this end, the literature formulates the time-optimal cost-function simply as

$$J(\mathbf{x}) = \frac{1}{2} t_1^2, \tag{5.21}$$

i. e. minimization of the square of the planned trajectory end-time. Note that this leads to desired results in the case in which one optimizes for a fixed trajectory length. For the case when one optimizes for a fixed trajectory duration, it is trivial that such a cost-function would be useless.

We would like to present here an elegant alternative that encodes the time-optimality character of the system implicitly. Intuitively, it relates to the sugar-on-a-stick phenomenon: a donkey has a stick glued on him such that the sugar cube connected to the stick end is visible by the donkey. Being hungry, the donkey will run towards the sugar cube as fast

as it can, while obviously never reaching it. This seems to be a highly efficient method to motivate the proverbially stubborn donkey.

Going back to navigation, the idea is to provide at every iteration of the controller a goal that cannot be reached given the optimization problem constraints, for example by constraining the trajectory length to be less than the local cost-maps radius. The time-optimal character is thus implicitly present, as the optimizer will find the trajectory that gets as close to the goal as possible, e.g. by maximizing velocities and minimizing the distance to the goal.

Accelerations In certain circumstances, one might want to penalize the accelerations of the system. This can be quite easily encoded into an integral cost term

$$J(\mathbf{x}) = \frac{1}{t_1 - t_0} \int_{t_0}^{t_1} k_1 \dot{v}^2(t) + k_2 \dot{\omega}^2(t) \mathrm{d}t, \tag{5.22}$$

with the scaling constrants k_1 and k_2. Note that this becomes a so-called power-cost due to the temporal normalization involved. The reason for this normalization term is the fact that when the duration of the trajectory is an optimization variable, we are interested to equally penalize motions of arbitrary duration to be penalized equally during high accelerations. Without this normalization, as the horizon duration increases, the cost-function term would become dominant.

Path Deviation Especially in industrial applications, navigation might be desired to be performed precisely on predefined route. In this case, one might use an integral cost term of the form

$$J(\mathbf{x}) = \int_{t_0}^{t_1} d_{path}^2(\mathbf{x}(t)) \mathrm{d}t, \tag{5.23}$$

where d_{path} can be computed using an EDT. Note that here, in general normalization is not advised, as this would lead to the optimizer potentially finding solutions with larger horizon and thus larger path deviation errors. It is worth noting here that for differentially flat systems, alternative generic formulations of the optimization problem together with its cost function exist, which possess the invariance property, i. e. after the agent converges to the path it will never deviate [66].

The above presented objectives are quite general and application independent in the context of navigation. However, when going closer to specific applications, numerous other objectives can be present, their modelling and efficient formulations varying greatly in difficulty and complexity.

6 Experimental Results

In this chapter, simulated as well as real-hardware results of discussed approaches of Optimal Local Path-Planning and Control are presented. To illustrate the capacities and flexibility of the discussed methods, results will be presented on three different autonomous platforms with different purposes: a Differential-Drive used for navigation in human-shared environments, an Autonomous Race-Car used for time-optimal driving through an (unknown) circuit as well as an IWS drive used in navigation with torso-orientation requirements.

6.1 General Considerations

Developed Libraries: MPC Module The implementation of the presented work has been done in C++, targeting high generality and efficiency of the developed code. To this end, besides the standard object-oriented programming paradigm, heavy use of template meta-programming as well as functional programming techniques have been used to provide the desired modularity and performance. Given the developed library, the library-user is provided with convenient definition and formulation of the platform model, optimization variables as well as simple interfaces that provide large flexibility in implementation specifics (ODE solver method, sensitivities computation method, numerical type of the variables etc.). Moreover, it allows modular definition and usage of multi-lattice evaluation (e.g. equal time, equal distance, parametric function knots etc.).

Having the model of the platform defined, the application designer can now focus on formulating the optimization problem. The library is designed such that the platform model and optimization problem specifics are as independent as possible. The motivation behind it is that typically, one might want to formulate and (or) evaluate a vast variety of optimization approaches given the same platform model. Thus, the library interfaces allow convenient definitions of cost-functions, equality constraints as well as hard and soft inequality constraints, together with the definition of the required dynamic models for evaluating them.

Problem formulations Throughout the thesis, specific formulations and practical considerations have been given, with an emphasis on the differential drive platform. Thus, Examples 2.1, 2.8, 2.12 2.13, present incrementally the approach and design for solving the ODE of the differential-drive model (with sensitivities). The ODE evaluation of other platforms has been designed in a similar fashion. Section 4.1 as well as Example

4.2 further present the considerations of the system trajectory evaluation, focusing on the minimal parametric representation and including efficient multi-lattice support. The constraints used throughout experimental results are presented in Examples 4.1 and 4.3 – 4.6. The main component of the cost-function is the end-state term discussed in-depth in Section 5.1, together with non-dominant terms (such as the implicit time-optimality), discussed throughout Chapter 5.

Optimization Solver Currently, the only supported optimization problem solver is the library *Optizelle*. The motivation behind the choice of the optimization library is its efficiency, State of the Art implemented algorithms as well as matrix-free design. For all the presented results, the optimization problem is solved using an Interior-Point Method combined with an Inexact Composite-Step SQP (that internally uses a Trust-Region Method and a Krylov sub-problem solver). The equality constraints system is preconditioned using a column pivoting QR decomposition of the equality constraints gradient, while the second order sensitivities are approximated using the *BFGS* scheme.

Developed Libraries: Environment-Processing Module As discussed in Chapter 5, the environment-processing algorithms are desired to be computed independent of the optimization problem. Because of this, we chose to split the computation of various functions over discretized grids from the general MPC library. Thus, we use a second processing thread for computing every cycle the Euclidean Distance Transforms, functions based on Fast Marching Methods as well as gradients of those. In the development of this module, we make use of the *GridMap* library [67].

System Framework Modules integration, inter-process communication as well as visualization and debugging tools are necessary capabilities when working in the field of Robotics and Navigation. One such open-source framework that is popular in this field is the *Robotics Operating System* (ROS) [68], providing a multitude of tools and academy-developed, maintained, open-source packages.

System Simulation The entire system is initially tested in simulation using the 3D-Physics Rigid-Body Simulator *Gazebo* [69].

Hardware and computation times The developed algorithms are tested running on two cores of an Intel i7 (2016) processor, clocked at 2.7 GHz for the simulated results and 3.2 GHz on the machine running on the real robot.

The run-time of solving the optimization problem for all presented results varies between $20 - 50$ ms, for problems parametrized by $20 - 40$ parameters, second order Runge-Kutta ODE solver on approximatively 20 equal time and 20 equal distance lattice points. Regarding the environment processing module, all the required fields are computed in approximatively 80 ms for a 640×640 pixels resolution.

Figure 6.1: Robotic Platform *Pioneer-3DX (left)* and its simulated model *(right)*.

6.2 Differential-Drive: Navigation in Human-Shared Environments

In this section, we are interested in developing an Optimal Local Path-Planner/Controller for a Differential-Drive platform, its main application being autonomous navigation in (unstructured) environments in which humans are present.

The agent used for simulated and real-world testing is a Pioneer P3DX, illustrated in Figure 6.1. In most of the presented experiments, we limit the agent wheel angular velocity to $10\,\mathrm{rad/s}$ (corresponding to a maximum linear velocity $\approx 1\,\mathrm{m/s}$), wheel angular acceleration to $3\,\mathrm{rad/s^2}$ and maximum lateral acceleration to $0.5\,\mathrm{m/s^2}$. Note that the maximum platform accelerations constraints are quite conservative, as the platform is physically capable of accelerating/decelerating approximately twice as fast. This not only results in smooth perceived motions but also illustrates the quality of the presented navigation approaches when the dynamic constraints of the platform are non-negligible. Before starting to analyse different approaches and results, we would like to note that in general, all desired navigation *behaviours* cannot be appropriately mapped into solving only one optimization problem. Moreover, we would like to possess redundant emergency behaviours in the cases in which the to-be-solved optimization problem becomes infeasible.

To address this aspect, in practice we use different optimization problem formulations for different behaviours. For example, we distinguish between the general case in which the agent has to navigate along a global route and the case close to the goal, where the agent is expected to reach it with a high accuracy.

Example 6.1 (State-Machine for the Differential-Drive Navigation Module). This example provides a more in-depth discussion regarding the implemented state-machine. Figure 6.2 presents graphically the states and their associated transition conditions.

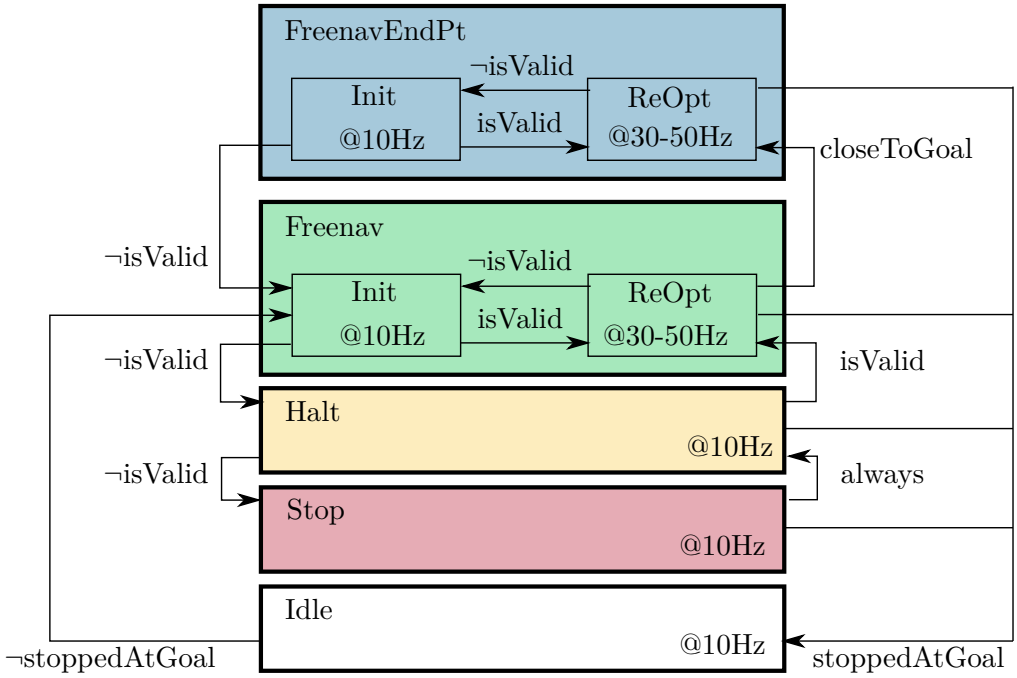


Figure 6.2: State-Machine for the Differential-Drive Navigation Module

We distinguish between 5 different states for the behaviour state-machine:

- FreeNav: The general case in which the agent is expected to navigate (loosely) along the global path while taking into account all the dynamic and environment-based constraints.
- FreeNavEndPoint: In comparison with the FreeNav behaviour, we formulate the optimization problem such that the end-state of the agent is imposed to coincide with a stop at the route end-point using equality constraints. This behaviour is desired to be enabled when the agent is sufficiently close to the route end-point, represented by the boolean predicate *closeToGoal*. In practice, *closeToGoal* evalutes to true when the end-point of the optimized trajectory is relatively close to the goal(for example $< 0.3\,\mathrm{m}$).

- Halt: In certain situations, the optimization problem of the FreeNav or FreeNavEndPoint behaviours might fail due to limited number of iterations or a poor initial solution. To this end, the predicate *isValid* evaluates the satisfaction of the constraints of the optimization problem that *has just been solved*. In this case where ¬isValid, we would like to find trajectories that simply stop the agent while not colliding with obstacles. Note that solving such an optimization should require in general less iterations, as the number of constraints are reduced as well as the cost-function has a simple structure.
- Stop: Redundant behaviour which is triggered when everything else fails. Note that in comparison with Halt, this does not guarantee non-collision with obstacles. Rather, it will make use of a simple kinematic sequence (no optimization involved) that stops the agent. Also note that Stop always transitions to Halt, resulting in recovery of the normal operation as soon as a solution that does not collide with obstacles is found.
- Idle: In cases in which it is clear that the goal cannot be reached, or after the agent reaches the goal (i. e. *stoppedAtGoal* evaluates to true), we would like the system to enter an Idle state.

Additionally, we would like to solve the optimization problems differently depending on which initial solution approach is used (as discussed in Section 4.3). For example, when starting with a null trajectory, we expect the solver to require more iterations in comparison with starting from an optimal trajectory computed in the previous cycle. Thus, expecting a reduced number of required iterations in a re-optimization situation, we can increase the frequency of the optimization module (the algorithms presented in Section 4.4 do not require a fixed cycle-time).

6.2.1 Navigation in Partially-Mapped Static Environments

In this experiment, we are interested in analysing the resulting trajectory of the agent in partially-mapped environments. Thus, the main goal is qualitative evaluation of the resulting trajectories when using the non-trivial cost function based on FMM. The following experiments are run using ground-truth measurements of the agent state, as provided by the 3D physics simulator.

Figure 6.3 illustrates a scenario in which local information allows a trajectory optimization (compared to the global path). In this example, the global path is computed assuming that the narrow path in the middle of the image is blocked. However, the MPC controller using the FMF_{damp} maps finds a trajectory that is shorter (with respect to its metric). Note that in this scenario, using a stronger reduction of the velocity map close to obstacles would result in the robot navigating on the left corridor.

Figure 6.4 illustrates the scenario complementary to the previous: the global path becomes invalid under local (online) information. In Figure 6.4a, the blocking obstacles are not yet in the agent sensing range, thus the planned trajectory closely follows the global

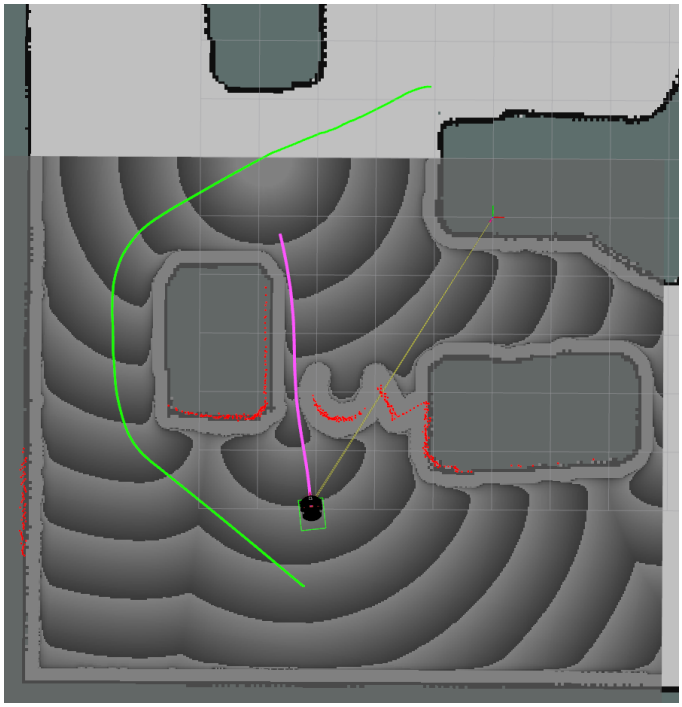


Figure 6.3: The optimized planned trajectory of the MPC controller *(magenta)* undertakes a shorter euclidean path to the goal in comparison to the global path *(green)*. *black*–mapped obstacles, *red*–sensed obstacles

path. However, in the cycle in which the blocking obstacles are sensed, the MPC finds an alternative trajectory towards the goal 6.4b.

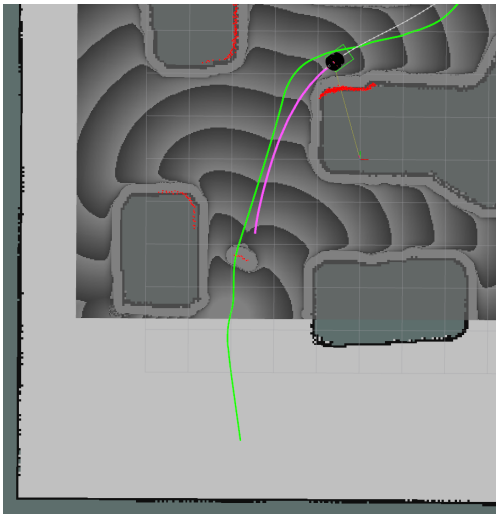
Note that by using the naive end-cost term of an euclidean norm to the goal, none of the two presented behaviours could be (systematically) achievable in practice.

6.2.2 Navigation using Safety Constraints

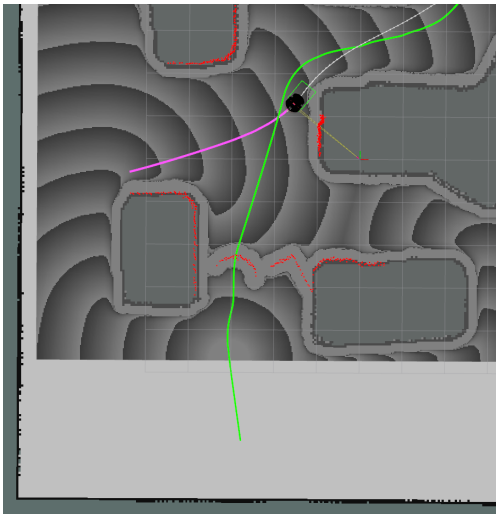
In those experiments, we are interested in validating the concepts of safe navigation constraints discussed in Section 3.3, whose practical formulation is given in Example 4.6. With this, two scenarios will be analysed: a cornering manoeuvre using simulation and navigation through narrow passages, making use of the real robot.

Cornering maneuver In this experiment, we compare the resulting trajectories when undertaking a cornering manoeuvre in confined space. We assume that the environment is static but (partially) unknown. We perform the experiment using three different formulations:

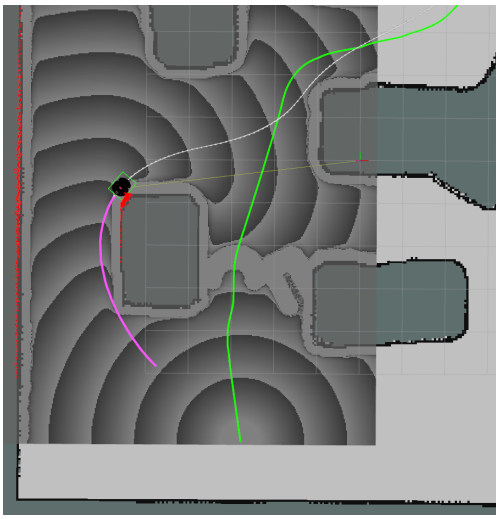
- F1: guaranteed-safe constraints



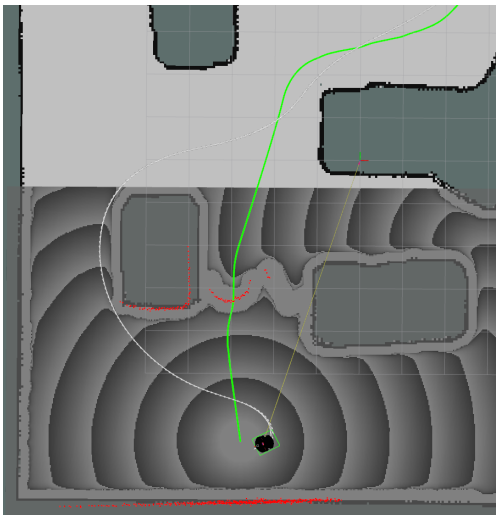
(a) MPC trajectory follows closely the global path



(b) Global path invalidated by unmapped obstacles; MPC trajectory finds alterative route



(c) Note the time-optimal trajectory as the lateral acceleration constraint is active



(d) Goal is reached

Figure 6.4: Temporal sequence illustrating the behaviour of the MPC controller when the global route becomes invalid due to non-mapped obstacles.

- F2: nominal-safe constraints
- F3: linear velocity end-state constraint

Recall that when the environment model is static *and known*, F3 guarantees safety. However, in this scenario, it is possible that an obstacle is present just around the corner, a situation in which F3 will result in a collision. Figure 6.5 illustrates the three different

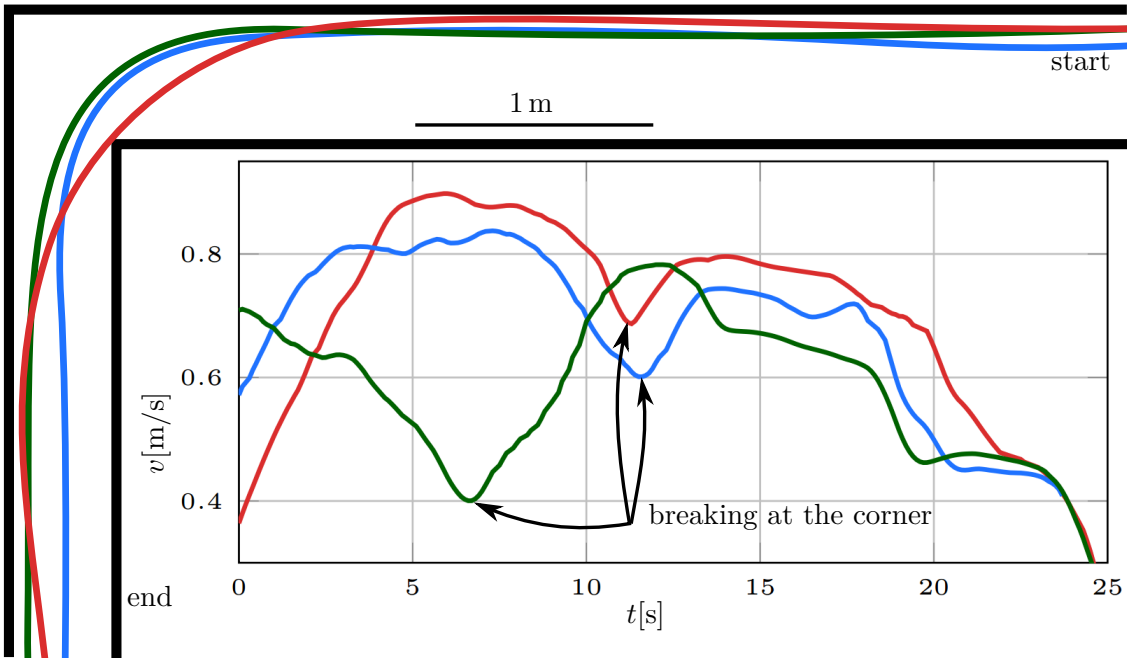


Figure 6.5: Evaluation of different safety constraints during a cornering manoeuvre in a static (partially) unknown environment. *Top-Left*: Executed trajectories using zero end velocity constraint (*red*), nominal safety constraints (*blue*) and guaranteed safety constraints (*green*). *Bottom*: the velocity profile of the respective trajectories. Note that the profiles have been temporally aligned at the end of the motion

trajectories as well as the agent linear velocity during the manoeuvre. As expected, F3 has the highest velocity profile and the trajectory resembles with a time-optimal manoeuvre: the radius of the turn is maximized as the lateral acceleration constraint is active. We can note that the F2 velocity profile closely resembles with F3. The trajectory, however, performs a motion that increases the visibility around the turn, a fact expected from the problem formulation. Considerably different is the velocity profile of F1. The reason is that according to the constraint formulation, the predicted trajectory always has to remain in the visible region. This induces a strong deceleration before the turn, hence the difference in the velocity profile.

Traversing narrow areas We evaluate the motions resulting from F1 and F2 under the random-walk model in narrow spaces. We set the maximum random walk velocity to 0.5 m/s and want to evaluate the trajectories and the velocities of the agent in corridors (doorways) of $1-2$ m. Of course, in such scenarios where the free-space width is small with respect to the random-walk maximum velocity, a better model of the environment is desired (e.g. constant velocity model). Nevertheless, this scenario allows us to validate expected behaviours. Figure 6.6 presents the resulting trajectories and velocity profiles.

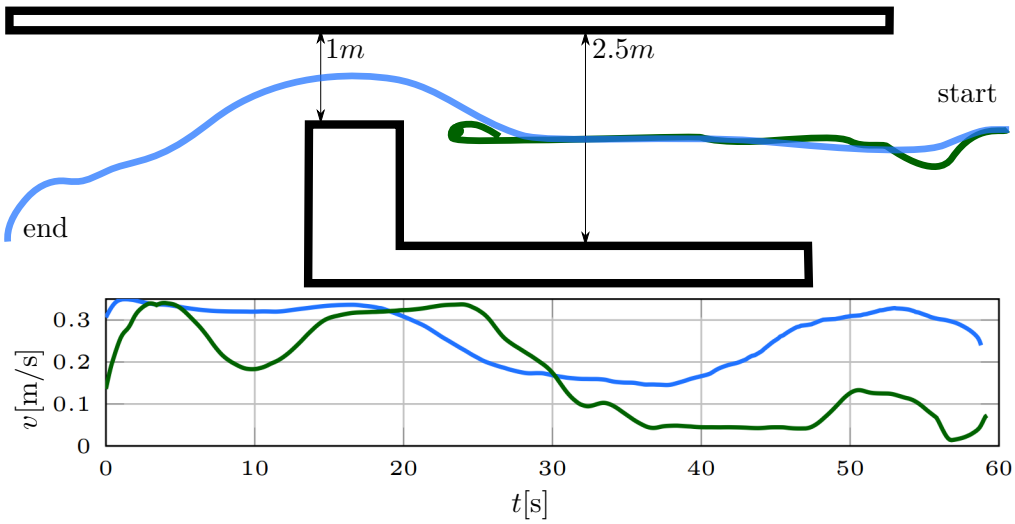


Figure 6.6: Navigation under uncertainty. *Top*: Executed trajectories using the nominal safety constraints (*blue*) and guaranteed safety constraints (*green*). *Bottom*: the velocity profile of the respective trajectories

In the case of F2, the trajectory is as expected: in order to maximize velocity, the agent should maximize its distance from obstacles (thus being allowed to decelerate towards a feasible invariant state for a longer period of time). Note that the trajectory resembles closely with following the Voronoi path [70], i. e. the path that maximizes the minimum distance to obstacles. However, this behaviour is implicit through the formulated constraints and environment model. Also, the velocity profile is expected: in the wider section, the agent has a higher velocity.

However, F1 fails to reach the goal. The reason is that due to the aggressive inflation of the environment over the forward simulation, the planned trajectory is very short: in this experiment, with an average of 0.1 m (in comparison with F2 which averaged at 1.7 m). The main argument lies within the fact that even though we make use of a non-trivial cost-function in our optimization problem (discretized solution of the Eikonal equation), the non-linear solver is not guaranteed to find a global minimum of the problem.

6.2.3 MPC vs Stabilized MHTP Comparison

In addition to the well known optimization-based control MPC, Section 3.2 introduced the concept of Stabilized MHTP (SMHTP). The details of the algorithmic additions for the concerned modules are given in Section 4.4. In the following experiments, we are interested in validating and evaluating the quality of the two approaches.

As presented previously, SMHTP requires a low-level controller that (exponentially) asymptotically stabilizes the agent along the optimization-module planned trajectory. The following examples will present two such controllers.

Example 6.2 (Lyapunov-based Trajectory-Following Control). A generic approach of designing (static) feed-back controllers for non-linear systems is based on Lyapunov-methods. For the differential drive model presented in Example 2.1, the control law

$$v_{fb}(t) = v_d(t)\cos(e_\theta) + k_1(\cos(\theta(t))e_x + \sin(\theta(t))e_y) \tag{6.1a}$$
$$\omega_{fb}(t) = \omega_d(t) + k_2 v_d(t)(\cos(\theta(t))e_y - \sin(\theta(t))e_x) + k_3 e_\theta, \tag{6.1b}$$

with

$$k_1 = k_3 = 2\xi\sqrt{b v_d^2(t) + \omega_d^2(t)} \tag{6.2a}$$
$$k_2 = b \tag{6.2b}$$

and the controller design parameters b, ξ , asymptotically stabilizes the error $\mathbf{e}$

$$\mathbf{e}^{\mathrm{T}} = \begin{bmatrix} e_x & e_y & e_\theta \end{bmatrix} = \begin{bmatrix} x_d(t) - x(t) & y_d(t) - y(t) & \theta_d(t) - \theta(t) \end{bmatrix} \tag{6.3}$$

along the trajectory $\begin{bmatrix} x_d(t) & y_d(t) & \theta_d(t) & v_d(t) & \omega_d(t) \end{bmatrix}^{\mathrm{T}}$. The stability proof of this control-law can be found in [28].

Example 6.3 (Input-Output Linearization Trajectory-Following Control). Another approach to design a simple feed-back law for non-linear systems is through Input-Output Linearization. Such an approach transforms the (initially non-linear) system into a linear system with respect to the error. The resulting system can be *exponentially* asymptotically stabilized by using linear control theory.

For the differential drive model presented in Example 2.1, we note that using the error

$$\mathbf{e}^{\mathrm{T}} = \begin{bmatrix} e_x & e_y \end{bmatrix} = \begin{bmatrix} x(t) - x_d(t) & y(t) - y_d(t) \end{bmatrix} \tag{6.4}$$

and the system inputs $\mathbf{u}^{\mathrm{T}} = \begin{bmatrix} v & \omega \end{bmatrix}$ results in a full relative degree $r = \dim(\mathbf{e}) = 2$, as $r = r_1 + r_2$ and $\mathbf{r}^{\mathrm{T}} = \begin{bmatrix} r_1 & r_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \end{bmatrix}$. However, the decoupling matrix

$$\mathbf{D}(\mathbf{x}) = \begin{bmatrix} \mathrm{L}_{g_1}\mathrm{L}_{\mathbf{f}}^{r_1-1} e_x(\mathbf{x}) & \mathrm{L}_{g_2}\mathrm{L}_{\mathbf{f}}^{r_1-1} e_x(\mathbf{x}) \\ \mathrm{L}_{g_1}\mathrm{L}_{\mathbf{f}}^{r_2-1} e_y(\mathbf{x}) & \mathrm{L}_{g_2}\mathrm{L}_{\mathbf{f}}^{r_2-1} e_y(\mathbf{x}) \end{bmatrix} \tag{6.5}$$

is singular. On the other side, using the fictitious input $\mathbf{u}^{\mathrm{T}} = \begin{bmatrix} \dot{v} & \omega \end{bmatrix}$, we still obtain $\mathbf{r}^{\mathrm{T}} = \begin{bmatrix} 1 & 1 \end{bmatrix}$ and additionally, the decoupling matrix

$$\mathbf{D}^{-1}(\mathbf{x}) = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ \frac{-\sin(\theta)}{v} & \frac{\cos(\theta)}{v} \end{bmatrix} \tag{6.6}$$

is non-singular for every $v \neq 0$. Thus, the control-law

$$\mathbf{u}_{fb} = \mathbf{D}^{-1}(\mathbf{x})(\mathbf{v} - \mathbf{b}(\mathbf{x})), \quad \mathbf{b}(\mathbf{x}) = \begin{bmatrix} \mathrm{L}_{\mathbf{f}}^{r_1} e_x(\mathbf{x}) \\ \mathrm{L}_{\mathbf{f}}^{r_2} e_y(\mathbf{x}) \end{bmatrix} = \mathbf{0} \tag{6.7}$$

allows the design of a linear state controller with respect to the new input $\mathbf{v}$. For example, one could set

$$\mathbf{v} = -\mathbf{K}_p\mathbf{e} - \mathbf{K}_d\dot{\mathbf{e}} = -\mathbf{K}_p \begin{bmatrix} x(t) - x_d(t) \\ y(t) - y_d(t) \end{bmatrix} - \mathbf{K}_d \begin{bmatrix} v(t)\cos(\theta(t)) - \dot{x}_d(t) \\ v(t)\sin(\theta(t)) - \dot{y}_d(t) \end{bmatrix}. \tag{6.8}$$

Keep in mind that such a control-law is designed for the system input $\mathbf{u}^{\mathrm{T}} = \begin{bmatrix} \dot{v} & \omega \end{bmatrix}$. As in reality we are interested in the input $\mathbf{u}^{\mathrm{T}} = \begin{bmatrix} v & \omega \end{bmatrix}$, we can create a dynamic controller by integrating $\dot{v}$ numerically. Note that this is valid in the context of SMHTP, as Theorem 3.1 allows the controller to evaluate previous temporal points.

Having introduced simple feed-back laws that asymptotically or exponentially stabilize the agent along a trajectory, we can now evaluate as well the SMHTP controller.

Figure 6.7 illustrates the scenario in which the following experiments will be performed. Note that the global path does not take into account unmapped obstacles present in the scene.

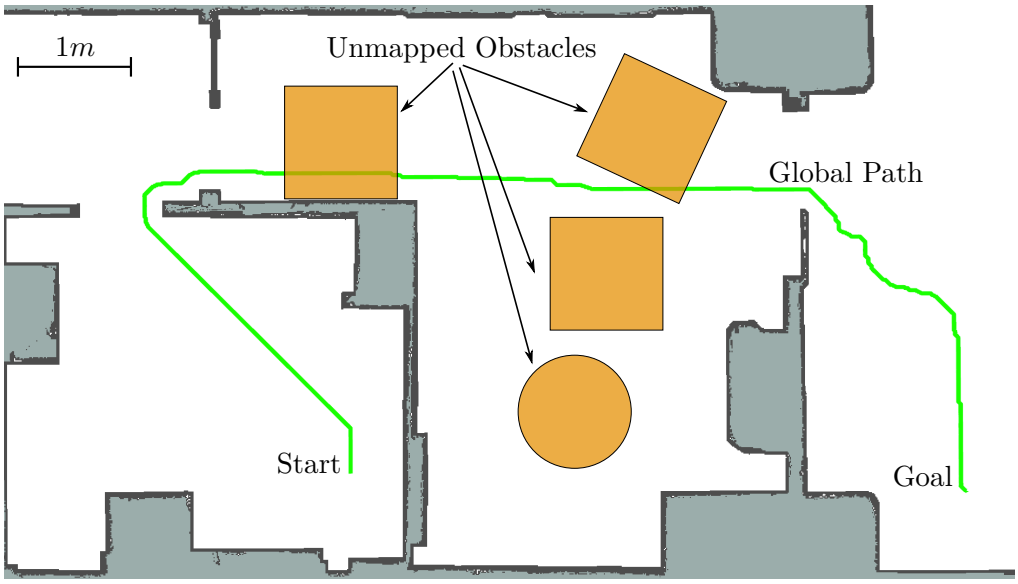


Figure 6.7: Testing scenario and computed global path

Figure 6.8 illustrates the undertaken trajectories of the agent through the static (partially unknown) environment using MPC as well as SMHTP with an asymptotic controller. Note that in the case of SMHTP, the trajectory deviates towards the opening between the two unmapped boxes. This deviation is due to the fact that in this run, the visibility of the environment was different (due to different trajectories and noise in the sensor data) and thus the environment representation considered induced planning a feasible trajectory through the two boxes for a longer period of time. However, as expected, as soon as the region is observed to be too narrow, the agent adapts its motion accordingly.

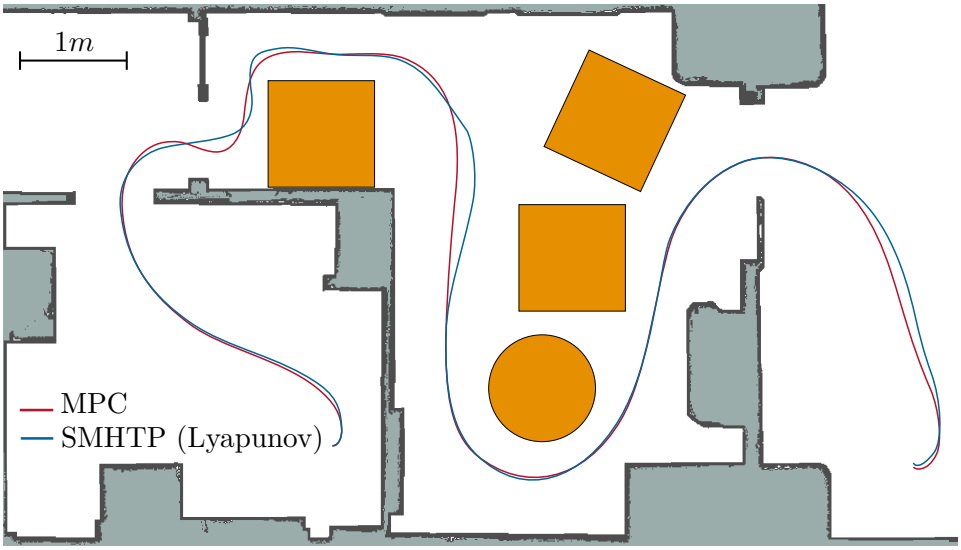


Figure 6.8: Trajectories of the agent using MPC and SMHTP with a Lyapunov Controller

The minimum distance to obstacles d_{obst} as well as the linear and angular velocities of the platform are illustrated in Figure 6.9 and Figure 6.10 for MPC as well as SMHTP. In the SMHTP case, the nominal (as computed by the optimization module) as well as applied velocities are presented. As no additional controller is involved, in the MPC case they coincide.

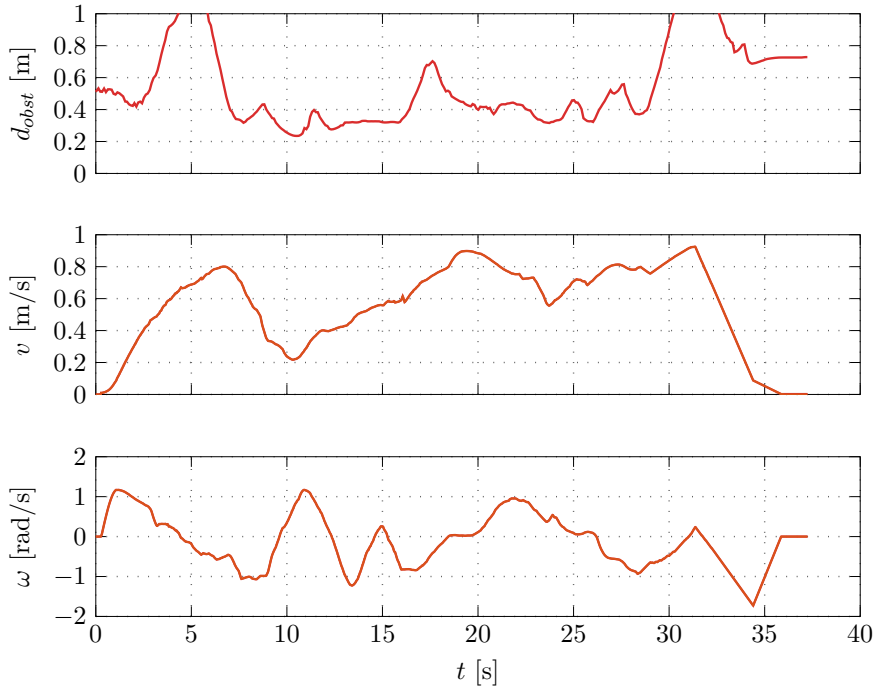


Figure 6.9: Distance to obstacles, linear and angular velocities along the MPC trajectory

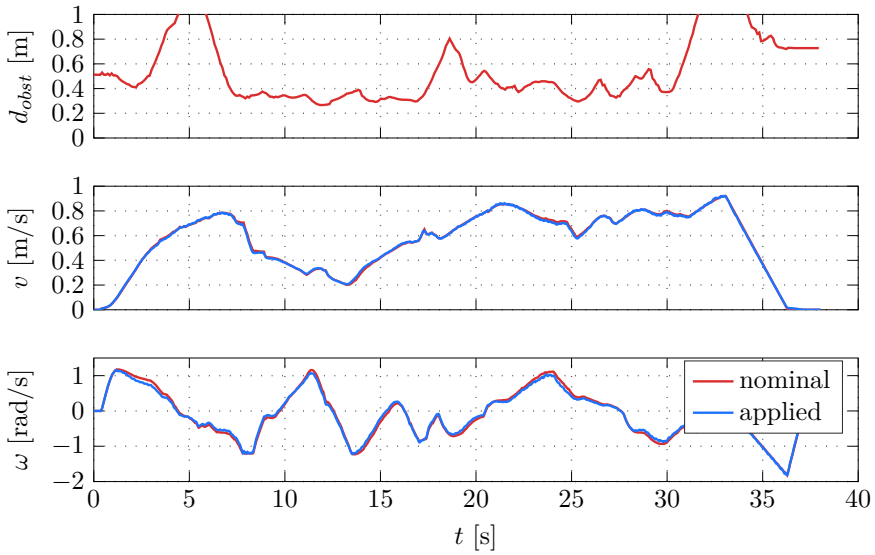


Figure 6.10: Distance to obstacles, linear and angular velocities along the SMHTP trajectory

At this point, one might ask why SMHTP would be beneficial. Resulting from the previous experiment, it requires relaxing the constraints of the optimization problem and does not induce optimal input sequences when perturbations are present. One argument could be that the low-level controllers, due to their reduced computational requirements can run at higher frequencies than the optimization module. Another argument that motivates the usage of SMHTP is parameter variations.

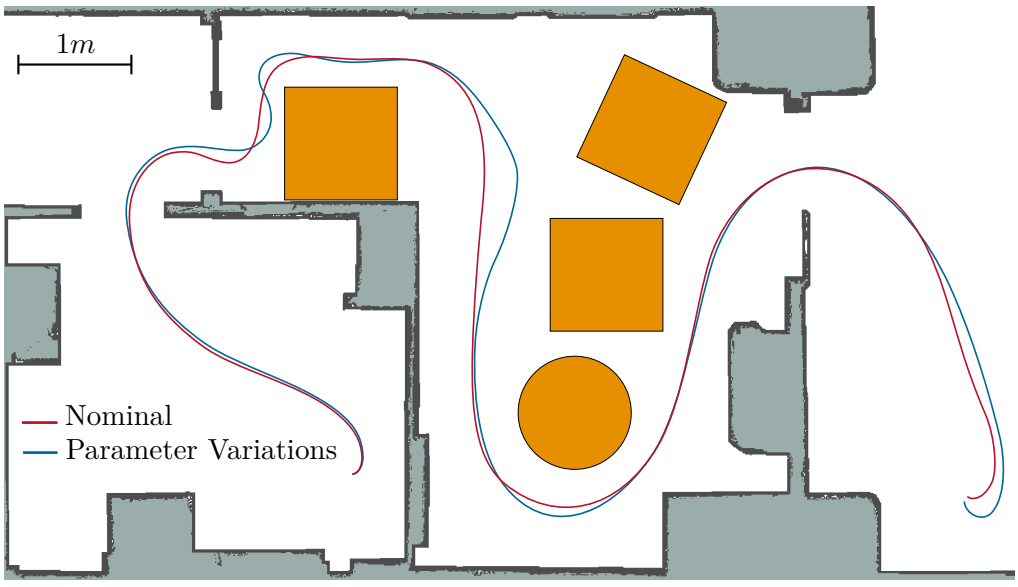


Figure 6.11: Trajectory of the MPC under modelling errors

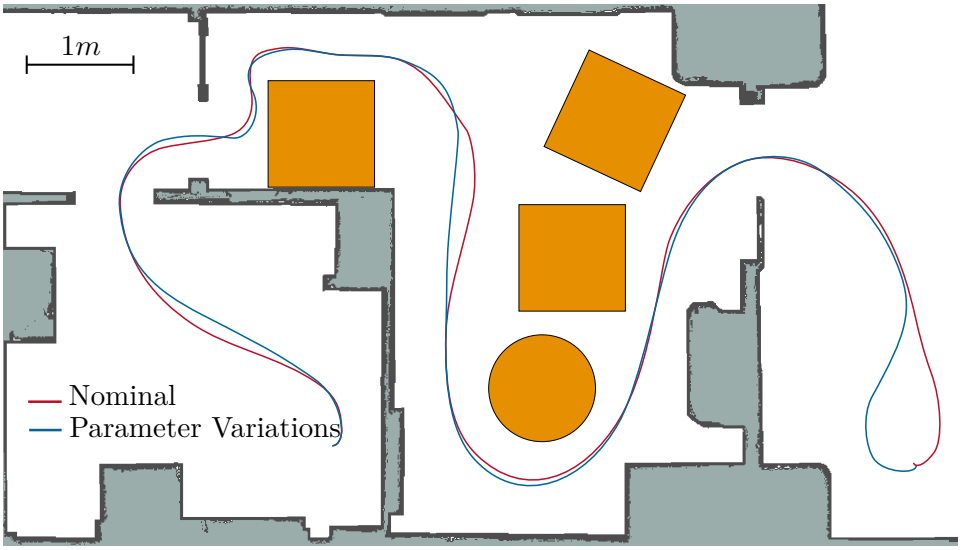


Figure 6.12: Trajectory of the SMHTP under modelling errors

Figure 6.11 and Figure 6.12 present the resulting trajectories when altering the model parameters of the controllers from nominal values (wheel radius $r = 0.97\,\mathrm{m}$ and wheel displacement $d = 0.33\,\mathrm{m}$) by approx. 10% ($r' = 0.8\,\mathrm{m}$, $d' = 0.3\,\mathrm{m}$). As expected, the MPC trajectory degrades, visible especially in the narrow areas at the beginning of the course. Nevertheless, even in the case of such variations, the control scheme continue to provide acceptable results. For the case of MHTP, the distortion of the trajectory in the same region is not as large. However, the deviations of the applied platform velocities compared with the nominal velocities are non-negligible, as seen in Figure 6.13.

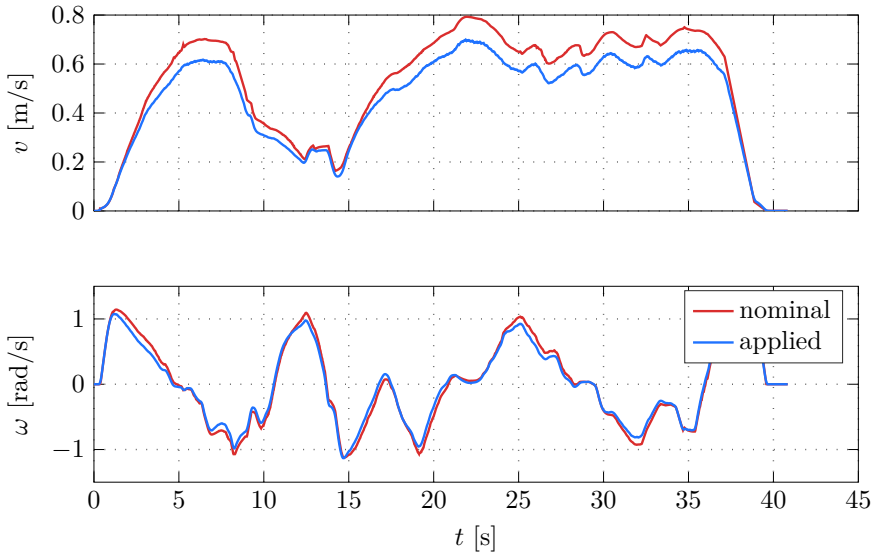


Figure 6.13: Linear and angular velocities along the SMHTP trajectory under modelling errors

6.2.4 Using State Observers

So far, all experiments have been conducted by providing the algorithms the ground-truth state of the agent. In practice, this is not the case. As discussed in the introduction, an entire topic of research regarding navigation is on the context of localization, i. e. the discipline of observers for the agent pose. Currently, due to their robustness, the most popular methods for localization are based on Particle-Filtering. However, from a system dynamics perspective, they typically tend to be relatively inaccurate.

Another point to consider regarding using only the localization observer for the control-algorithms is the fact that *in a non-linear system, the independence property does not hold:* one cannot design a stable controller and a stable observer independently and expect that using them together will result in a stable system. Of course, in practice, this is not taken into account and results are satisfactory. Nevertheless, we are interested in potentially allowing observers to be proper (i. e. designed in accordance with the controllers) while still making use of localization approaches as a black-box. This motivates the observer chaining structure presented in Figure 6.14.

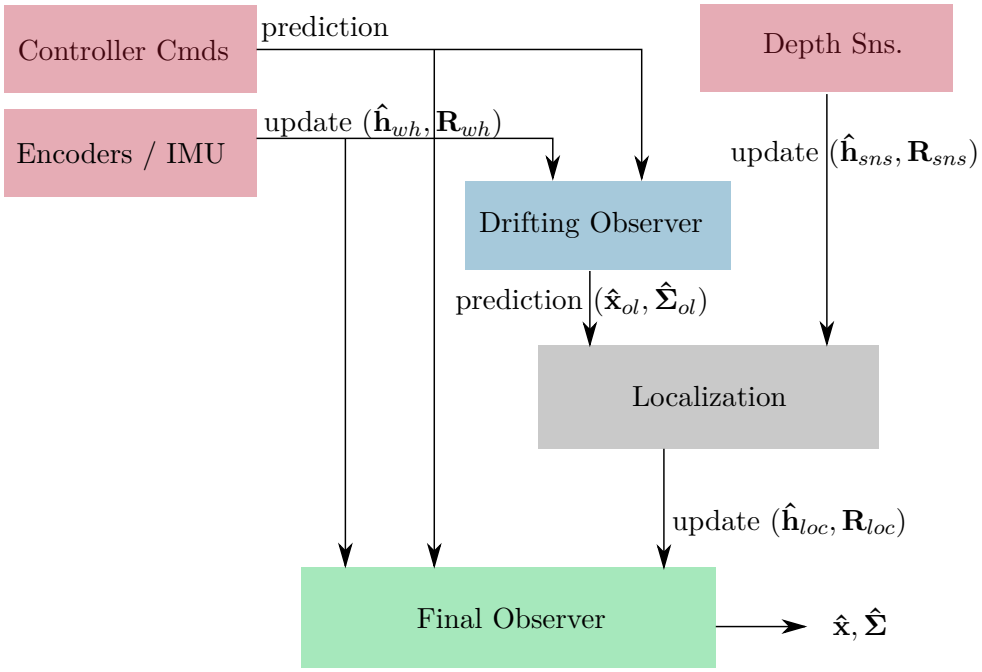


Figure 6.14: Diagram illustrating the observers structure

As we want to consider the localization module as a black box, we require an open-loop observer that provides the so called *odometry* to the localization module, i. e. a predicted state $\hat{\mathbf{x}}_{ol}, \hat{\mathbf{\Sigma}}_{ol}$ that is as qualitative as possible while still not providing updates in the pose-space of the agent. For correctness, note that a very slow filter can be applied such that the open-loop observer does not reach numerical instability as the run-time of the

system $\to \infty$. As localization is typically based on particle filtering, we are interested in post filtering its output in a final observer, that for increased quality takes into account all the inputs and measurements of the system. Note that here, parameter estimation of the platform-model can be performed. Moreover, note that the final observer can be designed together with the controller. Unfortunately, there exist no methods in the literature to construct observers that are stable in combination with an MPC. This is another argument that motivates the usage of SMHTP.

Example 6.4 (Kalman Filter for State and Parameter Estimation). In this example, we would like to present the design of an Extended-Kalman-Filter (EKF) observer for estimating the agent pose, velocities as well as wheels radius and displacement. As mentioned earlier, such an observer is not proven to be stable when used in combination with a controller. However, in practice it results in a generic observer that provides satisfactory results.

For generality, we are interested in the case in which we design an EKF that possesses multiple update types $\hat{\mathbf{h}}_u$, $u \in \mathcal{U}$. As such updates might come from sub-systems that are running at different frequencies, another requirement of the designed filter is to be able to process updates arriving at asynchronous time intervals. The pseudo-code of such an algorithm when processing a measurement is given in Algorithm 6.

Algorithm 6 $EKF\_async\_update(\hat{\mathbf{h}}_u, \mathbf{R}_u, \hat{t}_u)$
internal state: $\hat{\mathbf{x}}, \hat{\mathbf{\Sigma}}, t, t_u, \quad \forall u \in \mathcal{U}$
requirements: $\hat{t}_u \geq t$

// prediction step:
$\Delta t \leftarrow \hat{t}_u - t$
$t \leftarrow \hat{t}_u$
$\bar{\mathbf{x}} \leftarrow advance\_ode(\hat{\mathbf{x}}, \Delta t)$
$\bar{\mathbf{\Sigma}} \leftarrow \mathbf{\Phi}(\hat{\mathbf{x}}, \Delta t)\hat{\mathbf{\Sigma}}\mathbf{\Phi}(\hat{\mathbf{x}}, \Delta t)^{\mathrm{T}} + \mathbf{Q}^d(\hat{\mathbf{x}}, \Delta t)$

// correction step:
$\Delta t_u \leftarrow \hat{t}_u - t_u$
$t_u \leftarrow \hat{t}_u$
$\mathbf{S} \leftarrow \mathbf{C}_u(\bar{\mathbf{x}})\bar{\mathbf{\Sigma}}\mathbf{C}_u(\bar{\mathbf{x}})^{\mathrm{T}} + \mathbf{R}_u^d(\mathbf{R}_u, \Delta t_u)$
$\mathbf{K} \leftarrow \bar{\mathbf{\Sigma}}\mathbf{C}(\bar{\mathbf{x}})^{\mathrm{T}}\mathbf{S}^{-1}$
$\hat{\mathbf{x}} \leftarrow \bar{\mathbf{x}} + \mathbf{K}(\hat{\mathbf{h}}_u - \mathbf{h}_u(\bar{\mathbf{x}}))$
$\hat{\mathbf{\Sigma}} \leftarrow (\mathbf{E} - \mathbf{K}\mathbf{C}_u(\bar{\mathbf{x}}))\bar{\mathbf{\Sigma}}$

The prediction-step is independent of the type of update, with the discrete ODE solver step $advance\_ode(\mathbf{x}, \Delta t)$ that advances the system state $\mathbf{x}$ by the time interval Δt, the discrete state-transition matrix of the linearised system $\mathbf{\Phi}$ as well as the discrete prediction noise $\mathbf{Q}^d$. The update step requires functions that are dependant on the type of update, with the observation function $\mathbf{h}_u(\bar{\mathbf{x}})$, the observation function Jacobian

$\mathbf{C}_u(\bar{\mathbf{x}})$ and the discrete observation noise $\mathbf{R}_u^d(\mathbf{R}_u, \Delta t_u)$.

Thus, we want to design the functions *advance_ode*, $\mathbf{\Phi}$ and $\mathbf{Q}^d$ for the filter state model and the functions $\mathbf{h}_u$, $\mathbf{C}_u$ and $\mathbf{R}_u^d$ for every implemented update type $u \in \mathcal{U}$. In the following, we are interested in computing the above-mentioned functions for designing an observer for the Differential-Drive that accepts updates from wheel angular velocity measurements as well as agent pose (from the localization module). Moreover, we are interested in also estimating the platform linear and angular velocity as well as the kinematic model parameters: the wheel radius r_w ($\rho = 1/r_w$) and the wheels displacement d. Thus, our filter state and inputs are

$$\mathbf{x}^{\mathrm{T}} = \begin{bmatrix} x & y & \theta & v & \omega & \rho & d \end{bmatrix}, \quad \mathbf{u}^{\mathrm{T}} = \begin{bmatrix} \dot{v} & \dot{\omega} \end{bmatrix} \tag{6.9}$$

with the system dynamics

$$\dot{\mathbf{x}}^{\mathrm{T}} = \begin{bmatrix} v\cos(\theta) & v\sin(\theta) & \omega & \dot{v} & \dot{\omega} & 0 & 0 \end{bmatrix}. \tag{6.10}$$

Computing the (continuous-time) dynamic matrix $\mathbf{A}$ and input matrix $\mathbf{B}$, we obtain

$$\mathbf{A} = \frac{\partial \dot{\mathbf{x}}}{\partial \mathbf{x}} = \begin{bmatrix} 0 & 0 & -v\sin(\theta) & \cos(\theta) & 0 & 0 & 0 \\ 0 & 0 & v\cos(\theta) & \sin(\theta) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{B} = \frac{\partial \dot{\mathbf{x}}}{\partial \mathbf{u}} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}. \tag{6.11}$$

Regarding the (continuous-time) state transition noise, we assume the presence of terms in all the system states except the constant parameters, i. e.

$$\mathbf{Q} = \mathrm{diag}\left(\begin{bmatrix} nn_{xy} & nn_{xy} & nn_{\theta} & nn_{v} & nn_{\omega} & 0 & 0 \end{bmatrix}\right). \tag{6.12}$$

Note that there is no state transition noise present for the estimated model parameters, as they are of constant nature. For the wheels update, we have the observation function

$$\hat{\mathbf{h}}_{wh} = \begin{bmatrix} \rho\left(v + \frac{\omega d}{2}\right) \\ \rho\left(v - \frac{\omega d}{2}\right) \end{bmatrix} \tag{6.13}$$

and its Jacobian

$$\mathbf{C}_{wh} = \frac{\partial \hat{\mathbf{h}}_{wh}}{\partial \mathbf{x}} = \begin{bmatrix} 0 & 0 & 0 & \rho & \frac{\rho d}{2} & v + \frac{\omega d}{2} & \frac{\omega \rho}{2} \\ 0 & 0 & 0 & \rho & -\frac{\rho d}{2} & v - \frac{\omega d}{2} & -\frac{\omega \rho}{2} \end{bmatrix}. \tag{6.14}$$

Analogously, we have for the localization measurement

$$\hat{\mathbf{h}}_{loc} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{6.15a}$$

$$\mathbf{C}_{loc} = \frac{\partial \hat{\mathbf{h}}_{loc}}{\partial \mathbf{x}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{6.15b}$$

Now, we are interested to discretize the above-defined functions. The *advance_ode* can be discretized using various ODE discretization methods as discussed in Section 2.3. For the rest, formally we want to compute

$$\mathbf{\Phi} = \mathrm{e}^{\mathbf{A}\Delta t} \tag{6.16a}$$

$$\mathbf{\Gamma} = \int_0^{\Delta t} \mathrm{e}^{\mathbf{A}t}\mathbf{B}\ \mathrm{d}t \tag{6.16b}$$

$$\mathbf{Q}^d = \int_0^{\Delta t} \mathrm{e}^{\mathbf{A}t}\mathbf{Q}(\mathrm{e}^{\mathbf{A}t})^{\mathrm{T}}\ \mathrm{d}t \tag{6.16c}$$

$$\mathbf{R}_u^d = \frac{\mathbf{R}_u}{\Delta t_u} \tag{6.16d}$$

with the matrix exponential $\mathrm{e}^{\mathbf{A}\Delta t}$. In the following we will make use of the second order (Tustin) approximation of the matrix exponential [71]

$$\mathrm{e}^{\mathbf{A}\Delta t} \approx \left(\mathbf{E} + \frac{\mathbf{A}\Delta t}{2}\right)\left(\mathbf{E} - \frac{\mathbf{A}\Delta t}{2}\right)^{-1} \tag{6.17}$$

With this, we obtain

$$\mathbf{\Phi} = \begin{bmatrix} 1 & 0 & -\Delta t\, v \sin(\theta) & \Delta t\, \cos(\theta) & -1/2\, v \sin(\theta)\, \Delta t^2 & 0 & 0 \\ 0 & 1 & \Delta t\, v \cos(\theta) & \Delta t\, \sin(\theta) & 1/2\, v \cos(\theta)\, \Delta t^2 & 0 & 0 \\ 0 & 0 & 1 & 0 & \Delta t & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.18a}$$

$$\mathbf{\Gamma} = \begin{bmatrix} 1/2\, \cos(\theta)\, \Delta t^2 & -1/6\, v \sin(\theta)\, \Delta t^3 \\ 1/2\, \sin(\theta)\, \Delta t^2 & 1/6\, v \cos(\theta)\, \Delta t^3 \\ 0 & 1/2\, \Delta t^2 \\ \Delta t & 0 \\ 0 & \Delta t \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{6.18b}$$

$$\mathbf{R}_u^d = \frac{\mathbf{R}_u}{\Delta t_u} \tag{6.18c}$$

and the non-zero upper-triangular values of the symbolic discrete covariance matrix are given in (6.19). Note that the matrix $\mathbf{Q}^d$ contains entries from the e.g. v–related noise also in the terms relating to the pose states.

$$
\begin{aligned}
\mathbf{Q}^d(0,0) &= \frac{\Delta t \left(3\, v^2 (\sin(\theta))^2 nn_\omega\, \Delta t^4 + 20\, (\sin(\theta))^2 \Delta t^2 v^2 nn_\theta + 20\, (\cos(\theta))^2 \Delta t^2 nn_v + 60\, nn_{xy}\right)}{60} \\
\mathbf{Q}^d(0,1) &= -\frac{\Delta t^3 \sin(\theta) \cos(\theta) \left(3\, \Delta t^2 v^2 nn_\omega + 20\, v^2 nn_{\theta_{ch}} - 20\, nn_v\right)}{60} \\
\mathbf{Q}^d(0,2) &= -1/8\, v \sin(\theta)\, \Delta t^2 \left(nn_\omega\, \Delta t^2 + 4\, nn_\theta\right) \\
\mathbf{Q}^d(0,3) &= 1/2\, \cos(\theta)\, nn_v\, \Delta t^2 \\
\mathbf{Q}^d(0,4) &= -1/6\, v \sin(\theta)\, nn_\omega\, \Delta t^3 \\
\mathbf{Q}^d(1,1) &= \frac{\Delta t \left(3\, v^2 (\cos(\theta))^2 nn_\omega\, \Delta t^4 + 20\, (\cos(\theta))^2 \Delta t^2 v^2 nn_\theta + 20\, (\sin(\theta))^2 \Delta t^2 nn_v + 60\, nn_{xy}\right)}{60} \\
\mathbf{Q}^d(1,2) &= 1/8\, v \cos(\theta)\, \Delta t^2 \left(nn_\omega\, \Delta t^2 + 4\, nn_\theta\right) \\
\mathbf{Q}^d(1,3) &= 1/2\, \sin(\theta)\, nn_v\, \Delta t^2 \\
\mathbf{Q}^d(1,4) &= 1/6\, v \cos(\theta)\, nn_\omega\, \Delta t^3 \\
\mathbf{Q}^d(2,2) &= 1/3\, nn_\omega\, \Delta t^3 + nn_\theta\, \Delta t \\
\mathbf{Q}^d(2,4) &= 1/2\, nn_\omega\, \Delta t^2 \\
\mathbf{Q}^d(3,3) &= nn_v\, \Delta t
\end{aligned}
\tag{6.19}
$$

Given the presented observers structure, Figure 6.15 illustrates the ground-truth vs. observed pose of the agent during navigation. Note that in certain regions, a drift from the ground-truth pose emerges. The main reason for this deviation is the fact that the map of the environment is relatively inaccurate. Nevertheless, as the navigation-approach relies on agent-relative (local) information, sufficiently small deviations from ground-truth pose do not influence the quality of the emerging trajectories.

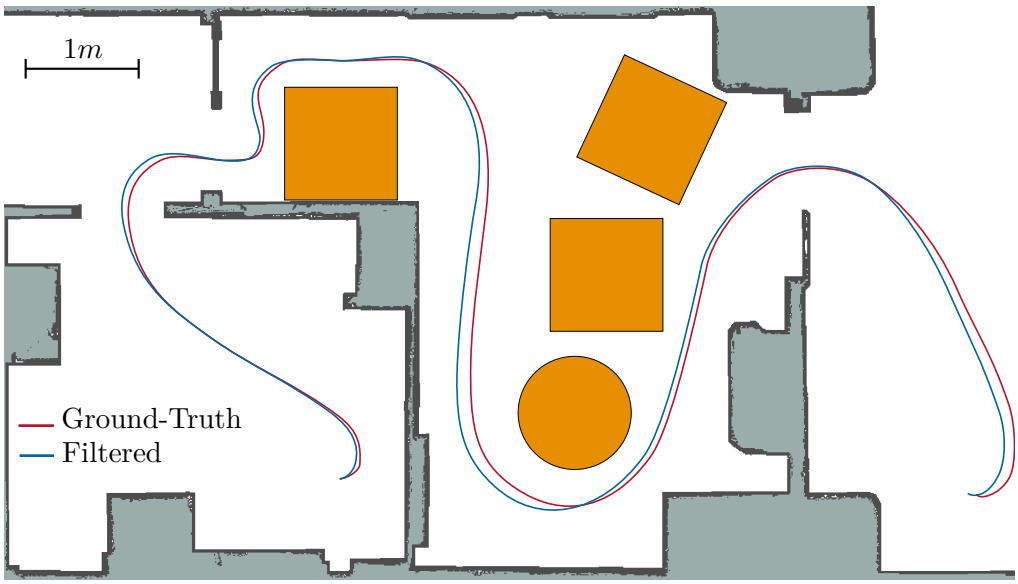


Figure 6.15: Estimated Pose of the MPC vs Ground-Truth Pose

Figure 6.16 presents the resulting (ground-truth) trajectories of the agent using the MPC

scheme, SMHTP with a Lyapunov controller and SMHTP with Input-Output Linearization controller with modelling errors of approx. 10%, similar with the setting from Subsection 6.2.3. It can be noted that all controllers behave qualitatively even in this case where the system state is observed and considerable modelling errors are present.

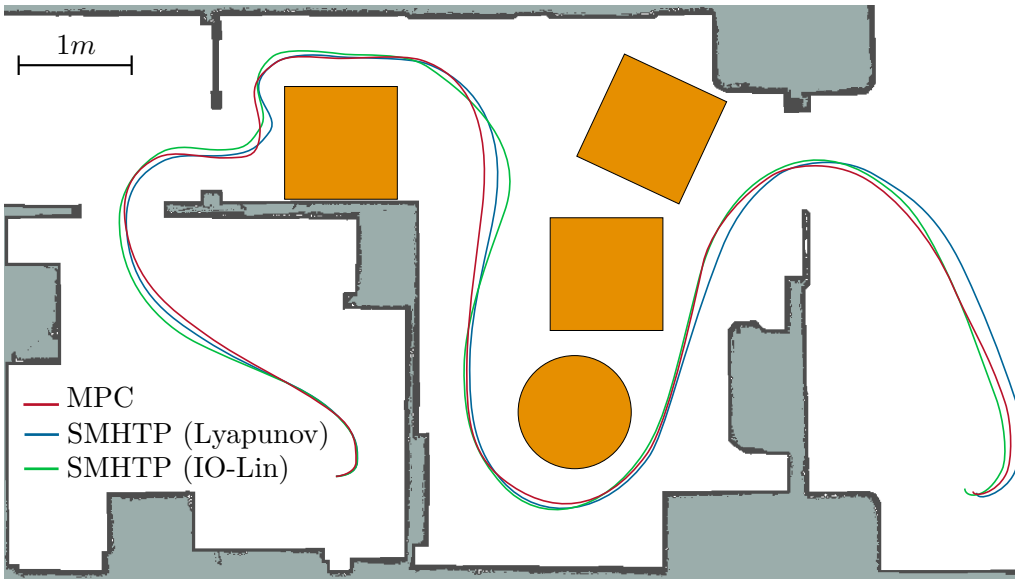


Figure 6.16: Ground-truth trajectories of MPC and SMHTP using state estimation under modelling errors

The nominal and applied agent velocities as well as the model parameter estimation evolution for the IO-Lin SMHTP are presented in Figure 6.17. Note that the model parameters quickly converge to their true values.

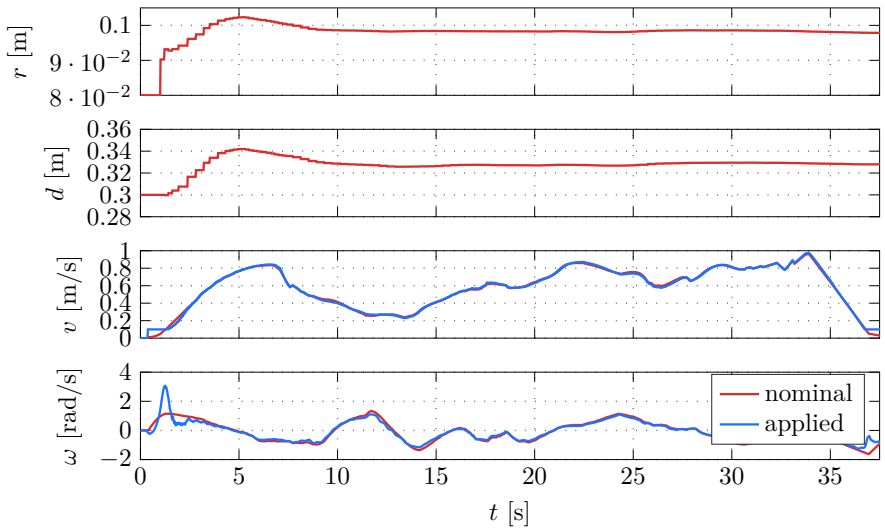


Figure 6.17: Adaptation of the observed model parameters as well as linear and angular velocities along the SMHTP trajectory

6.2.5 Real-Robot Testing

In the following, we would like to do a qualitative analysis of the presented navigation algorithms on the real-world robotic platform, using the other required modules such as Localization, Observers etc.

Navigation in Partially-Mapped Static Environments In this experiment, we let the robot navigate through a static office environment. As the radius of the agent is approximatively 0.15 m, a minimum distance to obstacles of 0.2 m has been enforced. Moreover, an additional velocity dependant distance term has been added: a constant factor (0.2) multiplied by the agent velocity. Thus, if the agent navigates at 1 m/s, it has to maintain a distance to obstacles of 0.4 m.

In the tested environment, non-convex unmapped obstacles have been added in certain regions. Moreover, the robot has to pass a relatively narrow doorway. The computed global-path and the executed agent trajectory are illustrated in Figure 6.18.

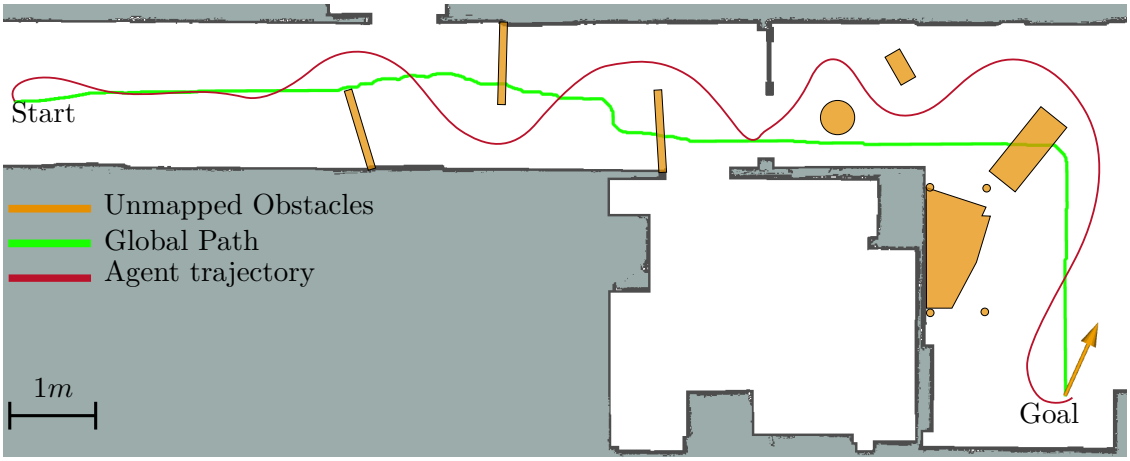


Figure 6.18: Resulting trajectory of the real platform in partially-mapped static environments

As visible in Figure 6.19 at $t = 20\,\text{s}$, the agent slows-down almost to a stop just before navigating through the (mapped) doorway. The reason of this unnecessary and sub-optimal slow-down is the fact that the Localization-module possessed relatively high inaccuracies in the orientation of the agent just before entering the doorway. Because of this, the navigation through the (narrow) doorway was rendered as infeasible, transitioning the optimization module into Halt mode. However, as the agent dynamics reduced, the Localization-module converged to the true pose of the agent and the system entered in normal operation.

Navigation near Dynamic Obstacles Working in the same environment setting, we now remove some of the static un-mapped environments and allow humans to move along the robot trajectory. Figure 6.20 illustrates the undertaken agent trajectory and key temporal

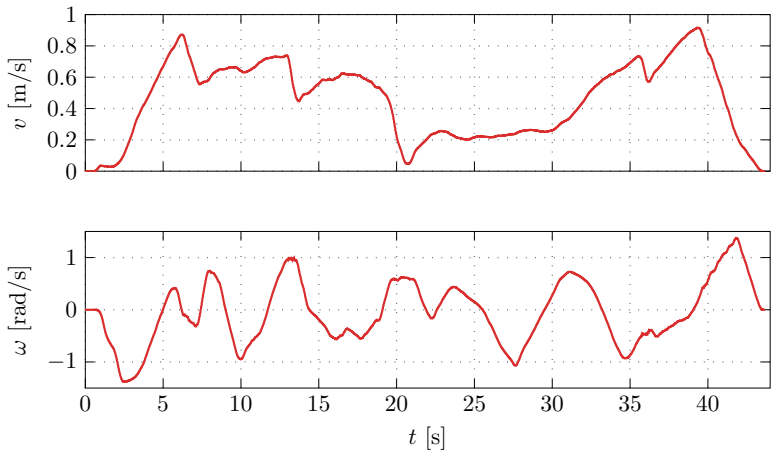


Figure 6.19: Linear and angular velocity commands along the real-robot trajectory in static environment

points where a human performs certain motions. In the first encounter (*yellow*), the human waits in the door-way. As expected, as soon as the doorway is in the local view of the agent, the agent smoothly navigates to a stop, as the goal cannot be reached. As soon as the human moves away, the agent continues its task. The second encounter is one in which the human initially stands still (*light blue*). As soon as the robot starts to navigate around the human, the human moves towards (*dark blue*) the trajectory of the agent. With this, we want to create an "annoying human" scenario, which requires the agent to heavily modify its trajectory. Even though only a random-walk model of the environment is assumed, we observe that the agent quickly adapts to the human motion.

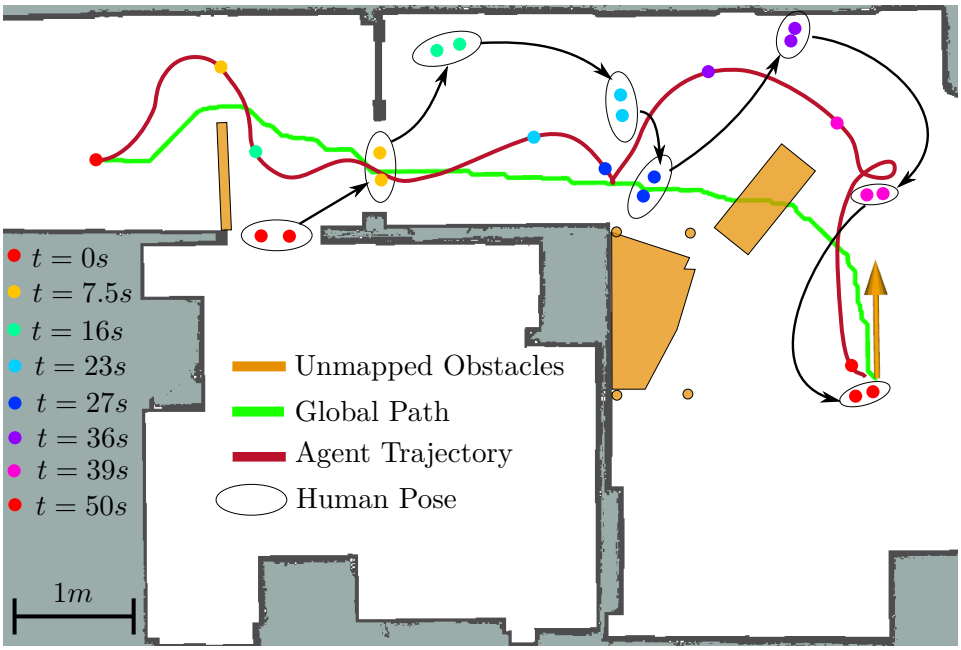


Figure 6.20: Resulting trajectory of the real platform navigating near humans

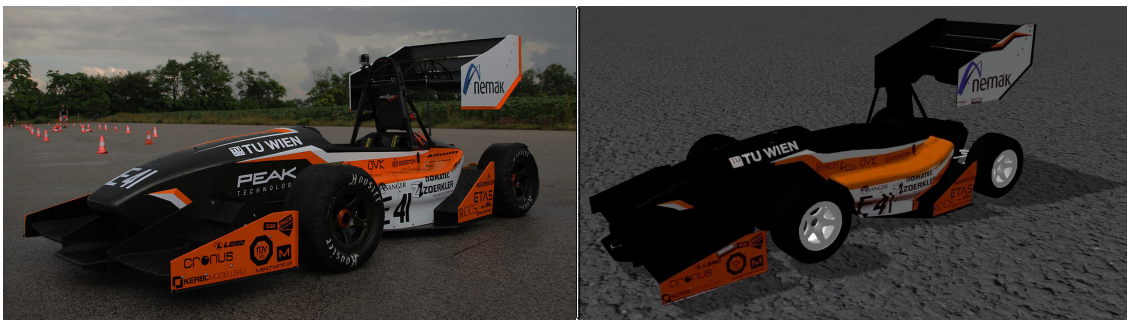


Figure 6.21: Formula-Student Race-Car *Edge8 (left)* and its simulated model *(right)*.

6.3 Ackerman-Drive: Navigation of the TU Autonomous Race-Car

In this section, we would like to evaluate a small set of experiments on a different platform: an electric autonomous race-car. The race-car has been designed and manufactured by the TU Racing Team. Moreover, the team has created a high-fidelity model of the platform for simulation-use in the Gazebo simulator, accounting for non-linear tire friction models, air drag, battery voltage drops etc. Details regarding the race-car and the developed simulation model can be found in [72]. Figure 6.21 illustrates the real and the simulated platform.

The main reason of considering experiments on such a platform is to evaluate the scalability of the proposed approaches. Note that in the following tests, velocities as high as 15 m/s (54 km/h) are achieved, which results in the platform advancing $0.3 - 1.5$ m between each optimization cycle, depending on the update rate. Thus, one of the main validated algorithms is the temporal synchronization and extrapolation of the navigation module, discussed in Section 4.4.

The simulated model is controlled by applying individual torques to the rear wheels. However, for the optimization module, a kinematic model similar with the one designed for the Differential-Drive has been used. Thus, a low-level controller that controls the longitudinal wheel-slip has been developed, asymptotically following the velocity-profile requested by the optimization module. Figure 6.22 illustrates the trajectory of the agent when navigating through an (unknown) circuit. Note that due to the non-circular shape of the car, its bounding box is approximated by two circular regions. Figure 6.23 presents the velocity and steering-angle profile of the platform during the course of one lap.

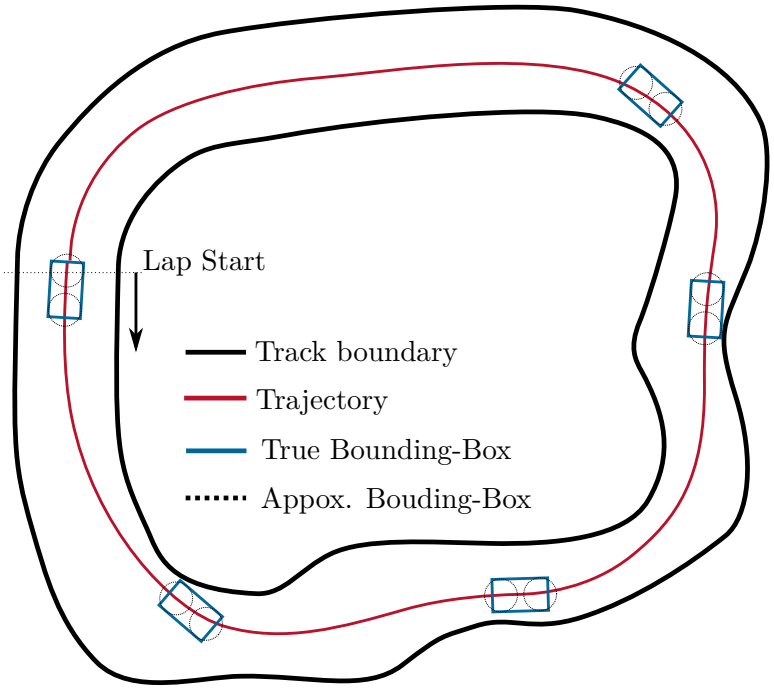


Figure 6.22: Resulting trajectory in a closed-circuit

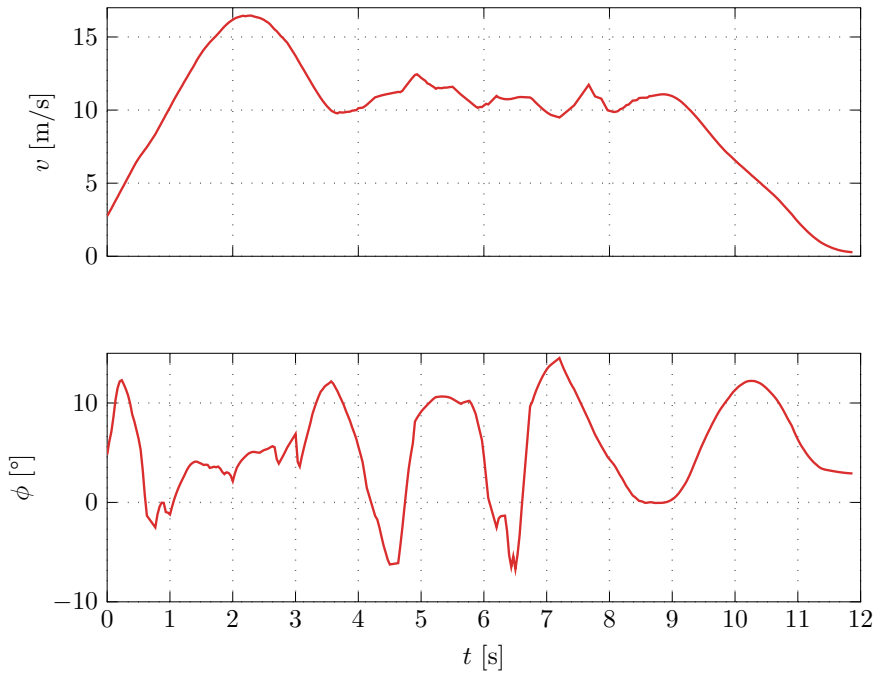


Figure 6.23: Resulting linear velocity and steering angle in a closed-circuit

6.4 IWS-Drive: Navigation with Controlled Torso-Orientation

In this section, we are interested in evaluating the emerging trajectories of the optimization module under various dominating cost-function terms, when applied to an IWS platform, introduced in Example 2.3.

Figure 6.24 illustrates the robotic platform *Blue* and the simulation model of its mobile base, making use of CAx models of the components as specified by the designer, together with component inertial information. General purpose surface friction parameters have been applied accordingly. Without loss of generality, the actuators are modelled taking into account damping and the interface has been designed to apply shaft torques as input. As 3D physics engines implemented in Gazebo (such as ODE, Bullet or Dart) perform typically poor in over-constrained closed kinematic chains (such as a I4WS) when force control is used, increased simulation accuracy is achieved through parameter-tuning of the engine (increased iteration count as well as relaxation of the stiffness of the formed contact joints). Additionally, quantisation noise is modelled for the angular sensors of the actuators as well as Gaussian noise of the depth sensor.

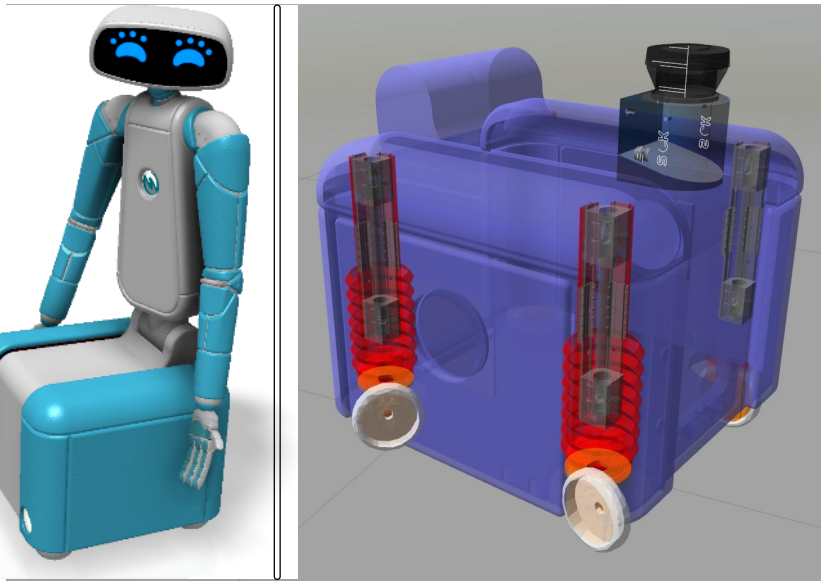
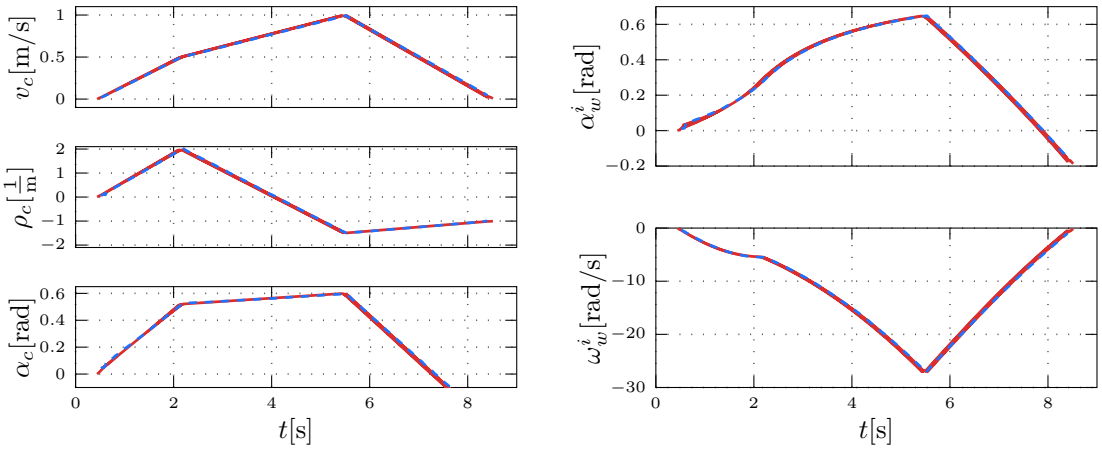


Figure 6.24: Robotic Platform *Blue (left)* and its simulated model *(right)*.

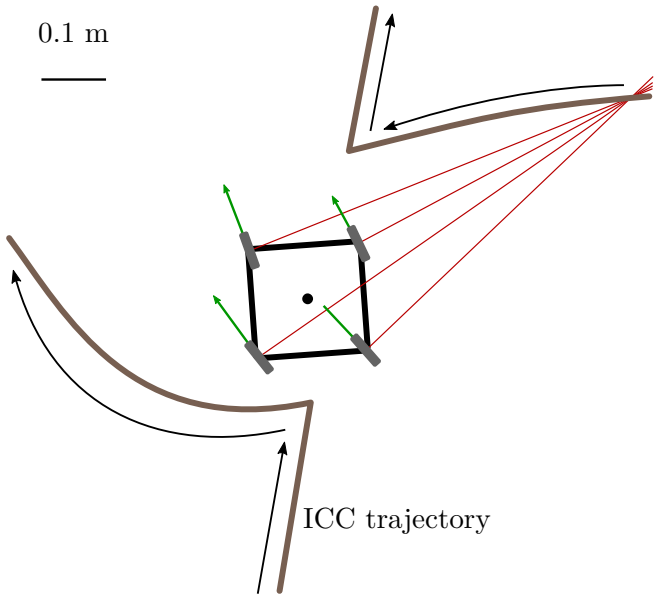
As discussed in Example 2.3, various parametrizations of the platform can be applied. In the following, the wheels actuators are controlled using de-coupled linear (PID-based) controllers, assuming independence in their models. These controllers are used to control the individual motors such that the desired chassis state is achieved.

Initially, we are interested in visualizing the evolution of the ICC parameters given a certain chassis state trajectory. For simplicity of data interpretation, the parametrization $\mathbf{p}^{\mathrm{T}} = \begin{bmatrix} v & \rho & \alpha \end{bmatrix}$ is used. Figure 6.25b presents the local view of the platform as it performs the sequence of control commands from Figure 6.25a. For visualization purposes, the

entire parameter trajectory has been computed initially; however, each cycle computation is performed on-line during normal operation. Desired versus applied parameter and wheel velocity trajectories are illustrated in Figure 6.25a, together with the temporal evolution of the orientation as well as rotation speed of the top-right wheel. As expected, when the ICC revolutes at a small radius relative to a wheel (big values of ρ), the non-linear character of the wheel motion becomes significant, as depicted in the right side of Figure 6.25a.



(a) Desired (red) vs. measured (blue) data: ICC parameters (*left*), orientation and velocity of a wheel (*right*)



(b) ICC trajectory in local frame of the I4WS when executing a parametrized trajectory from Figure 6.25a

Figure 6.25: Visualization of an ICC trajectory

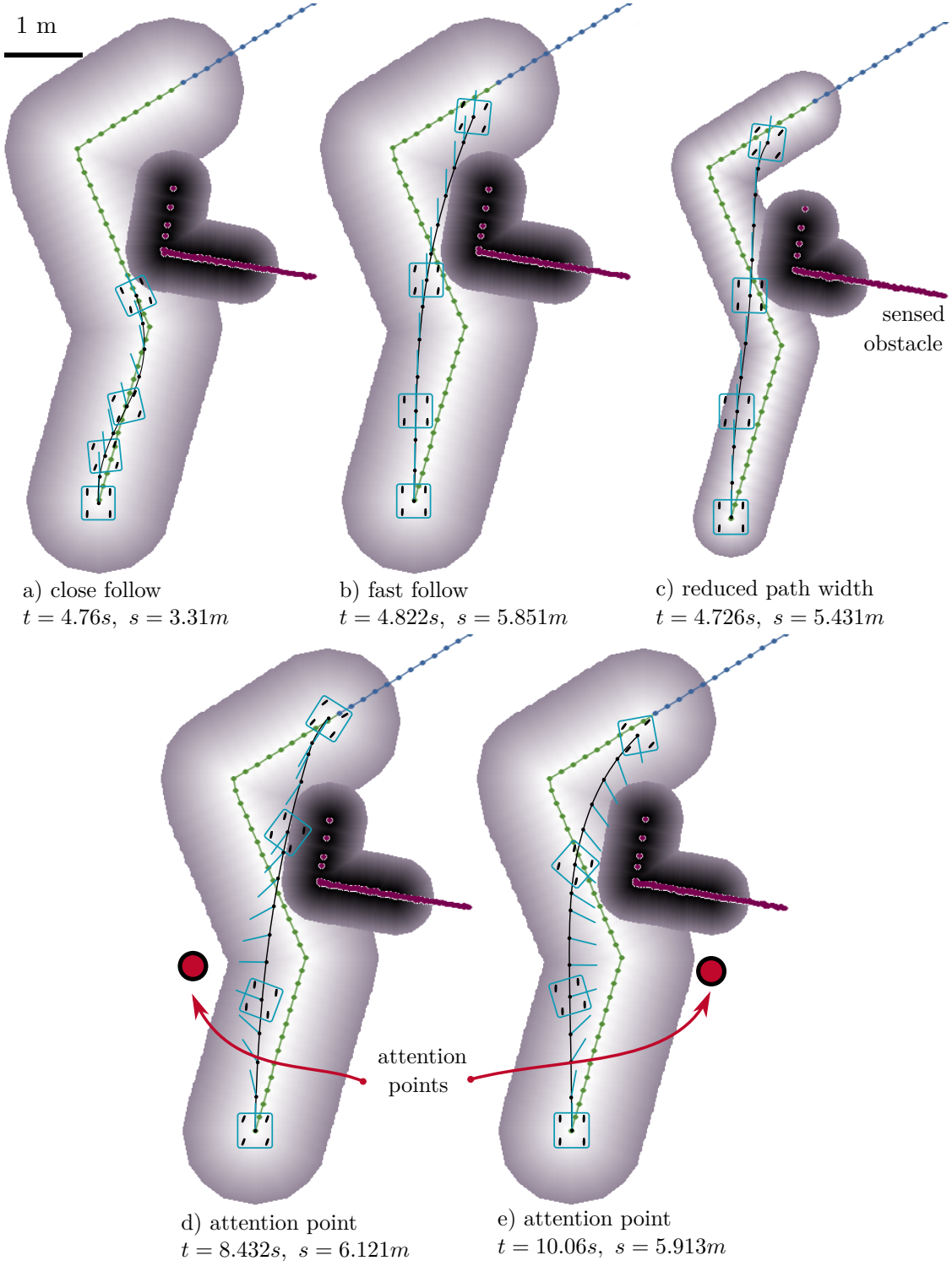


Figure 6.26: Generated trajectories under different dominating weights

Figure 6.26 presents the behaviour that can be obtained by the Local Path-Planner using only three control points. The active route for this instance is represented by the green dotted line and its afferent distance-field is created. The route far away is not considered locally and is in this instant inactive (blue). In the presented scenarios, the parametrization $\mathbf{p}^{\mathrm{T}} = \begin{bmatrix} v & \omega_{traj} & \omega_b \end{bmatrix}$ has been used as it performs superior to other parametrizations when attention-focus points are targeted. The agent has an initial velocity of 0.1 m/s and for all trajectories, the end-constraint of 0 base velocity is imposed. Furthermore, wheel actuator orientation, velocity and acceleration as well as platform base linear and lateral acceleration constraints have been enforced. Figure 6.26a presents the scenario where route accuracy is desired. As the number of control points is small, high path-following accuracy implies a reduction of the trajectory length. Figure 6.26b presents the generated trajectory with dominant costs towards time-optimality. Notice that the obstacle represents the only active spatial constraint. 6.26c illustrates the situation where maximum path deviation is reduced. Notice here the sensed obstacle distance field truncation further away from the path, as the path constraint makes this region automatically infeasible. Last but not least, Figures 6.26d and 6.26e showcase the same scenario when an attention-focus point (to the left and right respectively) is targeted. The increased duration of the generated trajectory in Figure 6.26e is mostly due to the internal actuator constraints of such motion.

7 Conclusions

This thesis focused on Optimal Local Path-Planning and Control. Chapter 2 presented generic concepts related to dynamic models, parametrizations as well as solving ODEs and their sensitivities. Moreover, it provided models for various agent platforms (Differential-Drive, Ackerman-Drive, Single-Track model, IWS-Drive) as well as practical approaches of the discussed topics building upon these models.

Chapter 3 formulated the dynamic optimization problem and methods of its usage in feed-back loops. Here, besides the well-known MPC scheme, we presented an alternative feed-back approach, the Stabilized MHTP, which in certain circumstances possesses several benefits over MPC. Moreover, we addressed the problem of guaranteeing safety in the context of Autonomous Navigation using MHTP. We started from general models of the agent and the environment (allowing the environment to be dynamic and assuming inexact knowledge regarding its model). From this, we defined the required assumptions that have to be made and provided two formulations (in form of constraints) that if once satisfied, will continue to be satisfied indefinitely, thus providing safety for the navigation task. As previously discussed, even though more expensive computationally, the proposed nominal safety constraint allows planning trajectories for arbitrary horizons even in the case where the uncertainty of the environment prediction is non-negligible.

In Chapter 4, practical implementation specifics of MHTP Local-Path Planning within the context of autonomous navigation have been addressed. Initially, two discretization methods of the dynamic optimization problem are proposed. Building upon the Minimal Parametric Discretization method, methods for constraints discretization on evaluation lattices of different natures (spatial, temporal, dynamic) have been presented. Attention has been given to the temporal synchronization of the iterative optimization module, obtaining finally algorithms that can function asynchronously. Last but not least, the Chapter discussed various details regarding optimization problem modelling and solvers specifics.

Motivated by unsatisfactory performance of straight-forward approaches, Chapter 5 addressed the convexity of the optimization problem with the objective of reaching to a goal subject to navigable space constraints. Two metrics defined by functions solving partial differential equations (Laplace and Eikonal, respectively) have been proposed in order to convexify the problem with respect to non-linear optimization. For a simplified setting (point-particle), we have proven that such mappings ensure that the gradient of the metric never vanishes and is always pointing towards the inside of the allowed drivable space. These properties make the optimization sub-problem have only one optimum, i. e. the global optimum. The required operations and computed fields have been summarized

in an environment-processing algorithm. Lastly, other typically desired cost-function terms and their induced behaviours have been discussed.

Chapter 6 presented simulated and real-platform results, validating the proposed methods using three different autonomous platforms. The qualitative and quantitative analysis of the resulting trajectories and behaviours satisfies the desired capabilities of motion: its optimal character and robustness. Moreover, it was observed that it behaves well even if the used environment models are relatively simplistic: the platform behaved satisfactory when close to human motion even when assuming a low-noise random-walk environment model. Temporal synchronization and extrapolation of the proposed algorithms have been validated by applying them on a real-sized autonomous Race-Car.

7.1 Future Work

Improvements and future research (and implementations) could be performed in various contexts presented throughout the Thesis.

Regarding parametrizations and ODE solvers, one possible improvement is to make use of higher order B-Splines to parametrize the ODEs. However, due to their recursive definition with respect to the Spline knots, computing their sensitivities when the knots locations are variable is expected to be relatively computationally-intensive. A class of ODE solvers whose accuracy versus computational requirement could be analysed are Symplectic Solvers [73]. Regarding sensitivities computation, integration of Automatic Differentiation Tools (AutoDiff) [74, 75] in the developed libraries is desired. However, due to the complexity of the developed code, fully using AutoDiff might considerably increase the run-time of the algorithm.

In Chapter 3, the two presented dynamic optimization problem discretization approaches lie at opposite ends of the spectrum: one approach solves the entire ODEs within the optimization problem while the other approach fully solves the ODEs external to the optimization problem. An advanced approach would be to combine the two discretization methods, by splitting the simulated trajectory into several intervals: each interval ODE being externally solved and the intervals being connected by equality constraints enforced in the optimization problem. This would still provide accurate ODE solutions on such intervals while allowing the entire trajectory ODE to be solved in parallel. Regarding initial solution initialization, the main two approaches used in the current implementation are the previous trajectory and the null trajectory. Even though the null trajectory has been extensively used to validate the robustness of the optimizer solution w.r.t. local minima, improved results are expected when using sampling methods to initialize the optimization problem. As a more long-term goal, evaluation of methods solving stochastic optimization problems is desired.

The nominal safety constraint possesses one practical drawback: the existence of trajectories towards a feasible invariant state has to be investigated for all the states along the trajectory. Future work is expected in the context of efficient evaluation of such trajec-

tories, for example by parallelizing their evaluation. Moreover, the presented technique of precomputing trajectory candidates towards feasible invariant states is not generally applicable, in particular in the case in which regions in the environment are explicitly excluded from the feasible invariant set. An example could be an expert explicitly not allowing agents to reside in certain regions of the environment. Another problematic situation would be in the case where the trajectories towards feasible invariant states do not tend to have a short duration. Moreover, the proposed approach could be improved by using more complex dynamic environment models.

The environment processing module leaves room for future work as well. Firstly, non-trivial dynamic obstacle perception and models are desired for improved navigation in the vicinity of humans. Moreover, in the context of Multi-Agent Navigation, distributed extensions could allow the agents to predict their motion while taking into account the predicted motion of neighbouring agents.

Last but not least, additional testing for all the real platforms is expected. Moreover, an implementation of a Local-Path Planning module based on the (dynamic) single-track model is desired for pushing the limits of the Autonomous navigation of the TU Race-Car.

Bibliography

[1] J. Alonso-Mora, A. Breitenmoser, M. Rufli, P. Beardsley, and R. Siegwart, "Optimal reciprocal collision avoidance for multiple non-holonomic robots," in *Distributed Autonomous Robotic Systems: The 10th International Symposium*, A. Martinoli, F. Mondada, N. Correll, G. Mermoud, M. Egerstedt, M. A. Hsieh, L. E. Parker, and K. Støy, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 203–216, ISBN: 978-3-642-32723-0. [Online]. Available: https://doi.org/10.1007/978-3-642-32723-0_15.

[2] M. Čáp, P. Novák, A. Kleiner, and M. Seleckỳ, "Prioritized planning algorithms for trajectory coordination of multiple mobile robots," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 3, pp. 835–849, 2015.

[3] P. Moral, *Mean Field Simulation for Monte Carlo Integration*, ser. Chapman and Hall/CRC Monographs on Statistics and Applied Probability Series. CRC Press LLC, 2016, ISBN: 9781138198739. [Online]. Available: https://books.google.at/books?id=bHJKvgAACAAJ.

[4] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2d lidar slam," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 1271–1278.

[5] M. Montemerlo and S. Thrun, "Fastslam 2.0," *FastSLAM: A scalable method for the simultaneous localization and mapping problem in robotics*, pp. 63–90, 2007.

[6] M. Phillips and M. Likhachev, "Sipp: Safe interval path planning for dynamic environments," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, IEEE, 2011, pp. 5628–5635.

[7] T. Howard, "Adaptive model-predictive motion planning for navigation in complex environments," PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Aug. 2009.

[8] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics Automation Magazine*, vol. 4, no. 1, pp. 23–33, Mar. 1997, ISSN: 1070-9932.

[9] W. Xu, J. Wei, J. M. Dolan, H. Zhao, and H. Zha, "A real-time motion planner with trajectory optimization for autonomous vehicles," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, May 2012, pp. 2061–2067.

[10] B. Chachuat, *Nonlinear and Dynamic Optimization: From Theory to Practice - IC-32: Spring Term 2009*, ser. Polycopiés de l'EPFL. EPFL, 2009. [Online]. Available: https://books.google.at/books?id=%5C_JOHYgEACAAJ.

[11] J. Rauch, *Hyperbolic Partial Differential Equations and Geometric Optics*, ser. Graduate studies in mathematics. American Mathematical Soc., 2012, ISBN: 9780821885093. [Online]. Available: https://books.google.at/books?id=rDbAC1%5C_SOewC.

[12] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *Robotics Automation Magazine, IEEE*, vol. 4, no. 1, pp. 23–33, Mar. 1997, ISSN: 1070-9932.

[13] M. Bader, A. Richtsfeld, M. Suchi, G. Todoran, W. Holl, W. Kastner, and M. Vincze, "Balancing centralized control with vehicle autonomy in agv systems," in *Proceedings 11th International Conference on Autonomic and Autonomous Systems (ICAS)*, vol. 11, May 2015, pp. 37–43.

[14] T. Schouwenaars, E. Feron, and J. How, "Safe receding horizon path planning for autonomous vehicles," in *Proceedings of the 40th Allerton Conference on Communication, Control and Computing*, Monticello, IL, 2002.

[15] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Aggressive driving with model predictive path integral control," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, IEEE, 2016, pp. 1433–1440.

[16] D. Q. Mayne, M. M. Seron, and S. Raković, "Robust model predictive control of constrained linear systems with bounded disturbances," *Automatica*, vol. 41, no. 2, pp. 219–224, 2005.

[17] D. Q. Mayne and E. C. Kerrigan, "Tube-based robust nonlinear model predictive control," *IFAC Proceedings Volumes*, vol. 40, no. 12, pp. 36–41, 2007.

[18] L. Martinez-Gomez and T. Fraichard, "Collision avoidance in dynamic environments: An ics-based solution and its comparative evaluation," in *Proceedings of the 2009 IEEE International Conference on Robotics and Automation*, ser. ICRA'09, Kobe, Japan: IEEE Press, 2009, pp. 2251–2256, ISBN: 978-1-4244-2788-8.

[19] T. Schouwenaars, J. How, and E. Feron, "Receding horizon path planning with implicit safety guarantees," in *American Control Conference, 2004. Proceedings of the 2004*, IEEE, vol. 6, 2004, pp. 5576–5581.

[20] B. Øksendal, *Stochastic Differential Equations: An Introduction with Applications*, ser. Hochschultext / Universitext. Springer, 2003, ISBN: 9783540047582. [Online]. Available: https://books.google.at/books?id=VgQDWyihxKYC.

[21] T. P. d. Nascimento, A. L. d. Costa, and C. C. Paim, "Axebot robot the mechanical design for an autonomous omnidirectional mobile robot," in *Electronics, Robotics and Automotive Mechanics Conference, 2009. CERMA '09.*, Sep. 2009, pp. 187–192.

[22] M. O. Tatar, C. Popovici, D. Mandru, I. Ardelean, and A. Plesa, "Design and development of an autonomous omni-directional mobile robot with mecanum wheels," in *Automation, Quality and Testing, Robotics, 2014 IEEE International Conference on*, May 2014, pp. 1–6.

[23] P. Ryan, *Euclidean and Non-Euclidean Geometry: An Analytic Approach*. Cambridge University Press, 1986, ISBN: 9780521276351. [Online]. Available: https://books.google.at/books?id=%5C_6VoRV-RwNwC.

[24] G. Todoran and M. Bader, "Expressive navigation and local path-planning of independent steering autonomous systems," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2016)*, Daejeon, Korea, Oct. 2016, pp. 4742–4749.

[25] D. Schramm, M. Hiller, and R. Bardini, "Single track models," in *Vehicle Dynamics*, Springer-Verlag, 2018, pp. 225–257.

[26] H. Pacejka, *Tyre and Vehicle Dynamics*, ser. Automotive engineering. Butterworth-Heinemann, 2006, ISBN: 9780750669184. [Online]. Available: https://books.google.at/books?id=wHlkbBnu9FEC.

[27] A. Isidori, *Nonlinear control systems*. Springer Science & Business Media, 2013.

[28] A. De Luca, G. Oriolo, and M. Vendittelli, "Control of wheeled mobile robots: An experimental overview," in *Ramsete: Articulated and Mobile Robotics for Services and Technologies*, S. Nicosia, B. Siciliano, A. Bicchi, and P. Valigi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 181–226, ISBN: 978-3-540-45000-9. [Online]. Available: https://doi.org/10.1007/3-540-45000-9_8.

[29] B. R. Rao and S. Ganapathy, "Linear time-varying systems – state transition matrix," in *Proceedings of the Institution of Electrical Engineers*, IET, vol. 126, 1979, pp. 1331–1335.

[30] M. Gasca, "Multivariate polynomial interpolation," in *Computation of Curves and Surfaces*, W. Dahmen, M. Gasca, and C. A. Micchelli, Eds. Dordrecht: Springer Netherlands, 1990, pp. 215–236, ISBN: 978-94-009-2017-0. [Online]. Available: https://doi.org/10.1007/978-94-009-2017-0_7.

[31] A. Gelman and G. Imbens, "Why high-order polynomials should not be used in regression discontinuity designs," *Journal of Business & Economic Statistics*, 2017. eprint: https://doi.org/10.1080/07350015.2017.1366909. [Online]. Available: https://doi.org/10.1080/07350015.2017.1366909.

[32] J. A. Cottrell, T. J. Hughes, and Y. Bazilevs, *Isogeometric analysis: Toward integration of CAD and FEA*. John Wiley & Sons, 2009.

[33] C. De Boor, "On calculating with b-splines," *Journal of Approximation theory*, vol. 6, no. 1, pp. 50–62, 1972.

[34] J. Butcher, *Numerical Methods for Ordinary Differential Equations*. Wiley, 2003, ISBN: 9780471967583. [Online]. Available: https://books.google.at/books?id=nYuDWkxhDGUC.

[35] W. H. Press, *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

[36] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005, ISBN: 0262201623.

[37] F. Fairman, *Linear Control Theory: The State Space Approach*. Wiley, 1998, ISBN: 9780471974895. [Online]. Available: https://books.google.at/books?id=kZQ9x0WQa%5C_IC.

[38] R. Freeman and P. V. Kokotovic, *Robust nonlinear control design: State-space and Lyapunov techniques*. Springer Science & Business Media, 2008.

[39] Y. Kuwata, T. Schouwenaars, A. Richards, and J. How, "Robust constrained receding horizon control for trajectory planning," in *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2005, pp. 15–18.

[40] F. Allgöwer, T. A. Badgwell, J. S. Qin, J. B. Rawlings, and S. J. Wright, "Nonlinear predictive control and moving horizon estimation–an introductory overview," in *Advances in control*, Springer, 1999, pp. 391–449.

[41] L. Grne and J. Pannek, *Nonlinear model predictive control: Theory and algorithms*. Springer Publishing Company, Incorporated, 2013.

[42] J. Rawlings and D. Mayne, *Model Predictive Control: Theory and Design*. Nob Hill Pub., 2009, ISBN: 9780975937709. [Online]. Available: https://books.google.at/books?id=3%5C_rfQQAACAAJ.

[43] B. Chachuat, "Nonlinear and dynamic optimization: From theory to practice," Tech. Rep., 2007.

[44] D. Liberzon, *Calculus of Variations and Optimal Control Theory: A Concise Introduction*. Princeton, NJ, USA: Princeton University Press, 2011, ISBN: 0691151873, 9780691151878.

[45] L. Pontryagin, *Mathematical Theory of Optimal Processes*, ser. Classics of Soviet Mathematics. Taylor & Francis, 1987, ISBN: 9782881240775. [Online]. Available: https://books.google.at/books?id=kwzq0F4cBVAC.

[46] P. B. Ryan, R. L. Barr, and H. D. Todd, "Simplex techniques for nonlinear optimization," *Analytical Chemistry*, vol. 52, no. 9, pp. 1460–1467, 1980.

[47] Z. Manchester and S. Kuindersma, "Derivative-free trajectory optimization with unscented dynamic programming," in *2016 IEEE 55th Conference on Decision and Control (CDC)*, Dec. 2016, pp. 3642–3647.

[48] I. Griva, S. G. Nash, and A. Sofer, *Linear and Nonlinear Optimization (2. ed.)*. SIAM, 2008, pp. I–XXII, 1–742, ISBN: 978-0-89871-661-0.

[49] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd. New York: Springer, 2006.

[50] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004, ISBN: 0521833787.

[51] D. Bertsekas, *Constrained Optimization and Lagrange Multiplier Methods*, ser. Athena scientific series in optimization and neural computation. Athena Scientific, 1996, ISBN: 9781886529045. [Online]. Available: https://books.google.at/books?id=-UQZAQAAIAAJ.

[52] P. T. Boggs and J. W. Tolle, "Sequential quadratic programming for large-scale nonlinear optimization," *Journal of Computational and Applied Mathematics*, vol. 124, no. 1, pp. 123–137, 2000.

[53] M. Heinkenschloss and L. N. Vicente, "Analysis of inexact trust-region sqp algorithms," *SIAM Journal on Optimization*, vol. 12, no. 2, pp. 283–302, 2002.

[54] M. S. Lau, S. Yue, K. Ling, and J. Maciejowski, "A comparison of interior point and active set methods for fpga implementation of model predictive control," in *Control Conference (ECC), 2009 European*, IEEE, 2009, pp. 156–161.

[55] C. Kanzow and A. Klug, "On affine-scaling interior-point newton methods for nonlinear minimization with bound constraints," *Computational Optimization and Applications*, vol. 35, no. 2, pp. 177–197, 2006.

[56] K. Chen, *Matrix Preconditioning Techniques and Applications*, ser. Cambridge Monographs on Applie. Cambridge University Press, 2005, ISBN: 9780521838283. [Online]. Available: https://books.google.at/books?id=d9UdanCqJ1QC.

[57] L. Trefethen and D. Bau, *Numerical Linear Algebra*, ser. Other Titles in Applied Mathematics. Society for Industrial and Applied Mathematics, 1997, ISBN: 9780898713619. [Online]. Available: https://books.google.at/books?id=bjLu6zjWbEC.

[58] T. Howard, C. Green, D. Ferguson, and A. Kelly, "State space sampling of feasible motions for high-performance mobile robot navigation in complex environments," *Journal of Field Robotics*, vol. 25, no. 6-7, pp. 325–345, Jun. 2008.

[59] A. Kolmogorov and S. Fomin, *Introductory Real Analysis*, ser. Dover Books on Mathematics. Dover Publications, 1975, ISBN: 9780486612263. [Online]. Available: https://books.google.at/books?id=z8IaHgZ9PwQC.

[60] K. Hoffman and R. Kunze, *Linear algebra*, ser. Prentice-Hall mathematics series. Prentice-Hall, 1971. [Online]. Available: https://books.google.at/books?id=I4kQAQAAIAAJ.

[61] K. H. Wray, D. Ruiken, R. A. Grupen, and S. Zilberstein, "Log-space harmonic function path planning," in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, IEEE, 2016, pp. 1511–1516.

[62] R. Duffin, "The maximum principle and biharmonic functions," *Journal of Mathematical Analysis and Applications*, vol. 3, no. 3, pp. 399–405, 1961, ISSN: 0022-247X. [Online]. Available: //www.sciencedirect.com/science/article/pii/0022247X6190066X.

[63] I. K. Youssef and A. A. Taha, "On the modified successive overrelaxation method," *Appl. Math. Comput.*, vol. 219, no. 9, pp. 4601–4613, Jan. 2013, ISSN: 0096-3003. [Online]. Available: http://dx.doi.org/10.1016/j.amc.2012.10.071.

[64] T. H. Cormen, *Introduction to algorithms*. 2009.

[65] J. V. Gómez, D. Álvarez, S. Garrido, and L. Moreno, "Fast methods for eikonal equations: An experimental survey," *CoRR*, vol. abs/1506.03771, 2015. [Online]. Available: http://arxiv.org/abs/1506.03771.

[66] T. Faulwasser, *Optimization-based solutions to constrained trajectory-tracking and path-following problems*, EPFL-BOOK-184941. Shaker Verlag, 2013.

[67] P. Fankhauser and M. Hutter, "A universal grid map library: implementation and use case for rough terrain navigation," in *Robot Operating System (ROS) – The Complete Reference (Volume 1)*, A. Koubaa, Ed., Springer, 2016, ch. 5, ISBN: 978-3-319-26052-5.

[68] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: An open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[69] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 3, Sep. 2004, 2149–2154 vol.3.

[70] "Handbook of computational geometry," J.-R. Sack and J. Urrutia, Eds., 2000.

[71] T. A. Bickart, "Matrix exponential: Approximation by truncated power series," *Proceedings of the IEEE*, vol. 56, no. 5, pp. 872–873, May 1968, ISSN: 0018-9219.

[72] M. Zeilinger, R. Hauk, M. Bader, and A. Hofmann, "Design of an autonomous race car for the formula student driverless (fsd)," in *Proceedings of the OAGM & ARW Joint Workshop (OAGM & ARW-17)*, Vienna, Austria, May 2017, pp. 57–62.

[73] E. Hairer, G. Wanner, and C. Lubich, "Symplectic integration of hamiltonian systems," in *Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 179–236, ISBN: 978-3-540-30666-5. [Online]. Available: https://doi.org/10.1007/3-540-30666-8_6.

[74] S. N. Laboratories, U. S. N. N. S. Administration, U. S. D. of Energy. Office of Scientific, and T. Information, *Sacado: Automatic Differentiation Tools for C++ Codes*. United States. National Nuclear Security Administration, 2009. [Online]. Available: https://books.google.at/books?id=F69UnQAACAAJ.

[75] D. Kourounis, L. Gergidis, M. Saunders, A. Walther, and O. Schenk, "Compile-time symbolic differentiation using c++ expression templates," *ArXiv preprint arXiv:1705.01729*, 2017.

Eidesstattliche Erklärung

Hiermit erkläre ich, dass die vorliegende Arbeit gemäß dem Code of Conduct – Regeln zur Sicherung guter wissenschaftlicher Praxis (in der aktuellen Fassung des jeweiligen Mitteilungsblattes der TU Wien), insbesondere ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel, angefertigt wurde. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Die Arbeit wurde bisher weder im In– noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Vienna, February 2018

Horatiu George Todoran