



Master's Thesis

Path-Following NMPC with Deep Learning Environment Motion Prediction for Mobile Robots

Jan-Christophe Swierczek-Jereczek

A thesis presented for the degree of Master of Science

Chair for measurement and control
Technische Universität Berlin
1st corrector: Pr. Dr.-Ing. habil. Rudibert King
2nd corrector: M.Sc. Florian Arnold

In cooperation with IAV GmbH
Research, Powertrain Mechatronics
Supervisor: Dr.-Ing. Sebastian Bernhard

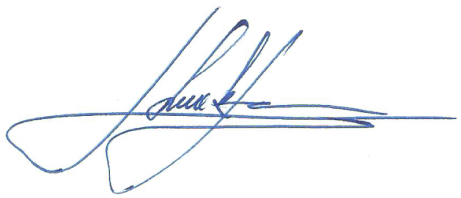
Acknowledgements

I would like to express my deepest thanks to my mother, Chantal Jereczek, for making it all possible and offering me her eternal, loving and unconditional support. The steadiness of our relation could let wonders appear like evidences. I know they are not and acknowledge them every single day.

Many thanks to Pr. King, for being an inexhaustible source of knowledge during the studies and a great instructor, who marvellously prepared me to the present work.

Statement on Oath

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.



Jan-Christophe Swierczek-Jereczek, Berlin December 7 2020.

Abstract

The first and foremost task of mobile robots is to complete an optimal motion from A to B in a cluttered static environment. Because of the obstacles, the allowed state region is non-convex, thus presenting a challenge for the underlying optimization. The straightforward application of a high-frequency, real-time efficient NMPC is therefore often computationally infeasible. In this thesis, the problem is solved by the use of a path planner, combined with a path-following formulation of NMPC. The path is improved by an elastic band algorithm and parametrized by an arc-length approach. The state constraints are described by a sequential convex set approximation of the free space, computed by the safe-corridor method. The results obtained on a simulator of the rover show that the proposed safe-corridor path-following NMPC can deal with any path and any static environment. This is shown to even hold for relatively large parametric and state disturbances. Thus, for the state of the art, it represents the first real-time efficient implementation of path-following NMPC on arbitrary non-convex environments.

Mobile robots can additionally have to deal with dynamic obstacles. It is of utmost importance to avoid them systematically. Humans can complete this task properly, thanks to their context understanding and the resulting on-going constraint forecasting of their motion. This mechanism is now transposed to robots. Thanks to a neural network, based on ConvLSTM-cells, a prediction of future occupancy grid maps is achieved. The path-following NMPC is slightly modified to incorporate the constraint prediction into the optimization problem. It is finally tested on a pedestrian simulation model and shows that the proposed approach allows the robot to automatically adapt its motion to the environment behavior. Thus, it represents a new variant of NMPC, in which the predictive aspect is extended to the optimization constraints.

Kurzfassung

Die zentrale Aufgabe eines mobilen Roboters ist es, eine optimale Bewegung von A nach B in einer statischen Umgebung zu gewährleisten. Da die Umgebung mit Hindernissen besetzt ist, ist die erlaubte Zustandsmenge nicht konvex und stellt somit eine Herausforderung für die zugrundeliegende Optimierung dar. Die direkte Applikation einer hochfrequenten, echtzeitfähigen NMPC ist somit oft rechnerisch unmöglich. In der vorliegenden Masterarbeit, wird das Problem gelöst, indem ein Pfadplanner mit einer Pfadfolge-Formulierung der NMPC kombiniert wird. Der Pfad wird durch den *elastic band* Algorithmus verbessert und anschließend durch einen *arc-length* Ansatz parametrisiert. Die Zustandsnebenbedingungen werden durch eine sequentielle Konvexe-Menge-Approximation des freien Raumes beschrieben. Sie wird durch die *safe-corridor* Methode berechnet. Die an einem Roversimulator erhaltenen Ergebnisse zeigen, dass die vorgeschlagene Erweiterung der Pfadfolge-NMPC mit beliebigen Pfaden und Umgebungen umgehen kann. Es wird gezeigt, dass dies auch für relativ hohe Parameter- und Zustandsstörungen gilt. Für den Stand der Forschung, ist es somit die erste echtzeitfähige Implementierung der Pfadefolge-NMPC auf beliebigen, nicht-konvexen Umgebungen.

Mobile Roboter können zusätzlich mit dynamischen Hindernissen zu tun haben. Es ist von großer Wichtigkeit, ihnen systematisch auszuweichen. Menschen erfüllen diese Aufgabe besonders gut, dank ihres Kontextverständnisses und der daraus resultierenden, permanenten Vorhersage der Bewegungsnebenbedingungen. Dieser Mechanismus wird nun auf Robotern übertragen. Mithilfe eines, auf ConvLSTM-Zellen basierten, neuronalen Netzes, wird eine Vorhersage der zukünftigen *occupancy grid maps* erzielt. Die Pfadfolge-NMPC wird leicht abgeändert, sodass die Prädiktion der Nebenbedingungen in das Optimierungsproblem aufgenommen wird. Dies wird abschließend auf einen Fußgänger*Innen-Simulator getestet und zeigt, dass die vorgeschlagene Herangehensweise dem Roboter ermöglicht, seine Bewegung automatisch an die der Umgebung anzupassen. Es stellt somit eine neue Variante der NMPC dar, in der das vorrausschauende Charakter auf die Zustandsnebenbedingungen erweitert wird.

Index of Notation

Abbreviations and Acronyms – Part 1

ALP	Arc-Length Parametrization
BIT	Batch-Informed Tree
BN	Batch Normalization
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DO	DropOut
DOA	Dynamic Obstacle Avoidance
EBA	Elastic Band Algorithm
EMP	Environment Motion Prediction
EMP-MPFC	Environment Motion Predicting MPFC
FCNN	Fully Connected Neural Network
F.l.t.r., f.t.t.b.	From left to right, from top to bottom
FSC-MPFC	Forecasting Safe-Corridor MPFC
GPM	Grid Prediction Model
GPU	Graphics Processing Unit

Abbreviations and Acronyms – Part 2

LIDAR	Laser Imaging Detection And Ranging
LMS	Least-Mean-Square
LSTM	Long Short Term Memory
MDA	Multi-Dimensional Array
MIMO	Multiple-Input/Multiple-Output
MPC	Model Predictive Control
NMPC	Nonlinear Model Predictive Control
NN	Neural Network
OCP	Optimal Control Problem
ODE	Ordinary Differential Equation
OGM	Occupancy Grid Map
OSM	Optimal Step Model
MPFC	Model-predictive Path-Following Control
ReLU	Rectified Linear Unit
RK4	4-th order Runge-Kutta method
RNN	Recurrent Neural Network
SLAM	Simultaneous Localization And Mapping
SCM	Safe-Corridor Method
SC-MPFC	Safe-Corridor MPFC
SGD	Stochastic Gradient Descent
SSM	State-Space Model
w.r.t.	with respect to

Mathematical Notation – Arrays

b	A scalar
B	A scalar
$\boldsymbol{b}$	A vector
$\mathbf{B}$	A matrix
B	A tensor
$[\mathsf{B}]$	A multi-dimensional array representing the tensor B
$\mathcal{B}$	A space
B	A point in space
i, j	Running indices
b_i	i-th entry of vector $\boldsymbol{b}$
B_{ij}	Entry of matrix $\mathbf{B}$ for i-th row and j-th column
$B_{i_1 \ldots i_{n_m}}$	Entry of component MDA $[\mathsf{B}]$
$\mathcal{T}^m$	Space of m-th order tensors
$n_{\boldsymbol{b}}$	Dimension of the vector $\boldsymbol{b}$
$n_{\mathbf{B}}$	Dimension of the square matrix $\mathbf{B}$
$n_{\mathcal{B}}$	Dimension of an element of $\mathcal{B}$
$\cdot$	Contraction operation
$*$	Convolution operation
$\odot$	Element-wise multiplication

Mathematical Notation – Control Theory – Part 1

$\oplus_0$	Initial value of the arbitrary variable $\oplus$
$\tilde{\oplus}$	Value of the arbitrary variable $\oplus$ predicted by the optimization
c_p	Parametric disturbance factor
$\boldsymbol{e}$	Trajectory-following error
$e_\perp$	Normal path-tracking error
E	End cost
$\boldsymbol{f}$	System dynamic
$\boldsymbol{h}$	State-output relation
H_c	Control horizon
H_p	Prediction horizon
$H_\mathcal{D}$	Constraint horizon
I	Cost functional
k	Arbitrary time step
$\overline{k}$	Current time step
k_E	Time step corresponding to end time t_E
$n_{\text{collision}}$	Number of collisions with dynamic obstacles
N_B	Number of time steps required to reach the target
q_i	i-th diagonal entry of $\mathbf{Q}$
$\mathbf{Q}$	Error weighting matrix
r_i	i-th diagonal entry of $\mathbf{R}$
$\mathbf{R}$	Input weighting matrix
RMSE_P	Root-mean square path-tracking error

Mathematical Notation – Control Theory – Part 2

t	Time
t_{B}	Time required to reach the target
$\bar{t}_{\mathrm{B}}$	Mean time required to reach the target
t_E	End time
$\mathrm{T}_{\mathrm{CPU}}$	Allowed time for the computation an NMPC step
$\boldsymbol{u}$, $\mathbf{U}$	Control variable and associated control sequence
$\mathcal{U}$	Admissible input set
$\boldsymbol{w}$	Reference trajectory
$\boldsymbol{x}$, $\mathbf{X}$	State vector and associated state sequence
$\mathcal{X}$	Admissible state set
$\mathcal{X}_E$	Admissible final state set
$\boldsymbol{y}$	Output vector
$\boldsymbol{z}$, $\mathbf{Z}$	Augmented state vector and associated sequence
δ_t	Time step for control update
$\Delta\boldsymbol{u}$, $\Delta\boldsymbol{\nu}$	Input rate
$\boldsymbol{\kappa}_i$	i-th coefficient of RK4
μ	Virtual input of the path parameter dynamic
$\overline{\mu}$, $\underline{\mu}$	Box constraint of the virtual input
$\boldsymbol{\nu}$, $\boldsymbol{\mathcal{V}}$	Input of the augmented SSM and associated sequence
$\boldsymbol{\xi}_m$	Coordinate of the rover's center of mass

Mathematical Notation – Environment and Path – Part 1

$a_{i,(j)}$, $b_{i,(j)}$	j-th coefficients of the linear interpolating function ρ_i
$\tilde{a}_i$, $\tilde{b}_i$, $\tilde{c}_i$	Coefficients of the i-th half-plane constraint of SCM
A	Start point
B	Goal point
c_e	Ellipse expansion factor
c_r	Scaling factor of repulsive EBA force
c_s	Scaling factor of smoothing EBA force
$\mathcal{C}$	Configuration space
$\mathcal{C}_0$	Free configuration space
$\mathcal{C}_1$	Occupied configuration space
$\mathcal{C}_{-1}$	Unexplored configuration space
$\mathcal{D}_k$	Convex approximation of the configuration space at time step k
$\mathbf{F}_s, \mathbf{F}_r$	Smoothening and repulsive forces of the EBA
$\mathrm{F}_1, \mathrm{F}_2$	Focal points of the ellipse for SCM
l_P	Path length
$\mathbf{M}_\tau$	OGM at time step τ (index τ left away if time-invariant)
$\mathbf{M}_s$	Static OGM
$\mathbf{M}_d$	Dynamic OGM
n_H	Number of half-plane constraints given by SCM
p	Occupancy probability of a cell
p_0	Threshold probability for converting a probabilistic OGM into a binary one

Mathematical Notation – Environment and Path – Part 2

$\mathbf{P}^*$	Path point matrix
$\mathbf{P}$	Extended path point matrix
$\mathcal{P}$	Set of path points
$r_{\min}$	Minimal radius of disk approximation
$r_{(i)}$	Radius of disk free space approximation for i-th path point
r_n	Ellipse half-axis in normal direction
r_r	Range radius w.r.t. the prediction horizon
r_t	Ellipse half-axis in tangential direction
β	Ellipse orientation in plane
$\Delta\mathbf{P}^*$	Approximation of the first path derivative
δ_r	Radius increment for free space fitting
δ_s	Spatial step for environment discretization
$\boldsymbol{\rho}$	Path parametrization function
θ	Path parameter
$\theta_{(i)}$	Path parameter for i-th path point
$\boldsymbol{\xi}_c$	Coordinates of ellipse center point
$\boldsymbol{\xi}_{\text{col}}$	Coordinates of collision point at ellipse expansion time
$\boldsymbol{\xi}_{(i)}$	Coordinates of i-th path point
$\boldsymbol{\xi}_{o,(i)}$	Coordinates of collision point at radius expansion time for i-th path point
$\boldsymbol{\xi}_{\text{P}}$	Coordinates of an arbitrary point P in world coordinate system

Mathematical Notation – System Identification & Mechanics

f_c	Dynamic of the time-continuous rover model
$F_{j,(i)}$	Force applied on the chassis by i-th wheel in j-th direction
$F_{A,(i)}$	Adhesion force on i-th body
$F_{N,(i)}$	Normal force on i-th body
F_F	Friction force
g	Gravitational acceleration on earth surface
$M_{(i)}$	Torque of i-th motor
M_F	Friction torque
$M_{\max}$	Maximal motor torque
l_{χ_i}	Half rover axis in χ_i-direction
$m_{(i)}$	Mass of i-th body
$\boldsymbol{p}$	Parameter vector
$\tilde{\boldsymbol{p}}$	Optimized parameter vector of the parameter identification
$\overline{\boldsymbol{p}}$	Initial guess for $\tilde{\boldsymbol{p}}$
R	Wheel radius
S_i	Center of mass of the i-th body
v	Longitudinal velocity of the rover
$v_{\max}$	Maximal rover velocity
ϕ	Rover orientation in 2D plane
$\Theta_{\chi_j}^{\mathrm{S}_i}$	Moment of inertia of i-th body around its center of mass S_i and axis χ_j
ξ_i	i-th spatial coordinate in world coordinate system
χ_i	i-th spatial coordinate in body-fixed coordinate system
$\psi_{(i)}$	angular position of i-th body

Mathematical Notation – Deep Learning – Part 1

$\oplus^{(l)}$	l-th layer tensor $\oplus$
$\oplus_{(\tau)}$	Tensor $\oplus$ at time step τ
A	Activation tensor
B	Bias tensor
C	Cell state tensor
$\tilde{\mathsf{C}}$	Cell state candidate
F_β	F_β-score
F^+	Amount of false positives
F^-	Amount of false negatives
F	Forget gate tensor
H	Hidden state tensor
I	Input gate tensor
l	Layer index
L	Total amount of layers
$[\mathbf{M}]_\tau$	Sequence of past OGMs used for prediction at time step τ
$[\hat{\mathbf{M}}]_\tau$	Sequence of predicted OGMs at time step τ
$\mathbf{M}_\tau$	OGM at time step τ
$\hat{\mathbf{M}}_\tau^{(i)}$	Predicted OGM at time step τ for the time step $\tau + \iota$
n_K	Number of used kernels
n_ι	Number of frames to predict
O	Output gate tensor
p_{DO}	Dropout probability

Mathematical Notation – Deep Learning – Part 2

S	Stride
T^+	Amount of true positives
T^-	Amount of true negatives
$\mathbf{tanh}$	Element-wise hyperbolic tangent function
$\mathbf{W}$	Weight tensor
$\mathbf{X}$	Input data tensor
$\mathbf{Y}$	Output data tensor
$\hat{\mathbf{Y}}$	Output tensor predicted by NN
$\mathbf{Z}$	Affine transformation of $\mathbf{A}$
α	Learning rate
Γ	Loss function
ι	Prediction index of GPM20
η	Leaky ReLU slope
$\boldsymbol{\sigma}$	Element-wise sigmoid function
τ	Arbitrary time index
$\mathbf{\Psi}$	Non-linear activation function

Contents

Introduction **1**

Goals 2

Structure of the Thesis 3

I Real-Time Path-Following NMPC for Robots in Static Environments 4

1 NMPC for Robotic Applications – State of the Art 4

1.1 Optimal Control – Problem Set 4

1.2 Time-Discrete NMPC 6

1.3 Tackling the Computational Burden 8

1.4 Path-Following NMPC 11

2 ROSBot 2.0 – A System Analysis 16

2.1 Introduction to ROSBot 2.0 16

2.2 Glassy-Box Model 17

2.3 Parameter Identification 20

3 Safe-Corridor Path-Following NMPC 23

3.1 Systematic Path Improvement and Parametrization 23

3.2 Safe-Corridor Method 26

3.3 Implementation and Results 29

3.4 A Brief Discussion about Stability and Robustness 34

II Environment Motion Predicting Path-Following NMPC 36

4 Artificial Neural Networks – State of the Art 36

4.1 Fully Connected Neural Networks . . . 37

4.2 Convolutional Neural Networks . . . 39

4.3 ConvLSTM – A Recurrent Convolution . . . 42

4.4 Time Series Prediction with Convolution . . . 46

5 Environment Motion Prediction – State of the Art 47

5.1 Model-Based vs. Data-Based Approaches . . . 47

5.2 GPM19 – A Data-Based Approach . . . 48

6 Environment Motion Prediction with GPM20 51

6.1 GPM20 Architecture . . . 51

6.2 Data Generation . . . 54

6.3 Training Results . . . 56

7 Forecasting Safe-Corridor Path-Following NMPC 64

7.1 Forecasting Safe-Corridor Method . . . 64

7.2 Implementation and Results . . . 66

Conclusions 73

Outcomes . . . 73

Future Work . . . 74

References V

Appendix VI

A Used Libraries and Frameworks . . . VI

B Pseudo-Codes of the Elastic Band Approach . . . VII

C MPFC Results on Static Environments . . . IX

D Run-Time Analysis . . . XIII

E Exemplary GPM19 Results XVI

F Comparison of Control Results on Dynamic Environments XVII

List of Figures

1 Examples of modern robots. 1

2 Exemplary situations motivating an environment motion predicting approach to MPFC. 3

3 A simplified model of the human motion strategy transposed to robotic applications. . . 3

4 Representation of a single NMPC step. 7

5 Exemplary OGM and planned path. 9

6 Partitioned vs. monolithic approach for path-following tasks with NMPC. 11

7 Parametrization of a spiral path. 12

8 Time evolution of the exemplary reference trajectory for the spiral path. 13

9 Bird's-eye view of the rover in the ξ_1-ξ_2-plane and free body diagram of the chassis. . . 17

10 Side view of the rover and resulting free body diagrams. 18

11 Auto-validation results. 21

12 Cross-validation results. 22

13 Exemplary results of EBA and ALP. 25

14 Comparison of disk and safe-corridor approach for convex set approximation. 28

15 Control strategy on static environments. 30

16 Spatial visualization of the MPFC results on the different test cases. 32

17 Transient behavior of the MPFC on the indoor example. 33

18 Spatial visualization and performance measures of the MPFC for the two parametric disturbance cases on the indoor example. 35

19 Example of a FCNN with $L = 3$. 38

20 Exemplary convolution. 40

21 Connectivity comparison of a shallow FCNN and a shallow CNN. 40
22 Exemplary zero-padding. 41
23 Illustration of the max-pooling operation. 41
24 Exemplary structure of a typical CNN for image classification with $L = 4$ and $L_C = 2$. . 42
25 A single layer RNN, unwrapped over the time dimension. 42
26 A single LSTM-layer, unwrapped over the time dimension. 43
27 The ConvLSTM cell. 45
28 Overview of the GPM19 architecture. 49
29 GPM20 architecture. 52
30 Representation of the different VADERE-environments. 55
31 Conversion from a particle representation into an OGM, with p the occupancy probability. 56
32 Representation of the additional VADERE-environments. 56
33 Ground truth, VEM and GPM20 predictions for a situation on the H terrain. 58
34 Comparison of T_+, F_-, F_+, T_-, for VEM and GPM20 on the example of Figure 33. . . . 59
35 GPM20 results for a sequence of the airport environment. 60
36 GPM20 results for a sequence of the dynamic indoor environment 61
37 Comparison of the F_2-score of GPM20 and VEM on known test environments. 62
38 Comparison of the F_2-score of GPM20 and VEM on unknown test environments. 63
39 Control strategy on dynamic environments. 64
40 Exemplary situation requiring a breaking procedure. 67
41 Final path followed by the two FSC-MPFC procedures based on the GPM20 predictions. 68
42 Exemplary situation of free space liberation by a pedestrian. 69
43 Exemplary situation of a transversal manoeuvre computed by FSC-MPFC$_{0.1}$. 69
44 Exemplary "freezing-robot" situation with GT-FSC-MPFC. 72
45 Comparative transient behavior of the MPFC on the H example. IX
46 Comparative transient behavior of the MPFC on the maze example. X
47 Comparative transient behavior of the MPFC on the outdoor example. XI

48 Visualization and performance measures of SC-MPFC for disturbed cases. XII

49 Box plots of the computation time for an MPFC step and for SCM computation. XIII

52 Exemplary multimodal prediction delivered by GPM19. XVI

53 Exemplary context understanding by GPM19. XVI

54 Critical control results obtained by SC-MPFC on dynamic environments. XVII

55 Control results obtained by GT-FSC-MPFC on dynamic environments. XVIII

56 Control results obtained by FSC-MPFC$_{p_0=0.1}$ on dynamic environments. XIX

57 Control results obtained by FSC-MPFC$_{p_0=0.03}$ on dynamic environments. XX

58 Paths taken by FSC-MPFC$_{p_0=0.1}$ on the GPM20 test environments. XXI

List of Tables

2.1 Mechanical parameters of the rover. 22

2.2 Initial guess $\overline{\boldsymbol{p}}$ and optimized values $\tilde{\boldsymbol{p}}$. 22

3.1 Coefficients of the cost functional I_z and numerical values of the implementation. 30

3.2 Comparison of performance measures for circular and safe-corridor fit of $\mathcal{C}_0$. 31

6.1 Model summary of GPM20. 53

7.1 Comparison of $n_{\text{collision}}$ and $\bar{t}_{\text{B}}$ for three control procedures. 70

Introduction

Over the last decades, the field of robotics opened new horizons. Its development not only allowed us to automatise and speed-up myriads of processes but also opened fully new perspectives. Robot arms nowadays achieve beyond-human precision and can assist us in complex tasks, such as surgical interventions. Mobile robots have shown high versatility in exploration environment, reaching from contaminated regions to fully unexplored ones, as illustrated during the Fukushima catastrophe and NASA's Mars rover missions. Moreover, the last decade flung them beyond the realm of land-based applications, as drones and underwater robots can explore skies and seas. Far from being the case at the beginning of the millennium, robots are entering our daily life with cleaning assistants, 3D-printers, and the upcoming autonomous cars. Furthermore, as commercial robots get cheaper, robotics is now accessible for small enterprises, start-ups, and even private individuals. The Husarion ROSBot 2.0 is an example of it and serves as a case study for the techniques, developed in the present thesis.

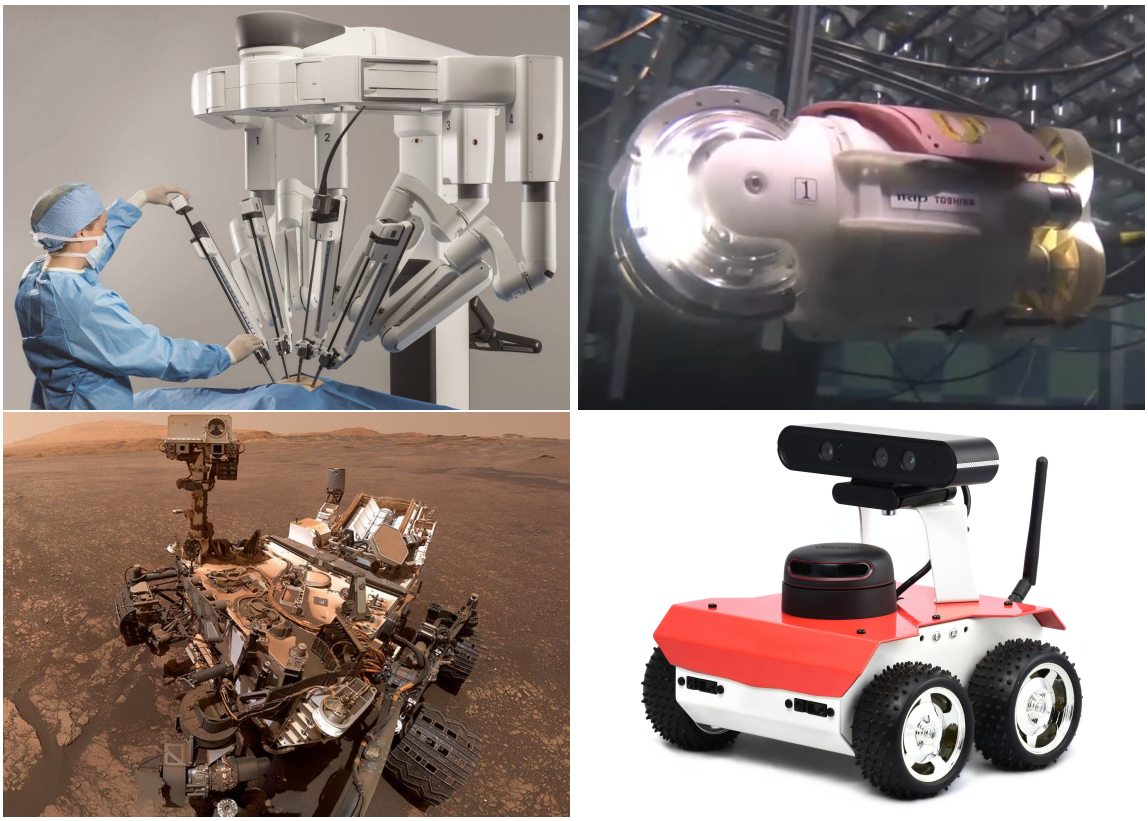


Figure 1: Examples of modern robots. F.l.t.r, f.t.t.b.: 1. Da Vinci, a new surgery assistant [1]. 2. ROV, used in Fukushima to access unreachable underwater regions [2]. 3. Curiosity, NASA's newest Mars rover [3]. 4. ROSBot 2.0, used for the present study case [4].

While the range of application is huge, the underlying basic task for autonomous systems is to ensure a precise movement of the robot, while avoiding any collision with the environment. The control methods are therefore similar, if not the very same, regardless of the case. As the future seems to make us, humans, interact more and more with our artificial autonomous fellows, it is important to design these controls as particularly safe. They should be conservative enough to not put anyone in danger while being smart enough to still be able to fulfil tasks in highly dynamic environments, such as typical city scenarios.

Goals

Humans perform particularly well on numerous tasks in static as well as in dynamic environments, such as moving an object from A to B or crossing a street safely. Various factors can explain our unconscious, daily and yet brilliant functionality:

1. Our perception allows us to not only measure our current position and movement but also the ones of surrounding obstacles. Our abstraction capacity allows us to understand which of the sensed features are relevant for our motion.
2. Given the abstract representation of the environment, our planning ability makes it easy to compute an efficient path linking start point A and goal point B, defined by the coordinates $\boldsymbol{\xi}_{\mathrm{A}}$ and $\boldsymbol{\xi}_{\mathrm{B}}$.
3. Our experience in motion allows us to learn an accurate "plant model" for our bodies.
4. Our resulting motion control, therefore yields precise results and fairly impressive computation performance, leading to a well-followed path.
5. Our context understanding leads us to perform an ongoing environment motion prediction (EMP), thus enabling us to adapt in dynamic situations. On the one hand, if a moving obstacle currently obstructing our path is very likely to leave it, we understand that no adaption is necessary. On the other hand, if a moving obstacle currently is not occluding the path but is very likely to do so in the future, we understand that a re-computation of the trajectory is needed. These two situations are represented in Figure 2 and respectively denoted by Situation 1 and Situation 2. The coordinate in i-th direction is hereby denoted by ξ_i.

A synthesis of this human strategy transposed to robots is depicted in Figure 3. The blocks are numbered accordingly to the previous enumeration and the ones to be extensively treated in this thesis are coloured in blue. This thesis aims to focus on point 4 and 5 and take a step more towards human performance by allowing robots to predict the motion of their environment through deep learning methods. As robots usually abstract their environment into a so-called *occupancy grid map* (OGM), this task results in foreseeing future OGMs. In the present thesis, such a grid prediction model is achieved by training a deep artificial neural network and its prediction is used in such a way that a "smart" avoidance of moving obstacles is performed. To do this, a nonlinear model-predictive path-following control (MPFC) is modified to incorporate the EMP into the optimization problem, thus potentially making any robot better suited for dynamic environments. The resulting novel framework is called "forecasting safe-corridor MPFC" (FSC-MPFC) and gathers different concepts, developed over the last decade. It is the first variant of environment motion predicting MPFC (EMP-MPFC) that can be found in literature.

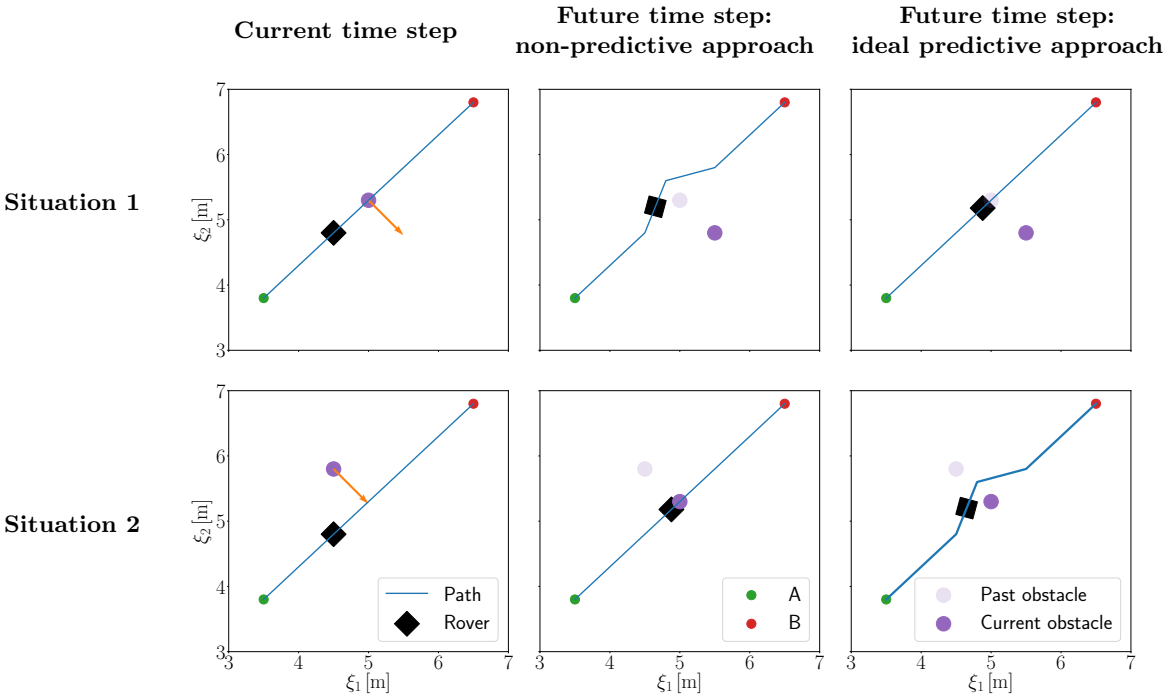


Figure 2: Exemplary situations motivating an environment motion predicting approach to MPFC.

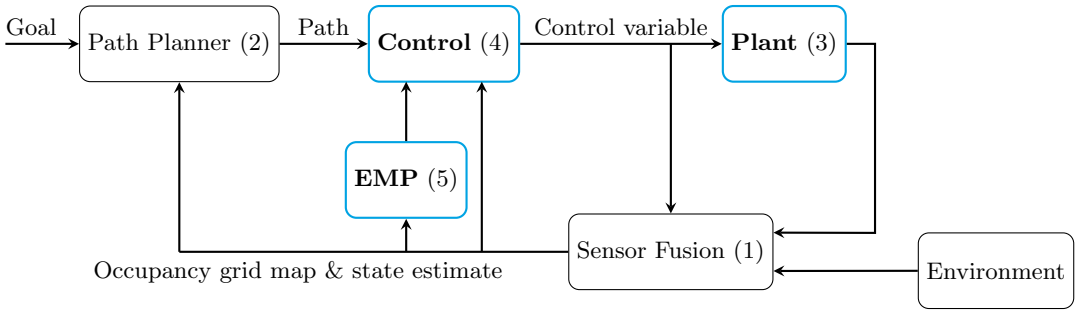


Figure 3: A simplified model of the human motion strategy transposed to robotic applications.

Structure of the Thesis

The present work is structured in two main parts:

I. The necessary steps to develop a real-time efficient MPFC for robotic applications in static environments are introduced. Therefore, the state of the art is first studied in chapter 1 and the case study is introduced in chapter 2. Improvements to the state of the art are proposed in chapter 3, where the final implementation and its results on rover simulations are shown.

II. The extension to EMP-MPFC suited for dynamic environments is introduced. Therefore the state of the art in neural networks and EMP are first studied in chapter 4 and chapter 5. The application of a deep-learning technique on environments including pedestrians is subsequently explained in chapter 6. Finally, MPFC is modified to be able to make use of the EMP given by the grid prediction model. Their combined performance is compared to the classical formulation of MPFC.

Part I

Real-Time Path-Following NMPC for Robots in Static Environments

Chapter 1

NMPC for Robotic Applications – State of the Art

The idea of model predictive control (MPC) was first mentioned in the 60s [5, 6] and its earliest application goes back to the end of the 70s [7]. Since then, it has grown to be one of the most powerful control techniques available nowadays and is still a richly published topic [8, 9, 10, 11]. It relies on the idea of iteratively solving an optimal control problem (OCP) over a simulated future time horizon, thus conferring its predictive character. Two main formulations of MPC exist: a time-continuous and a time-discrete one. The time-continuous one results, even for a finite prediction horizon, in an infinite-dimensional OCP which cannot be handled by a computer. In contrast, the time-discrete formulation allows transforming the dynamic optimization into a static one. It is thus more common for practical purposes and is, therefore, used in the present thesis.

An important question to address is: Why use MPC? Thanks to the state of the art in the field of numerical optimization, MPC is able to take input and state constraints into account and is, therefore, one of the few control techniques able to systematically do so. It can deal equally well with single-input/single-output and multiple-input/multiple-output (MIMO) systems. By extending it to a nonlinear formulation (NMPC), it can even deal with nonlinear state-space models (SSMs) and is therefore one of the best-suited techniques for nonlinear control applications [12]. However, it should be mentioned that, as NMPC is a model-based technique, the control performance depends on the model accuracy. Similarly to the linear quadratic control, it offers a direct formulation of requirements by shaping the cost function. Last but not least, the only control technique able to make use of an environment motion prediction (EMP) is the NMPC, as its core idea is per se to predict the future control results.

To introduce the needed concepts and notations, the OCP is formulated in section 1.1 and the NMPC architecture to solve it is presented in section 1.2. Subsequently, the limitations of NMPC and some methods used to overcome them in the field of robot motion are shown in section 1.3. MPFC is one of them and section 1.4 is dedicated to it, as it is a main element of the thesis. The present chapter serves as summary of the state of the art and is therefore very general. However, the case study is used to visualize some of the developed ideas and give a better understanding.

1.1 Optimal Control – Problem Set

We consider the following time-discrete nonlinear SSM for a MIMO system, constrained to some initial conditions:

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k), \quad \boldsymbol{x}, \boldsymbol{u} \in \mathbb{R}^{n_x}, \mathbb{R}^{n_u} \\ \boldsymbol{y}_k &= \boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{u}_k), \quad \boldsymbol{y} \in \mathbb{R}^{n_y} \\ \boldsymbol{x}(t=0) &= \boldsymbol{x}_0 \end{aligned} \qquad \text{With:} \quad \begin{cases} n_x,\, n_u,\, n_y \in \mathbb{N} \\ \boldsymbol{x}_k = \boldsymbol{x}(t=t_k) \\ t_k = k\,\delta_t, \quad k \in \mathbb{N}_0,\, \delta_t \in \mathbb{R}^+. \end{cases} \tag{1.1.1}$$

This difference equation, with $\boldsymbol{f}$ as right-hand-side, describes the evolution of the state $\boldsymbol{x}$ over the time t, given the input $\boldsymbol{u}$ and the initial state $\boldsymbol{x}_0$. The sampling time δ_t is used to update the values, while k describes an arbitrary time step. Generally speaking, not all of the states can be measured and the output $\boldsymbol{y}$ of the system can be formulated as a function $\boldsymbol{h}$ of state and input. Furthermore, we consider time-invariant models. Hence $\boldsymbol{f}$ and $\boldsymbol{h}$ depend only implicitly on t through $\boldsymbol{x}(t_k)$ and $\boldsymbol{u}(t_k)$. The basic problem of control theory is to compute an input $\boldsymbol{u}(t)$ allowing the output $\boldsymbol{y}(t)$ to follow a reference $\boldsymbol{w}(t) \in \mathbb{R}^{n_y}$ over a time interval $t \in [0, k_E\,\delta_t)$. The time step k_E hereby corresponds to the end time $t_E = k_E\,\delta_t$. The best-case scenario yields:

$$\boldsymbol{e}_k = \boldsymbol{w}_k - \boldsymbol{y}_k = \boldsymbol{0} \quad \forall\, k \in \{0, 1, 2, ..., k_E\}\,. \tag{1.1.2}$$

Thus, the goal can be formulated as a least-mean-square (LMS) problem w.r.t. the quadratic error $||\boldsymbol{e}_k||_2^2 = \boldsymbol{e}_k^{\mathrm{T}}\,\boldsymbol{e}_k$ summed over the time steps. However, shaping a more complex cost functional I is a way to take more specifications into account. A common example found in the literature [11, 12, 13] is expressed by:

$$I = \frac{1}{2}\sum_{k=1}^{k_E} \left(\boldsymbol{e}_k^{\mathrm{T}}\mathbf{Q}_e\boldsymbol{e}_k + \Delta\boldsymbol{u}_k^{\mathrm{T}}\,\mathbf{R}_{\Delta u}\,\Delta\boldsymbol{u}_k\right) + E(\boldsymbol{x}_{k_E}). \tag{1.1.3}$$

Hereby:

- The input variable is smoothened by penalizing its transient changes $\Delta\boldsymbol{u}_k = \boldsymbol{u}_k - \boldsymbol{u}_{k-1}$.
- Terminal costs E, depending on the end state $\boldsymbol{x}_E = \boldsymbol{x}(t = k_E\delta_t)$, are introduced. They can be necessary to enforce stability.
- Positive semi-definite symmetric weighting matrices $\mathbf{Q}_e, \mathbf{R}_{\Delta u} \in \mathbb{R}^{n_x \times n_x}, \mathbb{R}^{n_u \times n_u}$ scale the tracking cost and the input rate cost independently. Hence they allow to prioritize specifications.
- It is assumed that the weighted quadratic forms of the error and the input rate are bounded over a possibly infinite number of time steps k_E. In mathematical terms, I is a convergent series, thus delivering a finite scalar as output.

However, the minimization of I is rarely unconstrained, as:

1. Actuators have practical limitations, which result in an acceptable set of controls $\mathcal{U} \subset \mathbb{R}^{n_u}$. Taking the example of a rover, its motor torque is obviously bounded.
2. The allowed state region might be only a subset $\mathcal{X} \subset \mathbb{R}^{n_x}$. For example, the position of a rover cannot coincide with an obstacle.
3. To fulfil the state constraints, the state itself has to be given as optimization variable. This results in an additional constraint, as the state dynamic $\boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k)$ has to be fulfilled.
4. A terminal region $\mathcal{X}_E$ might be required to be achieved for the end time $t_E = k_E\,\delta t$. In the rover example, $\mathcal{X}_E$ could be any point of a planned path as proposed in [10].

As a time-discrete formulation is considered, the following state and input sequences are introduced:

$$\mathbf{X} = \Big[\boldsymbol{x}(0), \quad \boldsymbol{x}(\delta_t), \quad \ldots, \quad \boldsymbol{x}(k_E\,\delta_t)\Big], \quad \mathbf{X} \in \mathbb{R}^{n_x \times k_E+1} \tag{1.1.4}$$

$$\mathbf{U} = \Big[\boldsymbol{u}(0), \quad \boldsymbol{u}(\delta_t), \quad \ldots, \quad \boldsymbol{u}(k_E\,\delta_t)\Big], \quad \mathbf{U} \in \mathbb{R}^{n_u \times k_E+1} \tag{1.1.5}$$

The optimal values of $\boldsymbol{x}$ and $\boldsymbol{u}$ w.r.t. the costs are introduced as $\tilde{\boldsymbol{x}}$ and $\tilde{\boldsymbol{u}}$. Therefore, the OCP results in finding the associated sequences $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{U}}$ minimizing I:

$$\tilde{\mathbf{X}}, \tilde{\mathbf{U}} = \underset{\mathbf{X},\mathbf{U}}{\arg\min}\, I \tag{1.1.6a}$$

$$\text{subject to:} \quad \tilde{\boldsymbol{x}}_0 = \boldsymbol{x}(t=0) \tag{1.1.6b}$$

$$\tilde{\boldsymbol{x}}_{k+1} = \boldsymbol{f}(\tilde{\boldsymbol{x}}_k, \tilde{\boldsymbol{u}}_k) \quad \forall\, k \in \{0,1,2,\ldots,k_E-1\} \tag{1.1.6c}$$

$$\tilde{\boldsymbol{u}}_k \in \mathcal{U} \quad \forall\, k \in \{0,1,2,\ldots,k_E\} \tag{1.1.6d}$$

$$\tilde{\boldsymbol{x}}_k \in \mathcal{X} \quad \forall\, k \in \{0,1,2,\ldots,k_E\} \tag{1.1.6e}$$

$$\tilde{\boldsymbol{x}}_E = \boldsymbol{x}(t=t_E) \in \mathcal{X}_E. \tag{1.1.6f}$$

If an exact model of the system would be available and no disturbance would arise, solving the OCP stated in Equation 1.1.6a-f would be sufficient to achieve the wanted behavior. However, this is never fulfilled. The NMPC architecture, aiming to tackle this issue, is explained in the next section.

1.2 Time-Discrete NMPC

Throughout the present thesis, it is assumed that the state $\boldsymbol{x}_k$ can be exactly known, based on the measurement $\hat{\boldsymbol{y}}_k$ and a state estimator, such as a Kalman-filter. Thanks to the current state estimate, the feed-forward approach of Equation 1.1.6a-f can be turned into a feed-back method. For the current time step $\overline{k}$, the state $\boldsymbol{x}_{\overline{k}}$ is therefore set as initial condition, the OCP is solved on the simulated plant and the first entry of the resulting optimized control sequence is applied on the real plant. At the next time step $\overline{k}+1$, the state is estimated again and the optimization procedure is repeated with the new initial conditions, allowing to correct eventual discrepancies between simulation and reality. In fact, the predicted values $\tilde{\boldsymbol{x}}_k$, $\tilde{\boldsymbol{y}}_k$ and $\tilde{\boldsymbol{u}}_k$, output by the optimization procedure, can differ from the real ones $\boldsymbol{x}_k$, $\boldsymbol{y}_k$ and $\boldsymbol{u}_k$, for $k > \overline{k}$. This is due to model uncertainties, disturbances and the measurement noise. The optimized error $\tilde{\boldsymbol{e}}_k = \boldsymbol{w}_k - \tilde{\boldsymbol{y}}_k$ is therefore only an approximation of the future cost. Let us now take a look at the problem solved for each update.

The control $\boldsymbol{u}(t)$ is set as constant for $t \in [t_k, t_{k+1}]$, leading to a stair-shaped control sequence, as illustrated in Figure 4. Furthermore, to limit the number of optimization variables and make NMPC a real-time capable method, two horizons $H_c, H_p \in \mathbb{N}$ are introduced:

1. The prediction horizon $H_p \geq H_c$, is introduced to limit the dimensionality of the optimized state sequence $\tilde{\mathbf{X}}$. The SSM is only simulated over $k \in \{\overline{k}, \overline{k}+1, \ldots, \overline{k}+H_p\}$.

$$\tilde{\mathbf{X}}_{\overline{k}} = \Big[\tilde{\boldsymbol{x}}_{\overline{k}} \quad \tilde{\boldsymbol{x}}_{\overline{k}+1} \;\ldots\; \tilde{\boldsymbol{x}}_{\overline{k}+H_p}\Big], \quad \tilde{\mathbf{X}}_{\overline{k}} \in \mathbb{R}^{n_x \times (H_p+1)} \tag{1.2.1}$$

2. The control horizon H_c is introduced to limit the dimensionality of the optimized control sequence $\tilde{\mathbf{U}}_{\overline{k}}$. The input variable is optimized until $k = \overline{k}+H_c$ and subsequently hold constant until $\overline{k}+H_p$.

$$\tilde{\mathbf{U}}_{\overline{k}} = \Big[\tilde{\boldsymbol{u}}_{\overline{k}} \quad \tilde{\boldsymbol{u}}_{\overline{k}+1} \;\ldots\; \tilde{\boldsymbol{u}}_{\overline{k}+H_c-1} \quad \tilde{\boldsymbol{u}}_{\overline{k}+H_c} \;\ldots\; \tilde{\boldsymbol{u}}_{\overline{k}+H_c}\Big], \quad \tilde{\mathbf{U}}_{\overline{k}} \in \mathbb{R}^{n_u \times (H_p+1)} \tag{1.2.2}$$

For the current application, both horizons are set to be equal and are simply referred to as the prediction horizon H_p. The resulting problem solved for each NMPC step yields:

$$I = \frac{1}{2} \sum_{k=\overline{k}}^{\overline{k}+H_p} \left(\boldsymbol{e}_k^{\mathrm{T}} \, \mathbf{Q}_e \, \boldsymbol{e}_k + \Delta \boldsymbol{u}_k^{\mathrm{T}} \, \mathbf{R}_{\Delta u} \, \Delta \boldsymbol{u}_k \right) + E(\boldsymbol{x}_{\overline{k}+H_p}) \tag{1.2.3a}$$

$$\tilde{\mathbf{U}}_{\overline{k}}, \tilde{\mathbf{X}}_{\overline{k}} = \underset{\mathbf{X}_{\overline{k}}, \mathbf{U}_{\overline{k}}}{\arg\min} \, I \tag{1.2.3b}$$

subject to:

$$\tilde{\boldsymbol{x}}_{\overline{k}} = \boldsymbol{x}_{\overline{k}} \tag{1.2.3c}$$

$$\tilde{\boldsymbol{x}}_{k+1} = f(\tilde{\boldsymbol{x}}_k, \tilde{\boldsymbol{u}}_k) \quad \forall \, k \in \{\overline{k}, \overline{k}+1, ..., \overline{k}+H_p-1\} \tag{1.2.3d}$$

$$\tilde{\boldsymbol{u}}_k \in \mathcal{U} \quad \forall \, k \in \{\overline{k}, \overline{k}+1, ..., \overline{k}+H_p\} \tag{1.2.3e}$$

$$\tilde{\boldsymbol{x}}_k \in \mathcal{X} \quad \forall \, k \in \{\overline{k}, \overline{k}+1, ..., \overline{k}+H_p\} \tag{1.2.3f}$$

$$\tilde{\boldsymbol{x}}_{\overline{k}+H_p} \in \mathcal{X}_E. \tag{1.2.3g}$$

As illustrated in Figure 4, problem 1.2.3 is solved for the current time step $\overline{k}$ and yields control inputs until the prediction horizon $\overline{k}+H_p$. However, only the control $\boldsymbol{u}_{\overline{k}} = \tilde{\boldsymbol{u}}_{\overline{k}}$ will be applied, as the updated problem will be solved again for $k = \overline{k}+1$. This results in a so-called *moving horizon control*, which is sometimes used as synonym for MPC. The nonlinear optimization is usually handled by a gradient-based method, as these approaches tend to yield much faster computation time. As a drawback, the optimization process can "fall" into a local minimum, without being able to leave it [14]. Therefore, an eventual sub-optimality has to be accepted. Common gradient-based optimizers are e.g. sequential quadratic programming [15] or interior-point-method [16, 17]. Gradient-free methods turn out to be more robust towards local minima but are also drastically slower and thus do not allow real-time feasibility [14].

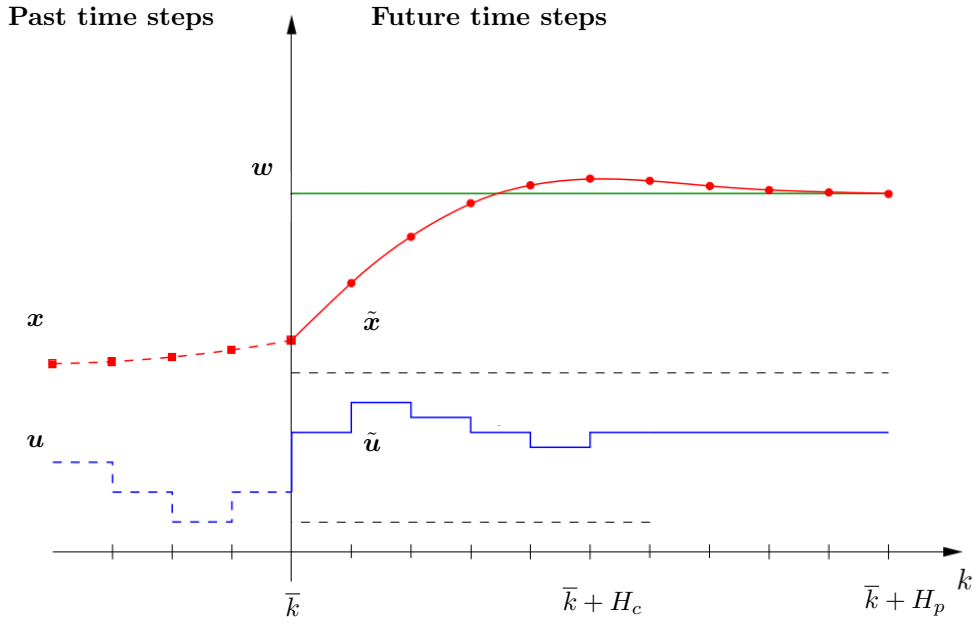


Figure 4: Representation of a single NMPC step, as presented in [12].

The basics of NMPC introduced in the present section are insufficient to apply NMPC on robots, as their need of high-frequency updates for the control variable tends to be incompatible with the relatively high computational burden of nonlinear optimization. State of the art solutions to make NMPC real-time efficient on robots are therefore presented in the upcoming section.

1.3 Tackling the Computational Burden

The large majority of the robots can be represented by SSMs that are commonly nonlinear, as rigid body rotations involve trigonometric functions. By moving in cluttered environments, they do not only have to respect input but also state constraints, which tend to be non-convex. As NMPC is able to deal with these difficulties, it seems to be a way to address a complete solution to the problem. Because of its computational burden, NMPC however remains under-represented in the field of robotics. Recent research has provided a partial solution to this problem and the additional improvement in computation power allows NMPC to leave its cradle, traditionally dominated by large time scale applications such as process engineering. Two of these techniques are presented below: the use of a global path planner and a partial convexification of the optimization problem, leading NMPC to solve a simpler problem.

Path Planning

The environment of a robot is described by a set of coordinates in space, commonly denoted as the configuration space $\mathcal{C} \subseteq \mathbb{R}^{n_\mathcal{C}}$, $n_\mathcal{C} \in \mathbb{N}$. It is commonly divided in free, occupied and unexplored subsets, respectively denoted by $\mathcal{C}_0$, $\mathcal{C}_1$ and $\mathcal{C}_{-1}$, with $\mathcal{C}_0 \cup \mathcal{C}_1 \cup \mathcal{C}_{-1} = \mathcal{C}$ and $\mathcal{C}_i \cap \mathcal{C}_j = \emptyset$ for all $i \neq j$. Hereby, the indices have been chosen in accordance with the fact that $\mathcal{C}$ is commonly approximated by a binary occupancy grid map (OGM) $\mathbf{M}$. This multi-dimensional array (MDA) can be understood as a spatial discretization of $\mathcal{C}$, in which free, occupied and unexplored cells respectively have values of 0, 1 and -1. Its size is determined by the explored space and the spatial discretization step δ_s. Notice that, even though $\mathcal{C}$ is represented by an MDA for computational purposes, it remains a set of coordinates.

Remark 1: The OGM is commonly partitioned in a static and a dynamic one, respectively denoted by $\mathbf{M}_s$ and $\mathbf{M}_d$, with $\mathbf{M} = \mathbf{M}_s + \mathbf{M}_d$. The former one only captures static obstacles, whereas the latter one only approximates the occupancies of dynamic ones.

Remark 2: Besides binary OGMs, other types of OGMs also exist. For example, probabilistic OGMs give an occupancy probability $\mathsf{p} \in [0, 1]$ for each cell. Thus, they give a certain confidence measure of the estimate. Notice that they do not represent a probability distribution in its mathematical definition. They are used in a later part of the present work.

ROSbot 2.0 – Example 1

For a rover moving in a 2D-plane, it holds $\mathcal{C} = \mathbb{R}^2$ and the OGM is a matrix. As an example, one can imagine the indoor environment depicted on the right side of Figure 5, where white and blue cells are respectively free and occupied. However, in practice, the mapping process underlies a certain perspective, as represented on the left side of Figure 5. This leads to the presence of gray, unexplored cells. Here, the robot only mapped its environment from the static perspective of point A. By moving around, the new perspectives would update the OGM and finally lead to the right illustration, in the case of a fully explored environment. This difference between the actual and the known map is a technical difficulty for the path planning task. Nonetheless, it is irrelevant for the development of a reliable control. It is therefore assumed throughout the thesis, that a bird's-eye view of the environment is given, as shown in the right plot of Figure 5, where $\mathcal{C}_{-1} = \emptyset$. If some sensors are external to the robot or if the sensing method allows a depth estimation (e.g. SONAR, computer vision), such a view of the environment can, in fact, be obtained.

The foremost way to take some computation load away from NMPC is to compute a collision-free path $\mathcal{P} \subset \mathcal{C}_0$, linking the start point A and the goal point B. This computation does not require to take the dynamics of the vehicle into account and thus results in a simpler, purely geometric problem, for

which very fast and robust solving techniques exist. An exemplary case is shown in Figure 5. Here the computation was performed by a Batch-Informed Tree Star (BIT*) algorithm. This planner is introduced in [18] and typically yields computation times under a second on any modern CPU. BIT* combines the graph-based and the sample-based approach to path planning, by randomly sampling the free space in a heuristically determined ellipse and subsequently linking the points by graph methods. An implementation of BIT* is available in the Open Motion Planning Library [19].

Remark: Path planners require a collision-checking algorithm. It is commonly based on $\mathbf{M}_s$ rather than on the full OGM $\mathbf{M}$, as dynamic obstacles are likely to move and path planners cannot deal with time-dependent information. This incapacity is due to their inexistent or at most only partial embedding of the system dynamics. This fact is an important statement of this section: as no path planner can make a precise use of the environment's transient behavior, the environment motion predictive character needs to be achieved by the control procedure. In Part II, a method to deal with a time-dependent $\mathbf{M}_d$ is developed. Throughout Part I, the OGM is assumed to be static $\mathbf{M} = \mathbf{M}_s$.

The path planner is commonly denoted as "global planner". The control aiming to follow the path is usually called "local planner". The use of a global planner can be understood as replacing the target position $\boldsymbol{\xi}_\mathrm{B}$ by the set $\mathcal{P}$ as reference $\boldsymbol{w}$ for the local planner. A method to turn the set $\mathcal{P}$ into a transient trajectory $\boldsymbol{w}(t)$ is explained in section 1.4. In summary, the objective of NMPC is reduced to follow the computed path, by taking the dynamics and the constraints into account, while simultaneously minimizing the costs.

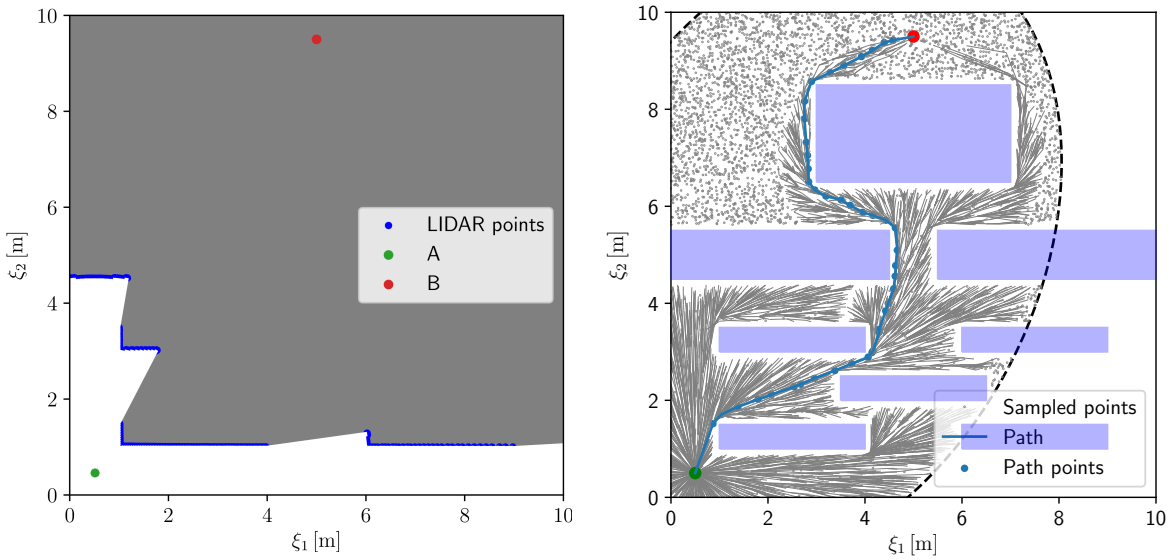


Figure 5: Left: OGM created by the measurement of the environment, while the rover stays in A. $\mathcal{C}_0$, $\mathcal{C}_1$ and $\mathcal{C}_{-1}$ are respectively represented by the white, grey and blue cells. Right: OGM resulting from a bird view of the environment. The path linking A and B is computed by BIT* [18, 19].

In most cases, the path planner is more than a mere helper for NMPC and turns out to be an indispensable step before tackling the optimization problem. As an example, one can set $\boldsymbol{y} = \boldsymbol{\xi}_m$, with $\boldsymbol{\xi}_m \in \mathbb{R}^{n_c}$ the coordinates of the centre of mass. If A and B are separated by a multitude of obstacles, setting $\boldsymbol{w} = \boldsymbol{\xi}_\mathrm{B}$ leads to several local minima w.r.t. the euclidean distance $||\boldsymbol{\xi}_m - \boldsymbol{\xi}_\mathrm{B}||_2$. Figure 5 represents such a cluttered situation. As already mentioned, local minima are problematic for gradient-based optimization methods, such as the ones commonly used in optimization-based control. NMPC as a stand-alone program would therefore not only fail to find a solution in a valid amount of time but might even fail to find a solution at all. Generating the reference $\boldsymbol{w}$ from the path $\mathcal{P}$ avoids such situations.

The input-output structure of a common path planner is:

- Input 1: The start point A, given by the localization procedure, and the goal point B, given by the user, are respectively defined by their coordinates $\boldsymbol{\xi}_{\mathrm{A}}, \boldsymbol{\xi}_{\mathrm{B}} \in \mathcal{C}$.
- Input 2: The static binary OGM $\mathbf{M}_s$.
- Output: The $n_P \in \mathbb{N}$ nodes of the path are denoted by $\boldsymbol{\xi}_{(i)}$, with $i \in \{1, 2, \ldots, n_P\}$. They are stacked column-wise in a matrix $\mathbf{P}^* \in \mathbb{R}^{n_\mathcal{C} \times n_P}$, which can be written out as in Equation 1.3.1. Any column of the path point matrix $\mathbf{P}^*$ is an element of the path $\mathcal{P}$. The path itself is the set of all points obtained by connecting consecutive nodes with straight segments. This relation is expressed mathematically in section 3.1.

$$\mathbf{P}^* = \Big[\boldsymbol{\xi}_{(1)} = \boldsymbol{\xi}_{(A)}, \quad \boldsymbol{\xi}_{(2)}, \quad \ldots, \quad \boldsymbol{\xi}_{(n_P-1)}, \quad \boldsymbol{\xi}_{(n_P)} = \boldsymbol{\xi}_{(B)}\Big] \tag{1.3.1}$$

Even though obtaining $\mathbf{P}^*$ already represents a major step towards a real-time NMPC, the optimization problem remains nonetheless nonlinear and even non-convex. Re-phrasing the optimization problem in a numerically comfortable way might be required for real-time feasibility. Methods allowing this are explained in the next sub-section.

A Step Towards Convex Optimization

Nonlinear optimization can be well handled in convex cases. For NMPC, a convex formulation can only be completely achieved by a linearization of $\boldsymbol{f}$. Otherwise, the associated dynamic constraints of Equation 1.2.3f leads to a non-convexity of the optimization domain. This simplification is widely used for applications, where NMPC's main task is to compute a control fulfilling the constraints. As the linearization can lead to major errors, the prediction horizon is set to be small, and the foreseeing aspect of NMPC almost vanishes. Typical values yield about $t_{\overline{k}+H_p} - t_{\overline{k}} = 0.1\,\mathrm{s}$, as used in [20, 21]. For such time scales, the linearization does not lead to high prediction errors and is, therefore, an acceptable simplification. However, the present thesis aims to use a longer prediction to adapt the robot motion to the EMP over the next seconds. In this case, linearizing $\boldsymbol{f}$ around the current state can lead to unacceptable prediction errors. As an example, let us assume that the rover operates a right angle turn from $\phi(t_{\overline{k}}) = 0$ to $\phi(t_{\overline{k}+H_P}) = \pi/2$, with ϕ the orientation in the 2D-plane. Equation 1.3.2 shows that, for this common manoeuvre, a linearization around the current state leads to a completely false description of the trigonometric functions. The inability to formulate the problem as convex shall thus be accepted, as the precision of the prediction is prioritized over its computation time.

$$\begin{aligned} &\text{Without linearization:} && \cos(\pi/2) &&= 0 \\ &\text{With linearization in } \phi = 0\text{:} && \cos(\pi/2) &&\simeq \cos(0) - \sin(0) \cdot \pi/2 = 1 \neq 0 \end{aligned} \tag{1.3.2}$$

Because of the geometry of common obstacles, the free space tends to be a non-convex set as it can be seen in Figure 5. This adds another degree of difficulty for the optimization problem if $\mathcal{X} = \mathcal{C}_0$. As a possible solution, the non-convex free space can be approximated by a convex domain $\mathcal{D} \subset \mathcal{C}_0$, fitted around the current position. This is done by a separate algorithm, applied before the NMPC step. Optimizing over a non-convex configuration space can thus be convexified by setting $\mathcal{X} = \mathcal{D}$ and accepting a certain loss of precision in environment description. As it has to be re-computed for each time step, we define $\mathcal{D}_k$ to be the convex set approximation of $\mathcal{C}_0$ at the time step k.

An example of a numerical convex set approximation of $\mathcal{C}_0$ is the spherical free-space approximation, shown in algorithm 1 of the appendix and used in [22, 23]. A sphere $\mathcal{D}_k$ is incrementally expanded by δ_r around the current position $\boldsymbol{\xi}_k$ as long as its intersection with the obstacle set is empty. If the next expansion step does not fulfil this condition, the expansion procedure breaks up. Exemplary results of this methodology can be found in section 3.2.

In summary, the state of the art tackles the problem of non-convexity by two main techniques. Whereas the linearization of $\boldsymbol{f}$ is not suited for the current problem, the convex set approximation of $\mathcal{C}_0$ shall be applied before the optimization. In addition to the dynamic constraints, another non-convexity arises for path following tasks, as explained in section 3.1. However, the results presented in section 3.3 show that the remaining non-convexities are well handled at implementation time.

A major step towards NMPC feasibility is the computation of a path. However defining the set $\mathcal{P}$ as reference $\boldsymbol{w}(t)$ is not supported by classical NMPC, as it does not represent any time-parametrized reference trajectory. In the next section, a state-of-the-art technique to tackle this issue is presented.

1.4 Path-Following NMPC

Common trajectory-tracking NMPC partitions the path-following task into two steps: the design of $\boldsymbol{w}(t)$ out of $\mathcal{P}$ and the subsequent optimization. In contrast, the model-predictive path-following control (MPFC), as proposed in [10], presents a single-step approach. Here, the trajectory to follow is determined within the optimization procedure (c.f. Figure 6). This not only leads to a monolithic formulation of the problem but also grants the optimality of the reference trajectory $\boldsymbol{w}$ w.r.t. the costs and its feasibility w.r.t. the constraints. These advantages lead the present work to make use of such a formulation.

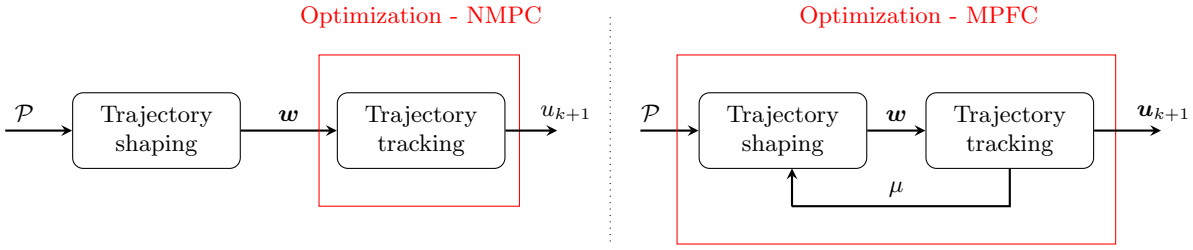


Figure 6: F.l.t.r.: partitioned vs. monolithic approach for path-following tasks with NMPC.

MPFC relies on the idea of introducing a so-called *path parameter* $\theta \in [\theta_0, 0] \subset \mathbb{R}^-$, allowing a spatial parametrization of the path $\mathcal{P}$ by the function $\boldsymbol{\rho}(\theta)$. The surjectivity of ρ can be easily achieved. As global planners do not allow self-intersecting paths, each path coordinate represents a unique image of $\boldsymbol{\rho}(\theta)$. The function $\boldsymbol{\rho}(\theta)$ is therefore bijective. Furthermore, it is required that the boundary points of the path correspond to the boundaries of the definition domain of θ.

$$\boldsymbol{\rho}(\theta) : [\theta_0, 0] \to \mathcal{P} \subset \mathbb{R}^{n_c} \tag{1.4.1a}$$

$$\forall\, \overline{\boldsymbol{\xi}} \in \mathcal{P}\ \exists!\, \overline{\theta} \in [\theta_0, 0]\ :\ \overline{\boldsymbol{\xi}} = \boldsymbol{\rho}(\overline{\theta}) \tag{1.4.1b}$$

$$\boldsymbol{\rho}(\theta_0) = \boldsymbol{\xi}_\mathrm{A}, \quad \theta_0 \in \mathbb{R}^- \setminus \{0\} \tag{1.4.1c}$$

$$\boldsymbol{\rho}(0) = \boldsymbol{\xi}_\mathrm{B} \tag{1.4.1d}$$

ROSbot 2.0 – Example 2

For a rover, one can imagine an exemplary spiral path, as represented in Figure 7. In this case, it is parametrized by:

$$\boldsymbol{\rho}(\theta) = \begin{bmatrix} \rho_1(\theta) \\ \rho_2(\theta) \end{bmatrix} = \begin{bmatrix} \frac{\theta}{4\,\pi}\cos(\theta) \\ \frac{\theta}{4\,\pi}\sin(\theta) \end{bmatrix}, \quad \theta \in [-4\,\pi, 0]. \tag{1.4.2}$$

The coordinates of the three exemplary points depicted in Figure 7 are given by:

$$\boldsymbol{\xi}_\mathrm{A} = \begin{bmatrix} \rho_1(-4\,\pi) \\ \rho_2(-4\,\pi) \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \end{bmatrix}, \quad \boldsymbol{\xi}_\mathrm{B} = \begin{bmatrix} \rho_1(0) \\ \rho_2(0) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad \boldsymbol{\xi}_\mathrm{C} = \begin{bmatrix} \rho_1(-3\,\pi) \\ \rho_2(-3\,\pi) \end{bmatrix} = \begin{bmatrix} 0.75 \\ 0 \end{bmatrix} \tag{1.4.3}$$

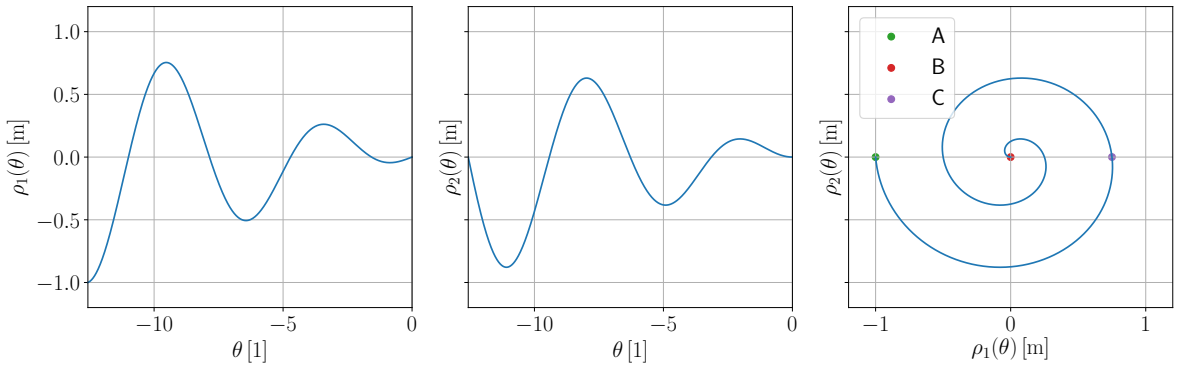


Figure 7: Parametrization of a spiral path. From left to right: $\rho_1(\theta)$, $\rho_2(\theta)$ and the resulting path $\mathcal{P}$.

To compute a trajectory following $\mathcal{P}$, the path parameter θ is now defined as time-dependent variable $\theta(t)$ and needs to evolve from θ_0 to 0 over time. By introducing a fictive control $\mu(t_k)$, able to influence the dynamic of $\theta(t)$, this evolution can be formulated as a control problem, with reference $w_\theta = 0$ and initial condition $\theta(t=0) = \theta_0$. A fictive dynamic f_θ relating θ and μ is introduced. Input constraints are imposed to prevent θ from growing too rapidly and state constraints are imposed, in order to remain in the parametrized domain of θ. The resulting OCP is summarized by:

$$I_\theta = \sum_{k=0}^{k_E} \theta_k^2 \tag{1.4.4a}$$

$$\tilde{\theta}(t_k), \tilde{\mu}(t_k) = \arg\min_{\theta,\mu} I_\theta \tag{1.4.4b}$$

subject to:

$$\tilde{\theta}(t=0) = \theta_0 \tag{1.4.4c}$$

$$\tilde{\theta}_{k+1} = f_\theta(\tilde{\theta}_k, \tilde{\mu}_k), \quad f_\theta : \mathbb{R}\times\mathbb{R} \to \mathbb{R} \tag{1.4.4d}$$

$$\tilde{\mu}(t_k) \in [\underline{\mu}, \overline{\mu}] \subset \mathbb{R} \tag{1.4.4e}$$

$$\tilde{\theta}(t) \in [\theta_0, 0]. \tag{1.4.4f}$$

The right-hand side f_θ, the initial θ_0 and the input variable boundaries $\underline{\mu}$ and $\overline{\mu}$ can be chosen arbitrarily, as θ is rather a mathematical helper than a physical state of the system. This statement is discussed at a later point in this section but shall be assumed to be true for now. In the present case, a first-order model relating θ_k and θ_{k+1} is used. A higher-order relation can however be chosen.

Case study: ROSbot 2.0 – example 3

The simplest dynamic to imagine for θ is an integrator behavior:

$$\theta_{k+1} = \theta_k + \mu_k \, \delta_t \tag{1.4.5a}$$

$$\theta_0 = -4\,\pi. \tag{1.4.5b}$$

To illustrate the introduced principle, the example of a spiral path is used with a polynomial input μ_k over $t_k \in \{0.3,\ 0.6,\ \ldots,\ 25.5\}\ s$, as depicted in Figure 8. The resulting evolution of $\theta(t_k)$ implies a time evolution of the reference $\boldsymbol{\rho}(\theta(t_k))$. The simulation outcomes are depicted in Figure 8. It appears that projecting the 3D plot into the ρ_1-ρ_2-plane results in the path shown in Figure 7.

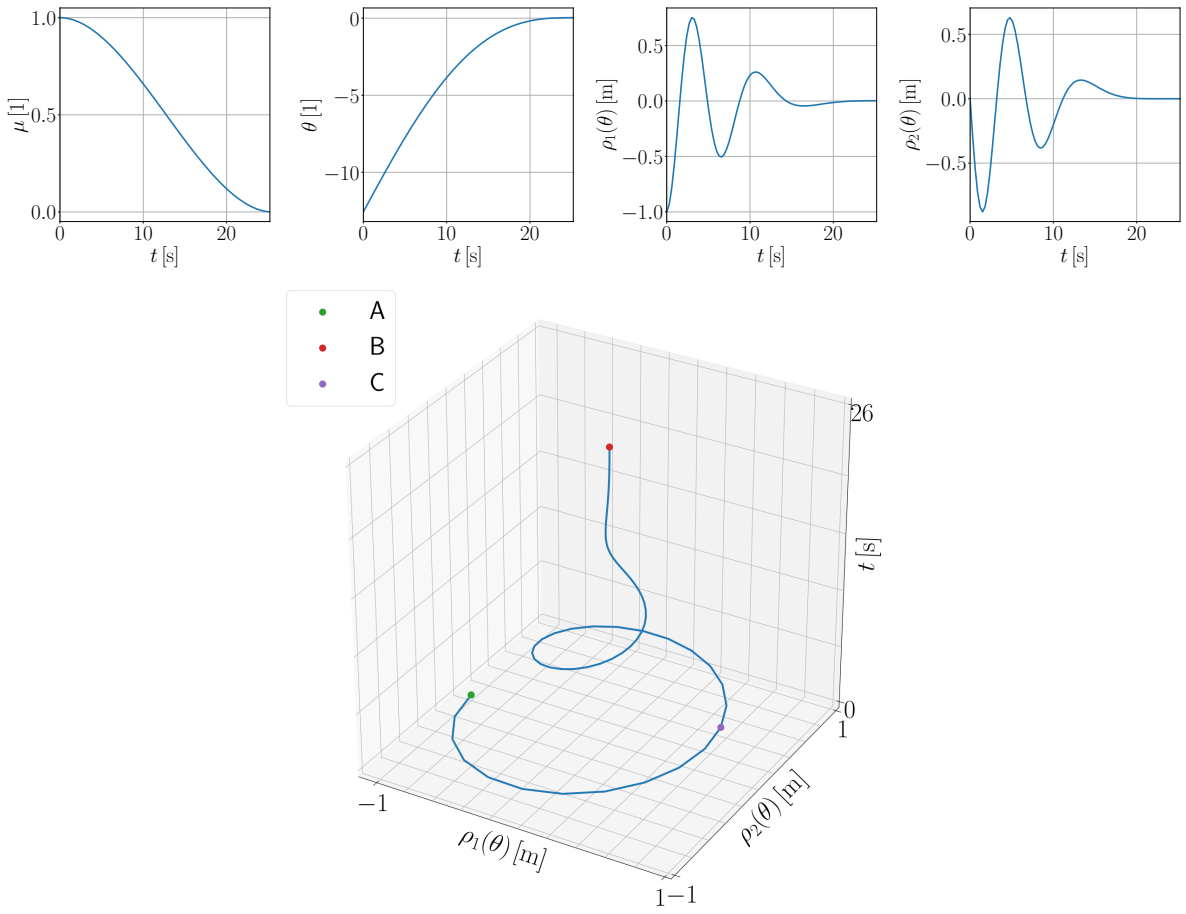


Figure 8: Time evolution of the exemplary reference trajectory for the spiral path.

By combining the OCPs for θ and $\boldsymbol{x}$, NMPC can jointly optimize trajectory-shaping and trajectory-tracking, as depicted in Figure 6. The SSM describing $\boldsymbol{x}(t)$ is therefore augmented with the additional virtual state θ and the corresponding dynamic, resulting in Equation 1.4.6.

$$\begin{aligned} &\boldsymbol{z}(t_{k+1}) = \boldsymbol{f}_z\big(\boldsymbol{z}(t_k), \boldsymbol{\nu}(t_k)\big) \\ &\boldsymbol{z}(t=0) = \boldsymbol{z}_0 = (\boldsymbol{x}_0,\ \theta_0)^{\mathrm{T}} \end{aligned} \qquad \text{With:} \quad \begin{cases} n_z = n_x + 1, \quad n_\nu = n_u + 1 \\ \boldsymbol{z} = (\boldsymbol{x},\ \theta)^{\mathrm{T}}, \quad \boldsymbol{z} \in \mathbb{R}^{n_z} \\ \boldsymbol{\nu} = (\boldsymbol{u},\ \mu)^{\mathrm{T}}, \quad \boldsymbol{\nu} \in \mathbb{R}^{n_\nu} \\ \boldsymbol{f}_z = (\boldsymbol{f},\ f_\theta)^{\mathrm{T}}, \quad \boldsymbol{f}_z : \mathbb{R}^{n_z} \times \mathbb{R}^{n_\nu} \to \mathbb{R}^{n_z} \end{cases} \tag{1.4.6}$$

The next step is to define an error and some appropriate costs. At this point, $\boldsymbol{y}_z$ is defined as shown in Equation 1.4.7. The resulting control error $\boldsymbol{e}_z$ is thus given by Equation 1.4.9.

$$\boldsymbol{y}_z(t_k) = \begin{bmatrix} \boldsymbol{\xi}_m(t_k), & \theta_k \end{bmatrix}^{\mathrm{T}}, \quad \boldsymbol{y}_z \in \mathbb{R}^{n_\mathcal{C}+1} \tag{1.4.7}$$

$$\boldsymbol{w}_z(t_k) = \begin{bmatrix} \boldsymbol{\rho}(\theta_k), & 0 \end{bmatrix}^{\mathrm{T}}, \quad \boldsymbol{w}_z \in \mathbb{R}^{n_\mathcal{C}+1} \tag{1.4.8}$$

$$\boldsymbol{e}_z(t_k) = \boldsymbol{y}_z - \boldsymbol{w}_z = \begin{bmatrix} \boldsymbol{\xi}_m(t_k) - \boldsymbol{\rho}(\theta_k), & \theta_k \end{bmatrix}^{\mathrm{T}}, \quad \boldsymbol{e}_z \in \mathbb{R}^{n_\mathcal{C}+1} \tag{1.4.9}$$

Case study: ROSbot 2.0 - example 4

For a rover:

$$\boldsymbol{y}_z(t_k) = \begin{bmatrix} x_1(t_k) \\ x_2(t_k) \\ \theta(t_k) \end{bmatrix} = \begin{bmatrix} \xi_{m,1}(t_k) \\ \xi_{m,2}(t_k) \\ \theta(t_k) \end{bmatrix} \quad \Rightarrow \quad \boldsymbol{e}_z(t_k) = \begin{bmatrix} x_1(t_k) - \rho_1(\theta_k) \\ x_2(t_k) - \rho_2(\theta_k) \\ \theta_k \end{bmatrix}$$

Remark: The further states are not penalised to keep a simple cost function. If strict following of the path is required, one might want to take the whole pose in the error term. The pose not only contains the robot position but also its orientation in space. However, for the current purpose, flexibility is prioritized over precision. Hence NMPC is not penalised if another angle than the one determined by the path planner is computed as optimal. This allows performing better on dynamic obstacle avoidance.

Diagonal, positive definite weighting matrices $\mathbf{Q}_z$ and $\mathbf{R}_{\Delta\nu}$ are introduced and the cost of NMPC defined in section 1.2 is adapted to the new variables $\boldsymbol{e}_z$ and $\boldsymbol{\nu}$:

$$\mathbf{Q}_z = \mathrm{diag}\big(q_1,\, q_2,\, \ldots\,,\, q_{n_\mathcal{C}+1}\big) \tag{1.4.10}$$

$$\mathbf{R}_{\Delta\nu} = \mathrm{diag}\big(r_1,\, r_2,\, \ldots\,,\, r_{n_\nu}\big) \tag{1.4.11}$$

$$I_z = \frac{1}{2} \sum_{k=\overline{k}}^{\overline{k}+H_p} \Big(\boldsymbol{e}_z^{\mathrm{T}}(t_k)\, \mathbf{Q}_z\, \boldsymbol{e}_z(t_k) + \Delta\boldsymbol{\nu}^{\mathrm{T}}(t_k)\, \mathbf{R}_{\Delta\nu}\, \Delta\boldsymbol{\nu}(t_k) \Big) \tag{1.4.12}$$

Whereas the coefficients $q_1, q_2, \ldots, q_{n_\mathcal{C}}$ are responsible for penalizing the path-tracking error, $q_{n_\mathcal{C}+1}$ penalizes any trajectory where $\theta \neq 0$ and thus ensures forward movement. As previously mentioned, the regularizing term, weighted with $\mathbf{R}_{\Delta\nu}$, smooths the control variables and damps the reaction of the system to the measurement noise. The sequences $\tilde{\mathbf{Z}}$ and $\tilde{\boldsymbol{\nu}}$ are defined as the new optimized state and input sequences. They are therefore the counterpart to $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{U}}$ after augmenting the SSM. The full problem can be now summarized as:

$$\tilde{\mathbf{Z}}_{\overline{k}},\, \tilde{\boldsymbol{\nu}}_{\overline{k}} = \underset{\mathbf{Z}_{\overline{k}},\, \boldsymbol{\nu}_{\overline{k}}}{\arg\min}\, I_z \tag{1.4.13a}$$

subject to:

$$\tilde{\boldsymbol{z}}_{\overline{k}} = \boldsymbol{z}_{\overline{k}} \tag{1.4.13b}$$

$$\tilde{\boldsymbol{z}}_{k+1} = \boldsymbol{f}_z(\tilde{\boldsymbol{z}}_k, \tilde{\boldsymbol{\nu}}_k), \quad \forall k \in \big\{\overline{k}, \overline{k}+1, ..., \overline{k}+H_p-1\big\} \tag{1.4.13c}$$

$$\tilde{\boldsymbol{u}}_k \in \mathcal{U}, \quad \forall k \in \big\{\overline{k}, \overline{k}+1, ..., \overline{k}+H_p\big\} \tag{1.4.13d}$$

$$\tilde{\boldsymbol{x}}_k \in \mathcal{X}, \quad \forall k \in \big\{\overline{k}, \overline{k}+1, ..., \overline{k}+H_p\big\} \tag{1.4.13e}$$

$$\tilde{\theta}_k \in [\theta_0, 0], \quad \forall k \in \big\{\overline{k}, \overline{k}+1, ..., \overline{k}+H_p\big\} \tag{1.4.13f}$$

$$\tilde{\mu}_k \in [\underline{\mu},\, \overline{\mu}], \quad \forall k \in \big\{\overline{k}, \overline{k}+1, ..., \overline{k}+H_p\big\}\,. \tag{1.4.13g}$$

Having gathered the problems of trajectory-shaping and trajectory-tracking into a single one, it appears that MPFC does not result in a structurally different problem than classical NMPC. In fact, the only difference is that the optimization has to deal with a system augmented by only one dimension. This adaption presents a major advantage, as it generates a reference $\boldsymbol{w}$ out of the path $\mathcal{P}$ within the optimization. However, some challenges still hinder a straightforward application:

1. In [10], it is stated that, θ being a fictive state, its dynamics, initial condition and corresponding input constraints can be chosen arbitrarily. However, through the optimization process, θ seems to have a major influence on the real states of the system. The choice of f_θ, θ_0 and $\underline{\mu} < \mu < \overline{\mu}$ can consequently either improve or deteriorate the results in a significant way.
2. The path planner outputs a set of points $\mathbf{P}$, from which a continuous function $\boldsymbol{\rho}$ needs to be derived.
3. The examples of MPFC applications, presented in [10, 24, 25, 26, 27], do not include position constraints. However, it is of major importance that no path deviation presenting collisions occurs. Completing this formulation by a computer efficient free-space approximation is crucial.
4. Until now, dealing with dynamic obstacles is not included at all in this strategy.

In the current chapter, a state-of-the-art formulation of MPFC [10, 24] has been presented. Additionally, the needed concepts for a real-time efficient implementation on a robot have been introduced. Hence, all the control theory knowledge, required for the later development of an EMP-MPFC, has been summarized. Solutions for the challenges 1-3 are presented in chapter 3. Dealing with challenge 4 is handled in chapter 7. To apply these solutions on the ROSBot 2.0, a case study is introduced in the next chapter. A plant model is derived and is subsequently used as a foundation for the simulation and optimization procedures.

Chapter 2

ROSBot 2.0 – A System Analysis

Whereas the state of the art has been introduced in a relatively general way, the present chapter aims to reduce the posed problem to the specific hardware architecture. The methods subsequently developed are nonetheless very general and can be transposed to almost any robot and even to a large variety of NMPC applications.

In section 2.1, the rover type, sensors, actuators and ready-to-use software packages of the ROSBot 2.0 are briefly discussed. A model of the rover is then derived in section 2.2 and the unknown parameters arising in it are identified in section 2.3.

2.1 Introduction to ROSBot 2.0

The ROSBot 2.0 is a relatively compact rover, mainly designed to complete exploration tasks. Because of its dimensions and its maximum speed $v_{\max} \simeq 1.25\,\mathrm{m\,s^{-1}}$, it is mainly designed for indoor applications. Therefore, it is assumed that the movement takes place on a horizontal plane. Each of the four rover wheels is actuated by a DC motor, controlled via quadrature encoding. The maximal torque is $M_{\max} \simeq 0.57\,\mathrm{N\,m}$. Any steering of the rover is achieved by a difference in the motor torques and a governable front axis is consequently not needed. Thanks to this actuator arrangement, the ROSBot 2.0 is able to turn without moving forward and thus follow non-smooth paths and sharp turns.

The main sensor used for the present considerations is a laser imaging, detection and ranging (LIDAR) device. By rapidly rotating around its own axis, the LIDAR emits numerous light beams within a couple of milliseconds. Where the light beams encounter an obstacle, they are reflected and the echoed signals are used to generate a point cloud of the environment. A camera can be additionally mounted and allows to generate a denser point cloud, based on visual features. This could also be used to extend the sensor methods to the wide field of computer vision. Furthermore, an inertial measurement unit allows the ROSBot to measure accelerations. By odometric computations, speed and position can be estimated. Further details concerning component specifications can be found in [4].

The simultaneous localization and mapping (SLAM) procedure takes advantage of these measurements to estimate the pose of the rover and OGM. Localization and mapping both mainly rely on the Bayes-rule and thus estimate a posterior probability, based on the previous estimate and the current measurement. The two most common techniques for these tasks are respectively the Kalman-Bucy-filter [28] and the log-odd updates of the OGMs [29]. The Husarion documentation however does not clearly state whether SLAM was implemented this way or not. As it plays only a secondary role

in the present thesis, the important takeaway, for now, is that information about the position and the environment is available. As they are obtained by posterior probability computation, the vast majority of the measurement noise is filtered out.

Remark: The LIDAR measurement consists of a point cloud, from which the ground plane and the obstacles above the robot's height are filtered out. It is then projected on a 2D grid and passed to the update procedure of the OGM. The resulting map therefore only takes relevant obstacles into account.

Because of the time constraint underlying a master's thesis, the decision was taken to drive all the investigation on a simulation model. However, the model itself is obtained by a parameter identification based on real measurements of the ROSBOT 2.0. This results in a good trade-off between real-world validity and time investment. In the next section, the aforementioned rover model needed for the MPFC is derived.

2.2 Glassy-Box Model

As the control variables of the rover correspond to the four motor torques, modeling the rover behavior results in deriving its equations of motion. To first visualize the simplified geometry of the system, a bird's-eye view is shown in Figure 9. I. The ξ_1-ξ_2-ξ_3 coordinate system is introduced as the world coordinate system. The body-fixed coordinate system χ_1-χ_2-χ_3 is translated by $\boldsymbol{\xi}_m$ and rotated by ϕ w.r.t. the world system. It is used to simplify the derivation of the equation of motion and finally transform them into the ξ-system. Furthermore, the sub-bodies of the rover system are numbered from 0 to 4, with 0 denoting the chassis and the following numbers denoting the wheels as illustrated in Figure 9. I. The half chassis lengths in χ_1 and χ_2 directions are defined as $l_{\chi_1} = 9.85\,\text{cm}$ and $l_{\chi_2} = 11.40\,\text{cm}$. The wheel radius is denoted by $R = 4.25\,\text{cm}$. The tangential direction is given by the velocity $\boldsymbol{v} \in \mathbb{R}^{n_c}$ and can be changed through the acceleration $\dot{\boldsymbol{v}}$, resulting from the motor torques. For the further considerations, the time derivative of an arbitrary variable a is noted as $\dot{a}$.

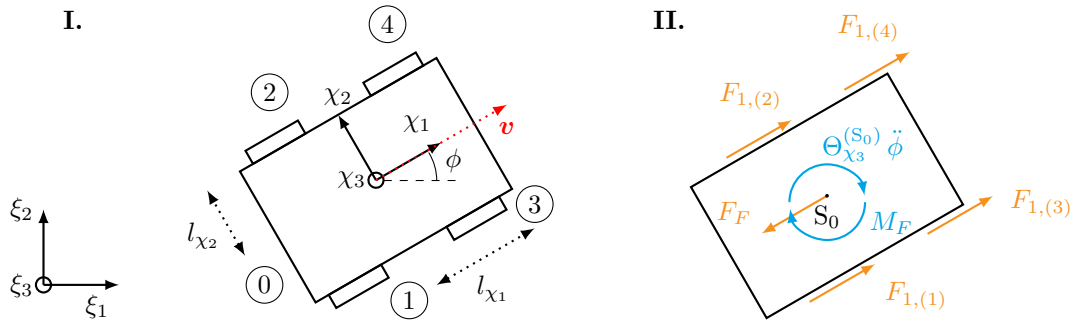


Figure 9: Bird's-eye view of the rover in the ξ_1-ξ_2-plane and free body diagram of the chassis.

The mass $m_{(i)}$ of the i-th body, its centre of mass S_i, its acceleration $\ddot{\chi}_{j,(i)}$ in the j-th direction and the gravitational acceleration g are introduced. For $i \in \{1, 2, 3, 4\}$, we additionally define the reaction force $F_{j,(i)}$ between the chassis and the i-th wheel, the reaction force of the ground plane $F_{N,(i)}$, the resulting adhesion force $F_{A,(i)}$, the motor torque $M_{(i)}$, the angular acceleration $\ddot{\psi}_{(i)}$ around the χ_2-axis and the corresponding moment of inertia $\Theta_{\chi_2}^{S_i}$ around S_i. For the chassis, $\ddot{\psi}_{(0)} = \ddot{\phi}$ and the corresponding $\Theta_{\chi_3}^{S_0}$ are defined around the χ_3-axis. The effect of air friction is gathered in a force F_F and a torque M_F.

A first free body diagram of the chassis is illustrated in Figure 9. II. It is completed by a flipped perspective in Figure 10. Here, the forces $F_{1,(2)}$, $F_{3,(2)}$, $F_{1,(4)}$ and $F_{3,(4)}$ are respectively hidden by

$F_{1,(1)}$, $F_{3,(1)}$, $F_{1,(3)}$ and $F_{3,(3)}$, as they point in the same direction and apply on the same point. They are consequently summed up. For the chassis, the balance of linear momentum and the balance of angular momentum around χ_3 are given by Equation 2.2.1. For the wheels, the balance of linear momentum and the balance of angular momentum around χ_2 are given by Equation 2.2.2.

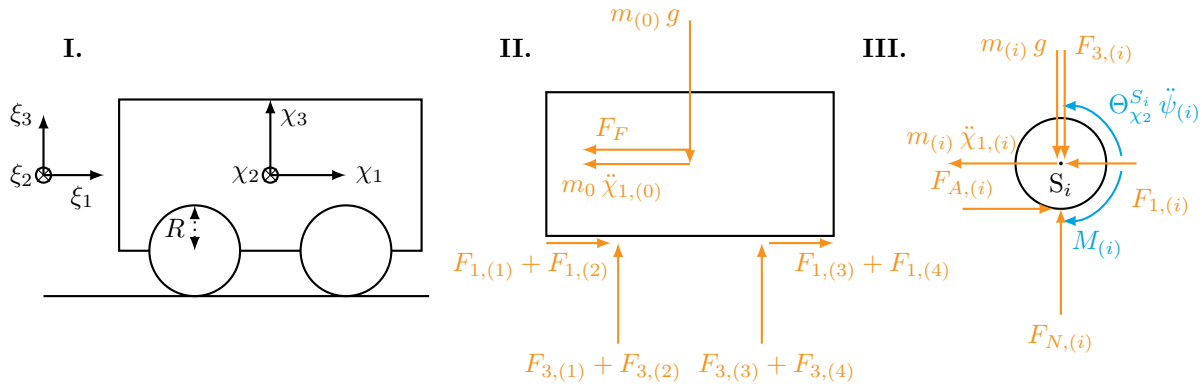


Figure 10: Side view of the rover and resulting free body diagrams. The four free body diagrams of the wheels present the same structure and are summarized by III. with $i \in \{1, 2, 3, 4\}$.

$$\text{Chassis:} \quad \begin{cases} \sum F_{\chi_1} = \sum_{i=1}^{4} F_{1,(i)} - m_{(0)}\, \ddot{\chi}_{1,(0)} - F_F = 0 \\ \sum F_{\chi_3} = \sum_{i=1}^{4} F_{3,(i)} - m_{(0)}\, g = 0 \\ \sum M^{S_0}_{\chi_3} = l_{\chi_2}\, (F_{1,(1)} + F_{1,(3)} - F_{1,(2)} - F_{1,(4)}) - \Theta^{S_0}_{\chi_3}\, \ddot{\phi} - M_F = 0 \end{cases} \tag{2.2.1}$$

$$i\text{-th wheel:} \quad \begin{cases} \sum F_{\chi_1,(i)} = F_{A,(i)} - F_{1,(i)} - m_{(i)}\, \ddot{\chi}_{1,(i)} = 0 \\ \sum F_{\chi_3,(i)} = F_{N,(i)} - m_{(i)}\, g - F_{3,(i)} = 0 \\ \sum M^{S_i}_{\chi_2} = M_{(i)} - \Theta^{S_i}_{\chi_2}\, \ddot{\psi}_{(i)} - F_{A,(i)}\, R = 0 \end{cases} \tag{2.2.2}$$

At this point, further simplifications are done:

1. As the maximum speed and yaw rate are relatively small, the centrifugal force arising in curves is neglected. This results in the absence of forces and thus of acceleration in χ_2-direction.

2. As the wheels have significantly lower mass and lengths compared to the chassis, it is assumed that:

$$m_{(i)}, \Theta_{(i)} \simeq 0, \quad \forall i \in \{1, 2, 3, 4\}. \tag{2.2.3}$$

   From assumption 1. and 2., it results that the mass of the chassis is equally distributed over the wheels and that the corresponding normal force can be computed by:

$$F_{N,(i)} = F_{3,(i)} = \frac{1}{4} \sum_{i=0}^{4} m_{(i)} = \frac{1}{4}\, m_{(0)}. \tag{2.2.4}$$

3. A no-slip condition holds between wheel and ground plane, resulting in:

$$\dot{\chi}_{1,(i)} = R\, \dot{\psi}_{(i)}. \tag{2.2.5}$$

4. The degrees of freedom are reduced, for the system to not be over-actuated:

$$M_{(1)} = M_{(3)} = u_1 \tag{2.2.6}$$
$$M_{(2)} = M_{(4)} = u_2. \tag{2.2.7}$$

5. As the velocity and angular velocity remain low, a linear model with parameter p_3 and p_4 is assumed for the air friction:

$$F_F = p_3\, v \tag{2.2.8}$$
$$M_F = p_4\, \dot{\phi}. \tag{2.2.9}$$

Remark 1: As the main goal is to extend the predictive character of the NMPC to its environment, the model derived in the present thesis is held relatively simple. Nonetheless, the methods used here could still be applied if a more complex model would be used.

Remark 2: Requirement 4 is motivated by more than mere model simplification. One can imagine, that by setting $M_{(1)} + M_{(2)} = -M_{(3)} - M_{(4)}$, the front and back wheel work symmetrically against each other, resulting in a zero dynamic even though a non-zero control input is given. Thanks to the current definition of u_1 and u_2, this undesirable behavior is avoided.

Finally, the kinematic constraint, given by the Euler formula for rigid body rotation, yields:

$$\dot{\chi}_{1,(0)} + l_{\chi_2}\dot{\phi} = \dot{\chi}_{1,(i)}, \quad \forall\, i \in \{1,3\} \tag{2.2.10}$$
$$\dot{\chi}_{1,(0)} - l_{\chi_2}\dot{\phi} = \dot{\chi}_{1,(i)}, \quad \forall\, i \in \{2,4\} \tag{2.2.11}$$

The equations obtained by combining the balance of momentum, the assumption implications and the kinematic constraints are now transformed into the ξ-system. As the angle ϕ, its rate $\dot{\phi}$ and longitudinal velocity v are scalars, they are coordinate-invariant and need no transformation. For the rover position in the ξ-system, a simple rigid body rotation has to be applied:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix}\xi_1\\ \xi_2\end{bmatrix} = \begin{bmatrix} v\cos(\phi)\\ v\sin(\phi)\end{bmatrix} \tag{2.2.12}$$

The state $\boldsymbol{x} = [x_1, \quad \ldots, \quad x_5]^{\mathrm{T}}$ and the parameters p_1 and p_2 are introduced. By combining the balance of momentum, the assumption results and the kinematic constraints, one obtains:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix}\xi_1\\ \xi_2\\ \phi\\ v\\ \dot{\phi}\end{bmatrix} = \begin{bmatrix} v\cos(\phi)\\ v\sin(\phi)\\ \dot{\phi}\\ \frac{2}{m_{(0)}\,R}(u_1+u_2) - p_3 v\\ \frac{l_{\chi_2}}{\Theta^{S_0}_{\chi_3}}(u_1-u_2) - p_4\dot{\phi}\end{bmatrix} \Rightarrow \frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix}x_1\\ x_2\\ x_3\\ x_4\\ x_5\end{bmatrix} = \begin{bmatrix} x_4\cos(x_3)\\ x_4\sin(x_3)\\ x_5\\ p_1(u_1+u_2) - p_3\, x_4\\ p_2(u_1-u_2) - p_4\, x_5\end{bmatrix} = \boldsymbol{f}_c(\boldsymbol{x},\boldsymbol{u}). \tag{2.2.13}$$

The obtained right-hand side $\boldsymbol{f}_c(\boldsymbol{x},\boldsymbol{u})$ is a continuous one, as indicated by the index c. However, for simulation and optimization, one needs to define a corresponding discrete model, as required in chapter 1. To achieve this, several methods can be used, such as the implicit Euler, the Dormand-Prince or the explicit 4th order Runge-Kutta (RK4) method. The latter one was chosen for this thesis, as it presents a good trade-off between accuracy and computation time. In [25], MPFC is implemented on a robot arm with few segments and RK4 leads to a better time performance than a collocation

method, without noticeable loss of precision. The dynamic discretized by RK4 yields:

$$\boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k) = \boldsymbol{x}_k + \frac{\delta_t}{6}(\boldsymbol{\kappa}_1 + 2\,\boldsymbol{\kappa}_2 + 2\,\boldsymbol{\kappa}_3 + \boldsymbol{\kappa}_4) \tag{2.2.14a}$$

With:

$$\boldsymbol{\kappa}_1 = \boldsymbol{f}_c(\boldsymbol{x}_k, \boldsymbol{u}_k) \tag{2.2.14b}$$

$$\boldsymbol{\kappa}_2 = \boldsymbol{f}_c(\boldsymbol{x}_k + \boldsymbol{\kappa}_1 \frac{\delta_t}{2}, \boldsymbol{u}_k) \tag{2.2.14c}$$

$$\boldsymbol{\kappa}_3 = \boldsymbol{f}_c(\boldsymbol{x}_k + \boldsymbol{\kappa}_2 \frac{\delta_t}{2}, \boldsymbol{u}_k) \tag{2.2.14d}$$

$$\boldsymbol{\kappa}_4 = \boldsymbol{f}_c(\boldsymbol{x}_k + \boldsymbol{\kappa}_3\,\delta_t, \boldsymbol{u}_k) \tag{2.2.14e}$$

As it can be seen in Equation 2.2.13, the model however still contains unknown parameters p_i with $i \in \{1, 2, 3, 4\}$. In the next section, a parameter identification is therefore conducted.

2.3 Parameter Identification

The interface of the ROSBot 2.0 only allows setting a target translation and rotation speed as control input. They are here denoted by v_t and $\dot{\phi}_\mathrm{t}$. The target translational velocity is proportional to the sum of torques, whereas the target rotational velocity is proportional to the torque difference between right and left wheels. The proportionality relation is captured within the factors p_1 and p_2.

$$\begin{cases} v_\mathrm{t} & \sim & u_1 + u_2 \\ \dot{\phi}_\mathrm{t} & \sim & u_1 - u_2 \end{cases} \quad \Rightarrow \quad \frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} p_1\, v_\mathrm{t} - p_3\, x_4 \\ p_2\, \dot{\phi}_\mathrm{t} - p_4\, x_5 \end{bmatrix} \tag{2.3.1}$$

One can observe that the unknown parameters only arise in the last two lines of the SSM, shown in Equation 2.3.1. Furthermore, the velocity states x_4 and x_5 can be estimated by odometry. The parameter identification can therefore be driven on this sub-system, which simplifies the problem. Conveniently, the parameters arise linearly. The ordinary differential equation (ODE) itself is linear and decoupled. For such cases, identifiability is given [28]. The straightforward approach for identifying a linear system is to excite it with a chirp signal and a pseudo-random binary signal (PRBS) in two separate experiments. The obtained system responses are respectively used for the auto-validation and the cross-validation [28].

A chirp signal consists of a sine function with linearly increasing frequency. For the present application, the frequency grows from $10^{-5}\,\mathrm{s}^{-1}$ to $1\,\mathrm{s}^{-1}$ over 30 seconds, as this corresponds to the typical frequency domain of the rover input. The resulting signal can be viewed in the top row of Figure 11. The PRBS is the discrete, binary and only pseudo-random counterpart of a white noise. It consists of steps between -1 and 1 with a simulated randomness determining the length of the steps. The minimum time for holding a step is set to one second in the present case. The resulting signal can be viewed in the top row of Figure 12.

Remark: Both signals are normally designed with an amplitude of 100% of the available input variable. This was tried out on the real system. However, because the PRBS is highly non-smooth, jolts arose in the rover motion. This leads to non-linearities, which cannot be captured by the model. For the present case, the amplitude was thus set to 75%. At first, this can seem to be a restriction of the model validity. However, the regularizing term, weighted by $\mathbf{R}_{\Delta\nu}$ in the MPFC, leads to a smooth evolution of the control input. As no such sharp evolution of the input occurs, the jolt is avoided and the input-velocity relation is operated in its linear domain for the current application.

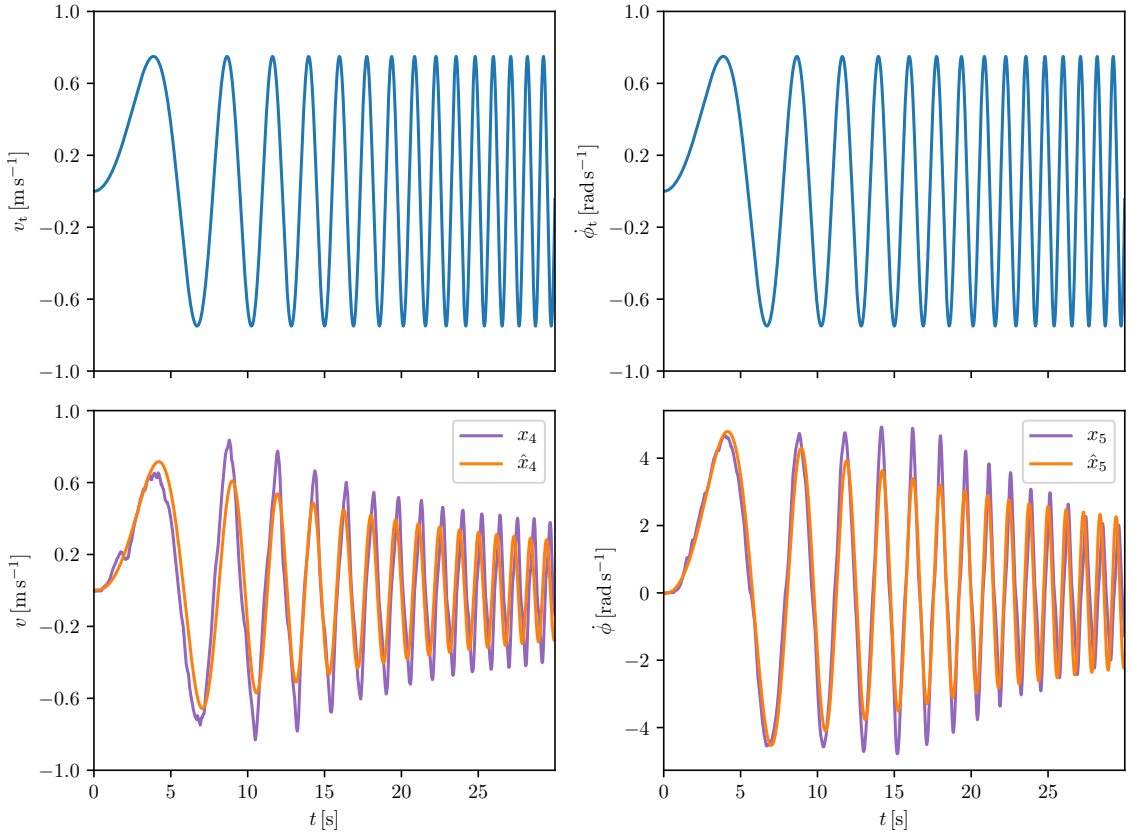


Figure 11: Auto-validation results.

The parameter identification consists of solving the LMS problem given in Equation 2.3.2. Here, $x_i(t_k)$ and $\hat{x}_i(t_k)$ respectively denote the simulated and the i-th state, sampled at time t_k. The optimization aims to find the parameter vector $\tilde{\boldsymbol{p}}$, minimizing the error between model and reality.

$$I_{\text{ID}} = \sum_{k=1}^{N_k} \big(\hat{x}_4(t_k) - x_4(t_k)\big)^2 + \big(\hat{x}_5(t_k) - x_5(t_k)\big)^2 \tag{2.3.2}$$

$$\tilde{\boldsymbol{p}} = \arg\min_{\boldsymbol{p}} I_{\text{ID}} \tag{2.3.3}$$

Based on the prior knowledge of the system, an initial guess $\overline{\boldsymbol{p}}$ can be given. In the present case, this is achieved by the measurement of the rover dimensions and mass. The moment of inertia around χ_3 can be approximated thanks to the formula for a rectangular shape. The friction parameters can be estimated by the steady-state version of Equation 2.3.1, with maximum speed and maximum control input. The mechanical parameters and the resulting initial guess are summarized in Table 2.1.

As the ODE system is decoupled, one can identify the pairs $\{p_1, p_3\}$ and $\{p_2, p_4\}$ independently. The chirp signal is first set as translational target velocity, while the rotational is set to zero. By an optimization driven in CasADi, $\{p_1, p_3\}$ is obtained. The comparison between the simulated results and the measured data is represented in the left column of Figure 11. The very same process is applied to determine $\{p_2, p_4\}$. This time, the chirp is set to be the rotational target velocity, while the translational is set to zero. The comparison between the simulated results and the measured data is represented in the right column of Figure 11. The resulting parameters are summarized in Table 2.2.

The auto-validation by itself does not give any intuition for how good the model is. In fact, the

m_0 [kg]	R [m]	l_{χ_2} [m]	$\Theta_{\chi_3}^{(S_0)}$ [kg m^2]
2.84	0.0425	0.11	0.00537

Table 2.1: Mechanical parameters of the rover.

i	1	2	3	4
$\overline{p}_i$	16.57	21.22	16.57	2.89
$\tilde{p}_i$	2.50	20.83	2.51	3.17

Table 2.2: Initial guess $\overline{\boldsymbol{p}}$ and optimized values $\tilde{\boldsymbol{p}}$.

optimization can overfit the data, thus showing a good error measure on the optimized case but giving a poor generalization. Therefore, a PRBS is applied on the real and the simulated system, to check if the parameter set correctly renders the physical behavior of the system for completely different inputs and outputs. This is a pure comparison, involving no optimization. Here again, the translational and rotational experiments are partitioned. The results are shown in Figure 12.

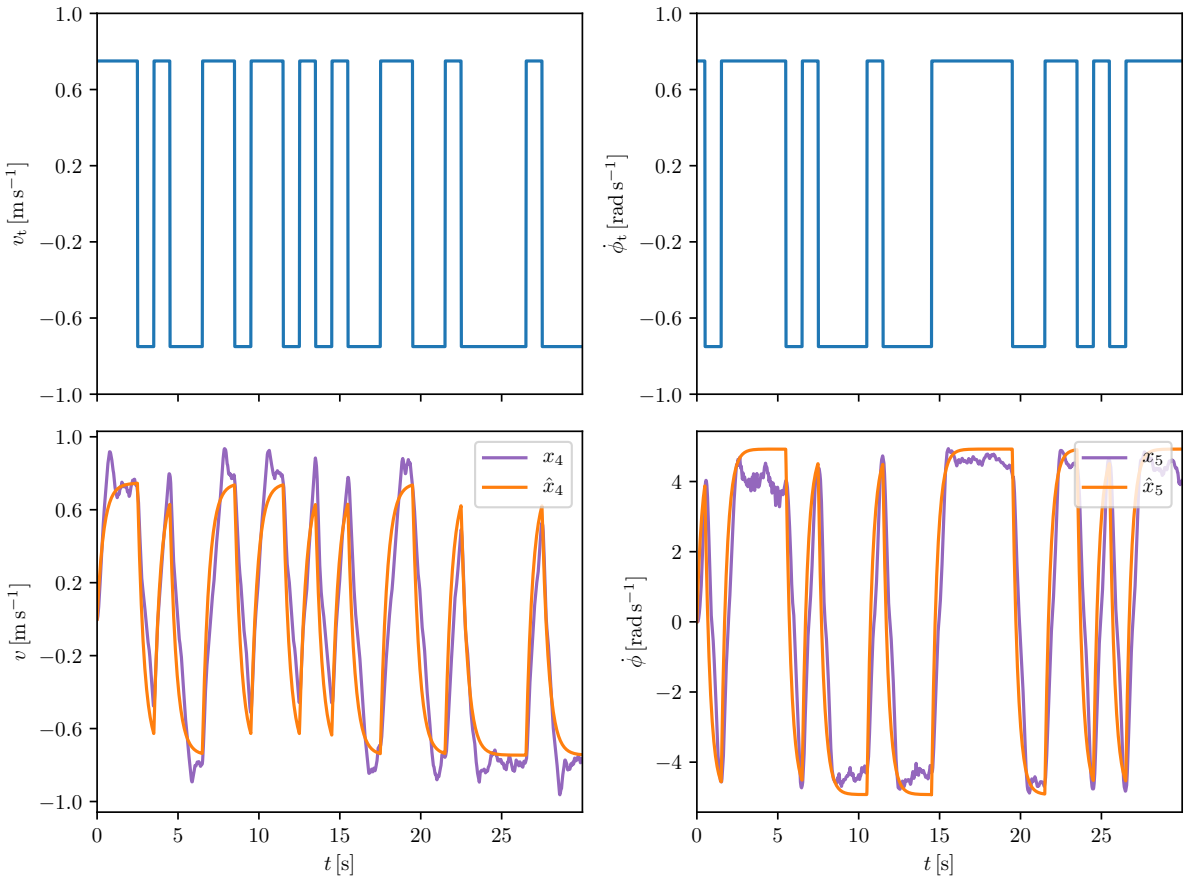


Figure 12: Cross-validation results.

The cross-validation shows that the system dynamics can be sufficiently approximated by the SSM and the optimized set of parameters. As the model is mainly used for the design of a feedback loop, the errors arising can be expected to be largely compensated by the control.

In this chapter, a system identification of the rover has been achieved. This allows for all further investigation to be driven on a realistic simulator. Additionally, as MPC is a model-based technique, its foundation has been built. In the next chapter, the MPFC algorithm is improved by additional features and subsequently tested in simulation.

Chapter 3

Safe-Corridor Path-Following NMPC

The current chapter applies the methods introduced in chapter 1 on the model derived in chapter 2. Additionally, it targets an improvement of the state-of-the-art formulation of MPFC. The aforementioned limitations of MPFC are tackled by a systematic path improvement and parametrization, introduced in section 3.1. In section 3.2, the non-convex state constraints are replaced by a so-called *safe-corridor*. This convex approximation of the free-space proposed in [30] motivates the naming of safe-corridor MPFC (SC-MPFC). Thereafter, the modified formulation of the problem is posed and the implementation results are discussed in section 3.3. In section 3.4, a brief stability discussion and a robustness analysis are subsequently provided. In conclusion, the improvements are compared to previous works on MPFC.

3.1 Systematic Path Improvement and Parametrization

In the present section, the path is first smoothed and moved away from obstacle vicinities, thanks to a post-processing technique. Subsequently, given the output $\mathbf{P}^*$ of a path planner, the arc-length parametrization $\boldsymbol{\rho}(\theta)$ of $\mathcal{P}$ is obtained by the use of piecewise linear functions. Finally, a systematic and physical way of choosing f_θ, θ_0 and $\left[\underline{\mu}, \overline{\mu}\right]$ is introduced.

Path Post-Processing by Elastic Band Approach

The output of a global planner can result in a highly non-smooth path, sometimes passing through a critical vicinity of some obstacles, as e.g. depicted in the lower-left plot of Figure 13. To smooth the result and get a more robust path by slightly moving it away from obstacles, techniques have been developed, such as the elastic band algorithm (EBA), proposed in [22]. As the name reveals, the core concept is to imagine the path as an elastic band, in which stress arises in non-smooth regions. These are characterized by high derivatives and the smoothing force $\mathbf{F}_s$ is, therefore, proportional to the finite difference approximation of the first derivative of $\mathbf{P}^*$, expressed by the matrix $\Delta\mathbf{P}^*$. At the boundary points, the finite difference is set to be zero, as start and goal point shall not be modified. Acting as a restoring force, $\mathbf{F}_s$ is scaled by the weight c_s and takes care of the path regularization:

$$\mathbf{F}_s = c_s\,\Delta\mathbf{P}^*, \qquad c_s \in \mathbb{R}, \quad \mathbf{F}_s,\, \Delta\mathbf{P}^* \in \mathbb{R}^{n_\xi \times n_P} \tag{3.1.1}$$

$$\begin{aligned}\Delta\mathbf{P}^* = \Bigg[\mathbf{0}, \quad & \frac{\boldsymbol{\xi}_{(3)} - 2\,\boldsymbol{\xi}_{(2)} + \boldsymbol{\xi}_{(1)}}{\|\boldsymbol{\xi}_{(3)} - \boldsymbol{\xi}_{(1)}\|_2^2}, \quad \ldots, \quad \frac{\boldsymbol{\xi}_{(i+1)} - 2\,\boldsymbol{\xi}_{(i)} + \boldsymbol{\xi}_{(i-1)}}{\|\boldsymbol{\xi}_{(i+1)} - \boldsymbol{\xi}_{(i-1)}\|_2^2}, \quad \ldots, \\ & \frac{\boldsymbol{\xi}_{(n_P)} - 2\,\boldsymbol{\xi}_{(n_P-1)} + \boldsymbol{\xi}_{(n_P-2)}}{\|\boldsymbol{\xi}_{(n_P)} - \boldsymbol{\xi}_{(n_P-2)}\|_2^2} \quad \mathbf{0}\Bigg].\end{aligned} \tag{3.1.2}$$

As represented in the lower-left plot of Figure 13, moving the path away from obstacles is done by introducing fictive bubbles, fixed around the path points. For the i-th path point $\boldsymbol{\xi}_{(i)}$, a bubble with radius $r_{(i)}$ is inflated until it reaches an obstacle. The disk approximation of the free space, presented in section 1.3, is hereby used. The contact point coordinate $\boldsymbol{\xi}_{o,(i)}$ determines the orientation of the resulting force $\mathbf{f}_{r,(i)}$. Its magnitude is determined by an inverse square relation to the radius $r_{(i)} = \|\boldsymbol{\xi}_{(i)} - \boldsymbol{\xi}_{o,(i)}\|_2$. It is finally scaled by a real positive number c_r. The generated forces are significant for close obstacles and almost none for far ones. In the vicinity of obstacles, the repulsive force consequently dominates, while the smoothing one dominates in sparsely cluttered regions. Notice that, in comparison to [22], the repulsive force is computed without the use of finite difference approximations.

$$\mathbf{f}_{r,(i)} = \frac{c_r}{||\boldsymbol{\xi}_{(i)} - \boldsymbol{\xi}_{o,(i)}||_2^2} \frac{\boldsymbol{\xi}_{(i)} - \boldsymbol{\xi}_{o,(i)}}{||\boldsymbol{\xi}_{(i)} - \boldsymbol{\xi}_{o,(i)}||_2} = c_r \frac{\boldsymbol{\xi}_{(i)} - \boldsymbol{\xi}_{o,(i)}}{||\boldsymbol{\xi}_{(i)} - \boldsymbol{\xi}_{o,(i)}||_2^3}, \quad c_r \in \mathbb{R}^+ \tag{3.1.3}$$

$$\mathbf{F}_r = \Big[\mathbf{0}, \quad \mathbf{f}_{r,(1)}, \quad \ldots, \quad \mathbf{f}_{r,(n_P-1)}, \quad \mathbf{0}\Big] \tag{3.1.4}$$

The smoothing and repulsive forces $\mathbf{F}_s, \mathbf{F}_r$ are applied on a fictive deformable body model of the path, until an equilibrium w.r.t. a certain tolerance is achieved. This is further explained by the pseudo-code of the implemented EBA, provided in section B of the appendix. This finally results in smoother paths with bigger margins w.r.t. the obstacle distance. They are therefore kinetically more meaningful and contribute to the robustness of the combination between local and global planner.

Over this process, some points turn out to be redundant, meaning that their consideration does not lead to a major improvement of the path quality. Hence they are pruned to reduce the complexity of the future path parametrization. In parallel, some path regions suffer from the contrary problem. They are resampled according to the path to refine its resolution where needed. An exemplary result of this technique is depicted in the lower-left plot of Figure 13. For a better understanding of the process, a video illustrating EBA can be found `here`[1]. To give an order of magnitude of the required computation time, the EBA results of Figure 13 take about 0.3 s on an Intel Core i5-8365U CPU. This method is not only fast to compute but also very general, as it can be applied on any path and in any dimension.

EBA remains a secondary implementation of the current thesis and the explanation given here therefore remains sparse. Nonetheless, a more detailed description of the algorithm is provided in section B the appendix. Having obtained a better path regarding robustness and smoothness, the relation between the path point matrix $\mathbf{P}^*$, the path $\mathcal{P}$ and the path parameter dynamic f_θ still have to be determined. This is done in the next section.

Arc-Length Parametrization

To systematically choose an initial value θ_0, a dynamic f_θ and input constraints $\overline{\mu}, \underline{\mu}$, θ is not interpreted as mere mathematical helper any more, but rather as physical value. If the total path length is denoted by l_P, the corresponding initial value is chosen as $\theta_0 = -l_P$. As $\theta_\mathrm{B} = 0$, one can easily imagine, that $\theta_{(i)}$ now represents the negative distance from the current point $\boldsymbol{\xi}_{(i)}$ to the goal point along the path. This methodology is a special case of what is called *arc-length parametrization* (ALP) in the field of differential geometry [31]. This very simple geometric interpretation allows the choosing of a generic dynamic for any path and yields:

$$\theta_{k+1} = \theta_k + \mu\,\delta_t, \qquad \mu \in [-v_\mathrm{max}, v_\mathrm{max}]. \tag{3.1.5}$$

[1] `https://www.youtube.com/watch?v=KQFTMNJjzrw&list=PLTKb-pD4cu3iyjE1YiPTW3SlWxkP4BHzQ&index=1`

Equation 3.1.5 is an integrator relation. Therefore, μ represents the speed with which the distance to the target point evolves. As the maximum speed of the vehicle is known, one can choose the input constraints into accord, without risking that this choice leads to physically unfeasible trajectories. Because of this choice, θ and μ are now represent distances and velocities, measured in m and $\mathrm{m\,s^{-1}}$. The matrix $\mathbf{P}$ is introduced as an extended version of $\mathbf{P}^*$. It contains an additional row with the distance from $\boldsymbol{\xi}_{(i)}$ to $\boldsymbol{\xi}_\mathrm{B}$ along the path. For a rover in a 2D-plane, this results in:

$$\mathbf{P} = \begin{bmatrix} \xi_{1,(1)} = \xi_{1,(\mathrm{A})} & \xi_{1,(2)} & \dots & \xi_{1,(n_P)} = \xi_{1,(\mathrm{B})} \\ \xi_{2,(1)} = \xi_{2,(\mathrm{A})} & \xi_{2,(2)} & \dots & \xi_{2,(n_P)} = \xi_{2,(\mathrm{B})} \\ \theta_{(1)} = -l_P & \theta_{(2)} & \dots & \theta_{(n_P)} = 0 \end{bmatrix} = \begin{bmatrix} \mathbf{P}_1 \\ \mathbf{P}_2 \\ \mathbf{P}_3 \end{bmatrix} \in \mathbb{R}^{3\times n_P}, \quad \mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3 \in \mathbb{R}^{n_P}. \tag{3.1.6}$$

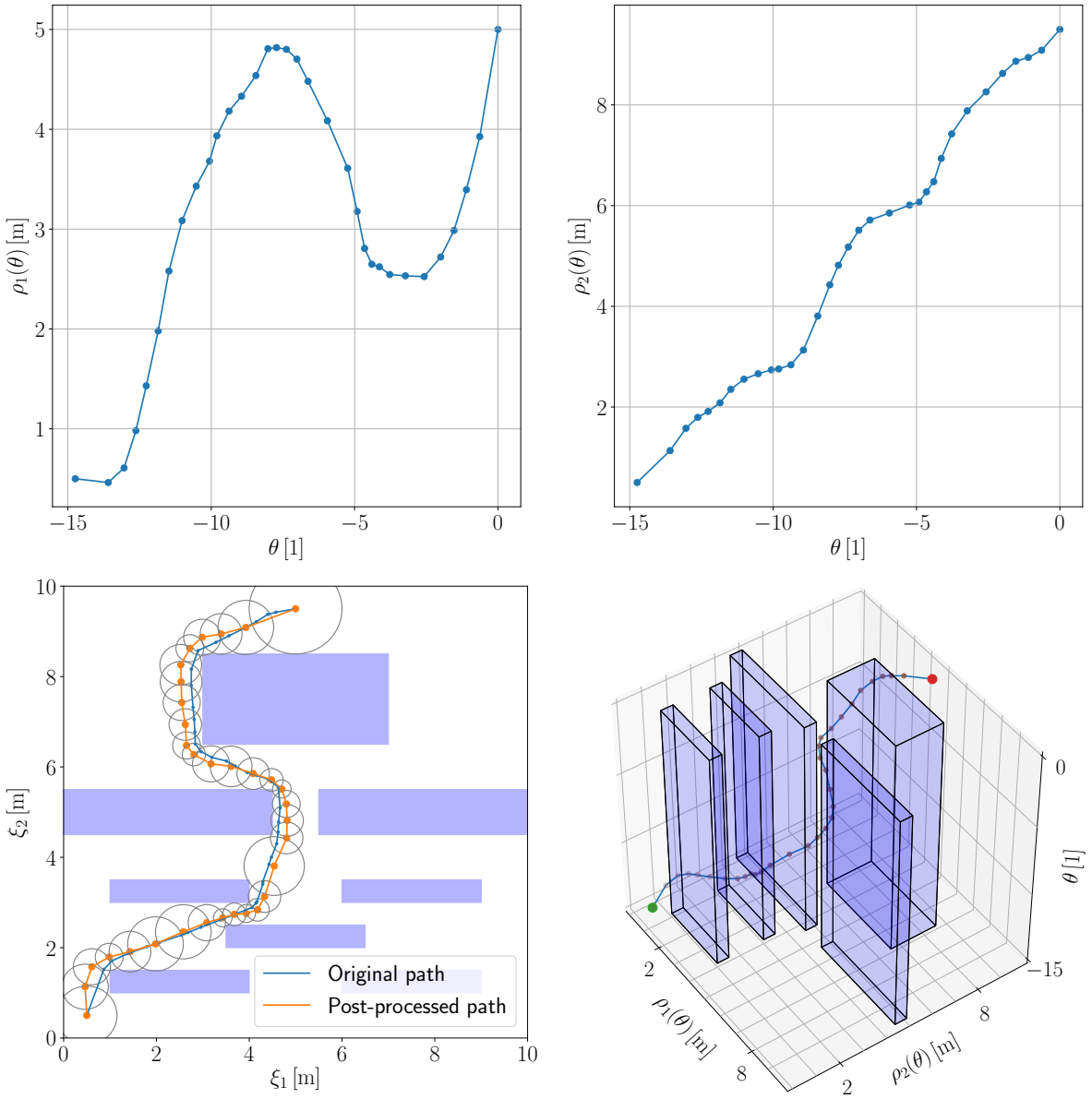


Figure 13: Exemplary results of EBA and ALP. Lower left: comparison of original and post-processed path by EBA. Upper: Exemplary parametrization ρ_1 and ρ_2 by ALP. Lower right: 3D visualization of the parametrization (some obstacles have been removed for the sake of clarity).

The parametrization problem can now be phrased as:

Find a function ρ_i, mapping $\mathbf{P}_3$ onto $\mathbf{P}_i$ for $i \in \{1, 2\}$.

An infinite number of function can fulfil this requirement. However, only one is fully consistent with BIT* and EBA. As both of them assume connecting path points by straight segments, ρ_i is defined as the piecewise linear interpolation of $\mathbf{P}_i$ over $\mathbf{P}_3$:

$$\rho_i(\theta) = \begin{cases} a_{i,(1)}\,\theta + b_{i,(1)}, & \text{if} \quad \theta_{(1)} \leq \theta < \theta_{(2)} \\ a_{i,(2)}\,\theta + b_{i,(2)}, & \text{if} \quad \theta_{(2)} \leq \theta < \theta_{(3)} \\ \vdots & \\ a_{i,(n_P-1)}\,\theta + b_{i,(n_P-1)}, & \text{if} \quad \theta_{(n_P-1)} \leq \theta \leq \theta_{(n_P)} \end{cases} \tag{3.1.7}$$

With:
$$\begin{aligned} a_{i,(j)} &= \frac{\xi_{i,(j+1)} - \xi_{i,(j)}}{\theta_{(j+1)} - \theta_{(j)}} \\ b_{i,(j)} &= \xi_{i,(j)} - a_{i,(j)}\,\theta_{(i)} \end{aligned} \qquad \forall i \in \{1, 2\}\,,\ \forall j \in \{1, 2, ..., n_P - 1\} \tag{3.1.8}$$

An exemplary result of ALP is depicted in Figure 13. This example illustrates the further non-convexity mentioned in section 1.3: as paths can be arbitrarily shaped, they can result in a non-convex reference $\boldsymbol{\rho}(\theta_k)$. Throughout the current section, a systematic approach has been developed to improve the path and obtain its parametrization as SSM, thus allowing to generalize MPFC. As mentioned in section 1.3, techniques allowing to approximate $\mathcal{C}_0$ by a convex set $\mathcal{D}$ already exist. In the next section, the recently proposed safe-corridor technique is presented.

3.2 Safe-Corridor Method

As explained in section 1.3, disk regions can be used to approximate $\mathcal{C}_0$. However, this shows some limitations, especially in the case of narrowings between obstacles or curves around them. Figure 14 shows that, for these cases, the free space is poorly fitted by the disk approach. This is due to the isotropy of a circle, which leads to a lack of degrees of freedom available for the fit.

Many other convex shapes with a higher complexity can be used to replace it and obtain a larger surface of $\mathcal{D}_k$. Convex polygons are a tempting option, as they allow the most general representation of a convex region. In fact, they rely on an arbitrary and possibly infinite number n_H of straight lines, enabling to approximate even curved shapes. The i-th edge is determined by a straight line, with parameters $\tilde{a}_i$, $\tilde{b}_i$, $\tilde{c}_i \in \mathbb{R}$ and fulfilling Equation 3.2.1a. Turning this into an inequality leads to a half-plane constraint. The convex polygon finally results of the combination of an arbitrary number of half-plane constraints, as shown in Equation 3.2.1b.

$$\tilde{a}_i\,\xi_1 + \tilde{b}_i\,\xi_2 + \tilde{c}_i = 0 \tag{3.2.1a}$$

$$\mathcal{D}_k = \left\{ \boldsymbol{\xi} \in \mathbb{R}^2 \ : \tilde{a}_i\,\xi_1 + \tilde{b}_i\,\xi_2 + \tilde{c}_i < 0, \quad \forall i \in \{1, 2, ..., n_H\} \right\} \tag{3.2.1b}$$

Remark 1: The disk approximation nonetheless turned out to be very useful, as its computation time is extremely small. This was taken advantage of within EBA, for which such computations have to be performed at least a hundred of times.

Remark 2: Using convex polygons can be easily generalized to higher dimensions. Instead of straight lines, hyperplanes are used and lead to a convex polytope as free space approximation.

Compared to many other convex shapes, the only drawback of a polygon fit is that a significantly higher

number of parameters has to be computed. The *safe-corridor method* (SCM), introduced in [30] and successfully used in [32], presents a solution to this problem. It offers a general and computationally efficient way to determine the number of edges and the associated $\tilde{a}$, $\tilde{b}$ and $\tilde{c}$. It merely assumes the knowledge of the path points and the OGM. It thus follows the same philosophy as the present thesis: *only rely on environment representations common to all robot concepts.* It is therefore implemented for the present work and its algorithmic structure is presented below. SCM relies on five major steps:

a. **Initialize sparse obstacle list.** The rover range over a prediction horizon is given by the product of the prediction time and the maximum velocity $r_r = H_P\,\delta t\,\|v\|_{\max}$. Thus, none of the obstacle points located outside of the circle with centre point $\boldsymbol{\xi}_m$ and radius r_r have to be considered in the NMPC. This results in a sparse list of obstacle coordinates.

b. **Initialize ellipse.** Compute the pair of path points $(\mathrm{F}_1, \mathrm{F}_2)$, containing the previous and the next path points that are driven by. These points are set as focal points of an ellipse, initialized with the half-axis r_t and r_n, respectively directed in tangential and normal direction of the path. Knowing $(\mathrm{F}_1, \mathrm{F}_2)$, one can compute the centre point coordinate $\boldsymbol{\xi}_c$ and the orientation β of the ellipse. Contrary to r_t and r_n, they remain constant over the fitting process. The ellipse inequality is given by Equation 3.2.2.

$$\mathcal{E} \;=\; \left\{ \boldsymbol{\xi} \in \mathbb{R}^2 \;:\; \frac{\left((\boldsymbol{\xi} - \boldsymbol{\xi}_c) \cdot (\cos\beta,\, \sin\beta)^{\mathrm{T}} \right)^2}{r_t^2} + \frac{\left((\boldsymbol{\xi} - \boldsymbol{\xi}_c) \cdot (\sin\beta,\, -\cos\beta)^{\mathrm{T}} \right)^2}{r_n^2} \leq 1 \right\} \tag{3.2.2}$$

1. **Inflate.** Increase r_t and r_n by a fixed factor $1 < c_e < 1.1$, resulting in a so-called *dilatation*:

$$r_t \leftarrow c_e\, r_t \tag{3.2.3}$$
$$r_n \leftarrow c_e\, r_n \tag{3.2.4}$$

2. **Collide.** Check if any obstacle point fulfils the ellipse equation for the updated half-axis. If not, **inflate** again. If yes, get the coordinates $\boldsymbol{\xi}_{\mathrm{col}}$ of the collision point between ellipse and OGM.

3. **Fit.** By setting c_e close to 1, one can make sure that the collision point is going to be approximately located on the ellipse contour. First, the ellipse and the collision point are translated by $-\boldsymbol{\xi}_c$ and rotated by $-\beta$, resulting in an ellipse, centred in $(0,0)^{\mathrm{T}}$ with $\beta = 0$. This transformation allows to easily determine the equation of the tangent passing by the collision point, as given in Equation 3.2.5. The straight line is subsequently transformed back into the original system of coordinates, with the ellipse centred in $\boldsymbol{\xi}_c$ and orientated with β. The coefficients $\tilde{a}$, $\tilde{b}$ and $\tilde{c}$ are finally obtained by the back-transformation of Equation 3.2.5.

$$\frac{\xi_{\mathrm{col},1}}{r_t^2}\,\xi_1 + \frac{\xi_{\mathrm{col},2}}{r_n^2}\,\xi_2 \; - 1 = 0 \tag{3.2.5}$$

4. **Remove.** All the obstacle points located on the illegal side of the new half-plane are removed, as they are already taken care of. The computation can thus be speeded-up by checking shorter lists during the **collide**-step.

5. **Check range.** If none of the half-axis has reached the maximal range value, go back to **inflate**. If one of them did, go back to **inflate** but only expand the other half-axis. If both of them have reached the maximal value, the ellipse results in the range circle, the procedure is interrupted and the half-plane parameters are returned.

A video showing the evolution of an exemplary fit can be found `here`[2] and more details can be obtained in [30]. Figure 14 shows the final results around four exemplary points on the path. It clearly outlines the significant improvement of the fitted surface, compared to a disk approximation of the free space. It does not only include larger portions of the path, but also of its surroundings. This results in a major advantage when it comes to avoiding dynamic obstacles, as shown in Part II. Furthermore, one can notice, that depending on the case, a different number of half-plane constraints are imposed. However, the code is strictly the same for all of them, as the method figures out by itself how many straight lines have to be fitted, based on the number of collision points. As depicted in Figure 14, a margin is introduced to take the rover dimensions into account.

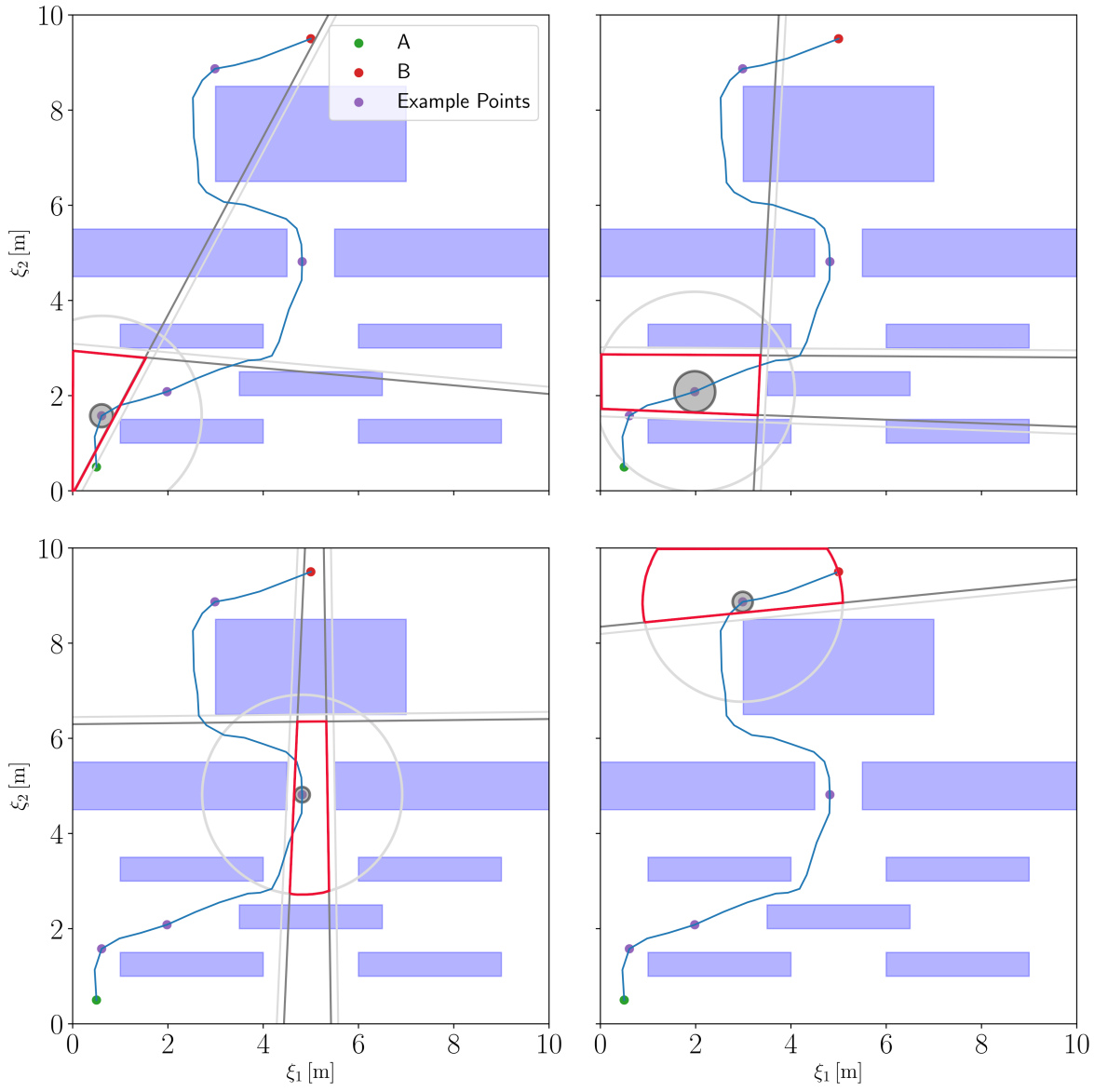


Figure 14: Comparison of disk and safe-corridor approximation of the free space. The dark grey circles show the resulting regions for the disk approach. The light and dark grey lines respectively represent the straight lines of the half-plane constraints before and after the application of margins. The safe-corridors are displayed in red.

[2] `https://www.youtube.com/watch?v=MmsXQUgwY2g&list=PLTKb-pD4cu3iyjE1YiPTW3SlWxkP4BHzQ&index=2`

Such a convex approximation has a further advantage than merely simplifying the optimization process. In fact, if the ellipse cannot be expanded at initialization, the environment is likely to have changed enough to require a new path planning, e.g. if the rover has explored new regions. An error message can be output and results in a communication interface from the local to the global planner, used whenever the local planner needs a new reference $\mathcal{P}$.

Throughout the past sections, all the necessary pre-requisites for SC-MPFC have been introduced. Its full formulation and implementation is summarized in the next section.

3.3 Implementation and Results

In this section, the problem formulation written out in Equation 1.4.13 is adapted to the case study. EBA and ALP are applied before the iterative optimization procedure. SCM is computed for each step, resulting in a time-variant state constraint $\tilde{x}_k \in \mathcal{D}_{\overline{k}}$, which is however fixed over a single optimization. This general strategy is depicted in Figure 15. The augmented dynamic, the associated costs and the resulting OCP to be solved iteratively yield:

$$\text{Dynamics:} \quad \begin{cases} \boldsymbol{z}_{k+1} = \begin{bmatrix} \boldsymbol{x}_{k+1} \\ \theta_{k+1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k) \\ \theta_k + \mu\, \delta_t \end{bmatrix} = \boldsymbol{f}_z(\boldsymbol{z}_k, \boldsymbol{\nu}_k) \\ \boldsymbol{y}_k = \begin{bmatrix} x_1(t_k), & x_2(t_k), & \theta_k \end{bmatrix}^{\mathrm{T}} \\ \boldsymbol{e}_k = \begin{bmatrix} x_1(t_k) - \rho_1(\theta_k), & x_2(t_k) - \rho_2(\theta_k), & \theta_k \end{bmatrix}^{\mathrm{T}} \end{cases} \tag{3.3.1}$$

$$\text{Costs:} \quad \begin{cases} \mathbf{Q}_z &= \operatorname{diag}\big(q_1,\, q_2,\, q_3\big) \\ \mathbf{R}_{\Delta\nu} &= \operatorname{diag}\big(r_1,\, r_2,\, r_3\big) \\ I_z &= \frac{1}{2} \sum_{k=\overline{k}}^{\overline{k}+H_p} \Big(\boldsymbol{e}_z^{\mathrm{T}}(t_k)\, \mathbf{Q}_z\, \boldsymbol{e}_z(t_k) + \Delta\boldsymbol{\nu}^{\mathrm{T}}(t_k)\, \mathbf{R}_{\Delta\nu}\, \Delta\boldsymbol{\nu}(t_k) \Big) \end{cases} \tag{3.3.2}$$

$$\text{OCP:} \quad \begin{cases} \tilde{\mathbf{Z}},\, \tilde{\boldsymbol{\nu}} &= \underset{\mathbf{Z},\, \boldsymbol{\nu}}{\arg\min}\, I_z \\ \text{subject to:} \quad \tilde{\boldsymbol{z}}_{k+1} &= \boldsymbol{f}_z(\tilde{\boldsymbol{z}}_k, \tilde{\boldsymbol{\nu}}_k) \\ \tilde{\boldsymbol{x}}_k &\in \mathcal{D}_{\overline{k}} \\ \tilde{\theta}_k &\in [-l_P, 0] \\ \tilde{\boldsymbol{u}}_k &\in \mathcal{U} = \{u_1, u_2 \in \mathbb{R} : ||u_1||, ||u_2|| < M_{\max}\} \\ \tilde{\mu}_k &\in [-v_{\max},\, v_{\max}]. \end{cases} \tag{3.3.3}$$

Remark: Neither terminal cost nor constraints were given. This choice is explained in section 3.4.

The coefficients of the weighting matrices are given in Table 3.1 and the implementation of SC-MPFC is now explained. To compute the Jacobians and Hessians needed for the optimization, the CasADi-library is used [33]. It offers a relatively high-level interface, where the problem can be formulated and the corresponding derivatives are automatically computed by the use of computational graphs, particularly well suited for such algorithmic differentiation tasks.

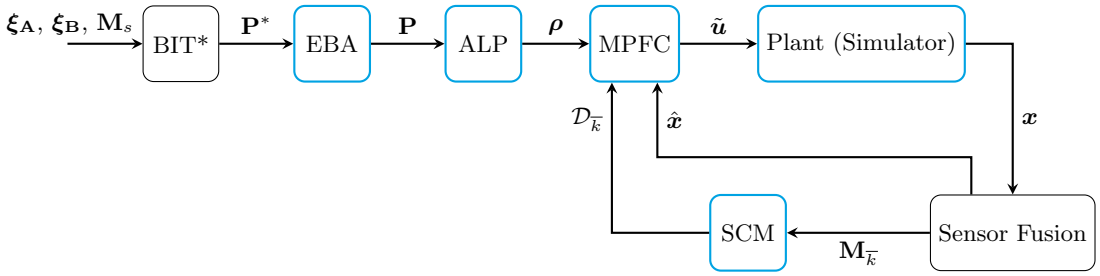


Figure 15: Control strategy on static environments. The blue blocks are self-implemented.

q_1	q_2	q_3	r_1	r_2	r_3	δ_t	$\mathrm{T_{CPU}}$	$\mathrm{T_{pred}}$	H_p
10^6	$2 \cdot 10^8$	$2 \cdot 10^8$	10^5	10^5	10^5	0.3 s	$0.9\,\delta_t$	2.1 s	7

Table 3.1: Coefficients of the cost functional I_z and numerical values of the implementation.

The optimization problem and the corresponding derivatives are finally passed to IpOpt, an open-source interior-point optimizer, well suited for nonlinear optimization tasks [16]. As such an optimizer needs to solve linear equation systems, the MA-27 solver of the HSL-library is additionally supplied [34], yielding faster results than the default solver. To ensure the real-time functionality of the system, the computation time is constrained to be smaller than $\mathrm{T_{CPU}} = 0.9\,\delta_\mathrm{t}$. The output of a control update is thus ensured for each time step. This can lead to a further sub-optimality, as the optimization might break up. Even for active time constraints, the results remained valid for any of the presented cases. This might be explained by the fact that the next local minimum is not exactly found but its vicinity is already reached, thanks to the rapidity of gradient-based approaches and to the use of a warm start for the optimization.

Even though the NMPC delivers an optimal solution, it can still be tricky to define a well-suited cost functional. In other words, minimizing the costs can still lead to bad performance, if they are not well defined. Tuning the coefficients of $\mathbf{Q}_z$ and $\mathbf{R}_{\Delta\nu}$ on a single example might have devastating effects: one might overfit the parameters, hence leading to a great performance on the test scenario on one hand, but possibly to a disastrous one on other cases. To avoid that, several examples, presenting different challenges, are introduced on a $10\,\mathrm{m} \times 10\,\mathrm{m}$ map:

- **The H terrain.** The static OGM resembles an H and the purpose is to drive the rover from the lower part of the H to its upper part. This results in manoeuvring the robot from a particularly non-convex region to another particularly non-convex one. This is normally not well handled by a stand-alone NMPC because of the pronounced local minima w.r.t. $||\boldsymbol{\xi}_m - \boldsymbol{\xi}_\mathrm{B}||_2$.
- **The indoor terrain.** This example was used until now to illustrate the path parametrization and improvement. It mainly represents a set of medium size obstacles, a narrow passage and a relatively big obstacle. They represent common shapes of indoor environments, such as tables, shelves or doors.
- **The maze terrain.** Mazes present two main challenges: their complex geometry makes it tough to find a path linking A and B and their narrowness is a challenge for the control procedure, as any significant deviation of the path might result in a collision.
- **The outdoor terrain.** In this case, rectangular obstacles of random size and position are

spawned. This stochastic example is introduced, to check the performance on an environment that was not designed by hand.

These environments, the corresponding paths to follow and the spatial results of the MPFC are depicted in Figure 16, showing that the method seems to be well suited for all kind of environments. In [23], it is pointed out that, in the case of planar rovers, a disk approximation of the free space is better suited for rover motion than hyperplane, log-barrier and exact-distance constraints. Therefore, SC-MPFC is now compared to MPFC with disk convexification. Videos showing the results of both approaches can be found `here`[3].

Measures are now introduced to allow quantitative comparisons. To study how fast the control is operating, the time t_{B}, needed to reach the vicinity of the goal point B, is introduced. As expressed in Equation 3.3.4, the vicinity is defined as the region where the euclidean distance to the target is less than one centimetre. In order to quantify the path-tracking error w.r.t. $\boldsymbol{\rho}$, a root mean square error RMSE_P is introduced in Equation 3.3.5. The error $e_\perp$ gives the normal distance to the path. It is averaged over the number of time steps N_{B}, needed to enter the vicinity of B.

$$t_{\mathrm{B}} = t_k \,:\, ||\xi(t_k) - \xi_{\mathrm{B}}||_2 \leq 0.01\,\mathrm{m} \quad \wedge \quad ||\boldsymbol{\xi}(t_{k-1}) - \boldsymbol{\xi}_{\mathrm{B}}||_2 > 0.01\,\mathrm{m} \tag{3.3.4}$$

$$\mathrm{RMSE}_P = \frac{1}{N_{\mathrm{B}}} \sum_{k=1}^{N_B} e_\perp(t_k), \quad \text{with:} \quad e_\perp = \underset{\boldsymbol{\xi}_{\mathcal{P}} \in \mathcal{P}}{\arg\min} \|\boldsymbol{\xi}_m - \boldsymbol{\xi}_{\mathcal{P}}\|_2 \tag{3.3.5}$$

	$t_{\mathrm{B}}^{\mathrm{Disk}}$ [s]	$t_{\mathrm{B}}^{\mathrm{SCM}}$ [s]	$\mathrm{RMSE}_P^{\mathrm{Disk}}$ [m]	$\mathrm{RMSE}_P^{\mathrm{SCM}}$ [m]
H	39.3	15.6	0.015	0.024
Indoor	39.3	17.4	0.017	0.037
Maze	224.1	36.6	0.010	0.036
Outdoor	36.3	16.8	0.016	0.031

Table 3.2: Comparison of performance measures for circular and safe-corridor fit of $\mathcal{C}_0$.

Intuitively speaking, these two performance measures quantify the path-following precision and the speed of the controlled rover. For the disk and SCM cases, these values are compared in Table 3.2. Using SCM speeds up the control by a factor 2 upto a factor 6 for the present cases. As can be seen in Figure 16 and in the videos, this effect is larger for paths that are on average closer to obstacles. The 3.2 cm mean deviation in the case of SC-MPFC remains very small compared to typical path lengths $l_P > 10\,\mathrm{m}$, to the half rover axis $l_{\chi_1} = 9.85\,\mathrm{cm}$, $l_{\chi_2} = 11.4\,\mathrm{cm}$ and to the error of the localization procedure. The best state of the art SLAMs typically make an error $e_{\mathrm{SLAM}} \in [0.001\,l_P,\ 0.01\,l_P]$ on the position estimate. The developed control thus yields a precision in the same, or even in a lower order of magnitude. In comparison to the disk approach, SCM approach leads to a negligible loss of precision and to a major improvement in time performance.

To visualize the transient behavior of the MPFC, the important variables are plotted in Figure 17. This is displayed for the indoor example. The corresponding figures for the H, the maze and the outdoor terrains are provided in section C of the appendix. It appears that the allowed range of the input variables μ, u_1, u_2 is far more used in the case of SC-MPFC. This leads the speed x_4 to be higher. However, because of the path-following formulation, the pose states θ, x_1, x_2, x_3 do not experience such

[3] `https://www.youtube.com/watch?v=FODWnKoGE_U&list=PLTKb-pD4cu3iyjE1YiPTW3SlWxkP4BHzQ&index=3`

a stretch in their magnitude. They go through the very same values than for the disk approach, which are however reached significantly faster, thanks to the higher input values.

Remark 1: Path-tracking precision can be prioritized over speed by simply setting q_1, q_2 higher. Achieving a better value of RMSE_P, therefore, results in a very easy task. This was not done for the present case as the loss of precision turned out to be an advantage. In fact, SC-MPFC recognizes and exploits shortcut and smoothing opportunities, as shown in Figure 16.

Remark 2: Optimization and simulation are dealt with by two different procedures. While the optimization relies on the symbolic RK4 given by CasADi, the simulator relies on SciPy's DOP853 initial value solver, with time step control.

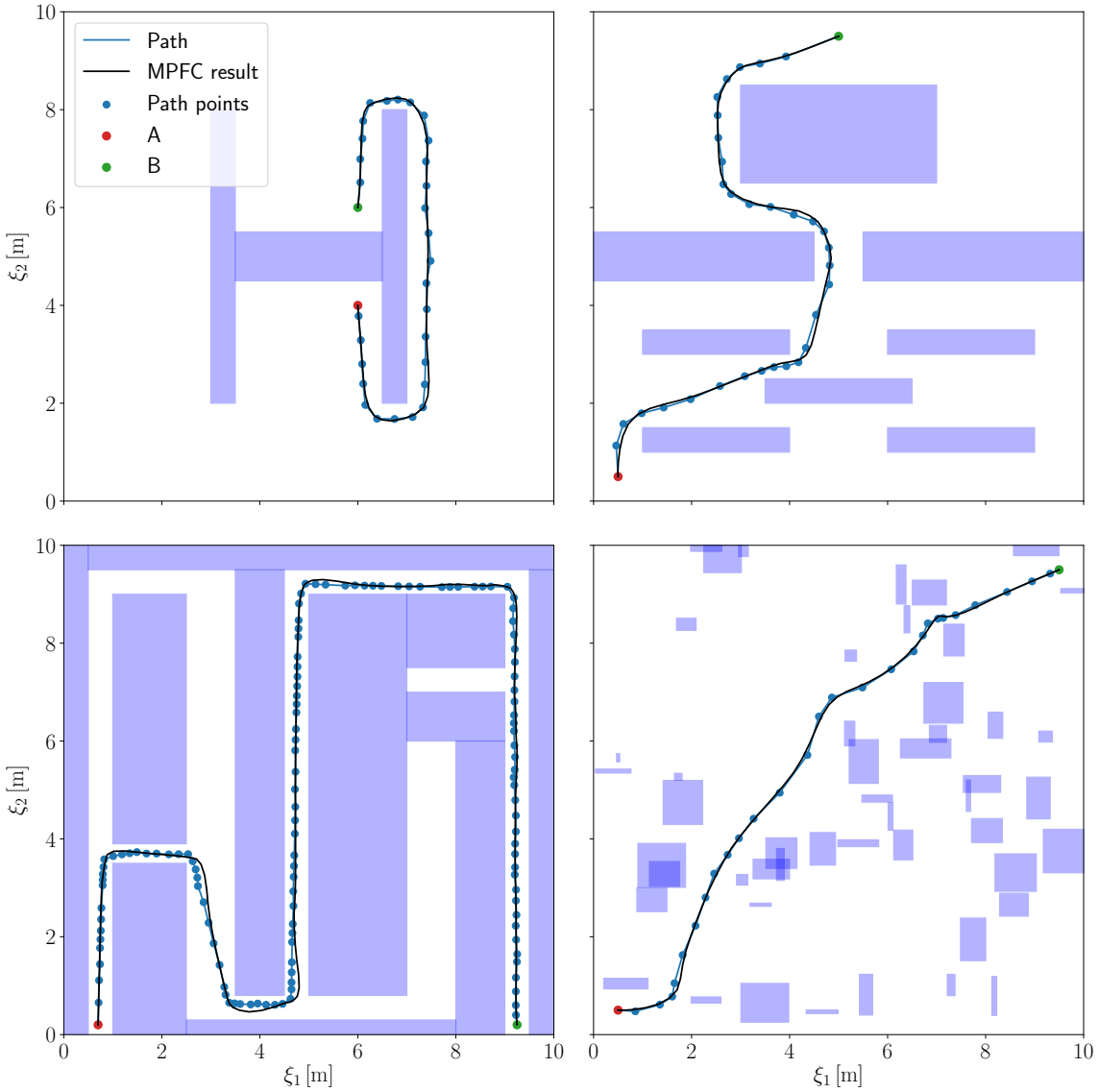


Figure 16: Spatial visualization of the MPFC results on the different test cases. F.l.t.r., f.t.t.b.: the H, the indoor, the maze and the outdoor example. The legend of the upper-left corner holds for all.

The computation time analysis of the method is provided in section D of the appendix and shows that, besides delivering good results, SC-MPFC is also able to respond to the real-time constraints. The implementation on the real robot is therefore only a small step away. In the next section, its reliability is further investigated by studying its stability and its robustness.

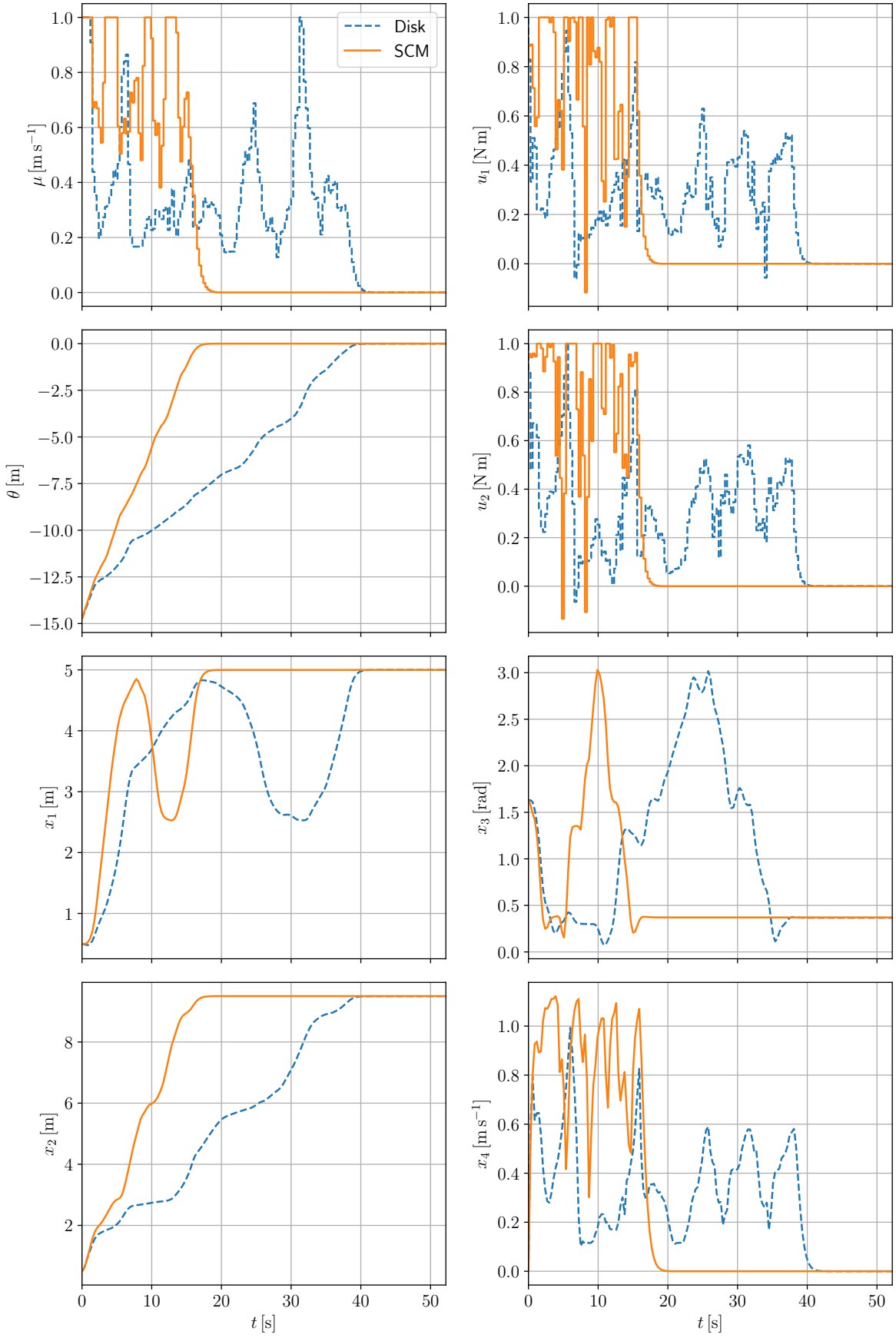


Figure 17: Transient behavior of the MPFC on the indoor example. Two different approaches for the approximation of $\mathcal{C}_0$ are compared: the disk and the safe-corridor method.

3.4 A Brief Discussion about Stability and Robustness

Stability is a hot topic of NMPC, as this method has been used for practical applications long before stability proofs were published. In 1987, the first proof using Lyapunov theory for linear cases was presented in [35]. For nonlinear cases, this task turns out to be even more challenging. Works providing stability theorems, such as [36], can be a help in this non-trivial quest. However, these cannot be straightforwardly applied in the present case. This is due to the fact, that the current control procedure presents even more difficulties than a usual NMPC:

- The state constraints $\mathcal{D}_k$ are time-variant.
- The reference $\boldsymbol{\rho}(\theta_k)$ and the resulting cost function are non-convex. In fact, they contain piecewise linear functions of θ, leading to piecewise convex cost regions but not to a globally convex cost.
- In the next part of this work, dynamic obstacles are also handled and lead to an even more complex problem, due to a further time-variant component.

The present work tackled the stability question rather by the simulation study shown in the past section. As the procedure seems to perfectly work for different environments and arbitrary path shapes, one can expect it to be relatively reliable in real applications. Regarding the fact that the ROSBot 2.0 can do no harm to a human, nor cause material damages other than to itself, implementing the procedure on the real system represents no danger.

Remark: The stability proof normally motivates the choice of the terminal constraints and the terminal costs. As this was not achieved here, they only represent further degrees of freedom to tune. In order to simplify the application, they were consequently not used.

A further important closed-loop property is its robustness. For linear systems, this can be easily studied and formally enforced. The nonlinear world however again turns out to be a capricious realm, where such property is mostly studied in simulation. This is now done for the present case by introducing two disturbances:

1. A parametric disturbance is applied to the system. Two different cases are introduced. In the first one, the inertial parameters p_1 and p_2, as well as the friction parameters p_3 and p_4, are increased by 30% of their initial value. In the second case, they are decreased by 30%. The resulting disturbance factor is denoted by $c_p = \{0.7,\ 1.3\}$. One could also imagine a combination of both, increasing some parameters while decreasing others. However, the two presented cases turn out to be the most critical ones, as one can imagine by a simple mechanical understanding of the system. The parametric changes are only made in the simulator. The MPFC hereby still assumes the parameters to yield their original value.
2. The initial condition is chosen, such that the original position is 20 cm away from the path start point A. To make it even more challenging, the orientation at $t = 0$ is chosen as $\phi_0 = \phi_{\mathrm{A},0} + \pi$. Hereby, the best possible orientation in A is denoted by $\phi_{\mathrm{A},0}$, computed by the knowledge of the next path point. In the disturbance case, the initial angle is turned by 180°and thus could not be worse.

Remark: As a good state estimator filters out the measurement noise, no such disturbance was applied during the current analysis.

The spatial results for the indoor case are depicted in Figure 18. Analogous results on the other environments are shown in section C of the appendix. The measures t_{B} and RMSE_P are displayed and show that the procedure handles significant disturbances without any great loss of performance

or precision, regardless of the environment. Videos illustrating the behavior of SC-MPFC for the two different parametric disturbances can be found `here`[4].

To obtain an even more robust control, many options are possible: a min-max optimization [11], a robust tube control [11], a multi-stage approach [37] or an unscented transformation [9] could be great tracks to improve SC-MPFC.

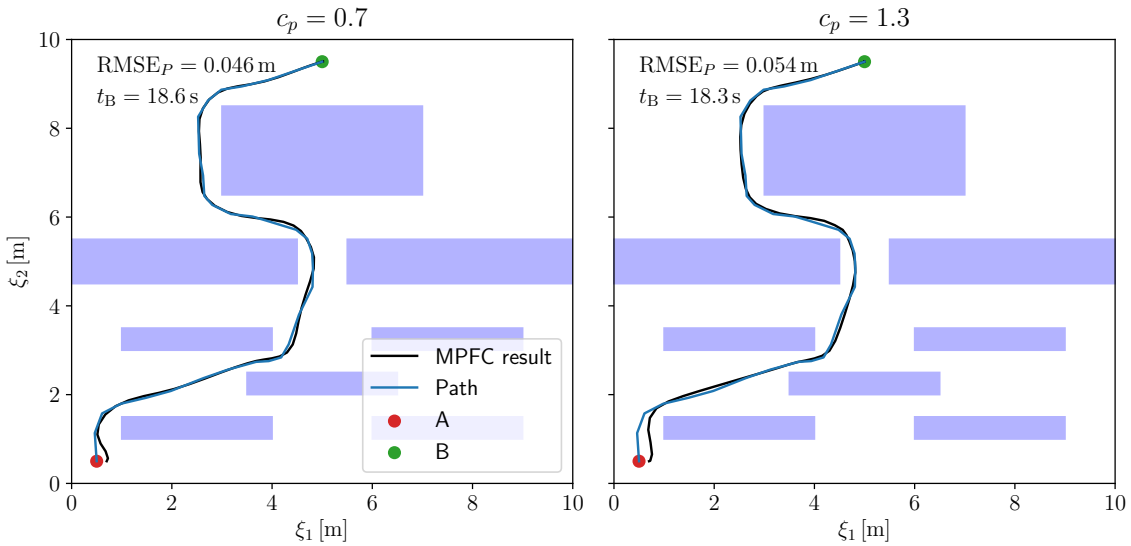


Figure 18: Spatial visualization and performance measures of the MPFC for the two parametric disturbance cases on the indoor example. In both cases, the original position is perturbed by 20 cm.

Conclusions on the Implementation

In summary, SC-MPFC seems to fulfil the requirements:

- It performs well on diverse environments with a single set of parameters.
- It yields an acceptable tracking error.
- It represents a major speed improvement compared to simpler constraint-convexifications.
- It is already real-time efficient.
- It seems to be relatively robust w.r.t. state and parametric disturbances.

Even though its stability could not be proven, the simulations are a great hint to believe that it is at least stable for very different application cases. A question now arises: what improvements have been achieved, compared to the state of the art? Examples of real MPFC applications can be found in [24, 25, 26, 27]. An important thing to notice is that none of those deal with non-convex state constraints, and mostly not even with state constraints at all. The present work tackles this problem for arbitrarily shaped environments and additionally presents a path improvement and a systematic parametrization with a physical meaning. These are already significant steps taken towards a more versatile and reliable MPFC.

In the next part, this is driven even further. The ability of SC-MPFC to deal with dynamic environments is developed and presents a novel achievement.

[4] `https://www.youtube.com/watch?v=BZUMCOcBgEI&list=PLTKb-pD4cu3iyjE1YiPTW3SlWxkP4BHzQ&index=7`

Part II

Environment Motion Predicting Path-Following NMPC

Chapter 4

Artificial Neural Networks – State of the Art

Artificial neural networks (NNs) denote trainable computing systems, characterised by their ability to scale better than any other machine learning technique to this date [38]. *Scaling* means that with an increasing amount of data, a NN can still improve its performance, while other powerful methods such as support vector machine tend to achieve a certain plateau. This highly desired behavior is due to the fact that the depth, and thus the complexity of a NN, can easily be increased, allowing it to learn more complex relations. The versatility of this technique makes it very appealing, as it over-performs some algorithms specifically designed for certain tasks. Pattern recognition, computer vision, time series prediction and natural language processing present the most remarkable examples for the great power of NNs. Related works can be found in [39, 40, 41].

Recent research allowed NNs to predict the future motion of complex dynamic environments [42, 43]. As forecasting the evolution of an OGM is nothing else than predicting the next frames of a video, network architectures for computer vision and for time series prediction are therefore combined. Since their first practical achievements in the 90s [44], convolutional NNs (CNNs) became one of the most common and powerful techniques for learning tasks in computer vision. Around the same time, recurrent NNs (RNNs) have experienced a major step forward by the introduction of the long short term memory (LSTM) cell in [45]. As machine learning tasks are getting more complex, it is no surprise that the combination of both arose in literature by the introduction of the ConvLSTM cell, presented in [46]. This was first done to forecast the short-term weather evolution and the idea was re-taken in [43] to predict the future motion of dynamic obstacles at a street crossing. In order to understand the function of ConvLSTM cells in section 4.3, several concepts need to be explained first. Fully connected NNs (FCNNs) and CNNs are therefore respectively introduced in section 4.1 and section 4.2.

A Brief Introduction in the Realm of Tensors

Before diving into specific architectures of NNs, understanding tensor operations can be a great way to simplify the notation and win generality. A tensor describes a multilinear relation between vector spaces and can be represented by a multi-dimensional array (MDA). The tensor $\mathbf{A}$ is a coordinate invariant quantity, whereas its component array $[\mathbf{A}]$ depends on the chosen coordinates. As for deep learning tasks, no change of coordinates is usually needed, the separation between a tensor and its representation tends to blur. The space of all tensors of order $m \in \mathbb{N}$, with dimensions $n_1, n_2, ..., n_m \in \mathbb{N}$, is denoted by $\mathcal{T}^m_{n_1,n_2,...,n_m}$. If the dimensions are not specified yet, it is simply written as $\mathcal{T}^m$. The entries of the component array are denoted by $A_{i_1,i_2,...,i_m}$, with $i_j \in \{1, ..., n_j\}$ and $j \in \{1, ..., m\}$. As an example,

the tensor $\mathbf{A} \in \mathcal{T}^3_{2,5,7}$ can be represented by $[\mathbf{A}] \in \mathbb{R}^{2\times5\times7}$ in a given coordinate system.

The contraction of an $(m+1)$-th order tensor $\mathbf{W} \in \mathcal{T}^{m+1}$ with an m-th order tensor $\mathbf{A} \in \mathcal{T}^m$ is defined by Equation 4.0.1. Einstein's sum convention is used w.r.t. the bond index i_{m+1}. The remaining indices are running indices. The result is a tensor $\mathbf{Z} \in \mathcal{T}^m$ and the relation can be written in a compact way as in Equation 4.0.2. For $m = 1$, this results in a simple vector-matrix multiplication. The tensor contraction can therefore be understood as a generalization of the matrix-vector product to higher dimensional arrays.

$$Z_{i_1\, i_2\, ...i_m} = W_{i_1\, i_2\, ...i_{m+1}} A_{i_{m+1}...i_{2m}} \tag{4.0.1}$$

$$\mathbf{Z} = \mathbf{W} \cdot \mathbf{A} \tag{4.0.2}$$

To obtain an affine relation, a bias tensor $\mathbf{B}$ of order m can be additionally introduced as in Equation 4.0.3. The tensor addition is defined element-wise as shown in Equation 4.0.4.

$$\mathbf{Z} = \mathbf{W} \cdot \mathbf{A} + \mathbf{B}, \quad \mathbf{Z}, \mathbf{A}, \mathbf{B} \in \mathcal{T}^m, \quad \mathbf{W} \in \mathcal{T}^{m+1} \tag{4.0.3}$$

$$Z_{i_1\, i_2\, ...i_m} = W_{i_1\, i_2\, ...i_{m+1}} A_{i_{m+1}...i_{2m}} + B_{i_1\, i_2\, ...i_m} \tag{4.0.4}$$

4.1 Fully Connected Neural Networks

Even though FCNNs are not used during this thesis, understanding their functionality is crucial to frame the basic principles, also holding for all further architectures. The idea of a single-layered FCNNs is to take an input data tensor $\mathbf{X}$ and forward-propagate it through an affine transformation, determined by a weight tensor $\mathbf{W}$, a bias tensor $\mathbf{B}$ and outputting $\mathbf{Z}$. A nonlinear activation function $\mathbf{\Psi}$ is then applied element-wise on $\mathbf{Z}$. Common activation functions are the sigmoid, the soft-max, the tanh and the rectified linear unit (ReLU). The output of such a computation is called the activation $\mathbf{A}$. Weights and biases are commonly denoted by the term *parameters*.

For an FCNN with L layers, this computation is sequentially repeated and thus determined by the layer-specific $\mathbf{W}^{(l)}$, $\mathbf{B}^{(l)}$ and $\mathbf{\Psi}^{(l)}$, with $l \in \{1, 2, .., L-1\}$. The computations performed are summarized in Equation 4.1.1 and a corresponding visualization is given in Figure 19. If $l = 0$, the activation $\mathbf{A}^{(0)}$ results in being the input $\mathbf{X}$. For the last layer $l = L$, the activation results in being the output $\mathbf{A}^{(L)} = \hat{\mathbf{Y}}$ of the NN. For any other layer, $\mathbf{A}^{(l)}$ corresponds to so-called *hidden states* $\mathbf{H}^{(l)}$. The tensor dimensions from a layer to another can vary, whereas the tensor order remains constant over the whole FCNN.

$$\mathbf{Z}^{(l)} = \mathbf{W}^{(l)} \cdot \mathbf{A}^{(l)} + \mathbf{B}^{(l)} \tag{4.1.1a}$$

$$\mathbf{A}^{(l+1)} = \mathbf{\Psi}^{(l)}\left(\mathbf{Z}^{(l)}\right) \quad \forall\, l \in \{1,\, 2,\, ...,\, L-1\} \tag{4.1.1b}$$

In supervised learning, a given input data $\mathbf{X}$ implies a ground truth $\mathbf{Y}$. For instance, let us illustrate this statement by a binary image classification task. If a cat picture is given as input data, the label $\mathbf{Y}$ would be "this is a cat". The aim of the training process consists in adapting the NN parameters, such that the output tensor $\hat{\mathbf{Y}}$ delivers a good approximation of the ground truth $\mathbf{Y}$. This problem can be formulated as minimizing an error. The scalar-valued loss function $\Gamma(\mathbf{Y}, \hat{\mathbf{Y}})$ measures the discrepancies between both tensors and is therefore used as the objective of the minimization problem.

An infinite number of possibilities is available for the choice of Γ. Based on information theory, some of them are however proved to be better suited, as e.g. the cross-entropy. This loss is used throughout this thesis and can be expressed, by the use of the sum convention:

$$\Gamma(\mathbf{Y}, \hat{\mathbf{Y}}) = -Y_{i_1,\ldots,i_m} \log \hat{Y}_{i_1,\ldots,i_m}. \tag{4.1.2}$$

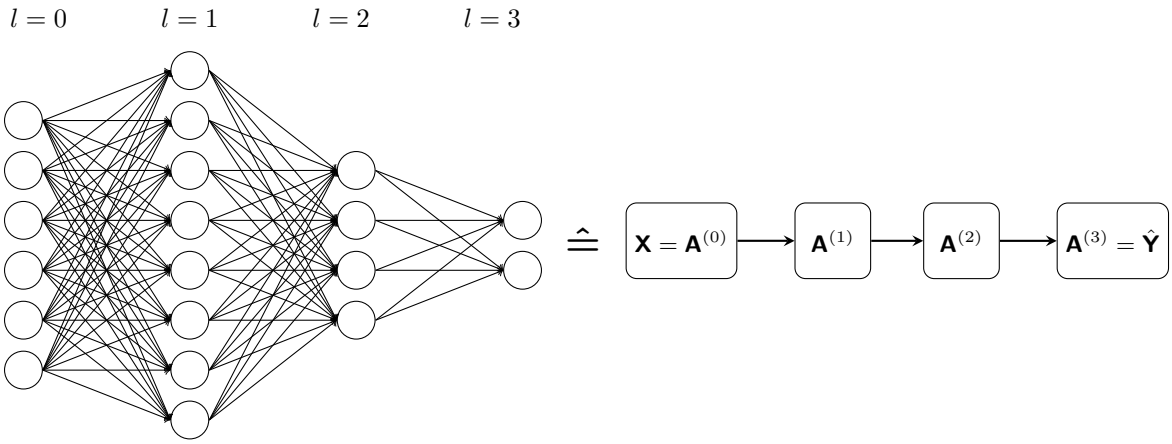


Figure 19: Example of a FCNN with $L = 3$.

The training process aims to minimize $\Gamma(\mathbf{Y}, \hat{\mathbf{Y}})$ by back-propagating the loss through the network and compute updates of the parameters in accordance to it. By the use of the chain rule, the gradients $\boldsymbol{\nabla}_{\mathbf{W}}\Gamma$ and $\boldsymbol{\nabla}_{\mathbf{B}}\Gamma$ can be computed, as shown in [38]. They give the direction of the steepest increase of $\Gamma(\mathbf{Y}, \hat{\mathbf{Y}})$ w.r.t. $\mathbf{W}$ and $\mathbf{B}$. The gradient descent algorithm, aiming to minimize Γ, thus takes the opposite direction by subtracting the gradient during the parameter update process, as shown in Equation 4.1.3 and Equation 4.1.4. To control the optimization procedure, the gradient is scaled by a learning rate α [47]. It is a non-learnable parameter, mostly chosen based on heuristics. Such quantities are denoted as *hyper-parameters*, in contrast to the trainable weights and biases.

$$\mathbf{W} \leftarrow \mathbf{W} - \alpha \boldsymbol{\nabla}_{\mathbf{W}}\Gamma \tag{4.1.3}$$

$$\mathbf{B} \leftarrow \mathbf{B} - \alpha \boldsymbol{\nabla}_{\mathbf{B}}\Gamma \tag{4.1.4}$$

The data available for training is commonly divided into three different sets: the train, the validation and the test set [48]. Parameter updates are repetitively performed using the train set, resulting in a decreasing training error Γ_{train}. Applying this procedure once on the whole train set is called *epoch*. The number of epochs used for training is denoted by n_e. After each epoch, the validation loss Γ_{val} is evaluated. As the network never used this data to update the parameters, it gives a measure for the generalization error. The optimal set of parameters is given by the minimal generalization error. The performance is finally estimated on the test set, which has neither been used for training, nor for the determination of the optimal training time. This results in a more rigorous measure of the network performance.

An intuitive training approach is to feed random single data-label pairs one after another into the network and compute updates for each of them. This results in the so-called *stochastic gradient descent* (SGD). Instead of this, a bundle of random $\{\mathbf{X}, \mathbf{Y}\}$-pairs can be drawn and used for the update procedure, resulting in mini-batch SGD. This presents two major advantages [47]. Firstly, the use of batches allows vectorization during the training process and thus reduces the time needed for each epoch. Secondly, the cost decrease is less noisy, as the obtained gradient is averaged over the batch. This can be of major importance in the vicinity of the global minimum. One could think of pushing the idea of using batches to the opposite extreme and feed the whole data into the network

for each parameter update. This results in the so-called *batch gradient descent* (BGD). However, the mini-batch approach presents three advantages [47]. On one hand, evaluating the totality of the $\{\mathbf{X}, \mathbf{Y}\}$-pairs would lead to high computation time for a single update of the weights. This makes the monitoring process slower and is undesired. On the other hand, the hardware underlies memory constraints and evaluating the whole data set in a single step might simply be infeasible. Finally, the slight noisiness of mini-batch SGD compared to BGD eases leaving saddle points of the loss function.

Remark: Let us assume, that the input of a NN is a tensor of arbitrary order m. In the case of mini-batch SGD, a stack of the data is performed, resulting in an input tensor of order $m+1$. For $m=1$, this means that the computations performed at training time can no longer be compactly described with simple matrix-vector multiplication, as the input itself is already a matrix. This motivates the use of the introduced tensor operations.

Further algorithms, more sophisticated than the gradient descent, can improve the optimization process. One of the most common ones is the ADAM-algorithm, which is used throughout the thesis with the set of hyperparameters presented in the original publication [49]. ADAM mainly consists of a modified gradient descent method, where a moving average filtering over the last updates is introduced.

To ease the training procedure, one can additionally apply a batch normalization (BN), as proposed in [50]. This allows normalizing the input of each layer, thus making the training procedure more robust and even faster. To achieve a better generalization of the NN, regularization techniques also have been developed. In the present thesis, dropout (DO) [51] is used as a regularization technique. It consists of randomly setting some weights to zero over a batch evaluation. Thus the network learns to not rely too strongly on few parameters but rather tries to equally improve them.

FCNNs have been successfully used for numerous applications and are capital to understand the more advanced deep learning concepts. They are however a pretty bad choice for a probabilistic prediction of future OGMs, which is a goal of this thesis. Two main problems are responsible for this: the curse of dimensionality and the vanishing gradient problem. They are respectively introduced in section 4.2 and section 4.3, where corresponding state-of-the-art solutions are also presented.

4.2 Convolutional Neural Networks

Even though FCNNs perform well on several tasks, the number of parameters grows exponentially with the size of the input data [38]. This curse of dimensionality would not only lead to longer training time but also to an enormous amount of data, needed to optimize the parameters. This is highly undesired and would particularly arise for image processing. In fact, pictures typically present thousands if not millions of pixel values, used as input of the NN.

This is why more appropriate architectures, scaling better with the input size, have been established. The most famous and common ones are convolutional NNs (CNNs), which are introduced below. Their key concept is to apply a convolution, denoted by "$*$", on the input, w.r.t. a so-called *kernel*. This allows to only sparsely connect the network. An exemplary convolution with input $[\mathbf{X}] \in \mathbb{R}^{2\times 4}$, kernel $[\mathbf{W}] \in \mathbb{R}^{2\times 2}$, stride $S=1$ and output $[\mathbf{A}] \in \mathbb{R}^{1\times 3}$ is depicted in Figure 20. In the case of a FCNN, the input matrix would have been flattened into a vector and densely connected to the next layer. For this example, the connectivity difference between FCNN and CNN is depicted in Figure 21. It appears that the number of parameters to optimize is way smaller in the case of a convolutional layer. As the convolution connects pixels that are near to each other, this operation can be understood as an efficient way of sparsely encoding of the local information.

A further advantage is that the convolution operation is translation-invariant, as it is shown in Figure 20. Thus, a feature can be extracted regardless of its position in the picture. This also leads to an invariance w.r.t. the input data shape, which is particularly important for image processing. In fact, resolution and picture size can dramatically vary from a picture to another. In comparison, a FCNN is very sensitive to translation, as a set of parameter is associated to each pixel. Depending on the pixel, the trained weights and biases can be very different. A simple translation of the feature within the picture can, therefore, result in a completely different output. Additionally, the nature of fully connected layers restrains the model to a fixed input data shape.

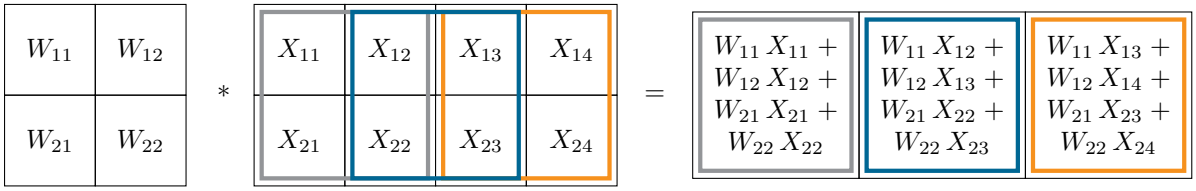


Figure 20: Exemplary convolution.

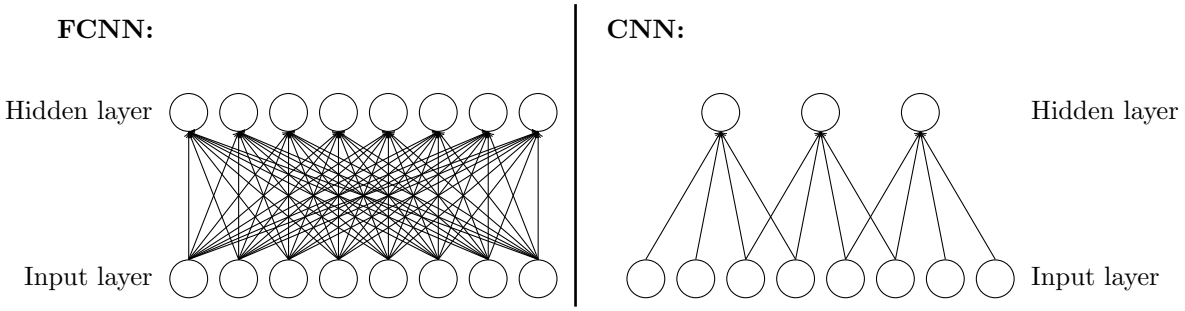


Figure 21: Connectivity comparison of a shallow FCNN and a shallow CNN.

As shown in Figure 20, the output dimension of a convolution is smaller than the input dimension. Additionally, the edges are barely taken into account if no further measures are taken. Both of these might be undesired effects. As a solution to this, a so-called *padding* is introduced [47]. It consists of artificially extending the input dimensions, as represented in Figure 22. A common way to do so is the zero-padding, extending the tensor by additional zero entries. The isomorphic zero-padding is the only one used throughout this thesis and consists of extending the input with zeros, such that the output dimension is the same as the input one.

In the field of computer vision, convolutions were used long before NNs blossomed. In fact, they allow filtering pictures by detecting edges, blurring, sharpening, contrasting, ... etc. the image. The underlying idea of CNNs is to learn the weights of the kernel instead of designing them by hand. Thus, a high number of degrees of freedom is available and allows the network to learn more complex relations. For each layer of convolution, several kernels are applied, leading to an output for each of them. The number of kernels used for one layer is denoted by n_K and the output consists of the n_K resulting MDAs. They are concatenated along the so-called *channel dimension*.

The pooling operation also plays an important role for CNNs, as it allows to shrink the matrix dimensions and reduce the feature space. A common variant is the max-pooling, represented in Figure 23. It mainly consists of applying a window with a certain stride and keeping only the maximum

value within this window. For Figure 23, a 2×2 window with a stride of 2 has been chosen. Only the feature with the highest numerical value is kept. This operation can thus be understood as only keeping the relevant information at encoding time. If another convolution is applied on top, the features extracted during max-pooling are combined and higher-level features arise.

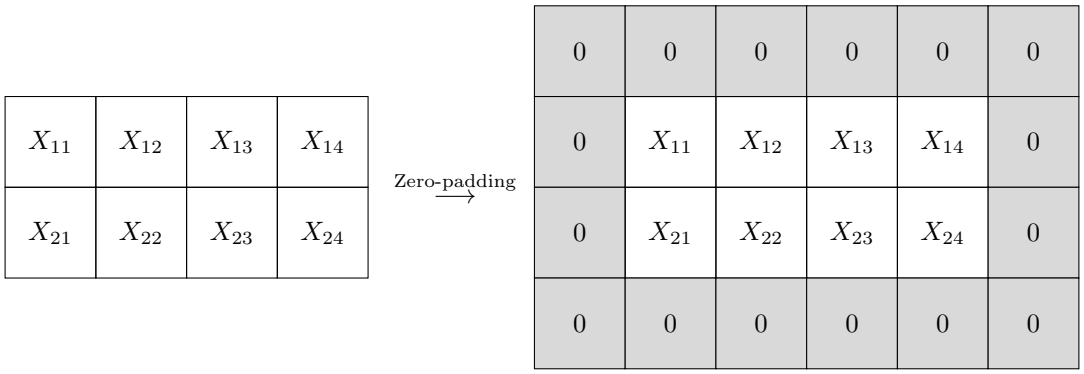


Figure 22: Exemplary zero-padding. For the output dimension to be consistent with the input dimensions, a convolution with a 3×3 kernel and $S = 1$ could be used.

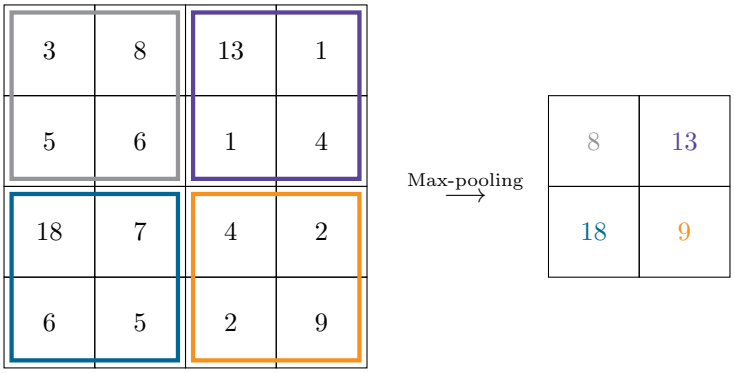


Figure 23: Illustration of the max-pooling operation.

Throughout the past and the current section, all the concepts were introduced to understand the architecture of most CNNs. They mainly consist of a succession of convolutions with padding, nonlinear activations and max-pooling. A bias can be additionally introduced to obtain further degrees of freedom. This is expressed by Equation 4.2.1, where L_{C} denotes the number of convolutional layers used. In Figure 24, an exemplary convolutional network with $L_{\mathrm{C}} = 2$ is represented. The repetitive convolution and pooling allow extracting more and more complex features over the depth. The highest level of encoding is commonly used by fully connected layers for classification tasks. However, this is by no mean a mandatory architecture, as it is shown in chapter 5.

$$\mathbf{A}^{(l+1)} = \mathrm{MaxPool}\left(\mathbf{\Psi}^{(l)}\left(\mathbf{W}^{(l)} * \mathbf{A}^{(l)} + \mathbf{B}^{(l)}\right)\right) \quad \forall\, l \in \{1,\, 2,\, ...,\, L_{\mathrm{C}} - 1\} \tag{4.2.1}$$

Even though the structure and connectivity differ between FCNNs and CNNs, the concept used at training time and introduced in section 4.1 remain the same. Batch-normalization, dropout, back-propagation and ADAM are applied accordingly to the structure of the convolution operation and appears to work similarly. As most of these operations are automatically processed by the majority of the deep learning programming frameworks, detailing them here would be of minor use.

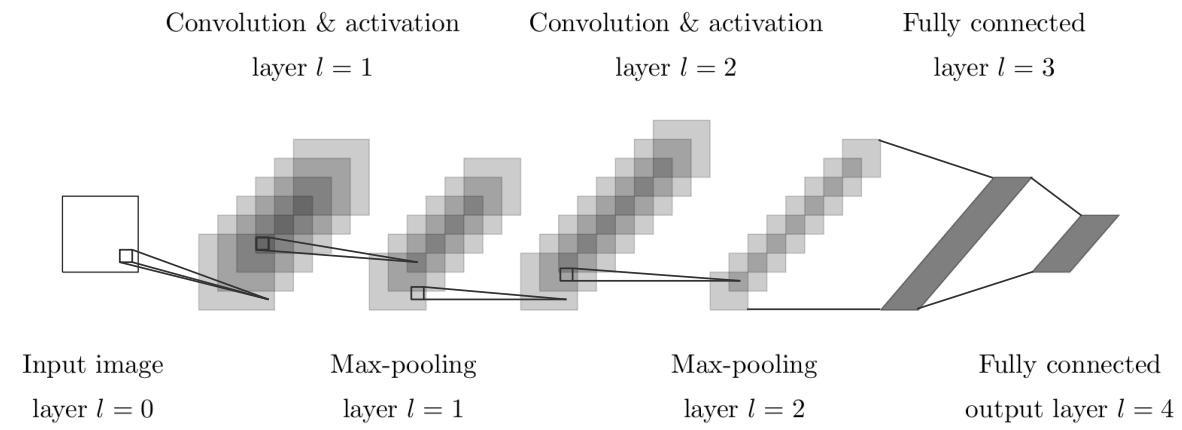


Figure 24: Exemplary structure of a typical CNN for image classification with $L = 4$ and $L_{\mathrm{C}} = 2$.

In the present section, all the deep learning tools needed for treating a single OGM have been introduced. However, considering sequences of OGMs requires the introduction of tools, specific for time series prediction tasks. They are therefore presented in the next section.

4.3 ConvLSTM – A Recurrent Convolution

RNNs constitute a wide class of models and are the default choice for many sequence-related tasks. In the present work, the focus is set on time series prediction, a subset of this problem class. The basic idea of RNNs is to feedback the output of a neuron into its input. Unwrapping this structure over time is represented in Figure 25 and can be interpreted as an integrating behavior, crucial for time series prediction. At time index τ, the input data tensor $\mathbf{X}_{(\tau)}$ and the hidden state of the previous time step $\mathbf{H}_{(\tau-1)}$ are fed into the neuron. Some computations (e.g. contractions or convolutions) are then internally performed, based on the trainable tensors $\mathbf{W}_x$ and $\mathbf{W}_h$. They respectively weight the input and the hidden state. The result $\mathbf{H}_{(\tau)}$ and the next input data tensor $\mathbf{X}_{(\tau+1)}$ are then fed into the same set of computation and lead to the obtention of $\mathbf{H}_{(\tau+1)}$.

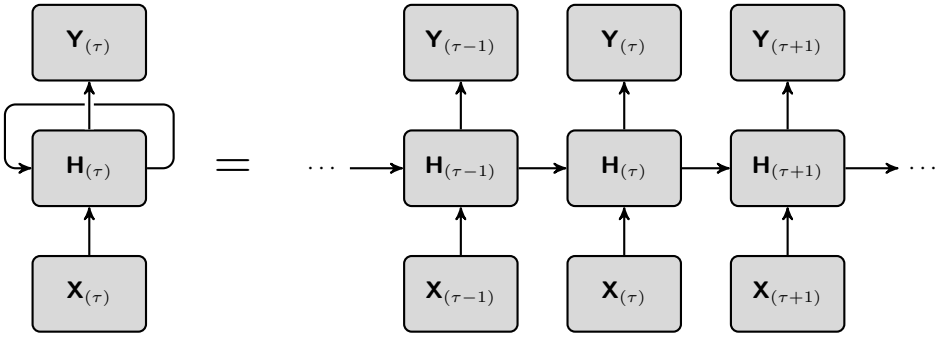


Figure 25: A single layer RNN, unwrapped over the time dimension.

It appears that the amount of tensor operations grows exponentially with τ. Thus, because the time dimension is involved, even a single layer RNN results in being a deep network. For the sake of simplicity in the upcoming explanations, the network depth is set to $L = 1$. The input tensor order is set to $m = 1$ and the weights $\mathbf{W}_x$ and $\mathbf{W}_h$ are consequently matrices. The inner cell operation is a contraction and the current output $\mathbf{Y}_{(\tau)}$ is given by Equation 4.3.1. Unwrapping this over time

results in Equation 4.3.2. Hereby, high powers of $\mathbf{W}_h$ appear, as $\tau \gg 1$. Assuming that $\mathbf{W}_h$ accepts an eigenvalue decomposition $\mathbf{W}_h = \mathbf{V} \cdot \mathbf{diag}(\boldsymbol{\lambda}) \cdot \mathbf{V}^{-1}$, one obtains Equation 4.3.3.

$$\mathbf{Y}_{(\tau)} = \mathbf{W}_h \cdot \mathbf{H}_{(\tau-1)} + \mathbf{W}_x \cdot \mathbf{X}_{(\tau)} \tag{4.3.1}$$

$$= \mathbf{W}_h^{\tau} \cdot \mathbf{H}_0 + \sum_{i=0}^{\tau-1} \mathbf{W}_h^i \cdot \mathbf{W}_x \cdot \mathbf{X}_{(\tau-i)} \tag{4.3.2}$$

$$\text{With:} \quad \mathbf{W}_h^i = \mathbf{V} \cdot \mathbf{diag}([\lambda_1^i, \, ..., \lambda_n^i]) \cdot \mathbf{V}^{-1} \tag{4.3.3}$$

Back-propagation over time is the key concept of the learning process and is very similar to the regular back-propagation. It allows to compute the gradients and thus the parameter update. Writing the forward-propagation over time is easier and sufficient to point out, that high powers of $\mathbf{W}_h$ will arise in the back-propagation computation. The critical consequences of such high powers in the gradients are now explained.

If $\lambda_j > 1$ for $j \in \{1, ..., n\}$, exploding values arise in $\mathbf{W}_h^i$, and thus in the gradient. This can be solved by introducing a saturation on the gradient values, resulting in a so-called *gradient clipping*. Eigenvalues $\lambda_j < 1$ are even more problematic, as $\lambda_j^i \simeq 0$ for $i \gg 1$. This results in a zero gradient, regardless of the computed loss. This gradient vanishing problem thus avoids the network to learn any long term dependencies. For the sake of simplicity, this was demonstrated for the contraction operation with $m = 1$ but can be similarly proved for higher orders and the convolution operation.

The vanishing gradient problem can be tackled and still leads to new research outcomes, as shown by the recent works on gated recurrent units and transformer models, respectively introduced in [52] and [53]. However, especially for time series prediction, the most common solution remains the LSTM-cell proposed in [45]. As depicted in Figure 26, its main idea is to introduce a cell state $\mathbf{C}$, which is fed back as an input of the recurrent cell for the next time step. This is a major difference compared to the basic RNN architecture and allows the cell to store information over time. Thanks to this additional degree of freedom, the vanishing gradient problem can be solved. In its classical formulation, the LSTM-cell takes vector-valued inputs and proceed to a set of computations, mostly relying on the contraction operation. However, a dense connection underlies LSTM-layers, which leads to several problems for image processing tasks, as pointed out in section 4.2.

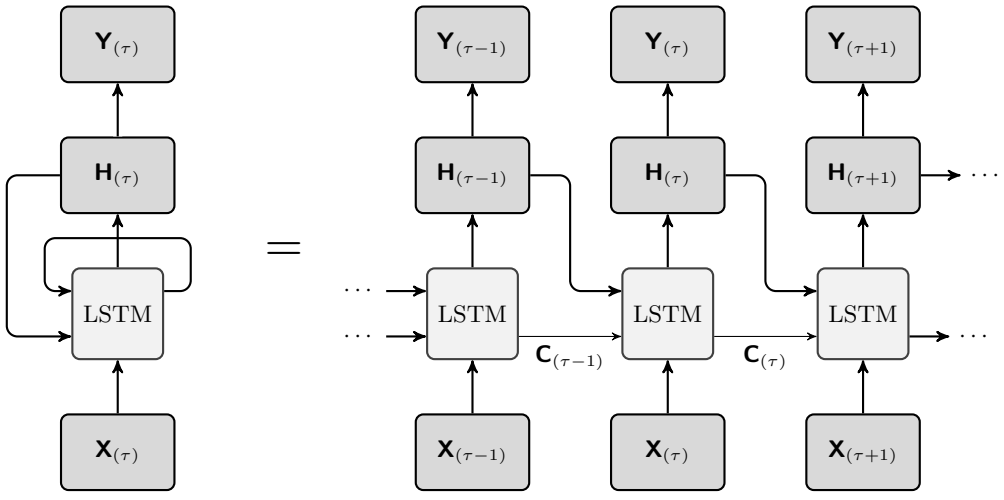


Figure 26: A single LSTM-layer, unwrapped over the time dimension.

For the current setting, a new approach to LSTM turns out to be favourable. The ConvLSTM-cell, introduced in [46], admits matrices (and even higher-order arrays) as input and relies on the convolution operation for the sake of sparsity. Its architecture is represented in Figure 27 and clearly outlines a more sophisticated algorithmic than the simple operation shown in Equation 4.3.1. The corresponding set of computation is listed in Equation 4.3.4 to Equation 4.3.9. The element-wise sigmoid and hyperbolic tangent functions are hereby denoted by $\boldsymbol{\sigma}$ and $\mathbf{tanh}$. If one replaces the convolution by a contraction and the matrix-valued states/inputs by vector-valued ones, the classical LSTM architecture arises.

$$\mathbf{F}_{(\tau)} = \boldsymbol{\sigma}(\mathbf{W}_{fx} * \mathbf{X}_{(\tau)} + \mathbf{W}_{fh} * \mathbf{H}_{(\tau-1)} + \mathbf{B}_f) \tag{4.3.4}$$

$$\mathbf{I}_{(\tau)} = \boldsymbol{\sigma}(\mathbf{W}_{ix} * \mathbf{X}_{(\tau)} + \mathbf{W}_{ih} * \mathbf{H}_{(\tau-1)} + \mathbf{B}_i) \tag{4.3.5}$$

$$\mathbf{O}_{(\tau)} = \boldsymbol{\sigma}(\mathbf{W}_{ox} * \mathbf{X}_{(\tau)} + \mathbf{W}_{oh} * \mathbf{H}_{(\tau-1)} + \mathbf{B}_o) \tag{4.3.6}$$

$$\tilde{\mathbf{C}}_{(\tau)} = \mathbf{tanh}(\mathbf{W}_{cx} * \mathbf{X}_{(\tau)} + \mathbf{W}_{ch} * \mathbf{H}_{(\tau-1)} + \mathbf{B}_c) \tag{4.3.7}$$

$$\mathbf{C}_{(\tau)} = \mathbf{F}_{(\tau)} \odot \mathbf{C}_{(\tau-1)} + \mathbf{I}_{(\tau)} \odot \tilde{\mathbf{C}}_{(\tau)} \tag{4.3.8}$$

$$\mathbf{H}_{(\tau)} = \mathbf{O}_{(\tau)} \odot \tanh(\mathbf{C}_{(\tau)}) \tag{4.3.9}$$

The memory cell output $\mathbf{C}$ is regulated by a forget gate, an input gate and an output gate. They respectively compute activations $\mathbf{F}$, $\mathbf{I}$ and $\mathbf{O}$, based on various learnable kernels $\mathbf{W}_{fx}$, $\mathbf{W}_{ix}$, $\mathbf{W}_{ox}$, $\mathbf{W}_{cx}$, $\mathbf{W}_{fh}$, $\mathbf{W}_{ih}$, $\mathbf{W}_{oh}$, $\mathbf{W}_{ch}$ and biases $\mathbf{B}_f$, $\mathbf{B}_i$, $\mathbf{B}_o$, $\mathbf{B}_c$. In Figure 27, $\{\mathbf{W}_{fx}, \mathbf{W}_{ix}, \mathbf{W}_{ox}, \mathbf{W}_{cx}\}$ and $\{\mathbf{W}_{fh}, \mathbf{W}_{ih}, \mathbf{W}_{oh}, \mathbf{W}_{ch}\}$ are gathered into $\mathbf{W}_x$ and $\mathbf{W}_h$. The resulting activations are combined to update the cell state $\mathbf{C}_{(\tau)}$ and the hidden state $\mathbf{H}_{(\tau)}$. During this process, the element-wise tensor multiplication "$\odot$", also known as Hadamard-product, is involved. At time τ, an intuitive understanding of the ConvLSTM-cell can be summarized as follows:

- The cell state is responsible for the memory property of the LSTM-layer. To update it, a candidate $\tilde{\mathbf{C}}_{(\tau)}$ is computed, based on the current cell inputs $\mathbf{X}_{(\tau)}$, $\mathbf{H}_{(\tau-1)}$ and on trainable parameters $\{\mathbf{W}_{cx}, \mathbf{W}_{ch}, \mathbf{B}_c\}$, as shown in Equation 4.3.7.
- The update $\mathbf{C}_{(\tau)}$ is performed by a linear combination of the candidate $\tilde{\mathbf{C}}$ and the previous cell state $\mathbf{C}_{(\tau-1)}$. They are respectively weighted by the activations $\mathbf{I}_{(\tau)}$ and $\mathbf{F}_{(\tau)}$, as shown in Equation 4.3.8.
- Both of these weights are computed based on the current cell inputs $\mathbf{X}_{(\tau)}$ and the previous hidden state $\mathbf{H}_{(\tau-1)}$. Two different sets of trainable parameters $\{\mathbf{W}_{ix}, \mathbf{W}_{ih}, \mathbf{B}_i\}$ and $\{\mathbf{W}_{fx}, \mathbf{W}_{fh}, \mathbf{B}_f\}$ are therefore used, as expressed in Equation 4.3.4 and Equation 4.3.5.
- Finally, the hidden state $\mathbf{H}_{(\tau)}$ is computed based on the updated cell state $\mathbf{C}_{(\tau)}$ and the output gate activation $\mathbf{O}_{(\tau)}$, as formulated in Equation 4.3.9. The output gate itself results of a convolution between the inputs and the kernels $\{\mathbf{W}_{ox}, \mathbf{W}_{oh}, \mathbf{B}_o\}$, as expressed in Equation 4.3.6.
- The hidden state $\mathbf{H}^{(l)}_{(\tau)}$ and the cell state $\mathbf{C}^{(l)}_{(\tau)}$ are fed back into the layer for the computation at $\tau + 1$. Only the hidden state is passed to the layer $l + 1$ for the computation at time τ. The cell state is therefore primarily determining the time dynamic within a layer, while the hidden state primarily determines the dynamic across the layers.

ConvLSTM-cells offer a convenient architecture for interpreting frames evolution over time. Hereby the original data format does not have to be flattened and densely connected. As a comparison, an approach used for similar purposes can be briefly sketched. The combined use of convolutional and LSTM layers can, in fact, address the same problematic. Nonetheless, it results in flattening the

output of the convolutional layers into vectors for the the LSTM layer, which subsequently densely connects the hidden states. This is not necessarily meaningful for EMP tasks, as dynamic obstacles (e.g. pedestrians) rather react on local context than on the whole feature map. As computation time and data amount are the common restrictions to face in the field of deep learning, a lower amount of weights to fit is desired. Learning convolutions over time clearly responds to this requirement, as it permits to train fewer parameters, while still encoding the local information.

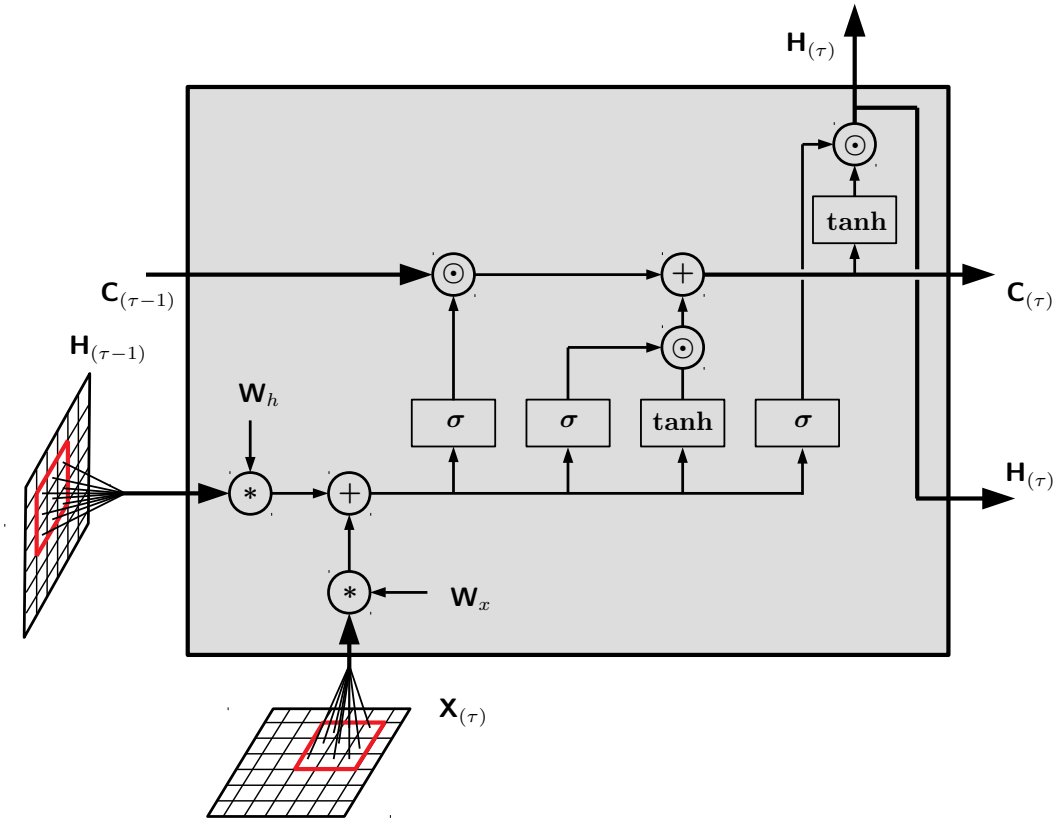


Figure 27: The ConvLSTM cell, as presented in [46].

Remark: The classical operations involved in CNNs do not include the time dimension. They can however be applied in a time-distributed way. For example, a time distributed max-pooling applies the max-pooling operation on each frame separately. The output is therefore also a sequence of frames. The main difference between time-distributed operations and recurrent cells is that no inner dynamic nor feedback is involved in the former one.

Equation 4.3.4 to Equation 4.3.9 show that, for a single ConvLSTM-layer, a relatively high number of parameter has to be trained. In some cases, this complexity has to be accepted for the sake of performance. A first alternative to this is the use of ConvGRU-layers, as presented in [54]. However, the convolution operation itself can already present a way to learn time-dependencies while involving less parameters than a ConvLSTM-layer. This idea is presented in the next section.

4.4 Time Series Prediction with Convolution

As shown in [55], 1D-convolutions can be used for time series predictions of scalar inputs. To illustrate this idea, let us take an arbitrary, time-dependent, scalar function $g(t)$. As expressed in Equation 4.4.3, applying a specific kernel on a sequence of sampled function value results in computing the numerical integration of the first order backward difference shown in Equation 4.4.1.

$$\Delta g(t) = \frac{g(t) - g(t - \delta_t)}{\delta_t} \simeq g'(t) \tag{4.4.1}$$

$$[-1, \quad 1] * \begin{bmatrix} g(t - \delta_t) & g(t) \end{bmatrix} = g(t) - g(t - \delta_t) \tag{4.4.2}$$

$$= \Delta g(t)\, \delta_t \tag{4.4.3}$$

Finally, computing the function value for the next time step can be achieved by the use of a subsequent layer:

$$g(t) + [-1, \quad 1] * \begin{bmatrix} g(t - \delta_t) & g(t) \end{bmatrix} \simeq g(t + \delta_t) \tag{4.4.4}$$

This can already be a good approximation of the function value for the future time step. However, by using several layers with trainable kernel weights, more complex integration behaviors can be learned. This is a simpler structure than the LSTM-cell. It does not rely on a cell state, which presents both, an advantage and a drawback. One the one hand, the cell state does not have to be initialized or built up over time. One the other hand, long-term dependencies cannot be captured. This is even more obvious for small kernels, which imply a small range of passed values used for prediction.

3D-convolutions are nothing but the generalization of the common 2D-convolution to a higher dimension. In the very same way that a vector-valued kernel applied along the time dimension of a scalar variable can learn some of its temporal dependency, a 3D kernel applied along the time-dimension of a 2D variable can deliver a time series prediction. An exemplary application of this idea for next-frame prediction can be found in [56]. Consequently, the time evolution of OGMs can be captured by 3D-convolutions [47] and this assessment is used in chapter 6.

In chapter 6, ConvLSTM, 3D-convolutions and the common operations related to it will play a central role for the design of an EMP technique. All the deep learning concepts needed to understand it have been introduced in the present chapter. The next chapter aims to partially render the state of the art in the field of EMP and find out what approach is the best suited for the present application.

Chapter 5

Environment Motion Prediction – State of the Art

Dynamic obstacle avoidance (DOA) is a vast topic and a task that can be handled in myriads of different ways. As already mentioned in the introduction and pictured in Figure 2, non-predictive approaches lead to naive behaviors of the robot and are, therefore, left aside for the current considerations. Predictive techniques partition this task into two steps: forecasting the environment motion and making use of this prediction. This chapter focuses on the first step and aims to partially render the diversity of EMP techniques. They can be roughly classified into two categories: model-based and data-based approaches. In section 5.1, their advantages and drawbacks are briefly compared. In section 5.2, GPM19, a data-based EMP technique suited for an application on the ROSBot 2.0, is illustrated. As autonomous driving is a major mover for the research in the field of EMP, techniques developed for cars are used as inspiration source for the present work.

The second step of DOA, consisting of incorporating the EMP into the control strategy, is treated in chapter 7. The EMP technique itself is developed in chapter 6 and strongly relies on GPM19.

Remark: Reinforcement learning approaches for DOA are also appearing, as proposed in [57, 58]. Here, the task is not partitioned in EMP and subsequent control, as the control is also learned from the data. Even though these methods show promising results, they represent a completely different approach and are thus left aside for the present considerations.

5.1 Model-Based vs. Data-Based Approaches

Model-based approaches make an EMP based on mathematical models specifically hand-designed for this task. They present a wide range of applications and complexity. A simple example is given by the velocity extrapolation model (VEM). Here, the dynamic obstacle is represented by a point in space and its current velocity is approximated by a first order backward finite difference, using the previous position. It is subsequently integrated over time to predict the next positions [42]. Such an assumption is unable to frame the changes in direction and speed. It ignores potential interactions with the environment and thus only gives good estimates for short-term predictions.

As autonomous driving requires the motion prediction of other cars, it encourages the research towards developing models that are more complex than VEM. For example, the probabilistic model presented in [59] allows multimodal predictions, meaning that multiple scenarios can be forecasted at the same time. For example, a car can be predicted to drive straight, turn right or turn left with a certain

likelihood depending on its current state. Additionally, the behavior of cars influence each-other, allowing to render the interaction between agents.

Model-based techniques present a low model complexity, leading to four major advantages. First, their computation tends to be fast [59]. Second, only little data is needed to optimize the model parameters. Third, overfitting is almost impossible with such a small number of parameters and leads to a good robustness. Finally, their simple structure allows to have insights on how the prediction mechanism works. However, in the field of autonomous driving, they rely on two major assumptions:

1. Only the motion of other cars has to be predicted, as no other dynamic obstacles (e.g. pedestrians and cyclists) are considered.
2. The agents obey a certain lane law, as cars only drive on pre-defined regions of their environment. Static obstacles are not incorporated to the model, as they are assumed to be outside the lanes.

The first assumption is particularly restrictive, as cars interact with pedestrians, cyclists, ... etc. Therefore, the research trend in the field of EMP for autonomous driving shifted towards data-based techniques [60, 61, 62, 63], as they do not make any assumption on the agent type and behavior. For example, data-based techniques can compute EMPs for cars and pedestrians with a single model, as shown in [64]. As the ROSBot 2.0 has to move in environments possibly presenting multiple agent types, the data-based approach seems to be a better choice.

Moreover, the second assumption does not hold for the ROSBot 2.0, as it moves among pedestrians, which certainly do not obey strict lane laws. Additionally, the motion of the agents is determined by the geometry of the static obstacles, which can be arbitrarily complicated. At this stage, it appears that the EMP technique for the ROSBot 2.0 has to be capable of encoding the environment geometry to make meaningful predictions. As NNs do not make assumptions about lane laws and are capable to encode spatial information (for example by the use of convolutional layers), they appear to be a reasonable choice for the current work.

The data-based approach proposed in [43] gathers these advantages and is the main inspiration source for the development of the ROSBot 2.0 EMP strategy. Therefore, it is presented in the next section.

5.2 GPM19 – A Data-Based Approach

Data-based approaches to EMP can rely on various representations of the environment. For robotic applications, a very general and common one is the representation as OGM. Furthermore, any other representation can be easily converted into an OGM. In [42, 43, 65, 66], future OGMs are predicted, based on the knowledge of previous sequences and on the use of NNs. Within this series of articles, the model performing the best and involving the minimal amount of parameters is presented in [43]. In the rest of the thesis, this grid prediction model (GPM) will be referred to as GPM19. It is now briefly analysed to understand its structure, its input-output format and its results.

The autoencoding structure of GPM19 is depicted in Figure 28. As it can be observed, the sequence of past OGMs is fed into the network and a prediction for the upcoming time step is output. This is achieved by first encoding the input through three consecutive time-distributed 2D-convolution layers. Recall from chapter 4 that time distribution is nothing but a sequential application of the same operation. Each layer is combined with a time-distributed max-pooling operation, allowing to shrink the input dimension from a sequence of 480×480 OGMs to a sequence of 18×18 matrices. This procedure presents two major advantages:

1. The encoded representation allows to capture more global information without using bigger kernels nor deeper networks. For example, using a 5×5 kernel on a 480×480 input only encodes relatively local information. Because of the scale difference, applying it on a 18×18 input can capture a relatively large neighbourhood of higher-level features.

2. From a practical point of view, more kernels can be used on the reduced feature space without leading to memory overflows at implementation time. In fact, using 128 kernels on a mini-batch of 480×480 OGM sequences might be impossible for the hardware. In comparison, doing so on 18×18 hidden state matrices is feasible with common GPUs.

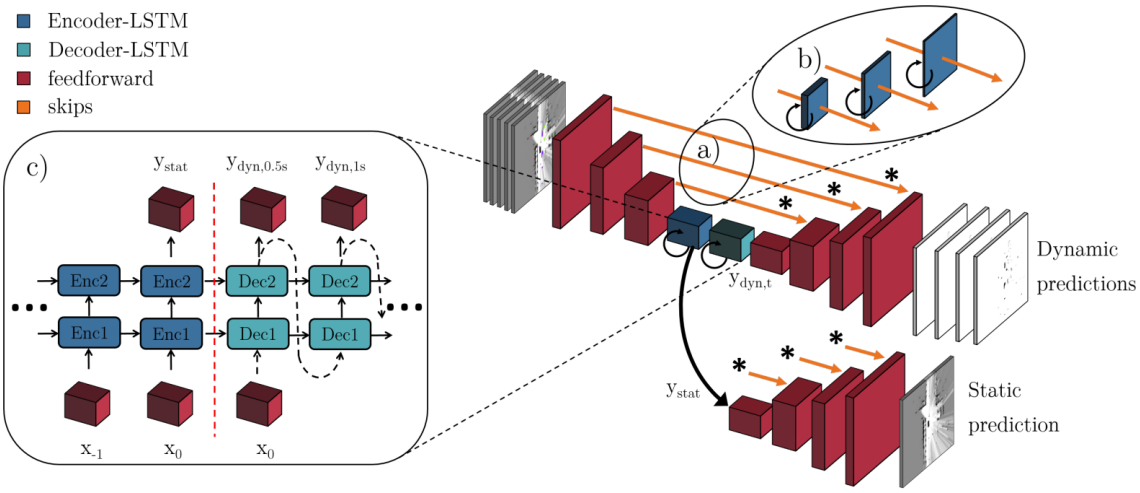


Figure 28: Overview of the GPM19 architecture developed in [43].

Subsequently, two encoder ConvLSTM-layers (represented in dark blue in Figure 28) as presented in section 4.3 are applied to capture the time dependency of the input data. Finally, to respect the required output format, the hidden states have to be gradually brought back to a sequence of 480×480 OGMs. As pictured in Figure 28, this decoding procedure is handled by two separate branches, respectively dedicated to the recognition of the static OGM and the prediction of the dynamic OGMs:

1. The static branch takes the output of the encoder and feeds it through a set of three layers involving time-distributed transposed 2D-convolutions. This operation can be seen as the counterpart of the 2D-convolution and allows a trainable upscaling of the encoded sequences. As for the convolution operation, it presents the major advantage of proceeding to a sparse, space-invariant connection between neighboured features [47]. The last layer consist of a time distributed 2D-convolution.

2. The dynamic branch first feeds the output of the encoder into two decoder ConvLSTM-layers (represented in cyan in Figure 28). Like the static branch, it then feeds the hidden states into three layers of transposed convolutions. It is stressed out that their structure is equal to the one of the static branch but relies on a different set of trainable parameters. Thus, more degrees of freedom are available at decoding time.

In summary, during the encoding process, the size of the hidden state shrinks over the depth, while the channel dimension grows. Over the decoding process, the exact opposite occurs: the channel dimension decreases and the latent representation is scaled back to the input size.

As explained in section 4.3, vanishing gradients can arise whenever the chain of tensor operations is relatively large. This does not only arise for sequence models but also for deep NNs. In the

former case, the depth of the time dimension is mainly responsible, whereas only the layer-depth is involved in the latter one. As GPM19 presents more than only a couple of layers, one can imagine that vanishing gradients might make the training procedure more complicated. A solution to this problem was presented by the introduction of the ResNets in [67]. Hereby, the key concept is to use skip connections. They allow feedforwarding the output of a layer $\bar{l}$, not only into the layer $\bar{l}+1$, but also into deeper layers $l > \bar{l}+1$. The powers of the parameter tensor arising in the back-propagation computation can thus be held lower and avoid vanishing values of the gradient. Such skip connections are used between the encoding and decoding layers of corresponding size. Thus, they allow the decoding layers to additionally dispose of the informations before the full encoding is performed. They are represented by yellow arrows in Figure 28 and consist of concatenating the hidden states along the channel dimension.

Remark: GPM19 takes dynamic OGMs as input. They are introduced in [68] and consist of probabilistic OGMs, extended with further informations. For each cell, a speed estimate for the two directions of the plane is additionally supplied along the channel dimension. As shown in [43], GPM19 performs slightly better with this supplementary information but can also deal with OGMs without speed estimates.

GPM19 was trained on real-world data, recorded by a static vehicle equipped with LIDAR devices. This was done for two different street crossings, where several agents interact and multiple scenarios can typically arise. It was shown on the test data, that the behavior of cars, pedestrians and cyclist can be well predicted. For more details, refer to [43] and section E of the appendix. Additionally to its performance, several properties of GPM19, makes it particularly appealing:

1. The input-output format relies on OGMs and is thus perfectly suited for robotic applications.
2. The EMP of different agent types is tackled by a single model.
3. The interaction with the static environment is automatically encoded and used for prediction. Thus, turning around static obstacles can be foreseen.
4. The interaction between agents does not rely on any assumption and is directly learned from the data. As shown in [43] and in section E of the appendix, an agent is automatically predicted to slow down if dynamic obstacles enter its vicinity.
5. By separating the output into two channels, cells are automatically labelled as static or dynamic.
6. For each detected agent, a model-based technique would need to extract a label, as e.g., “human” or “car”. GPM19 does not and directly understands the motion behavior from the data.
7. GPM19 outputs a probabilistic OGM (c.f. chapter 1, p.9), allowing to understand how sure the prediction is. It permits to react more conservatively if the future occupancies seem uncertain.
8. Compared to particle representations, the OGM format allows to define occupancy regions rather than specific psoitions. By the use of a NN, these regions can have arbitrarily complex shapes, as shown for the multimodal example in [43] and section E of the appendix.
9. Thanks to the very general structure of deep learning methods, this model might be extended with minimal efforts. For example, if bounding boxes of obstacles were available, they could also be fed into a very similar network.

The advantages presented by GPM19 motivate the present work to follow this approach. The next chapter aims to develop and implement a similar GPM.

Chapter 6

Environment Motion Prediction with GPM20

The two past chapters were dedicated to the state of the art. In the present one, this knowledge is used to develop GPM20, a modified version of GPM19. In section 6.1, the architecture presented in Figure 28 is adapted. In section 6.2, the data generated in simulation and used for training GPM20 is introduced. Finally, in section 6.3, the training procedure and its results are studied.

6.1 GPM20 Architecture

First, $\mathbf{M}_\tau \in \mathbb{R}^{n_\mathrm{M} \times n_\mathrm{M} \times 1}$ is defined as the ground truth OGM at arbitrary time step $\tau \in \mathbb{N}$. Its size is denoted by $n_\mathrm{M} \in \mathbb{N}$. The prediction made by GPM20 for the future time step $\tau + \iota$ is denoted by $\mathbf{M}_\tau^{(\iota)} \in \mathbb{R}^{n_\mathbf{M} \times n_\mathbf{M} \times 2}$, with $\iota \in \{1, 2, \ldots, n_\iota\}$ and n_ι the number of frames to predict. Two channels are used for the output, one for the static and one for the dynamic OGM, which are respectively indexed with s and d:

$$\mathbf{M}_\tau^{(\iota)} = \left[\mathbf{M}_{s,\tau}^{(\iota)}, \mathbf{M}_{d,\tau}^{(\iota)}\right] \tag{6.1.1}$$

For the present considerations $n_\mathrm{M} = 96$ is set. However, one of the major advantages of the convolution operation is that it can be applied for any n_M bigger than the kernel size. GPM20 takes a sequence $[\mathbf{M}]_\tau$ of past OGMs at time step τ, and outputs a probabilistic forecast $[\hat{\mathbf{M}}]_\tau$ of future OGMs. It is fixed that both sequences have the same length n_ι:

$$[\mathbf{M}]_\tau = [\mathbf{M}_{\tau-n_\iota}, \quad \mathbf{M}_{\tau-n_\iota+1}, \quad \ldots, \quad \mathbf{M}_\tau], \quad [\mathbf{M}]_\tau \in \mathbb{R}^{n_\iota \times n_\mathbf{M} \times n_\mathbf{M} \times 1} \tag{6.1.2}$$

$$[\hat{\mathbf{M}}]_\tau = \left[\left[\mathbf{M}_{s,\tau}^{(1)}, \mathbf{M}_{d,\tau}^{(1)}\right], \quad \ldots, \quad \left[\mathbf{M}_{s,\tau}^{(n_\iota)}, \mathbf{M}_{d,\tau}^{(n_\iota)}\right]\right], \quad [\hat{\mathbf{M}}]_\tau \in \mathbb{R}^{n_\iota \times n_\mathbf{M} \times n_\mathbf{M} \times 2} \tag{6.1.3}$$

Remark: Because of its separation in two output channels, GPM20 is capable of labelling cells as static or dynamic. This represents a major improvement for the ROSBot 2.0 as, until now, SLAM only outputs a single binary OGM $\mathbf{M} = \mathbf{M}_s + \mathbf{M}_d$ As explained in Part I, the knowledge of $\mathbf{M}_s$ allows the global planner to compute more pertinent paths than based on $\mathbf{M}$.

The architecture of GPM20 is shown in Figure 29. The hyper-parameter choices are summarized in Table 6.1. The number of kernels per layer, the stride applied during convolution/pooling and the DO-probability are denoted by $n_K, S \in \mathbb{N}$ and $\mathsf{p}_\mathrm{DO} \in [0, 1]$. The encoder of GPM20 consists of three

layers, transforming the input dimension $n_\iota \times 96 \times 96 \times 1$ to $n_\iota \times 12 \times 12 \times 80$. The two first layers involve a 3D-convolution while the last one consists of a ConvLSTM-cell. This is a major difference to GPM19, as its encoder only consists of time-distributed 2D-convolution, which are only able to capture spatial information. In comparison, GPM20 learns the feature space reduction directly in a sequence-orientated way. For each encoding layer, DO, leaky ReLU, BN and max-pooling are applied. The leaky ReLU activation is defined as in Equation 6.1.4, with $\eta = 0.1$ for the present consideration.

$$\text{LeakyReLU}(\mathsf{X}) = \begin{cases} \mathsf{X}, & \mathsf{X} \geq 0 \\ \eta \cdot \mathsf{X}, & \mathsf{X} < 0 \end{cases} \tag{6.1.4}$$

Before decoding, two ConvLSTM-layers are applied and learn relations on the latent space. This is half the amount of layers used in GPM19 at this stage and represents the second major difference between the GPMs. Even though GPM20 is shallower, it still delivers a sufficient performance, as shown in section 6.3. This can be explained by the fact that the temporal information is already exploited at encoding time.

The third major difference to GPM19 is that a single decoder is used, while still allowing a separation between static and dynamic predictions. Thus, less parameters are trained and informations common to both decoding procedure do not have to be extracted twice. The decoder of GPM20 presents a mirrored version of the encoder, thus also involving a ConvLSTM and two 3D-convolution layers. For each of them, the output of the recurrent cells is upscaled by a time-distributed transposed 2D-convolution. As in GPM19, skip connections are exploited at decoding time. Finally, the output layer consists of a 3D convolution with two kernels. One is responsible for extracting the informations of the static OGM, while the other one handles the extraction of dynamic predictions. The final activation function is a sigmoid, such that the output value range is between 0 and 1. This permits to interpret the cell value as occupancy probability.

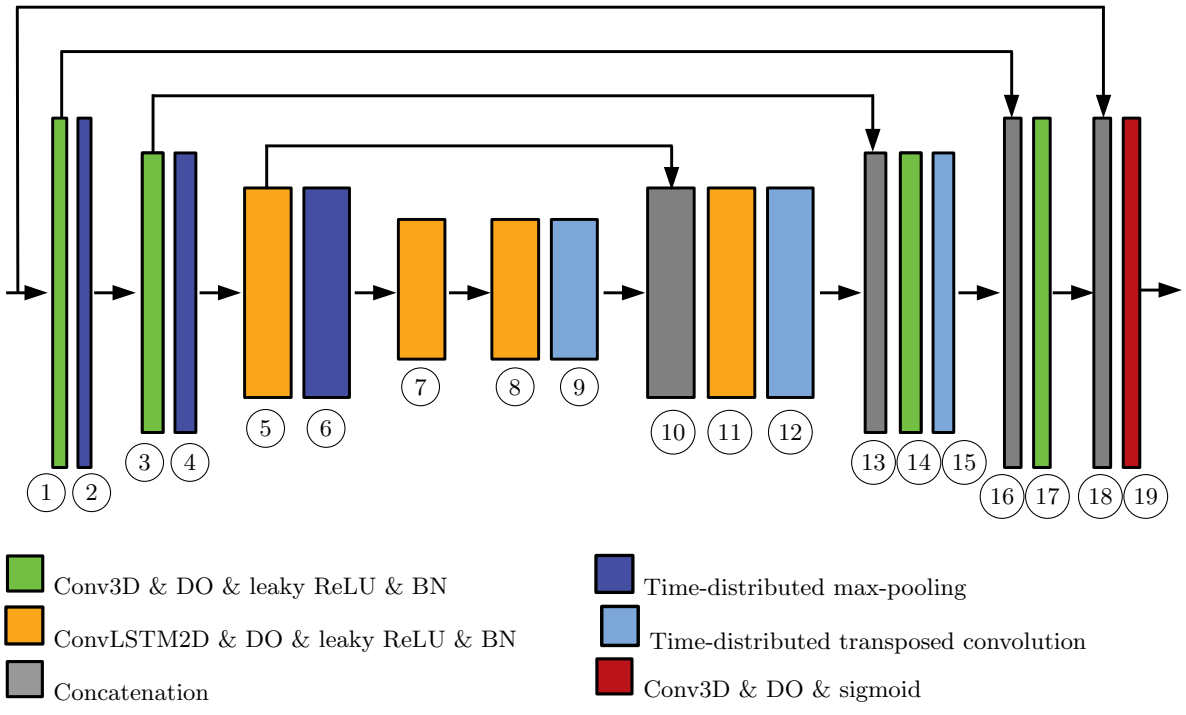


Figure 29: GPM20 architecture. The numbering corresponds to Table 6.1.

N°	Type	Input	Output dim.	n_K	Kernel /window size	S	p_{DO}	Trainable params
1	Conv3D	$\mathbf{X}$	$n_\iota \times 96 \times 96 \times 20$	20	7×7	1	0.1	6,880
2	Max-pooling	$\mathbf{H}^{(1)}$	$n_\iota \times 48 \times 48 \times 20$	-	2×2	2	-	0
3	Conv3D	$\mathbf{H}^{(2)}$	$n_\iota \times 48 \times 48 \times 40$	40	5×5	1	0.1	100,040
4	Max-pooling	$\mathbf{H}^{(3)}$	$n_\iota \times 24 \times 24 \times 40$	-	2×2	2	-	0
5	ConvLSTM2D	$\mathbf{H}^{(4)}$	$n_\iota \times 24 \times 24 \times 80$	80	5×5	1	0.2	600,240
6	Max-pooling	$\mathbf{H}^{(5)}$	$n_\iota \times 12 \times 12 \times 80$	-	2×2	2	-	0
7	ConvLSTM2D	$\mathbf{H}^{(6)}$	$n_\iota \times 12 \times 12 \times 80$	80	5×5	1	0.2	1,112,320
8	ConvLSTM2D	$\mathbf{H}^{(7)}$	$n_\iota \times 12 \times 12 \times 80$	80	5×5	1	0.2	1,280,320
9	Transp. Conv.	$\mathbf{H}^{(8)}$	$n_\iota \times 24 \times 24 \times 80$	80	2×2	2	-	14,460
10	Concatenation	$\mathbf{H}^{(5)}$, $\mathbf{H}^{(9)}$	$n_\iota \times 24 \times 24 \times 160$	-	-	-	-	0
11	ConvLSTM2D	$\mathbf{H}^{(10)}$	$n_\iota \times 24 \times 24 \times 60$	60	5×5	1	0.2	1,200,240
12	Transp. Conv2D	$\mathbf{H}^{(11)}$	$n_\iota \times 48 \times 48 \times 60$	60	2×2	2	-	6,440
13	Concatenation	$\mathbf{H}^{(3)}$, $\mathbf{H}^{(12)}$	$n_\iota \times 48 \times 48 \times 120$	-	-	-	-	0
14	Conv3D	$\mathbf{H}^{(13)}$	$n_\iota \times 48 \times 48 \times 40$	40	5×5	1	0.1	500,040
15	Transp. Conv2D	$\mathbf{H}^{(14)}$	$n_\iota \times 96 \times 96 \times 40$	40	2×2	2	-	1,620
16	Concatenation	$\mathbf{H}^{(1)}$, $\mathbf{H}^{(15)}$	$n_\iota \times 96 \times 96 \times 80$	-	-	-	-	0
17	Conv3D	$\mathbf{H}^{(16)}$	$n_\iota \times 96 \times 96 \times 20$	20	$7 \times 7 \times 7$	1	0.1	411,620
18	Concatenation	$\mathbf{X}$, $\mathbf{H}^{(17)}$	$n_\iota \times 96 \times 96 \times 21$	-	-	-	-	0
19	Conv3D	$\mathbf{H}^{(18)}$	$n_\iota \times 96 \times 96 \times 2$	2	$7 \times 7 \times 21$	1	0.1	96,728

Table 6.1: Model summary of GPM20.

In general, the indices k and τ, respectively used for the control and the EMP, are different. To keep MPFC and GPM20 consistent with each other, the frames are sampled with $\delta_t = 0.3\,\mathrm{s}$ and both indices are equal $k = \tau$. The number of predicted frames $n_\iota = 7$ is set to make it correspond with the prediction horizon of the MPFC. As the network operates in a symmetric sequence-to-sequence fashion, the seven most recent frames are used as input of the GPM.

GPM20 is implemented in Keras with TensorFlow back-end. The 5.36 million of trainable parameters represent less than 40 % of the 14.65 million parameter used in GPM19. This was achieved by avoiding a second decoding branch and keeping the network shallower by encoding and decoding the network in a sequence-orientated way. The mean inference time on Nvidia GeForce GTX 1080 is 0.127 s and leads to the obtention of $n_\iota = 7$ next frames. In comparison, obtaining a single frame with GPM19 on the same hardware requires 0.063 s [43]. For the whole output sequence, GPM19 would need 0.441 s, which is roughly 3.5 times more than with GPM20.

As a stand-alone procedure, GPM20 matches the real-time requirements. However, as EMP and MPFC are serial procedures, the total computing time for an EMP-MPFC starts being too large for a real-time efficient implementation. This is studied in detail in section D of the appendix and shows that the combination of both nonetheless remains close to real-time efficiency. Moreover, recall that the present work is mainly thought as a proof of concept.

A common saying in the field of deep learning is that NNs obey to the law of "trash-in, trash-out". This means that even a well chosen architecture can fail to perform well if the training data is not suited. Furthermore, GPM20 presents a relatively high number of parameters, requiring also a large number of frames for the training. In the next section, the data generation for the training procedure is explained, aiming to understand better, what GPM20 is trained on.

6.2 Data Generation

As gathering real-world data was not possible within the scope of the present thesis, an alternative solution has been chosen. Because humans are the foremost type of dynamic obstacle the ROSBot 2.0 has to face, the pedestrian simulation software Vadere is used to generate artificial data of dynamic environments. The simulation is based on the optimal step model (OSM), which was first introduced in [69], further improved in [70, 71, 72] and finally implemented into Vadere [73].

In brief, OSM relies on the use of potential fields. Minimizing the sum of the potential fields generates attractive forces, directed towards the target, and repulsive forces, directed away from obstacles for every agent of the simulation. The motion of a pedestrian is constrained to a certain range, according to typical pedestrian velocities. To partially render the randomness of human motion, Gaussian noise is added during the position update. By a system of queuing, OSM avoids any overlapping of the agents. An exemplary simulation of pedestrians moving on the H terrain can be viewed `here`[1]. OSM renders the macroscopic behavior while fulfilling the properties of the microscopic scale. This means that overall measures, such as the pedestrian flow, the average velocity, or the time to evacuate a room, yield similar values in simulation and in reality. Parallelly, while zooming in, the simulated individuals avoid collisions and reach their targets as required. Vadere relies on a source/sink system, meaning that pedestrians appear in certain regions and disappear in others. They are respectively represented in green and red in Figure 30. They can represent doors, gates or simply edges of the OGM. For each source, a list of target sinks and a time stride for pedestrian spawning is attributed. More details about OSM and Vadere can be found in [73].

To avoid overfitting on a single static OGM, the pedestrian motion is simulated on diverse environments, shown in Figure 30. Among those, one can find the H terrain, used to test the validity of the control method in Part I. In simulation, the real OGMs $\mathbf{M}_{s,\tau}$ and $\mathbf{M}_{d,\tau}$ are known for any time step τ and can be given as ground truth for the static output channel at training time. For real-world data, this is be possible, as cells can easily be classified as static or dynamic by post-processing the data.

The result output by Vadere is a list of coordinates over time. In order to convert this to an OGM representation, a space discretization is applied with the step $\delta_s = 0.1\,\mathrm{m}$. As all environments are set to have a dimension of $10\,\mathrm{m} \times 10\,\mathrm{m}$, the resulting OGM is $\mathbf{M} \in \mathbb{R}^{100\times100}$. The prediction by GPM20 is done on a symmetrically cropped version with dimensions 96×96. A fixed width of $0.3\,\mathrm{m}$ is attributed to each individual and the cells around their center point are accordingly filled. This results in transforming the particle representation of the environment into a binary OGM, respectively represent in the left and middle sub-figures of Figure 31.

[1] `https://www.youtube.com/watch?v=YwUCTs-a7t0&list=PLTKb-pD4cu3iyjE1YiPTW3SlWxkP4BHzQ&index=9`

The network is developed to not take only binary OGMs as an input. To achieve this, Gaussian noise with a standard deviation of 0.1 is applied on the input of GPM20. Thus, the NN is forced to learn meaningful features rather than rely on the specific 0 and 1 values of a binary OGM. To keep the noisy data within the legal probability range, the absolute value of the noise is subtracted for all occupied cells and added for all unoccupied ones. The result of this procedure is represented in the right sub-figure of Figure 31.

The amount of generated data is crucial for the training and validation process. Each environment has been simulated over 1,200 seconds for the train set. The validation and test set consist of results simulated over 240 seconds. This roughly corresponds to a 70-15-15% split of the data. For training, 15,952 $\{\mathbf{X}, \mathbf{Y}\}$-pairs are available, with $[\mathbf{X}] \in \mathbb{R}^{7\times96\times96}$. Therefore, a total amount of 1,029,095,424 pixel values is used. As it more than the number of trainable parameters by several orders of magnitude, the data is rich enough for the training procedure.

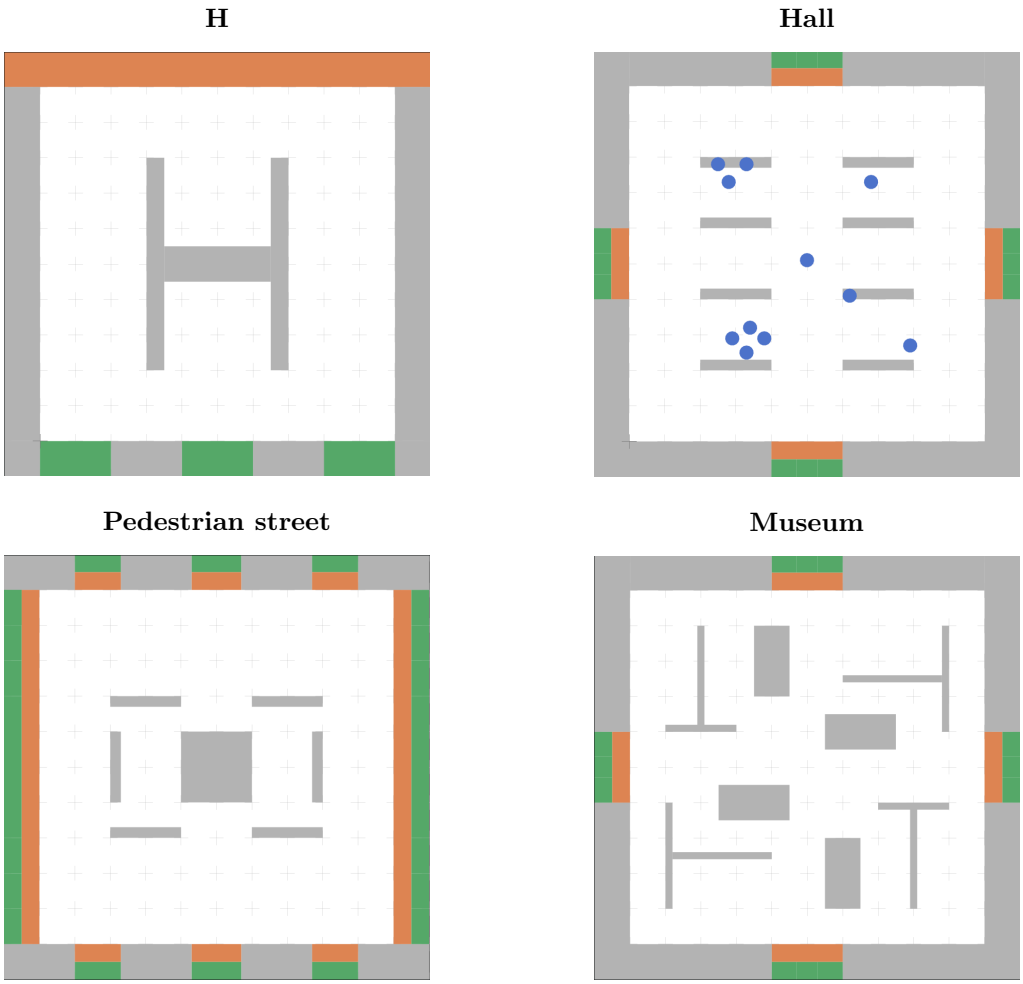


Figure 30: Representation of the different VADERE-environments. The green, red and gray regions respectively represent the sources, sinks and obstacles. The blue points represent static pedestrians.

Compared to GPM19, the current data set presents the drawback of not representing a real-world situation with multiple agent types. However, in the publications leading to the implementation of GPM19 [42, 43, 65, 66], only two static environments are represented in the data set. Both are needed for the sake of good generalization at training time, which means that the static environments used at test time have already been seen by the network. Thus, an important question remains unanswered: how does the network performs on fully unknown environments?

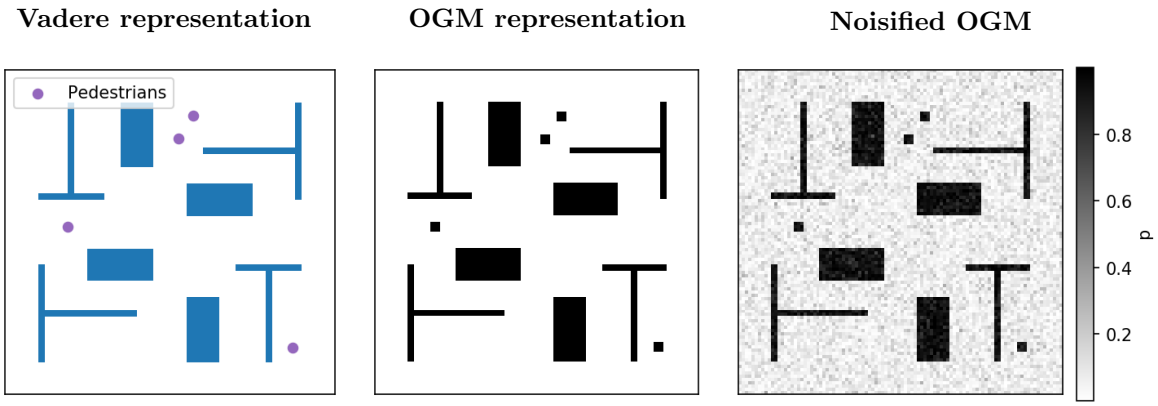


Figure 31: Conversion from a particle representation into an OGM, with p the occupancy probability.

An advantage of the present work is that it relies on an artificial data generation and thus allows an arbitrary large amount of static environment for training and testing. To study how good the network generalizes on fully unknown environments, this advantage is exploited and two further examples are created, as depicted in Figure 32. To put that into the jargon of machine learning, the test set is extended such that it comes from a different distribution than the train and the validation set. In the next section the training procedure is briefly described and the results on the test set are studied.

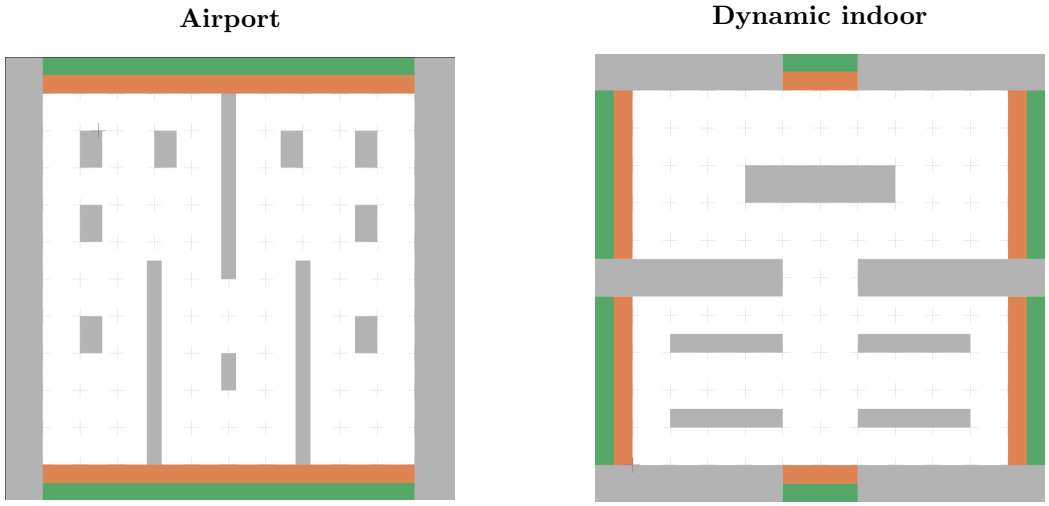


Figure 32: Representation of the additional VADERE-environments, used to test the generalization performance of GPM20.

6.3 Training Results

Binary cross-entropy is a well suited loss for probability estimation tasks [47] and it is therefore used as objective of the ADAM-optimizer. With the notation introduced in section 6.1, the loss yields:

$$\Gamma = -\sum_{\iota=1}^{n_\iota} \left(\mathbf{M}_{s,\tau+\iota} \odot \log\left(\hat{\mathbf{M}}_s\right) + \mathbf{M}_{d,\tau+\iota} \odot \log\left(\hat{\mathbf{M}}_{d,\tau}^{(\iota)}\right) \right) \tag{6.3.1}$$

Notice that for next-frame prediction tasks, the ground truth is obtained by a simple time shift of the input data. GPM20 is trained over 30 epochs. First, to get an intuition of its resulting behavior on

the test set, an exemplary result on the H terrain is depicted in Figure 33. The prediction output by VEM is also displayed as a baseline model. Before diving into a quantitative analysis, several features of the GPM20 predictions can already be observed:

- Blurred occupancies appear: a likely occupancy region is predicted instead of a single position.
- Then main difference to VEM is that pedestrians can be predicted to turn around the corner of obstacles. An example of this can be observed in the lower corners of the H.
- The static OGM is perfectly predicted and the input noise is filtered out.

To simplify the analysis of the results, a new terminology is introduced.

1. True positive: the cell is predicted as occupied and matches the ground truth.
2. True negative: the cell is predicted as unoccupied and matches the ground truth.
3. False positive: the cell is predicted as occupied and does not match the ground truth.
4. False negative: the cell is predicted as unoccupied and does not match the ground truth.

The total amount of true positives, true negatives, false positives and false negatives is denoted by $T_+, T_-, F_+, F_- \in \mathbb{N}$. To compute these quantities for GPM20, the predicted probabilistic OGM has to be converted into a binary one. Therefore, a threshold $\mathsf{p}_0 = 0.1$ is introduced. This means that a cell is only predicted as occupied if its occupancy probability p yields more than 10%.

Remark: At this point, one could think of designing GPM20 rather as a binary classifier. However, p_0 is a meaningful degree of freedom that can be used by the subsequent control to adapt the DOA strategy. Its usefulness is made clearer in section 7.2. In addition, a binary classifier is not able to render the degree of confidence in the prediction, which is a desired information.

The corresponding map of T_+, T_-, F_+, F_- is shown in Figure 34. In comparison to VEM, GPM20 appears to output a larger number of true positives and a smaller amount of false negatives at the expanse of more false negatives. Regarding the DOA developed in chapter 7, this is clearly a favourable behavior. In fact, true positives are crucial for the DOA, as they allow the rover to avoid cells that are occupied in the future. False negatives are critical, as the rover does not recognize the cell as occupied and might collide with the unforeseen object. Finally, false positives are by far less critical: cells that are unoccupied in the future are avoided, as they are recognized to be likely occupied. This simply results in a more conservative DOA and is an acceptable consequence, as long as it is not too pronounced. In a certain sense, polite humans adopt a similar strategy: rather avoid larger regions and not bump into someone than risking a collision for a minimal gain of time.

With Figure 35, the qualitative analysis is pushed a little further, by taking a look at an exemplary result of GPM20 on the airport environment, which was not used during the training. It appears that the labelling and predicting of the static environment is perfect, even though this map has never been seen before by the network. Furthermore, for the top left pedestrian, a multimodal prediction is made and can be particularly observed at $\iota > 4$. Even though this leads to false positives, it is a desired behavior of the network, as both scenarios are possible: going straight or turn.

A prediction on the other unknown terrain is shown in Figure 36. This time, another desired property of GPM20 can be observed. As two pedestrian are getting close to each-other, the fact that they are influencing each-other's motion is predicted and leads to a larger blurred region. For both of the unknown environments, the qualitative performance seems to correspond to the formulated requirements.

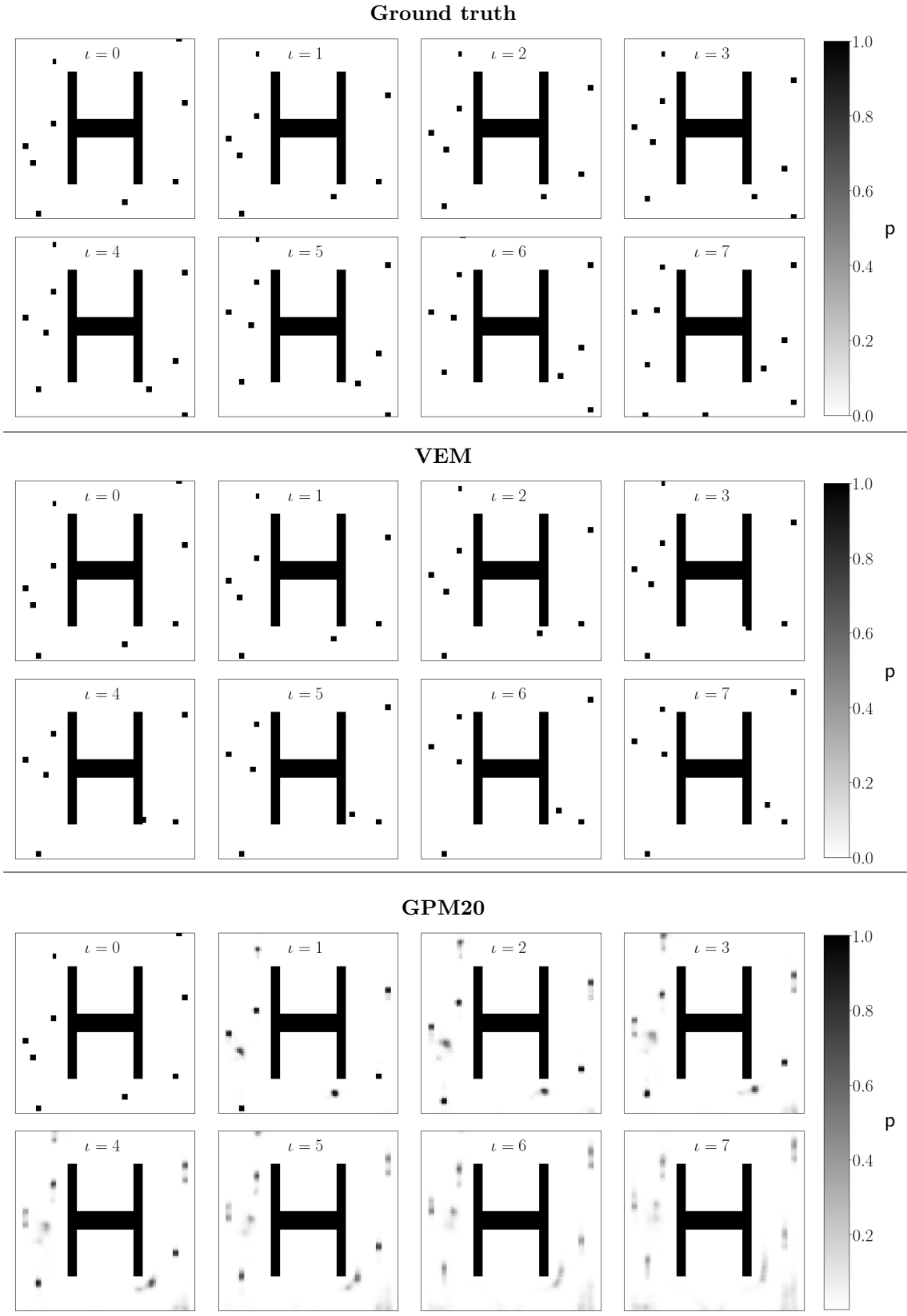


Figure 33: Ground truth, VEM and GPM20 predictions for a situation on the H terrain.

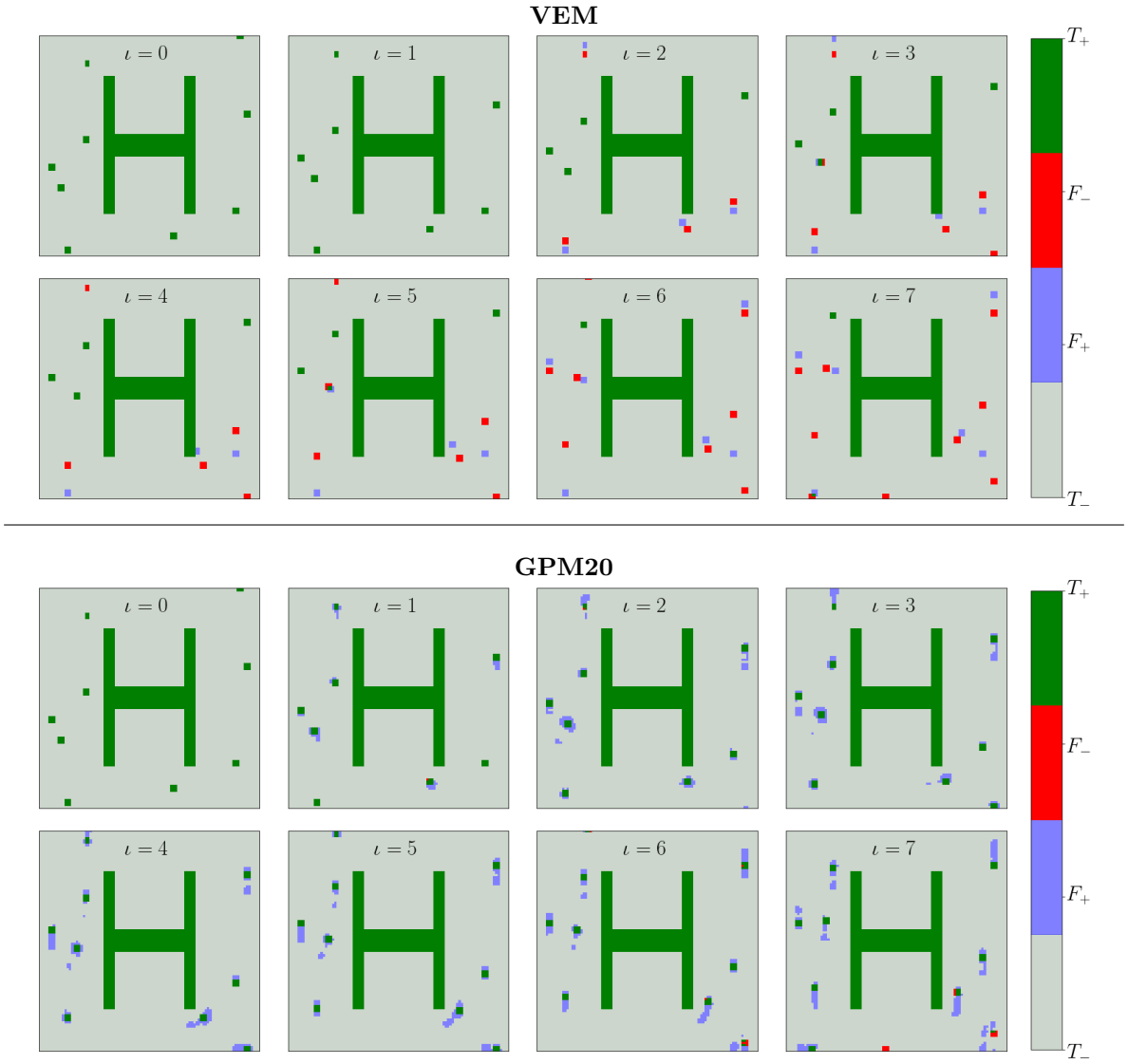


Figure 34: Comparison of T_+, F_-, F_+, T_-, for VEM and GPM20 on the example of Figure 33.

After analysing the results in a qualitative way, statistical tools are now presented to measure the overall performance of GPM20. Precision and recall are introduced as shown in Equation 6.3.2. To reduce the comparison between models to a single scalar, both of these measures are gathered into a weighted harmonic mean, called the F_β-score. It is defined in Equation 6.3.3 and is commonly used for this kind of problems [42, 65, 43, 74]. The higher $\mathsf{F}_\beta \in [0, 1]$, the better the performance.

$$\text{Precision} = \frac{T_+}{T_+ + F_+}, \qquad \text{Recall} = \frac{T_+}{T_+ + F_-} \tag{6.3.2}$$

$$\mathsf{F}_\beta = \frac{(1+\beta^2) \cdot T_+}{(1+\beta^2) \cdot T_+ + \beta^2 \cdot F_- + F_+} \tag{6.3.3}$$

For the present case, the recall value is more important. In fact, a recall near to 1 implicates a high number of true positives and a low number of false negatives. Therefore, the recall is weighted twice as much as the precision by setting $\beta = 2$. Notice that the resulting F_2-score still takes precision into account and addresses the requirements formulated in the qualitative analysis: maximize T_+, minimize F_- and try to keep F_+ low to avoid a very conservative DOA.

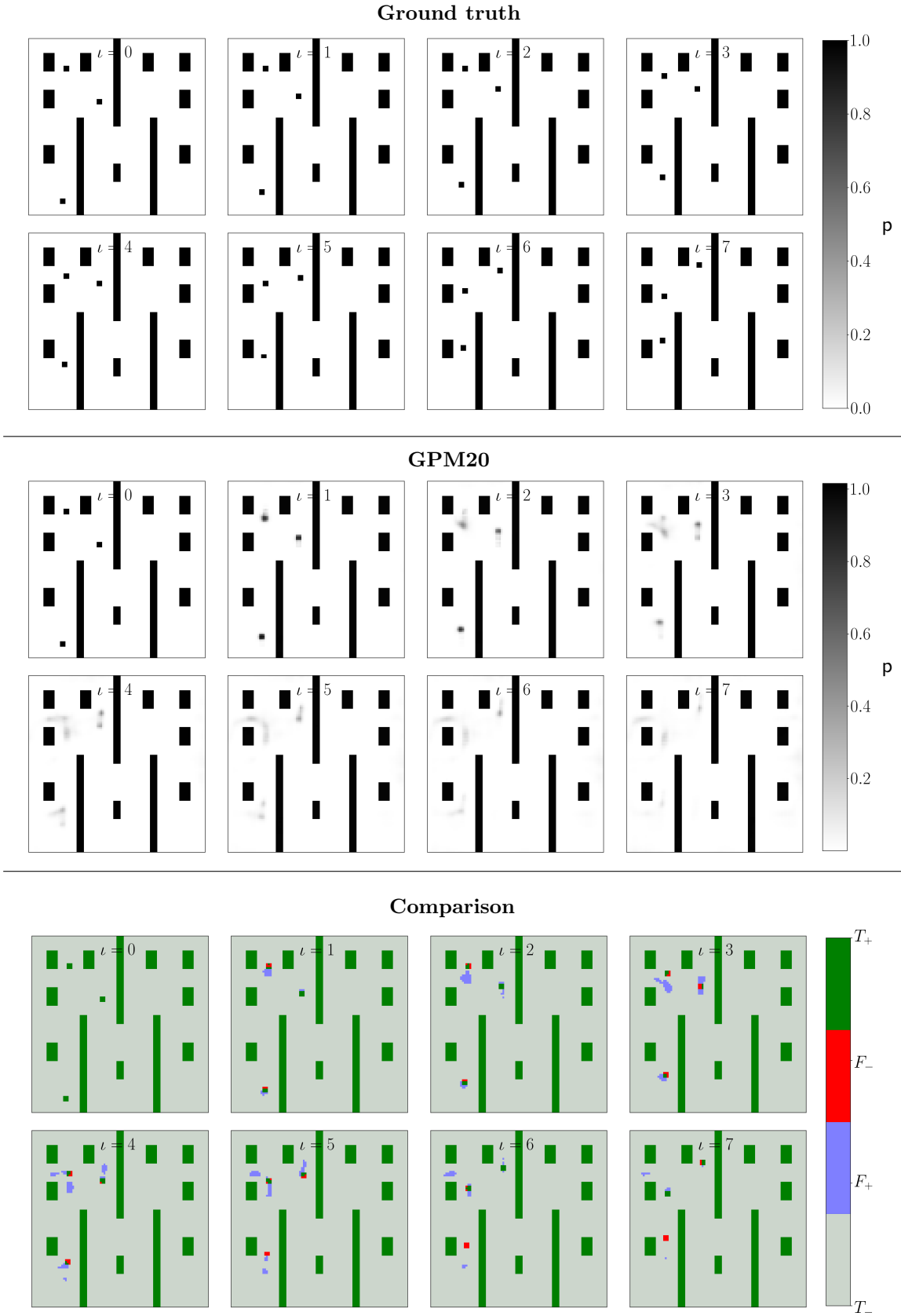


Figure 35: GPM20 results for a sequence of the airport environment.



Figure 36: GPM20 results for a sequence of the dynamic indoor environment

Remark: The F_2-score was not used as objective of GPM20 as it would require to design GPM20 as a binary classifier. As already mentioned, this would lead to a loss of flexibility in the interpretation of the GPM20 output.

To statistically measure the performance of GPM20, the whole test set is predicted by VEM and GPM20. This represents 770 sequence-to-sequence predictions per environment. For each prediction, the F_2-score is computed w.r.t. the cells occupied by pedestrians. For each scenario, it is then averaged over the inference time step k. This delivers a mean F_2-score for each of the prediction steps indexed by ι, as shown in Figure 37. It appears that GPM20 outperforms VEM regardless of the scenario and the prediction index ι. Whereas VEM delivers a good prediction for the next time step $\iota = 1$, the prediction improvement obtained by GPM20 gets evident for $\iota > 2$. Similar curves of VEM and GPM20 can be observed for the H terrain, the pedestrian street and the museum scenario. For the hall example, the performance decrease of both model is slower. This is due to the presence of static pedestrians, which are easier to predict correctly.

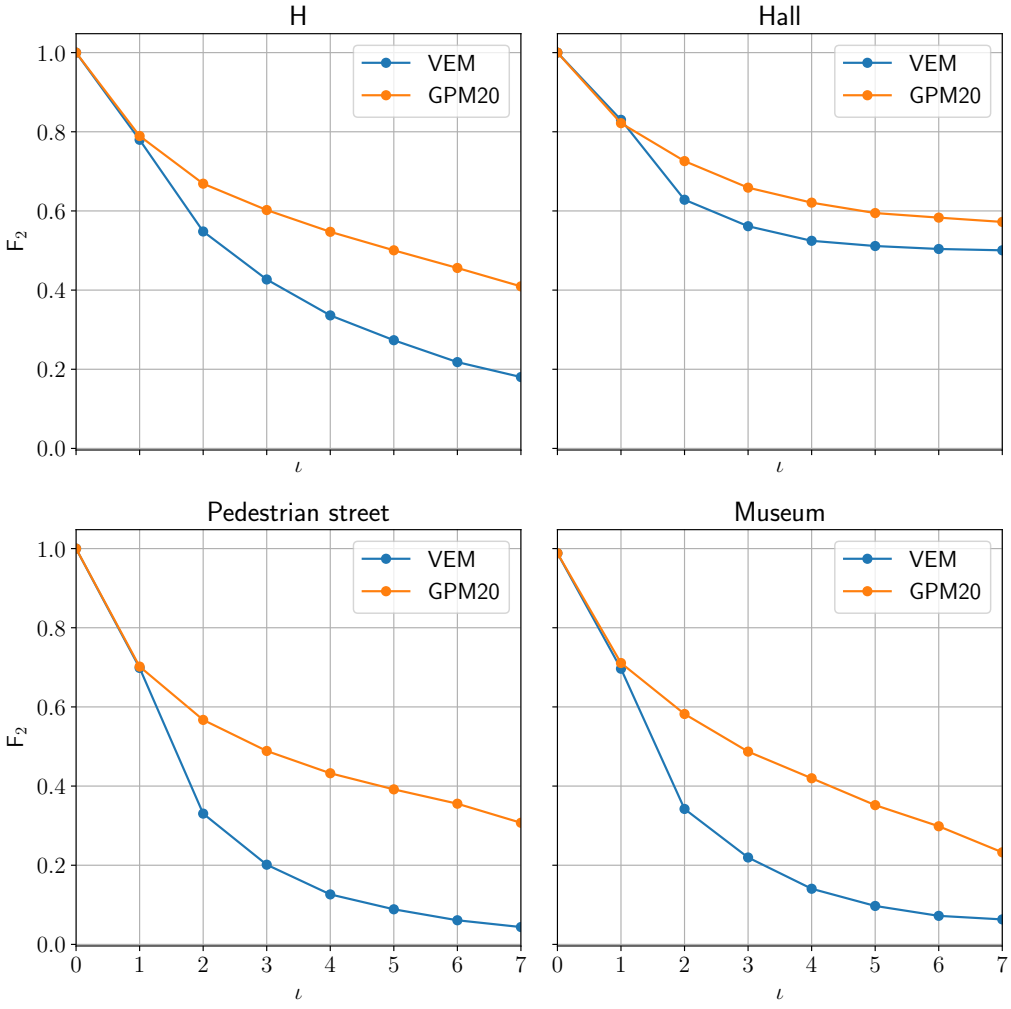


Figure 37: Comparison of the F_2-score of GPM20 and VEM on known test environments.

The same curve is computed for the unknown terrains and is shown in Figure 38. The very same observations can be made and the performance of GPM20 seems to not suffer from the fact that the model was not trained on these environments. This implicates a good generalization of GPM20, which largely outperforms VEM even on new environments.

Besides the performance improvement, GPM20 delivers practical advantages, which are now briefly outlined. VEM is straight-forward to implement in simulation, as the particle representation of the pedestrians is given as output of VADERE. However, for real situations, a conversion from the OGM into a particle representation would be needed. Additionally, to compute the velocity by finite difference, a tracking of each agent would be required. Finally, a conversion back into an OGM format would have to be completed, as it is the base for the convex free-space approximation. In comparison, GPM20 takes the output of the sensor fusion as input and directly computes the prediction.

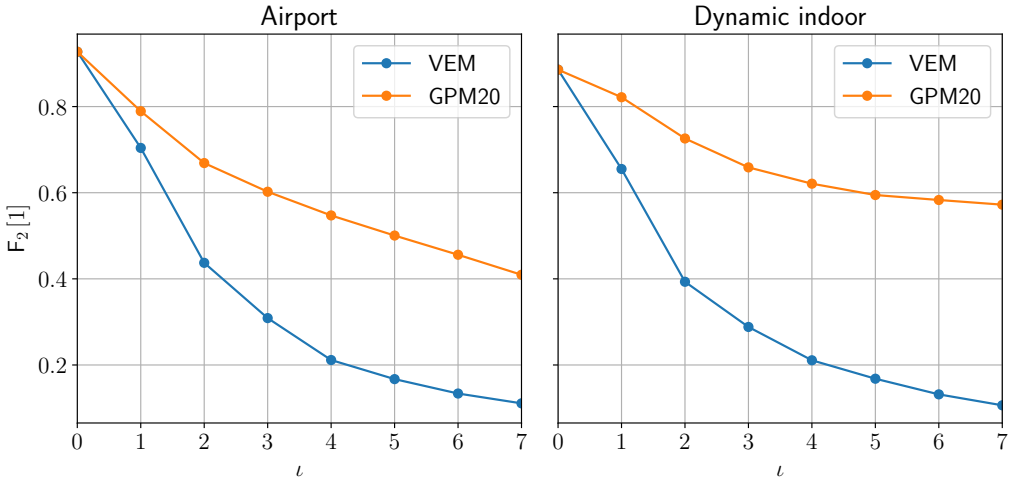


Figure 38: Comparison of the F_2-score of GPM20 and VEM on unknown test environments.

A direct performance comparison between GPM19 and GPM20 is impossible, as neither the data nor the model of GPM19 are open-source. Additionally the data of GPM19 is bonded to the vehicle perspective and results in being very different than the one of GPM20. Nonetheless, the study of GPM20's ability to generalize is a new assessment and an important property for a robotic applications, as some of the explored environments might be observed for the first time.

Additionally, regarding the model structure of GPM19 and GPM20, certain remarks can be done:

1. The reduced number of parameters of GPM20 improves the inference time.
2. For GPM20, the time dimension is involved during the encoding and decoding processes. In comparison to GPM19, this possibly leaves more freedom to learn a meaningful reduced representation.
3. GPM19 operates in a sequence-to-one fashion. This means that, to get the prediction at $\iota = 7$, the inference is first performed for all the previous ι and the result is then fed back as input. Thus, seven inferences are needed in total. In comparison, as GPM20 operates in a sequence-to-sequence way, a single inference is needed to obtain predictions until $\iota = 7$. This allows vectorizing some of the computations along the time dimension and contributes to the inference time improvement. Notice that the speed-up factor of 3.5, mentioned in section 6.1, would get larger with increasing n_ι.

A performant EMP is a great step towards a versatile DOA but remains useless if it is not correctly used by the control procedure. Therefore, in the next chapter, GPM20 is combined with SC-MPFC developed in Part I.

Chapter 7

Forecasting Safe-Corridor Path-Following NMPC

In the past chapter, an automated way of obtaining a "context-understanding" prediction of the future OGMs has been achieved. The goal is now to incorporate this knowledge into SC-MPFC. Therefore, the procedure is modified into a forecasting safe-corridor MPFC (FSC-MPFC). The global control strategy is summarized by Figure 39. It is emphasized that FSC-MPFC is a variant of the more general concept of EMP-MPFC.

In section 7.1, the formulation of FSC-MPFC is proposed. In section 7.2 it is compared to SC-MPFC, pointing out that taking the EMP into account allows a versatile DOA. The results of FSC-MPFC based on the true future motion and the occupancy estimations of GPM20 are compared.

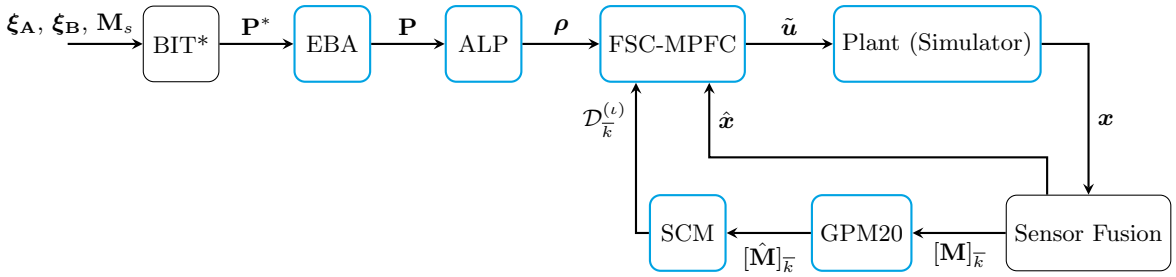


Figure 39: Control strategy on dynamic environments.

7.1 Forecasting Safe-Corridor Method

Making use of the EMP in the MPFC can be done in several ways. However, an important distinction can be done. Either the predicted occupancies are incorporated into the cost functional or into the state constraints. The former option can be understood as a soft-constraining, where estimated collisions are penalized but not necessarily forbidden. The latter delivers a more conservative approach by not allowing any collision with cells exceeding an occupancy probability of p_0. In the present work, the second option is chosen as interface between GPM20 and the MPFC.

The used methodology is explained below. In chapter 3, SCM has been introduced, in order to simplify the state constraints of the optimization. This procedure has already shown its ability to fit a fairly large convex region in a reasonable amount of time. However, it imposes the same state constraints

over the whole prediction horizon, even though the legal region $\mathcal{D}_{\overline{k}}$ is merely determined by the current OGM. This non-predictive approach w.r.t the environment motion is summarized in Equation 7.1.1 and appears to leave some room for improvements. Hereby, the operation "SCM" denotes the algorithmic use of SCM.

$$\text{Non-predictive constraining:} \quad \begin{aligned} \tilde{\boldsymbol{x}}_k &\in \mathcal{D}_{\overline{k}} \quad \forall\, k \in \left\{\overline{k}, ..., \overline{k} + H_p\right\} \\ \mathcal{D}_{\overline{k}} &= \texttt{SCM}(\mathcal{P},\, \boldsymbol{\xi}_{\overline{k}},\, \mathbf{M}_{\overline{k}}) \end{aligned} \tag{7.1.1}$$

The idea of FSC-MPFC is to apply SCM on the predicted OGMs $\hat{\mathbf{M}}_{\overline{k}}^{(\iota)}$, thus obtaining a convex legal state region $\mathcal{D}_{\overline{k}}^{(\iota)}$ for each time step $\iota \in \{0, 1, ..., H_p\}$. To optionally ignore the less reliable predictions made for distant time steps, a constraint horizon $H_{\mathcal{D}}$ is introduced. After this step, the legal region is hold constant. This approach is summarized in Equation 7.1.2. For the further considerations, $H_{\mathcal{D}} = 4$ is set, meaning that the predictions up to $t_k + 1.2\,\mathrm{s}$ are incorporated into the optimization problem. Previously, the constraints were time-variant, but constant for the time steps within an MPFC iteration. They are now also changing over the time steps of the MPFC iteration but are still hold constant over the optimization process.

$$\text{Predictive constraining:} \quad \begin{aligned} \tilde{\boldsymbol{x}}_{\overline{k}+\iota} &\in \mathcal{D}_{\overline{k}}^{(\iota)} \quad \forall\, \iota \in \{0, 1, ..., H_p\} \\ \mathcal{D}_{\overline{k}}^{(\iota)} &= \begin{cases} \texttt{SCM}(\mathcal{P},\, \boldsymbol{\xi}_{\overline{k}},\, \hat{\mathbf{M}}_{\overline{k}}^{(\iota)}), & \iota \leq H_{\mathcal{D}} \\ \mathcal{D}_{\overline{k}}^{(H_{\mathcal{D}})}, & \iota > H_{\mathcal{D}} \end{cases} \end{aligned} \tag{7.1.2}$$

The ellipse used as helper for forecasting SCM is built around the current position. If this procedure is successful for all time steps within the constraint horizon, it can be argued that the constraints can be systematically fulfilled. In fact, as expressed in Equation 7.1.3, the current position is an element of any predicted safe-corridor. This is an important assessment, as it states that, even though the state constraints are now changing over time for a single MPC step, they can be fulfilled if an ellipse can be fitted and if the rover is able to stay around its current position.

$$\boldsymbol{\xi}_m(t_{\overline{k}}) \in \bigcap_{\iota=0}^{n_\iota} \mathcal{D}_{\overline{k}}^{(\iota)} \Rightarrow \bigcap_{\iota=0}^{n_\iota} \mathcal{D}_{\overline{k}}^{(\iota)} \neq \emptyset. \tag{7.1.3}$$

As the rover yields a relatively low inertia, breaking procedures are fast. Thus, remaining in a small vicinity of the current position is no big challenge and the state constraints are fulfilled for cases where the ROSBot 2.0 does not drive at its full speed in the vicinity of dynamic obstacles. The blurred prediction of GPM20 tend to avoid such behaviors of the rover, as constraints are imposed in a rather conservative way.

In some cases, no ellipse can be fitted. In fact, the success of this procedure does not merely depend on the system and is largely determined by the environment. An exemplary situation of this on a dense, and thus challenging environment, is provided in section 7.2. This is not a major drawback, as the error message returned by SCM allows the rover to interrupt its motion. Thus, for these cases, collisions are recognized and and avoided by a static behavior rather than by a dynamic one.

In the next section, the simulation results of FSC-MPFC are studied.

7.2 Implementation and Results

In the present section, four control procedures are compared to each-other:

1. Presented in Part I, SC-MPFC is the non-predictive approach w.r.t. the environment motion.
2. GT-FSC-MPFC consists of the FSC-MPFC taking the ground truth of the future OGMs as input to fit the safe-corridors. Even though the exact knowledge of future OGMs is only possible for such offline examples, studying the results of GT-FSC-MPFC allows to understand how well FSC-MPFC performs, independently of the EMP delivered by GPM20.
3. FSC-MPFC$_{\mathsf{p}_0=0.1}$ consists of the FSC-MPFC combined with GPM20. The output of GPM20 is converted into a binary OGM with the threshold $\mathsf{p}_0 = 0.1$ and used for forecasting SCM.
4. FSC-MPFC$_{\mathsf{p}_0=0.03}$ consists of the same procedure, however with $\mathsf{p}_0 = 0.03$. Compared to $\mathsf{p}_0 = 0.1$, this means that lower occupancy probabilities are recognized as forbidden. Thus, the margins to pedestrians are expected to be larger and the control to be more conservative.

To illustrate their respective behavior on environments containing dynamic obstacles, the rover is required to follow various paths within the GPM20 test environments. As the rover is not present in the VADERE simulation, humans do not react to the presence of the robot. A good DOA is harder to achieve on such examples, as the other agents do not try to avoid the rover. In a later part of this section, the control performances are quantitatively compared over several examples. Before diving into this, the variety of manoeuvres enabled by EMP-MPFCs is shown on isolated examples. Notice that all the control parameters are the same as in Part I and can be found in Table 3.1.

Automatic Breaking

The first example is a sequence of the H terrain. For the four control procedures, videos of the results can be viewed `here`[1]. They are recommended to watch, as they give an intuitive understanding of the DOA. As depicted in Figure 40, $k = 14$ appears to be a critical time step because SC-MPFC leads to a collision with a pedestrian. This is due to the fact that SC-MPFC does not take the future environment motion into account during the constraint computation.

In contrary, none of the three predictive approaches present a collision. The rover is forced to slow down and lets the person pass by, which corresponds to a desired "polite" behavior. Notice that the breaking procedure occurs a couple of time steps before the rover even gets in a critical vicinity of the obstacle, as it can be observed for $k = 12$. In this sense, FSC-MPFC presents similarities with the environment motion predicting human strategy. Notice that the GPM20 prediction appears to be good enough for the DOA and that knowing the ground truth of the future motion is not needed.

Tuning p_0 is a simple way to achieve a more or a less conservative behavior. This is illustrated by the comparison between FSC-MPFC$_{\mathsf{p}_0=0.1}$ and FSC-MPFC$_{\mathsf{p}_0=0.03}$. Whereas GT-FSC-MPFC and FSC-MPFC$_{\mathsf{p}_0=0.1}$ present a similar margin to the avoided pedestrian, FSC-MPFC$_{\mathsf{p}_0=0.03}$ displays a larger one. Such a distance to humans might be desired. Not only the security requirement can be fulfilled with a higher robustness but the psychological effect on humans is positive. In fact, most pedestrians not only avoid collisions but tend to preserve a certain personal space around them. Whereas some parameter-tuning processes can be tedious, tuning p_0 involves a single value which is directly associated to the level of aggressivity of the control procedure. Notice that if GPM20 would have been designed as a binary classifier, this adaption capacity of FSC-MPFC would not be given.

[1] `https://www.youtube.com/watch?v=JCXgynKGBe4&list=PLTKb-pD4cu3iyjE1YiPTW3SlWxkP4BHzQ&index=10`

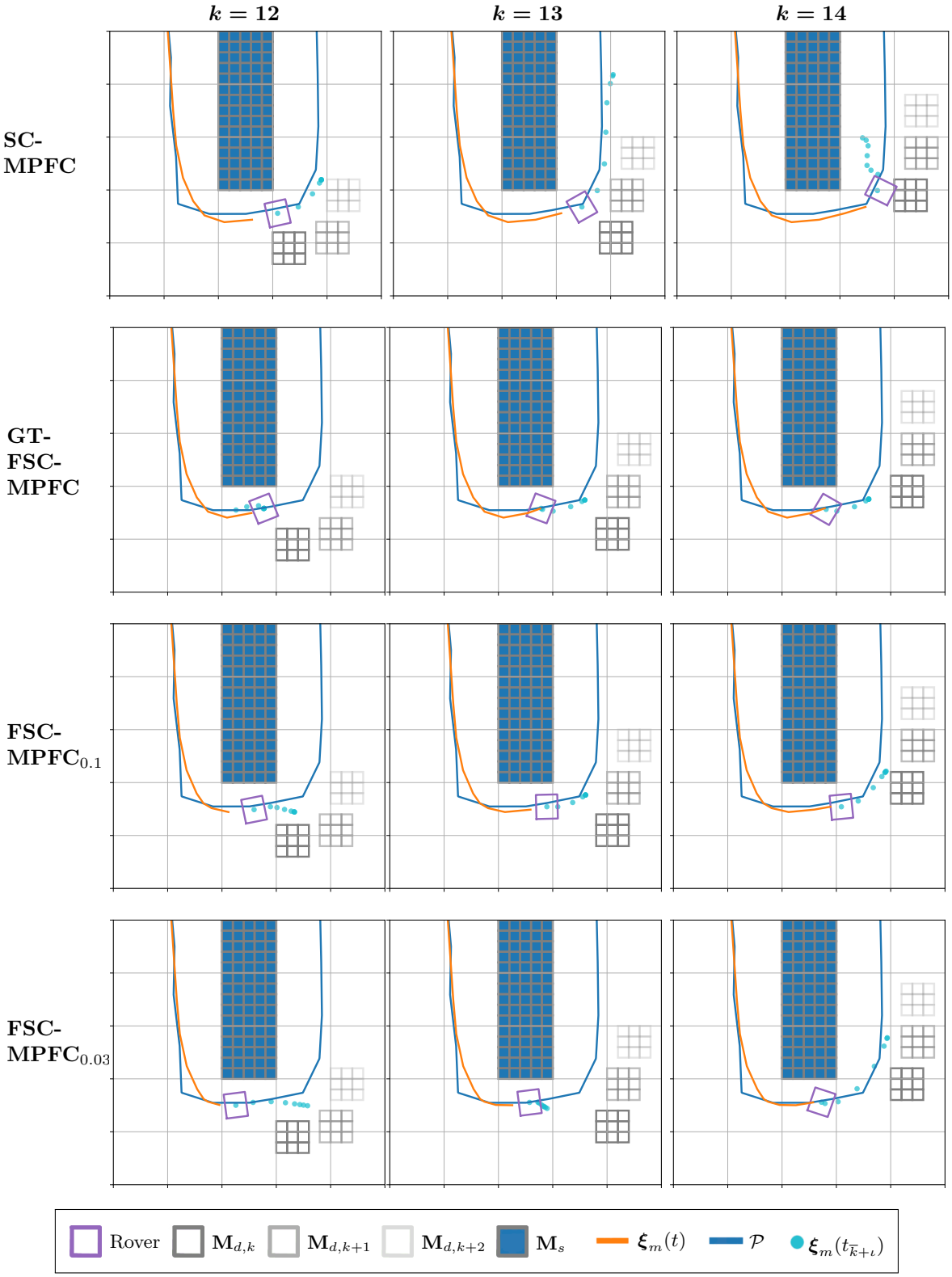


Figure 40: Exemplary situation requiring a breaking procedure.

Remark: Whereas the predicted position of the rover are represented for the whole prediction horizon (cyan dots), the positions of the dynamic obstacles are only represented for $\iota \in \{0, 1, 2\}$ for the sake of clarity. The opacity used to represent them decreases with increasing ι. Thus the most transparent gray cells represent the occupancy for $\overline{k} + 2$.

The path followed by the combination of GPM20 and FSC-MPFC is displayed in Figure 41. It shows that, even though pedestrians are successfully avoided, the path computed by the global planer is tracked with a similar precision than on a static environment (c.f. Part I). Additionally, whether $\mathsf{p}_0 = 0.1$ or $\mathsf{p}_0 = 0.03$, the spatial results are very similar. In the latter case the time to reach the goal point is $t_\mathrm{B} = 16.5\,\mathrm{s}$ and in the former one $t_\mathrm{B} = 16.2\,\mathrm{s}$. Notice that in the static case presented in Part I, $t_\mathrm{B} = 15.6\,\mathrm{s}$. Thus, for the conservative case, the time to target is only 4% larger than a case presenting no dynamic obstacles.

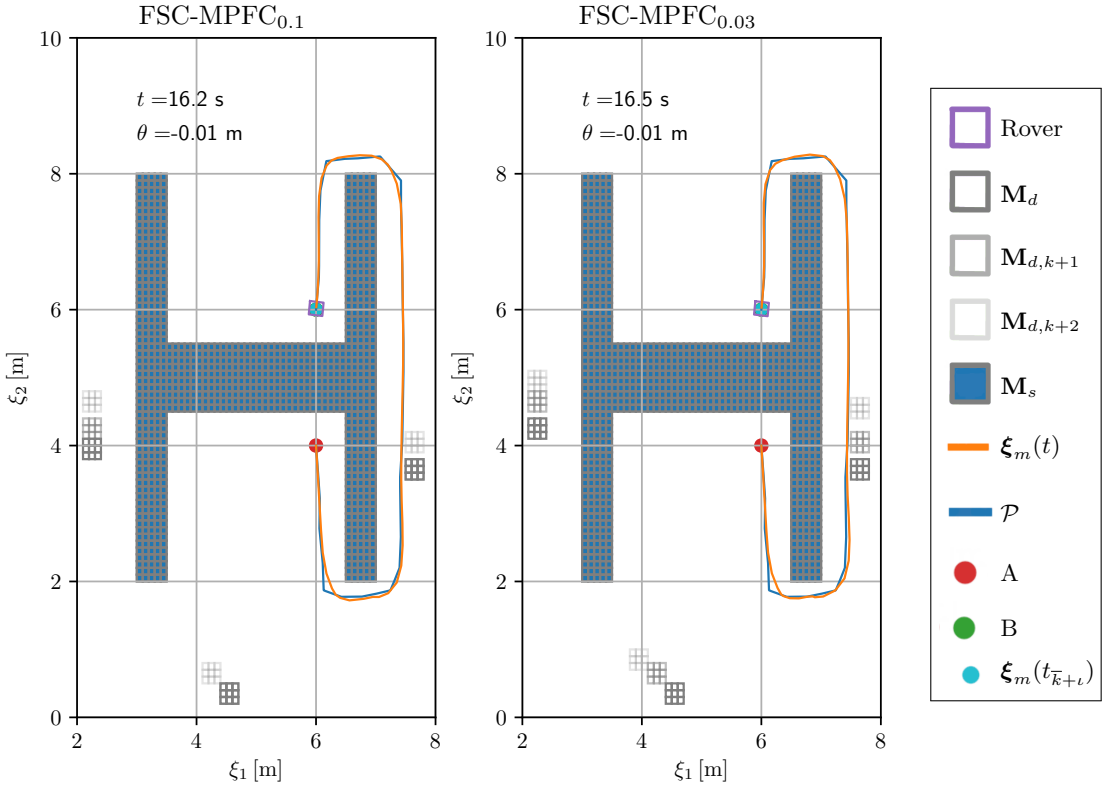


Figure 41: Final path followed by the two FSC-MPFC procedures based on the GPM20 predictions.

Free Space Recognition

A second advantage of predictive approaches is that they allow a recognition of the future free-space. To illustrate this, another sequence of the H example is simulated, where SC-MPFC does not lead to any collision. Thus, the time performance of FSC-MPFC can be compared to SC-MPFC, as no DOA is required. As depicted in Figure 42, some predicted positions of FSC-MPFC violate the displayed constraint $\mathcal{D}_{\overline{k}}^{(0)}$. This is the simple consequence of the fact that future positions $\boldsymbol{\xi}_m(t_{\overline{k+\iota}})$ only respect the corresponding future constraint $\mathcal{D}_{\overline{k}}^{(\iota)}$. As the pedestrian is walking upwards, its future motion liberates some space and the control recognizes that no breaking is needed. In comparison, SC-MPFC does not exploit this behavior. This leads GT-FSC-MPFC to outperform SC-MPFC, as they respectively yield a time to target of $t_\mathrm{B} = 15.6\,\mathrm{s}$ and $t_\mathrm{B} = 16.2\,\mathrm{s}$. The result for GT-FSC-MPFC is particularly interesting as it corresponds to the time needed on a static environment, as shown in Part I. This means that, based on the ground truth, the free space recognition given by the control method allows to move as fast as if no dynamic obstacles were present. In comparison, due to the blurred occupancy predictions of GPM20, FSC-MPFC$_{\mathsf{p}_0=0.03}$ reaches the goal at $t_\mathrm{B} = 15.9$. Thus, even for the conservative case, the performance results in being better than with a non-predictive approach. This effect can get particularly large if agents take similar paths and follow each-other, such as in corridors, lanes, ... etc.

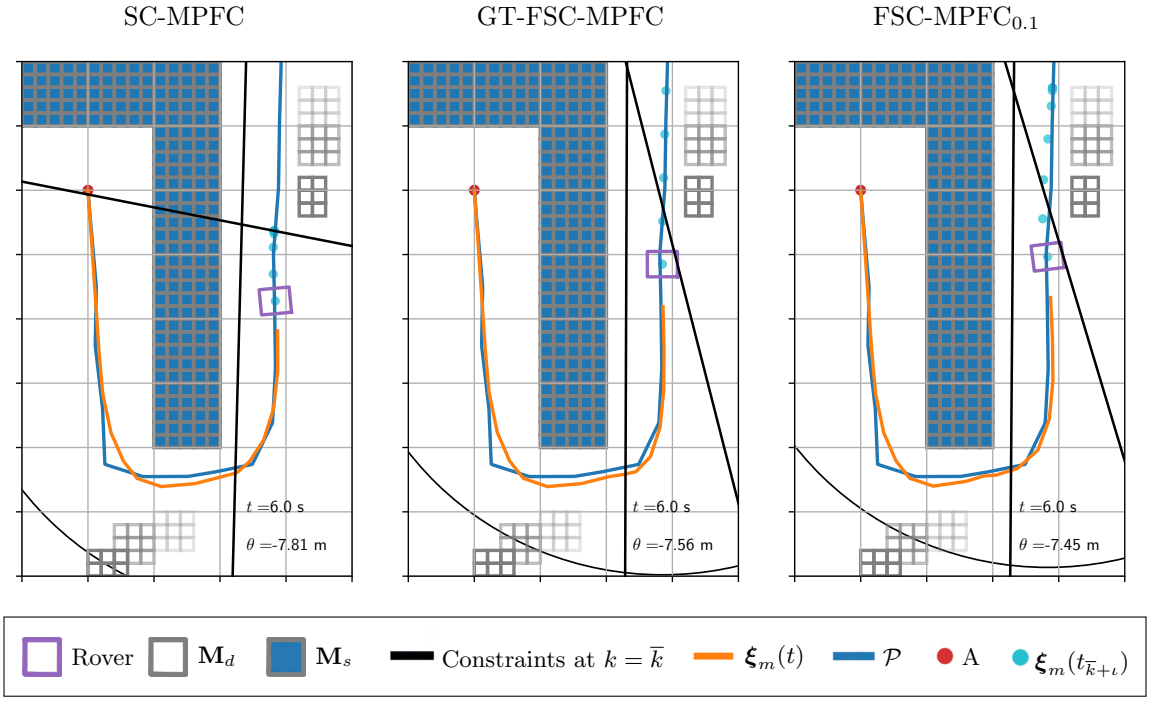


Figure 42: Exemplary situation of free space liberation by a pedestrian.

Transversal Manoeuvres

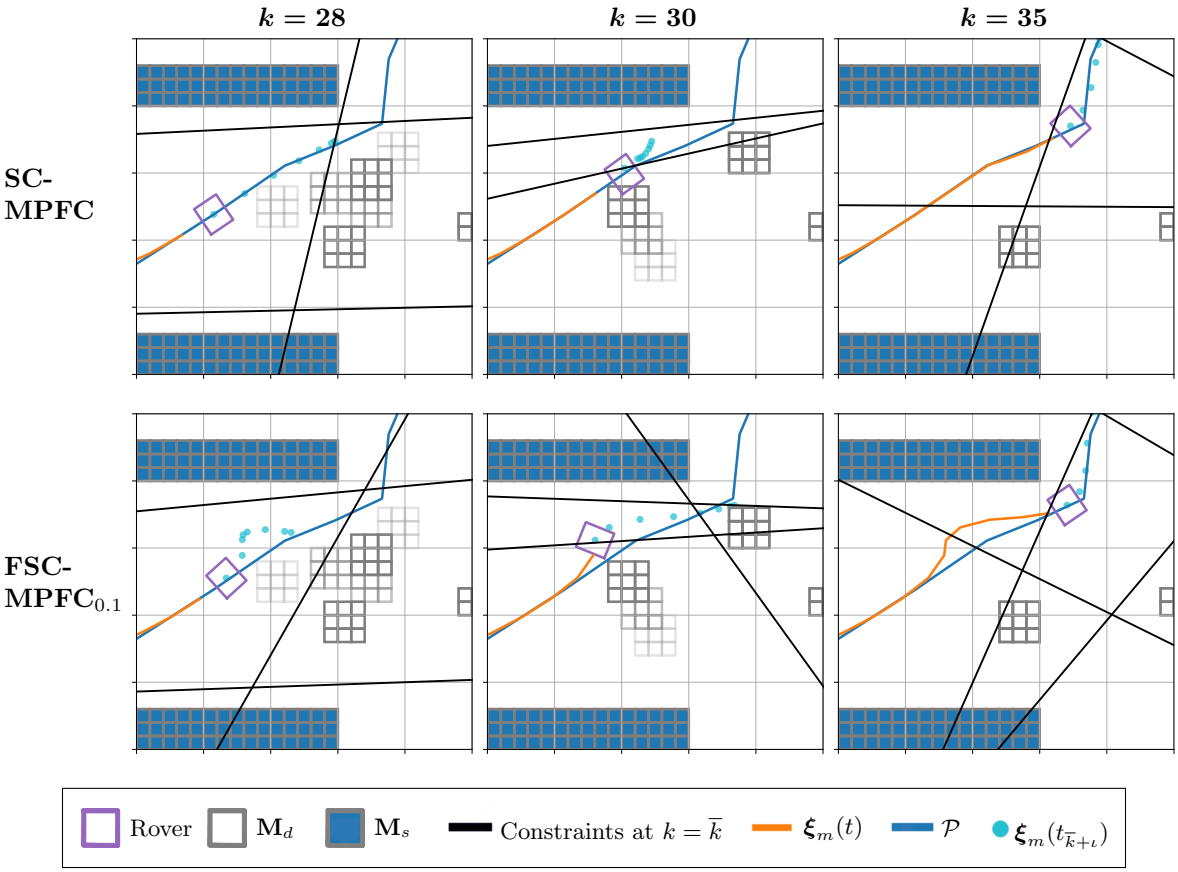


Figure 43: Exemplary situation of a transversal manoeuvre computed by FSC-MPFC$_{0.1}$.

Until now, only changes in the longitudinal dynamic of the rover have been observed. However, FSC-MPFC also allows more complex avoidance trajectories as it is depicted in Figure 43. Here, a sequence of the hall example is simulated. It appears that FSC-MPFC computes a detour around the dynamic obstacle and allows the rover to avoid the collision while still moving forward. This time the DOA takes place in the transversal direction rather than in the longitudinal one. This shows a certain versatility of the method, deciding which of both avoidance strategies is the most sensed depending on the case. Here again, the non-predictive approach leads to a collision as it is shown in Figure 43.

Achievements of FSC-MPFC

Examples of collision avoidances on the other test environments are illustrated in section F of the appendix and show that FSC-MPFC$_{\mathsf{p}_0=0.1}$ is able to generalize to various situations. The comparative videos for SC-MPFC, GT-FSC-MPFC and FSC-MPFC$_{\mathsf{p}_0=0.1}$ can be found `here`[2]. The total number of collisions over the scenarios and the mean time to reach B are respectively denoted by $n_{\text{collision}}$ and $\bar{t}_{\text{B}}$. For three of the control procedures, these values are computed and gathered in Table 7.1. Notice that the time to target is not defined for SC-MPFC, as collisions have occurred.

Remark: FSC-MPFC$_{\mathsf{p}_0=0.03}$ is not considered for the present comparison, as the less conservative FSC-MPFC$_{\mathsf{p}_0=0.1}$ yields to better results without leading to any collision.

	SC-MPFC	GT-FSC-MPFC	FSC-MPFC$_{\mathsf{p}_0=0.1}$
$\bar{t}_{\text{B}}$	not defined	19.6s	19.6 s
$n_{\text{collision}}$	8	0	0

Table 7.1: Comparison of $n_{\text{collision}}$ and $\bar{t}_{\text{B}}$ for three control procedures.

Table 7.1 and the associated videos show that the primary goal of DOA is completed, as no collisions occur for the two predictive approaches. This can be expected from GT-FSC-MPFC but is not an evidence while using GPM20. In fact false negatives could lead to a collision. The absence of such collisions is mostly due to the fact, that the F_2-score of GPM20 is particularly high for small values of ι. Moreover, for the GPM20 test cases, the mean value $\bar{t}_{\text{B}}$ is the same, whether the ground truth or the GPM20 prediction is used for the constraint forecasting. Thus, for the present control purpose, relying on GPM20 appears to be as good as relying on the ground truth.

A further result, outlined by the videos and the spatial plots given in section F of the appendix, is that MPFC performs well on the GPM20 test environments without changing any of the control parameters. Even though $\mathbf{Q}_z$ and $\mathbf{R}_{\Delta\nu}$ were tuned based on the H, the indoor, the maze and the outdoor examples, the feedback loop seems to generalize well enough on other environments.

Recalling the goals set in the introduction of this thesis, it appears that FSC-MPFC operates as wished:

1. If a collision is about to happen, it is forecasted by GPM20 and adaptations are automatically computed within FSC-MPFC.
2. The performance of GT-FSC-MPFC and FSC-MPFC$_{\mathsf{p}_0=0.1}$ is comparable.

[2] `https://www.youtube.com/watch?v=hbSWOihMsGo&list=PLTKb-pD4cu3iyjE1YiPTW3SlWxkP4BHzQ&index=11`

3. The control and the EMP both perform well on environments they have not been tuned or trained on. The generalization ability of FSC-MPFC might be its most significant achievement.
4. Future free space liberation is recognized and the rover does not break unnecessarily.
5. The margins to dynamic obstacles can be tuned by a single parameter $\mathtt{p}_0$.

At this point, an advantage linked to the use of MPFC shall be reminded. Compared to SC-MPFC, the only adaption in the control architecture of FSC-MPFC is the use of several safe-corridors instead of a single one. If, as shown in section 1.4, a partitioned approach to trajectory-shaping and trajectory-tracking would have been used, a meaningful way to adapt the trajectory-shaping procedure would be required. As both tasks are handled in a monolithic way within the optimization this is automatically computed and fulfils the (possibly only local) optimality criteria.

For the state of the art in the field of MPFC, this is the first implementation able to deal with dynamic environments. In addition, it should be mentioned that FSC-MPFC represents the first combination of control and next-frame prediction in the literature. Finally, no other work on a constraint-forecasted NMPC have been found. Thus, a new research track is opened by the present work. As it is a new approach, many methodological improvements are still possible, as discussed below.

Limitations

As much as the robot was not observed to bump into pedestrians, some simulations display humans bumping into the rover. Such a situation on the street scenario is depicted in Figure 44. The ground truth was assumed to be known, as it represents the best-case scenario w.r.t. the EMP. Here, the robot stopped because of the collision recognition. However, FSC-MPFC does not manage to compute an avoidance, leading the rover to "freeze". The collision arises in simulation because pedestrians do not take the rover into account. For real situations, it can be assumed that pedestrians are able to avoid a static rover and that it is not the robot's task to systematically go out of the way.

The corresponding video of the simulated street scenario can be found `here`[3]. It appears that the rover manages to avoid 5 out of 6 collisions and that, as mentioned, the only failure is a simulation artefact. Even though this example outlines a limitation, it also points out that FSC-MPFC can even deal with relatively dense dynamic environments.

Possible Improvements

As already mentioned, the blurred occupancies predicted by GPM20 are essentially a desired effect. However, for particularly dense environments, this can easily lead to a freezing rover, as the whole free-space might be recognized as likely to be occupied in the future. This might be tackled by a less conservative EMP. Alternatively, this can also be solved within the control procedure, as explained below.

FSC-MPFC converts the EMP into hard constraints. In comparison, a soft-constraining method would rely on the introduction of costs able to penalize future collisions rather than completely forbid them. This could present some advantages:

[3] `https://www.youtube.com/watch?v=7q6jCY-8e_c&list=PLTKb-pD4cu3iyjE1YiPTW3SlWxkP4BHzQ&index=17`

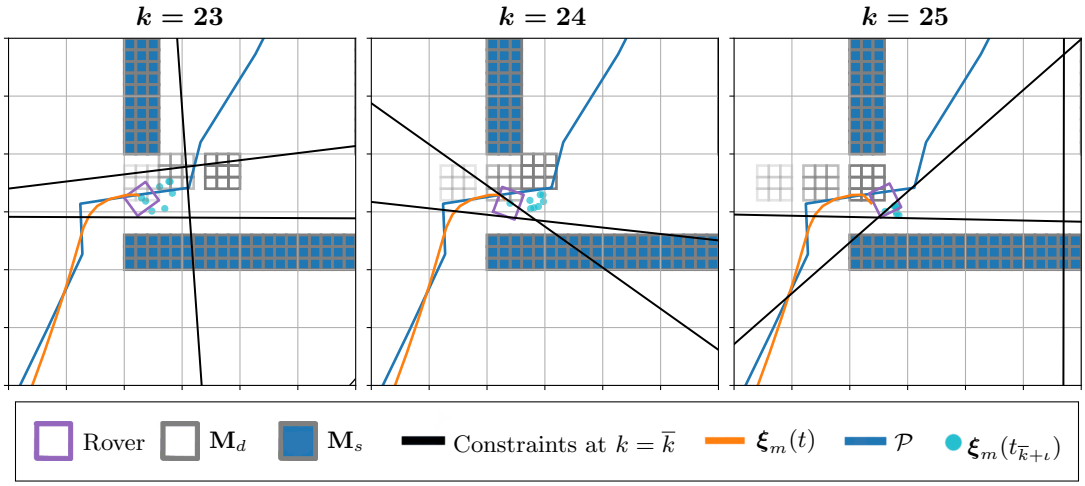


Figure 44: Exemplary "freezing-robot" situation with GT-FSC-MPFC.

1. By weighting a collision-cost function with the future occupancy probabilities, a soft-constraining approach could make full use of the output value range of GPM20. Cells that are very likely to be occupied would be automatically avoided, while positions displaying a small collision probability could still be driven through.

2. As the maximal output value of GPM20 decreases over the prediction time step ι, further time steps would be less weighted. This means that further EMPs lead to softer restrictions than nearer ones, which is very similar to the human DOA strategy. Because of this effect, introducing a constraint horizon $H_{\mathcal{D}}$ might not be needed any more.

3. As no hard constraints are applied, it does not have to be proven that they are systematically fulfilled. Thus, the rather informal argumentation provided in section 7.1 would not be necessary.

However, soft-constraining requires to shape a suited collision-cost function, which is likely to lead to further non-convexities in the control objective while introducing more parameters to tune. Because of the restrained time scope, the present work could not cover this interesting research track which is left for future works.

Conclusions

Outcomes

First, the present work led to a valid SSM of the ROSBot 2.0, allowing the use of model-based control techniques. To ease the optimization inherent to NMPC, several state-of-the-art techniques were combined:

1. EBA has been implemented, allowing to obtain an improved path w.r.t. to its smoothness and its average distance to obstacles.
2. A generic approach to the path parametrization has been achieved by ALP, combined with mechanical considerations. It allows to give a physical signification to the path parameter θ and to pose the optimization problem in a meaningful way. It avoids arbitrary choices for the dynamic, the initial value and the constraints of θ, which do not have to be tuned.
3. SCM has been implemented, leading to larger free convex regions than with a disk approximation.

In the literature, no example can be found of a joint use of these techniques in combination with MPFC. Together, they led to a novel achievement for the state of the art, as SC-MPFC is able to deal with arbitrarily shaped static environments. Further properties of this control procedure make it appealing:

1. It is suited for diverse paths and environments with a single set of control parameters.
2. Compared to a disk approximation of the free space, the surface improvement delivered by SCM appeared to have significant repercussions on the time performance of the control.
3. Even for relatively high disturbances, collisions with static obstacles are avoided. Hereby, the path remains tracked with a sufficient precision and the loss in time performance is negelectable.
4. The procedure is real-time efficient. However, speeding it up can increase the robustness and the reactivity of SC-MPFC.

In the second part of the thesis, the focus shifted to working in dynamic environments. To be able to forecast the environment motion, GPM20 was trained on simulation data. In addition to its clear performance improvement compared to a VEM, it presents many other advantages:

1. It presents a simpler architecture than GPM19, as it only uses one decoder. By the use of 2D-ConvLSTM and 3D-convolutions, it takes the time dimension into account at encoding and decoding time. This allowed to perform well, even with a shallower network than GPM19. Finally, the sequence-to-sequence input-output format allows a vectorized inference.

2. Compared to model-based techniques, it does not require to transform the OGM into another representation. No labelling of the objects is needed and the training is fully automatised.
3. The output of GPM20 is a probabilistic OGM, thus allowing "blurred" and multi-modal predictions. They allow to be more conservative for uncertain situations and represent a highly desired property.
4. The generality of NNs allows GPM20 to be suited for any kind of agent and environment, as long as the training data are accordingly provided.

Many examples of such GPMs have already been published. However, all of them have been developed as stand-alone models. The current thesis presents a novel achievement by combining it with a control method. To allow this, SC-MPFC has been modified into an FSC-MPFC, where a safe-corridor is not only fitted for the current OGM, but also for the future ones, forecasted by GPM20. This essentially results in incorporating estimated future state constraints into the optimization.

As shown in the last chapter, FSC-MPFC is able to avoid dynamic obstacles that could not have been avoided with SC-MPFC. The performance obtained relying on GPM20 appeared to be close to the one obtained with the knowledge of the future ground truth. By decreasing p_0, larger margins to pedestrians have been obtained. The compelling aspect of FSC-MPFC, is that it relies on a minimal number of hand-tuned parameter. The DOA is performed in a fully automated way and was shown to work for different situations of potential collision without needing any adaption.

The current work presents the first implementation of MPFC on arbitrary environments and the first combination of video-frame prediction with optimization-based control. As already mentioned, FSC-MPFC can be considered as an extension of the predictive nature of NMPC to the state constraints imposed by the motion of the environment. FSC-MPFC seems to be the first example of such a constraint-forecasted NMPC. Moreover, the developed techniques are by no mean restricted to rovers, nor to robots or to path-following tasks. In fact, the FSC-MPFC is a valid framework for any path-following task and the idea of constraint forecasting can be applied on any MPC.

Future Work

The present thesis not only achieved to deliver a proof of concept for an EMP-MPFC, but also opens several doors for future works. In the field of control theory, one can think of:

1. **A proof of stability.** Proving the stability of SC-MPFC would be a major achievement and might require adaptions of the control procedure, such as the introduction of suited end costs.
2. **An implementation on the real system.** SC-MPFC is only a step away of being implemented on the ROSBot 2.0. This would represent a great improvement compared to the current PID-controller of the rover.
3. **A slip compensation.** Especially if slip arises at the contact point between wheels and ground, a more complex SSM might be needed. An online estimation of this effect combined with a robust formulation of NMPC could be a great improvement. Time efficient approaches to NMPC relying on parameter estimation have been proposed in [9] and [37].
4. **An EMP-MPFC based on soft-constraining.** As mentioned in section 7.2, a soft-constraining approach could make a full use of the output value range of GPM20.

In the field of deep learning, possible improvements might be:

1. **Train on data including the perspective and the own motion.** For most of the robotic application, the mapping process underlies a certain perspective. Additionally, as the robot moves, its perspective changes over time. Both of these effects are easy to simulate and could be applied on the training data to obtain situations that are closer to reality.
2. **Train on real data.** Gathering real data seems the most reasonable option to properly train the GPM. It is the best way to include the reaction of other agents to the robot motion in the data set. However, this is by far more expansive in time than the simulated option and might be left aside until good results are obtained on virtual environments.
3. **Depth estimation by GPM.** If the GPM is trained with suited data, it could additionally deliver a depth estimation of the obstacles in the OGMs. This could be a major helper for the path planner, as more information about the environment would be available.
4. **Integrating computer vision to GPM.** Throughout the current work, the camera has only been used to densify the point cloud generated by the LIDAR. However, computer vision could also be used for segmentation and labelling tasks. Furthermore, it could be used as input of a modified GPM20 and allow to recognize additional features, such as the pedestrian's orientation. This could be a great improvement to estimate the likelihood of future occupancies.

Finally, an EMP-MPFC could be implemented on a real rover. A comparison to a reinforcement learning control could outline the advantages and drawbacks of both approaches.

References

[1] Intuitive. Da Vinci - Surgeon Assistant. https://www.intuitive.com/en-us/products-and-services/da-vinci/systems, last access: 2020/09/14.

[2] Toshiba. ROV Investigates Fukushima Daiichi Unit 3. https://www.toshiba-energy.com/en/nuclearenergy/topics/fukushima-robot.htm, last access: 2020/09/18.

[3] Nasa. Raw images of mars rover curiosity. https://mars.nasa.gov/msl/multimedia/raw-images/?order=soldesc,instrument_sortasc,sample_type_sortasc,date_takendesc&per_page=50&page=2&mission=msl, Last access: 2020/09/14 2020.

[4] Husarion. ROSBot 2.0 Documentation. https://husarion.com/manuals/rosbot-manual/, last access: 2020/08/31.

[5] A. I. Propoi. "Use of Linear Programming Methods for Synthesizing Sampled-Data Automatic Systems". *Autom. Rem. Control*, 24(7):837–844, 1963.

[6] E. B. Lee and L. Markus. *Foundations of Optimal Control Theory.* New York, Wiley, 1967.

[7] J. Richalet, A. Rault, J. L. Testud, and J. Papon. "Model Predictive Heuristic Control: Applications to Industrial Processes". *Automatica*, 14(5):413 – 428, 1978.

[8] C. E. Garcia, D. M. Prett, and M. Morari. "Model Predictive Control: Theory and Practice - A Survey". *Automatica*, 25(3):335–348, 1989.

[9] Thomas Heine, Michael Kawohl, and Rudibert King. "Robust Model Predictive Control using the Unscented Transformation". *Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control*, 2006.

[10] T. Faulwasser. *NMPC for Tracking and Path Following.* PhD thesis, Fakultät für Elektrotechnik und Informationstechnik, Otto-von-Guericke-Universität Magdeburg, 2012.

[11] J. B. Rawlings, D. Q. Mayne, and M. M. Diehl. *Model Predictive Control: Theory, Computation and Design. 2nd Edition.* Nob Hill Publishing, 2019.

[12] R. King. "Nichtlineare Regelungen". Fachgebiet Mess- und Regelungstechnik, TU Berlin, 2019.

[13] R. King. "Mehrgrößenregelung im Zeitbereich". Fachgebiet Mess- und Regelungstechnik, TU Berlin, 2019.

[14] K. Graichen. "Methoden der Optimierung und Optimalen Steuerung". Institut für Mess–, Regel– und Mikrotechnik. Fakultät für Ingenieurwissenschaften und Informatik. Universität Ulm, 2017.

[15] P. T. Boggs and J. W. Tolle. "Sequential Quadratic Programming". *Acta Numerica*, 4(4):1–51, 1995.

[16] R. H. Byrd, M. E. Hribar, and J. Nocedal. "An Interior Point Algorithm for Large-Scale Nonlinear Programming". *SIAM Journal on Optimization*, 9(4):877–900, 1999.

[17] A. Wächter. *An Interior Point Algorithm for Large-Scale Nonlinear Optimization with Applications in Process Engineering.* PhD thesis, Carneggie Melon University, 2002.

[18] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot. "Batch Informed Trees (BIT*): Sampling-based Optimal Planning via the Heuristically Guided Search of Implicit Random Geometric Graphs". *ArXiv: 1405.5848v7*, 2015.

[19] M. Moll I. A. Şucan and L. E. Kavraki. "The Open Motion Planning Library". *IEEE Robotics & Automation Magazine*, 19(4):72–82, 2012.

[20] R. Krenn et al. "Model Predictive Traction and Steering Control of Planetary Rovers". *12th Symposium on Advanced Space Technologies in Robotics and Automation*, 2013.

[21] K. Belda and O. Rovný. "Predictive Control of 5 DOF Robot Arm of Autonomous Mobile Robotic System". *21st International Conference on Process Control*, 2017.

[22] S. Quinlan and O. Khatib. "Elastic Bands: Connecting Path Planning and Control". *Proceedings IEEE International Conference on Robotics and Automation*, 1993.

[23] A. Thombra. "Optimization-Based Motion-planning Using Nonlinear Model Predictive Control". Master's thesis, Hamburg University of Technology, 2020.

[24] Timm Faulwasser et al. "Implementation of Nonlinear Model Predictive Path-Following Control for an Industrial Robot". *ArXiv: 1506.09084v2*, 2016.

[25] M. H. Arbo, E. I. Grøtli, and J. T. Gravdahl. "On Model Predictive Path Following and Trajectory Tracking for Industrial Robots". *ArXiv: 1703.02279v1*, 2017.

[26] B. Hall. "Sensitivity-Based Economic NMPC with a Path-Following Approach in Python". Master's thesis, Norwegian University of Science and Technology, 2017.

[27] Shuyou Yu et al. "MPC for Path Following Problems of Wheeled Mobile Robots". *IFAC Papers Online*, 51(20):247–252, 2018.

[28] R. King. "Struktur- und Parameteridentifikation". Fachgebiet Mess- und Regelungstechnik, TU Berlin, 2017.

[29] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics.* MIT Press, 2005.

[30] Sikang Liu et al. "Planning Dynamically Feasible Trajectories for Quadrotors Using Safe Flight Corridors in 3-DComplex Environments". *IEEE Robotics an Automation Letters*, 2(3):1688 – 1695, 2017.

[31] P. Renteln. *Manifolds, Tensors and Forms.* Cambridge University Press, 2014.

[32] Kartik Mohta et al. "Fast, Autonomous Flight in GPS-Denied and Cluttered Environments". *ArXiv: 1712.02052v1*, 2018.

[33] J. A. E. Andersson et al. "CasADi – A Software Framework for Nonlinear Optimization and Optimal Control". *Mathematical Programming Computation*, 11(1):1–36, 2019.

[34] "HSL. A Collection of Fortran Codes for Large Scale Scientific Computation.". `http://www.hsl.rl.ac.uk/`, October 2020.

[35] M. Sznaier and M. J. Damborg. "Suboptimal Control of Linear Systems with State and Control Inequality Constraints". *26th IEEE Conference on Decision and Control*, 26:761–762, 1987.

[36] M. Böck et al. "Fortgeschrittene Methoden der Nichtlinearen Regelung". Technische Universität Wien, Institut für Automatisierungs- und Regelungstechnik, 2016.

[37] S. Lucia, T. Finkler, and S. Engell. "Multi-Stage Nonlinear Model Predictive Control Applied to a Semi-Batch Polymerization Reactor Under Uncertainty.". *Journal of Process Control*, 23(9):1306–1319, 2013.

[38] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[39] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". *ArXiv: 1506.02640v5*, 2016.

[40] Z. Long, Y. Lu, and B. Dong. "PDE-Net 2.0: Learning PDEs from Data with a Numeric-Symbolic Hybrid Deep Network. *ArXiv: 181204426v2*, 2019.

[41] Tom B. Brown et al. "Language Models are Few-Shot Learners". *ArXiv: 2005.14165v4*, 2020.

[42] S. Hörmann, M. Bach, and K. Dietmayer. "Dynamic Occupancy Grid Prediction for Urban Autonomous Driving: A Deep Learning Approach with Fully Automatic Labeling". *ArXiv: 1705.08781v2*, 2017.

[43] M. Schreiber, S. Hörmann, and K. Dietmayer. "Long-Term Occupancy Grid Prediction Using Recurrent Neural Networks". *ArXiv: 1809.03782v2*, 2019.

[44] Y. Lecun et al. "Gradient-Based Learning Applied to Document Recognition". *Proceedings of the IEEE*, 86(11):2278 – 2324, 1998.

[45] S. Hochreiter and J. Schmidhuber. "Long Short-Term Memory". *Neural computation*, 9(8):1735–1780, December 1997.

[46] X. Shi et al. "Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting". *ArXiv: 1506.04214v2*, 2015. arXiv: 1506.04214.

[47] C. C. Aggarwal. *Neural Networks and Deep Learning*. Springer, 2018.

[48] Andrew Ng. *Machine Learning Yearning*. 2018.

[49] D. P. Kingma and J. L. Ba. "ADAM: A Method For Stochastic Optimization". *ArXiv: 1412.6980v9*, 2015.

[50] S. Ioffe and C. Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". *ArXiv: 1502.03167v3*, 2015.

[51] N. Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[52] K. Cho et al. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". *ArXiv: 1406.1078v3*, 2014.

[53] A. Vaswani et al. "Attention Is All You Need". *ArXiv: 1706.03762*, 2017.

[54] N. Ballas et al. “Delving Deeper into Convolutional Networks for Learning Video Representation”. *ArXiv: 1511.06432v4*, 2016.

[55] Y. Liu et al. “Time Series Prediction Based on Temporal Convolutional Network”. *IEEE/ACIS 18th International Conference on Computer and Information Science (ICIS)*, 2019.

[56] Sandra Aigner and Marco Körner. “FutureGAN: Anticipating the Future Frames ofVideo Sequences using Spatio-Temporal 3dConvolutions in Progressively Growing GANs”. *ArXiv: 1810.01325v2*, 2018.

[57] R. Güldenring. “Applying Deep Reinforcement Learning in the Navigation of Mobile Robots in Static and Dynamic Environments”. Master’s thesis, Universität Hamburg, 2019.

[58] Y. Zhang, X. Wei, and X. Zhou. “Dynamic Obstacle Avoidance Based on Multi-Sensor Fusion and Q-Learning Algorithm”. *IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference*, 2019.

[59] E. Galceran et al. “Multipolicy Decision-Making for AutonomousDriving via Changepoint-based Behavior Prediction”. *Autonomous Robot*, 41:1367–1382, 2017.

[60] B. Kim et al. “Probabilistic Vehicle Trajectory Prediction overOccupancy Grid Map via Recurrent Neural Network”. *IEEE 20th International Conference on Intelligent Transportation Systems*, 2017.

[61] H. Cui et al. “Multimodal Trajectory Predictions for AutonomousDriving using Deep Convolutional Networks”. *International Conference on Robotics and Automation*, 2019.

[62] R. Greer, N. Deo, and M. Trivedi. “Trajectory Prediction in Autonomous Driving with a Lane HeadingAuxiliary Loss”. *ArXiv: 2011.06679v1*, 2020.

[63] L. Hou. “Interactive Trajectory Prediction of SurroundingRoad Users for Autonomous Driving UsingStructural-LSTM Network”. *IEEE Transactions on Intelligent Transportation Systems*, 21(11):4615–4625, 2020.

[64] Y. Ma et al. “TrafficPredict: Trajectory Prediction for Heterogeneous Traffic-Agents”. *Thirty-Third AAAI Conference on Artificial Intelligence*, 2019.

[65] Stefan Hoermann et al. “Object Detection on Dynamic Occupancy Grid Maps Using Deep Learning and Automatic Label Generation”. *IEEE Intelligent Vehicles Symposium (IV) Changshu, Suzhou, China*, 2018.

[66] Nico Engel et al. “Deep Object Tracking on Dynamic Occupancy Grid Maps Using RNNs”. *21st International Conference on Intelligent Transportation Systems (ITSC) Maui, Hawaii, USA*, 2018.

[67] K. He et al. “Deep Residual Learning for Image Recognition”. *ArXiv: 1512.03385v1*, 2015.

[68] Dominik Nuss et al. “A Random Finite Set Approach for Dynamic Occupancy Grid Maps with Real-Time Application”. *ArXiv: 1605.02406v2*, 2016.

[69] M. J. Seitz and G. Köster. “How Update Schemes Influence Crowd Simulations”. *Journal of Statistical Mechanics: Theory and Experiment*, 2014.

[70] I. von Sivers and G. Köster. “How Stride Adaptation in Pedestrian Models Improves Navigation”. *ArXiv: 1401.7838v1*, 2014.

[71] I. von Sivers and G. Köster. "Dynamic Stride Length Adaptation According to utility and Personal Space". *ArXiv: 1401.7838v3*, 2015.

[72] I. von Sivers et al. "Modelling Social Identification and Helping in Evacuation Simulation". *Safety Science*, 89:288–300, 2016.

[73] B. Kleinmeier et al. "Vadere: An Open-Source Simulation Framework to Promote Interdisciplinary Understanding". *ArXiv: 1907.09520v1*, 2019.

[74] D. Powers. "Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation". *Mach. Learn. Technol.*, 2, 2008.

Appendix

A Used Libraries and Frameworks

General

- PYTHON 3.7 has been used as a common interface for all the other libraries and framework.
- NUMPY 1.18.1 and SCIPY 1.4.1 were used for the numerical computation done within Python. MATPLOTLIB 3.1.3 was used for visualization purposes.
- ROS KINETIC was used for the communication with the ROSBOT 2.0 during the parameter identification process.

Optimization

- For algorithmic differentiation the Python interface of CASADI 3.5.1 has been used.
- The optimization problem formulated by CASADI was passed to IPOPT 3.13.2, a nonlinear optimizer using an interior point method.
- The interior point methods needs to iteratively solve linear sub-problems. This was handled by the MA27-solver of HSL.

Machine Learning

- TENSORFLOW 2.2.0 has been used as back-end for building and training the model.
- KERAS 2.4.3 has been used as high-level interface with TENSORFLOW, allowing a faster and simpler development of the deep learning strategy.
- TENSORBOARD has been used for monitoring during the training process.

B Pseudo-Codes of the Elastic Band Approach

Remark 1: As all codes are written in PYTHON 3.7, arrays are zero-indexed.

Remark 2: The written algorithms are vectorized. For example, algorithm 1 can have scalar or vector-valued input/output.

Remark 3: Recall that:

$$\mathbf{P}^* = \Big[\boldsymbol{\xi}_{(1)} = \boldsymbol{\xi}_{(A)}, \quad \boldsymbol{\xi}_{(2)}, \quad \dots, \quad \boldsymbol{\xi}_{(n_P-1)}, \quad \boldsymbol{\xi}_{(n_P)} = \boldsymbol{\xi}_{(B)}\Big].$$

Algorithm 1: Disk-Fitting of the Free Space

Function `Disk`($\overline{\boldsymbol{\xi}}$, r):
  return $\left\{\boldsymbol{\xi} \in \mathbb{R}^2 : ||\boldsymbol{\xi} - \overline{\boldsymbol{\xi}}||_2 \leq r\right\}$

Function `FitDisk`($\overline{\boldsymbol{\xi}}, \mathcal{C}_1, r_{\min}, \delta_r$):
  $r \leftarrow r_{\min}$
  while `Disk`($\overline{\boldsymbol{\xi}}$, $r + \delta_r$) $\cap\, \mathcal{C}_1 = \emptyset$
    $r \leftarrow r + \delta_r$
  $\boldsymbol{\xi}_o \leftarrow$ `mean`(`Disk`($\overline{\boldsymbol{\xi}}$, $r + \delta_r$) $\cap\, \mathcal{C}_1$)
return $r, \boldsymbol{\xi}_o$

Algorithm 2: Point Spawning and Pruning

Function `SpawnPoints`($\mathbf{P}^*, \boldsymbol{r}, \boldsymbol{\xi}_o$):
  $i = 0$
  while $i <$ `len`($\mathbf{P}^*$):
    if $\|\boldsymbol{\xi}_{(i+1)} - \boldsymbol{\xi}_{(i)}\|_2 < \boldsymbol{r}_{(i+1)} + \boldsymbol{r}_{(i)}$:
      $\mathbf{P}^* \leftarrow$ `insert`$\Big(0.5\,(\boldsymbol{\xi}_{(i+1)} + \boldsymbol{\xi}_{(i)}), \mathbf{P}^*, i+1\Big)$
      $\boldsymbol{r} \leftarrow$ `insert`$\Big($`FitDisk`$(\boldsymbol{\xi}_{(i+1)}), \boldsymbol{r}, i+1\Big)$
    $i \leftarrow i + 1$
return $\mathbf{P}^*, \boldsymbol{r}, \boldsymbol{\xi}_o$

Function `PrunePoints`($\mathbf{P}^*, \boldsymbol{r}, \boldsymbol{\xi}_o$):
  $i = 0$
  while $i <$ `len`($\mathbf{P}^*$):
    if $\|\boldsymbol{\xi}_{(i+2)} - \boldsymbol{\xi}_{(i)}\|_2 < \boldsymbol{r}_{(i+2)} + \boldsymbol{r}_{(i)}$:
      `delete`$\Big(\boldsymbol{\xi}_{(i+1)}, \mathbf{P}^*\Big)$
      `delete`$\Big(\boldsymbol{r}_{(i+1)}, \boldsymbol{r}\Big)$
    $i \leftarrow i + 1$
return $\mathbf{P}^*, \boldsymbol{r}, \boldsymbol{\xi}_o$

Algorithm 3: Point Spawning and Pruning

Function `GetCentralDistance`($\mathbf{P}^*$) :

$\boldsymbol{d} \leftarrow \texttt{norm}(\texttt{1Dconvolution}([-1, 1], \mathbf{P}^*, \texttt{padding} = \text{valid}), \texttt{axis=0})$

return $\boldsymbol{d}[1:] + \boldsymbol{d}[:-2]$

Function `EBA`($\mathbf{P}^*, \mathcal{C}_1, r_{\min}, \delta_r, \epsilon, c_s, c_r$):

$\overline{\mathrm{F}} \leftarrow \infty$

$\boldsymbol{r}, \mathbf{P}_o \leftarrow \texttt{FitDisk}(\mathbf{P}^*, \mathcal{C}_1, r_{\min}, \delta_r)$

while $\overline{\mathrm{F}} > \epsilon$:

$\mathbf{P}^*, \boldsymbol{r}, \mathbf{P}_o \leftarrow \texttt{PrunePoints}(\mathbf{P}^*, \boldsymbol{r}, \mathbf{P}_o)$

$\Delta\mathbf{P}^* \leftarrow \texttt{1Dconvolution}([-1, 0, 1], \mathbf{P}^*, \texttt{padding} = \text{valid})$

$\Delta\mathbf{P}^* \leftarrow \texttt{divide}\big(\Delta\mathbf{P}^*, \texttt{GetCentralDistance}(\mathbf{P}^*)\big)$

$\Delta\mathbf{P}_o \leftarrow \texttt{divide}(\mathbf{P}^* - \mathbf{P}_o, \texttt{power}(\boldsymbol{r}, 2))[1:-2]$

$\mathbf{F} \leftarrow c_s\, \Delta\mathbf{P}^* + c_r\, \Delta\mathbf{P}_o$

$\mathbf{P}^*[1:-2] \leftarrow \mathbf{P}^*[1:-2] + \mathbf{F}$

$\overline{\mathrm{F}} \leftarrow \texttt{mean}\Big(\texttt{norm}(\mathbf{F}, \texttt{axis} = 0)\Big)$

$\boldsymbol{r}, \mathbf{P}_o \leftarrow \texttt{FitDisk}(\mathbf{P}^*, \mathcal{C}_1, r_{\min}, \delta_r)$

$\mathbf{P}^*, \boldsymbol{r}, \mathbf{P}_o \leftarrow \texttt{SpawnPoints}(\mathbf{P}^*, \boldsymbol{r}, \mathbf{P}_o)$

return $\mathbf{P}^*$

In the present pseudo-code, a first order model is chosen as deformable body model for the elastic band. This means that the update of the path point matrix $\mathbf{P}^*$ is directly determined by the virtual force $\mathbf{F}$. Alternatively, a second-order model can be chosen by introducing a velocity matrix in the integration chain.

For the present work, the while-loop of EBA was replaced by a for-loop, as a small fixed number of iterations already yields good results. This avoids long computation time if the force does not converge fast enough.

C MPFC Results on Static Environments

Concave Example

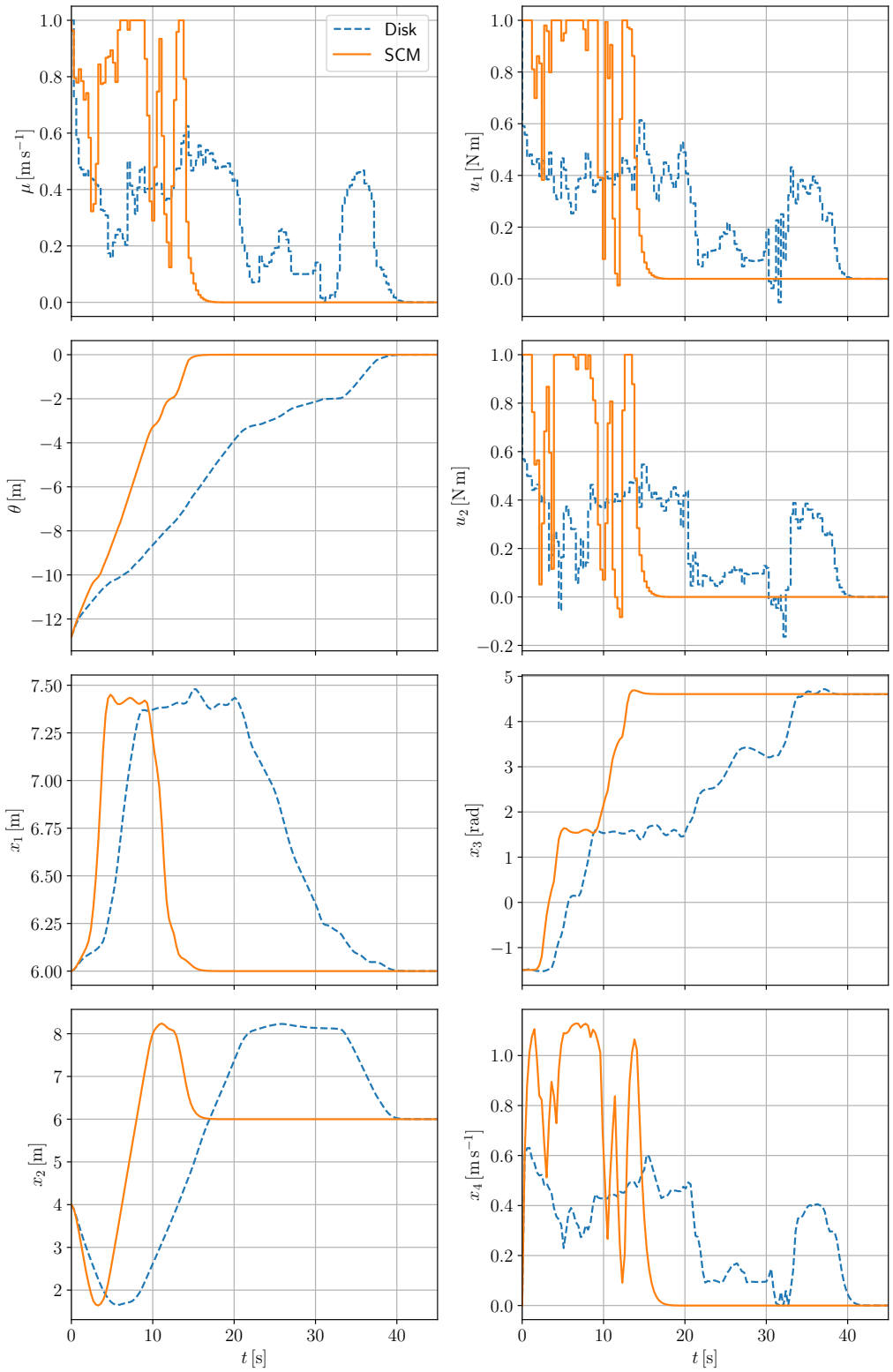


Figure 45: Comparative transient behavior of the MPFC on the H example.

Maze Example

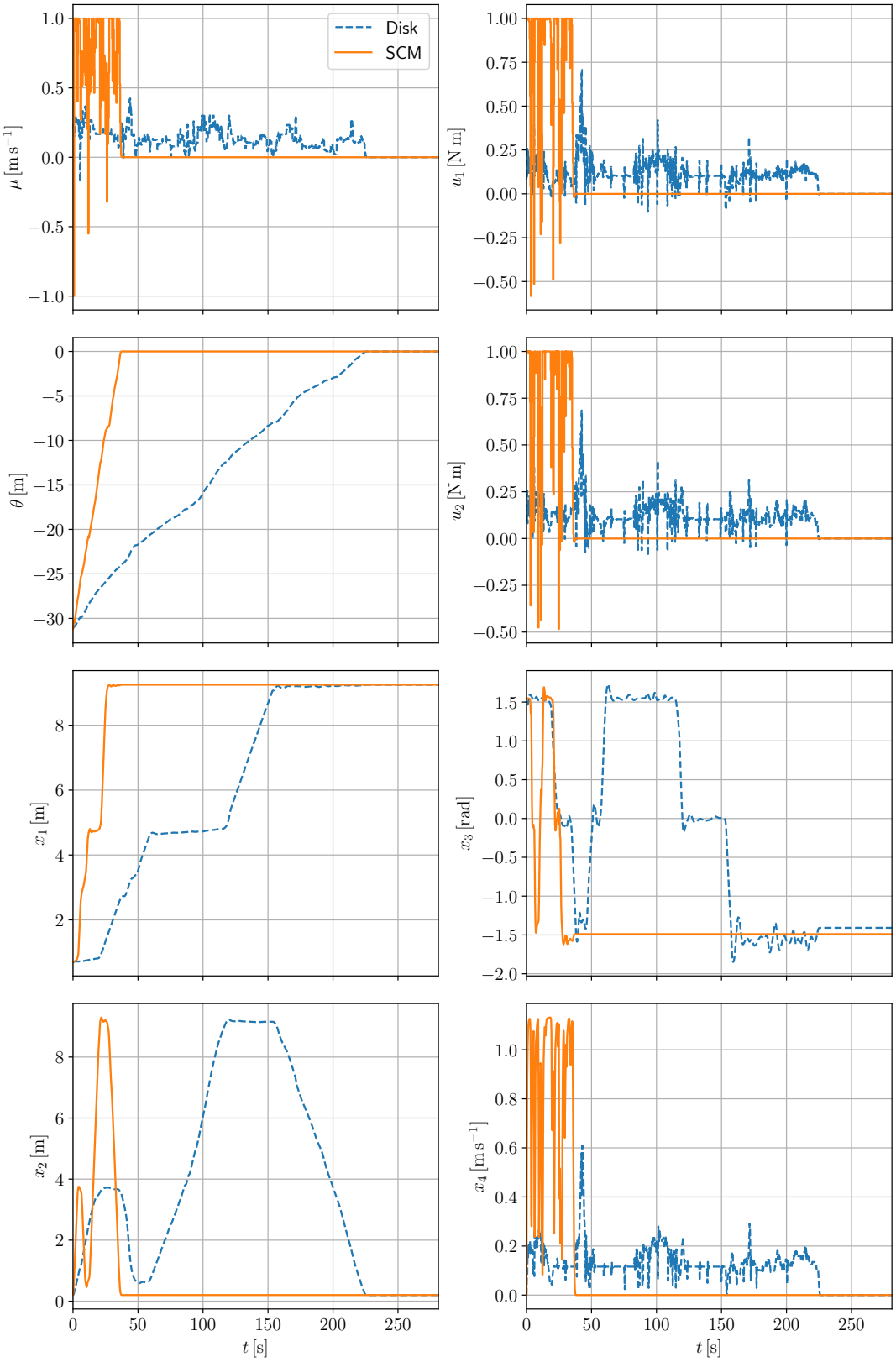


Figure 46: Comparative transient behavior of the MPFC on the maze example.

Outdoor Example

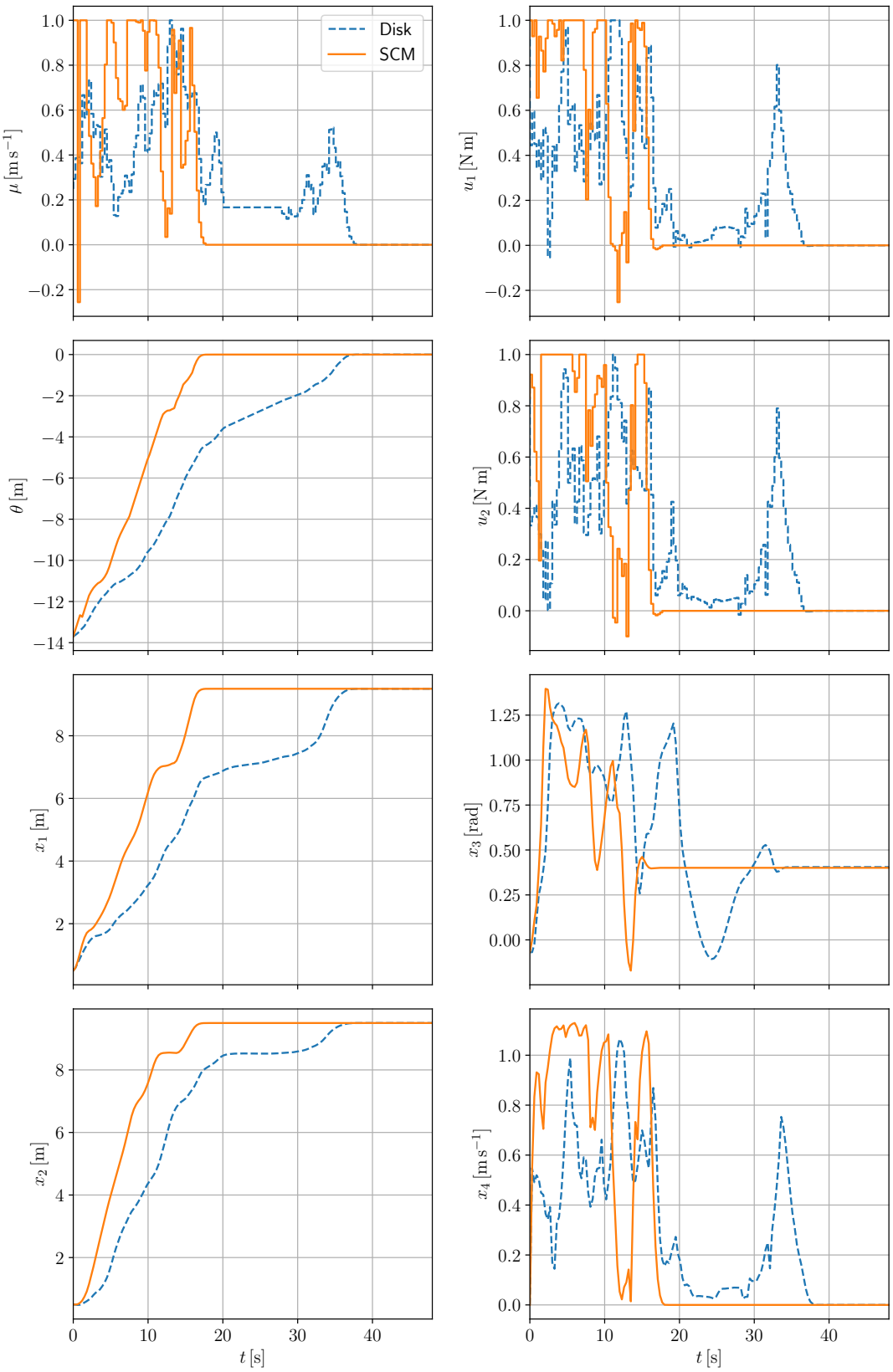


Figure 47: Comparative transient behavior of the MPFC on the outdoor example.

SC-MPFC Results for Disturbed Cases

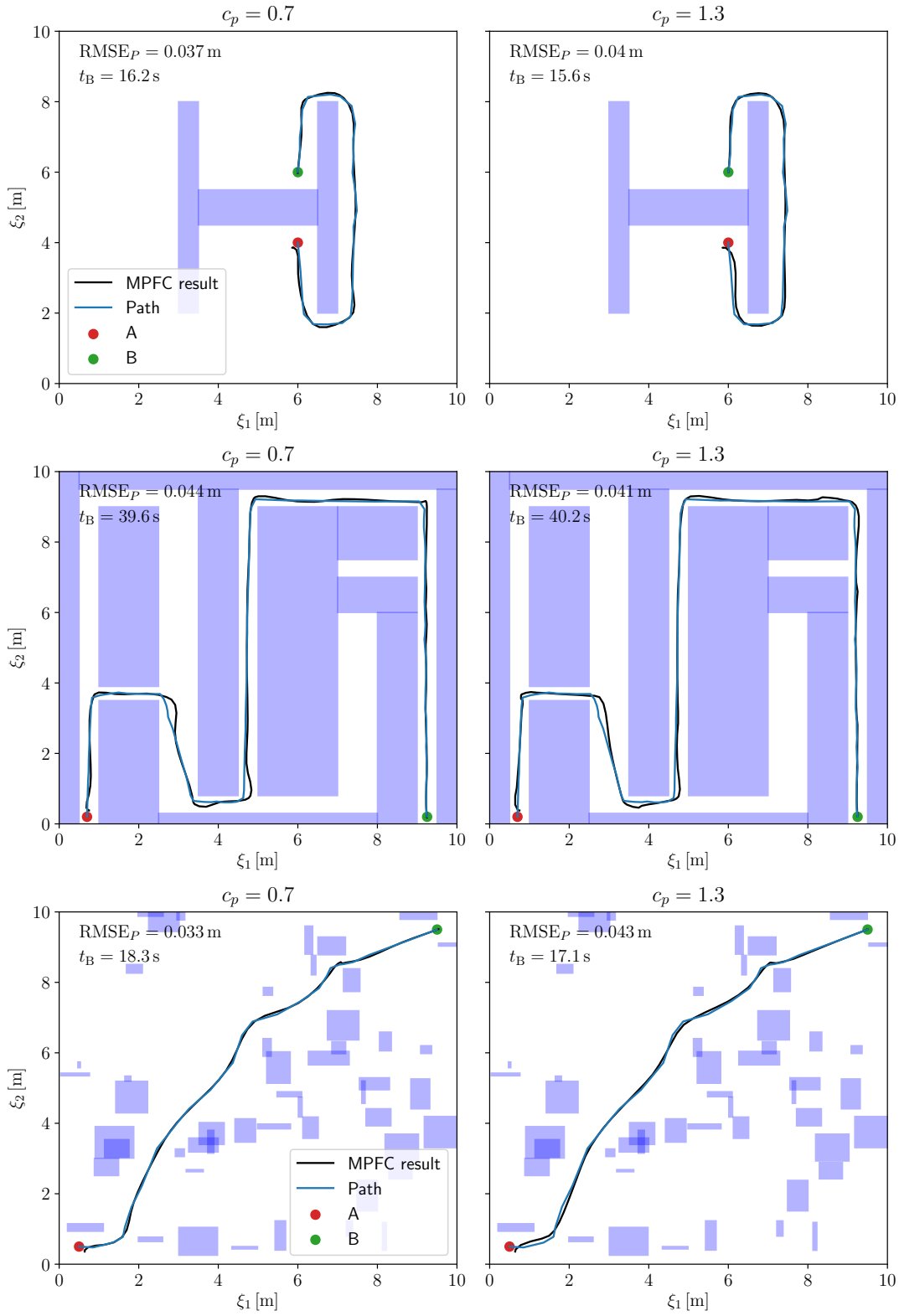


Figure 48: Spatial visualization and performance measures of SC-MPFC for the two parametric disturbance cases on the H, the maze and the outdoor example. For all cases, the original position is disturbed by 20 cm and the original orientation by π.

D Run-Time Analysis

For the present considerations, all the computations are performed on an Intel Core i5-8365U CPU. In the present section, the run-time of SC-MPFC and FSC-MPFC are studied one after another.

Real-Time Efficiency of SC-MPFC

As the ROSBot 2.0 yields a maximal speed of $v_{\max} = 1.25\,\mathrm{s}$, it is a relatively slow system that does not require high frequencies for the control update. Therefore, $\delta_t = 0.3\,\mathrm{s}$ is set and the optimization has to be computed within $\mathrm{T_{CPU}} = 0.27\,\mathrm{s}$ to match the real-time requirements. In some rare cases, this time is in fact needed for the optimization to deliver a good result. However, Figure 49 shows that the average computation time of the optimization is about 3 times lower. Moreover, about 75% of the SC-MPFC steps need less than 0.1 s. For the most challenging maze example, only 21.1% of the MPFC steps were broken up at $\mathrm{T_{CPU}}$. For the H, the indoor and the outdoor terrains, this value was even smaller, respectively yielding 14.3%, 3.1% and 13.1%. Thus, the real-time requirement is fulfilled. Additionally it appears that δ_t could even be chosen lower, thus possibly leading to a more performant and robust control. This was not done here, as the present work is foremostly a proof of concept. The available time was invested in the development of an environment motion predicting NMPC rather than diving into run-time optimization.

The small cushion of $\delta_t - \mathrm{T_{CPU}} = 0.03\,\mathrm{s}$ is used for the computations done before the optimization. The large majority of this time is used for the safe-corridor computation. Figure 49 shows that not a single SCM evaluation took more than the available time. The mean and quartile values are even by far lower. Thus, the implemented SCM is well-suited for real-time applications.

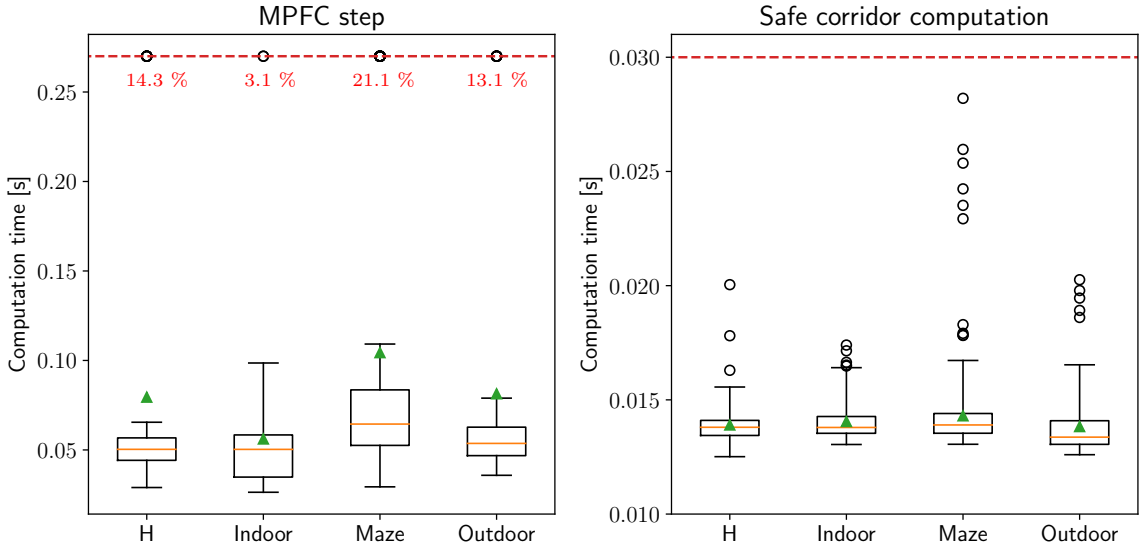


Figure 49: Box plots of the computation time for an MPFC step and for SCM. The boxes represent the first and the third quartile. The median, the mean value and the outliers are respectively represented by the middle line, the green triangle and the black circles. The whiskers yield three time the inter-quartile interval. The dotted lines represent the maximum computation time available.

Finally, as depicted in Figure 50, the average total computation time roughly yields $0.1\mathrm{s} = 1/3\,\delta_t$ for all scenarios. This even holds for the challenging maze example, where the mean total computation time for an iteration does not exceed 0.13 s.

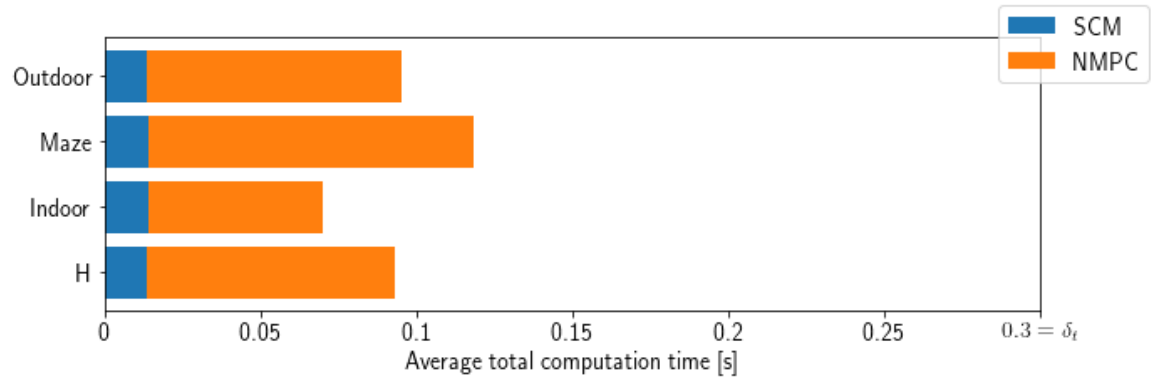


Figure 50: Mean total computation time of SC-MPFC for the various scenarios.

Run-Time of FSC-MPFC

For FSC-MPFC, the real-time requirement could not be fulfilled, as several ellipses have to be fitted and the inference of the neural network has to be performed. As depicted in Figure 51, the average time of the NN inference and of forecasting SCM already roughly sum up to the real time limit δ_t. To stay consistent with the previously studied SC-MPFC, the time available for optimization is limited to $T_{CPU} = 0.9\delta_t = 0.27\,s$. Even though the used hardware does not allow the implementation of FSC-MPFC for now, it appears that the method is not far from being real-time capable. In fact, reducing its run-time by half would already be enough for an application.

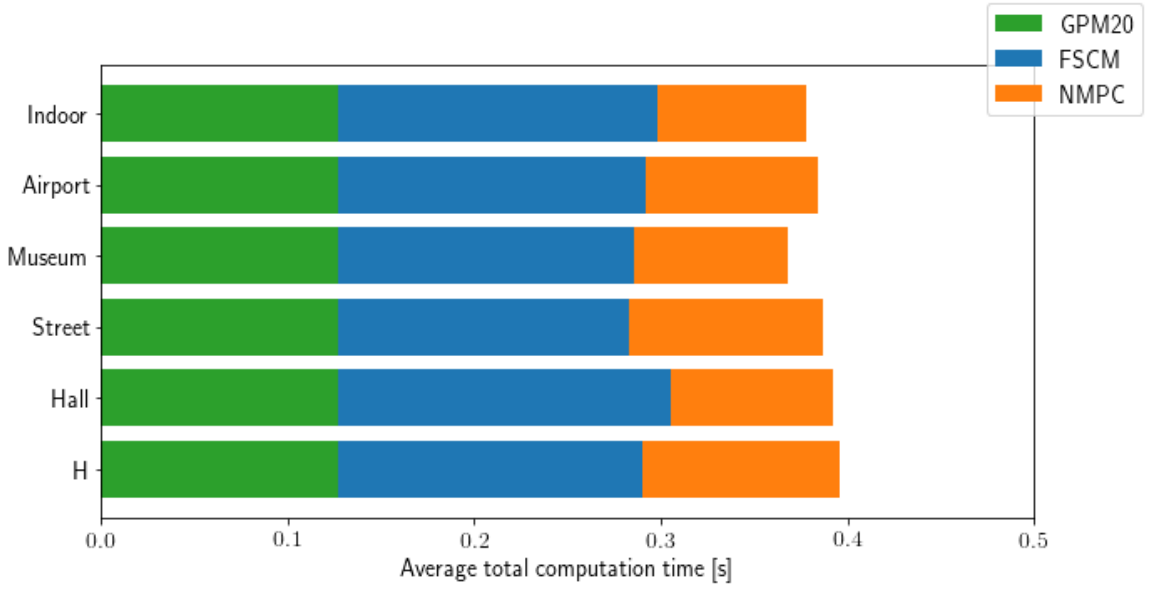


Figure 51: Mean total computation time of FSC-MPFC for the various scenarios.

Conclusions and Possible Improvements

Whereas SC-MPFC matches the real-time requirements of the ROSBot 2.0, FSC-MPFC does not. Still it appears that this could be achieved without large effort. Many improvements would also benefit SC-MPFC and allow it to yield a higher frequency of the control update. Hereby, several tracks can be exploited:

1. **Training a smaller GPM**. If a similar performance could be achieved with a smaller network than GPM20, the inference time would decrease. This could take away a significant part of the computational load of FSC-MPFC.

2. **Using an other programming language**. Even though PYTHON has been used as language for the programming interface, most of the code relies on run-time optimized C++ and FORTRAN libraries. However, using another language, specifically designed for an optimized run-time, such as JULIA, could allow to speed up SC-MPFC and FSC-MPFC.

3. **The run-time of SCM might still be improved**.

4. **Using a more performant hardware**. The embedded hardware used so far is relatively performant for such a small robot. However, as shown by many domains of application, cloud computing gets more and more popular because of the many advantages it represents. Applying this on the ROSBOT 2.0 could lead to real-time capable methods with high frequency updates.

5. **Using CasADi's code generation**. CASADI developer Joel Anderson recently announced, that the code generation tool shall be extended[4]. In fact, until now, it is not compatible with IPOPT. However, such a tool allows reducing the run-time by about 4 up to 10 times[5] and would shrink the run-time of SC-MPFC and FSC-MPFC.

[4] https://groups.google.com/g/casadi-users/c/uQtbSleEIF0/m/m6I758h9AQAJ
[5] http://casadi.sourceforge.net/users_guide/html/node5.html

E Exemplary GPM19 Results

As mentioned in chapter 5, GPM19 is able to deliver multimodal predictions, as shown in Figure 52. Here two scenarios are included in the prediction, as the car is predicted to be likely to take the downward or the leftward street.

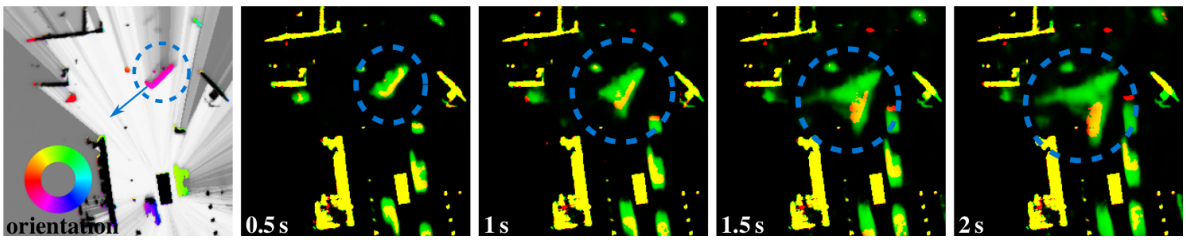


Figure 52: Exemplary multimodal prediction delivered by GPM19 [43].

Another appealing feature of GPM19 is its ability to render interactions between objects. In Figure 53, two situations are compared. On the left side, the car at the bottom of the figure drives upward without encountering any other agent. Thus, the car is predicted to continue driving without breaking. On the right side, an additional agent, circled in blue, is introduced on the map. Compared to the left picture, the prediction for a horizon of 1.5 seconds delivers a slight margin to the red line. Thus, GPM19 is able to partially understand how agents interact with each-other.

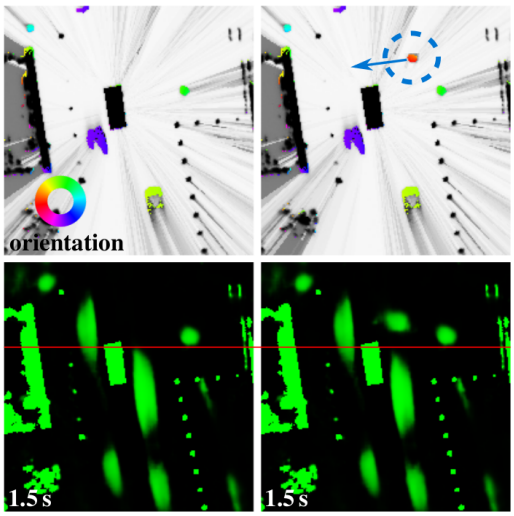


Figure 53: Exemplary context understanding by GPM19 [43].

F Comparison of Control Results on Dynamic Environments

Critical Time Steps – SC-MPFC

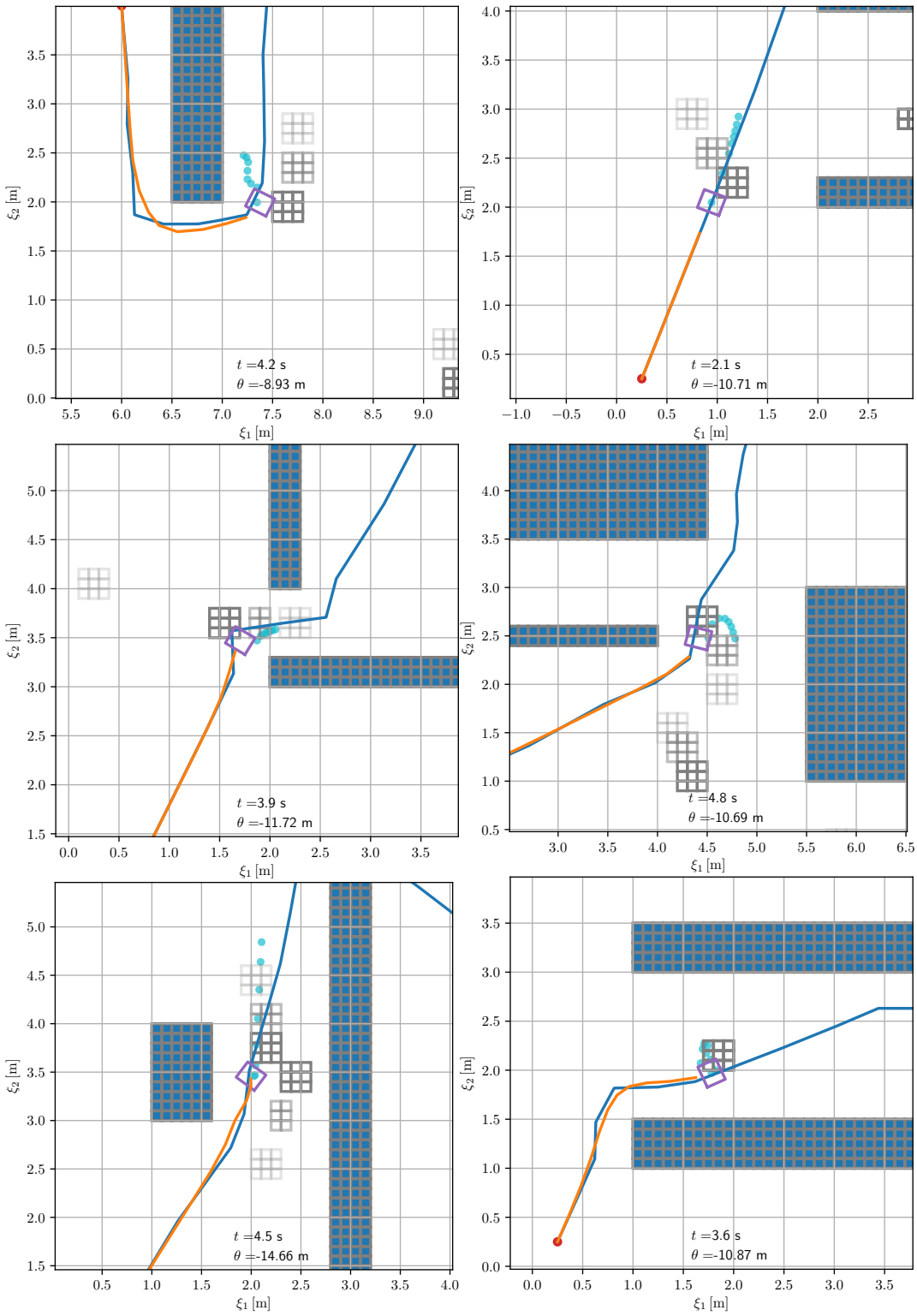


Figure 54: Critical control results obtained by SC-MPFC on dynamic environments.

Critical Time Steps – GT-FSC-MPFC

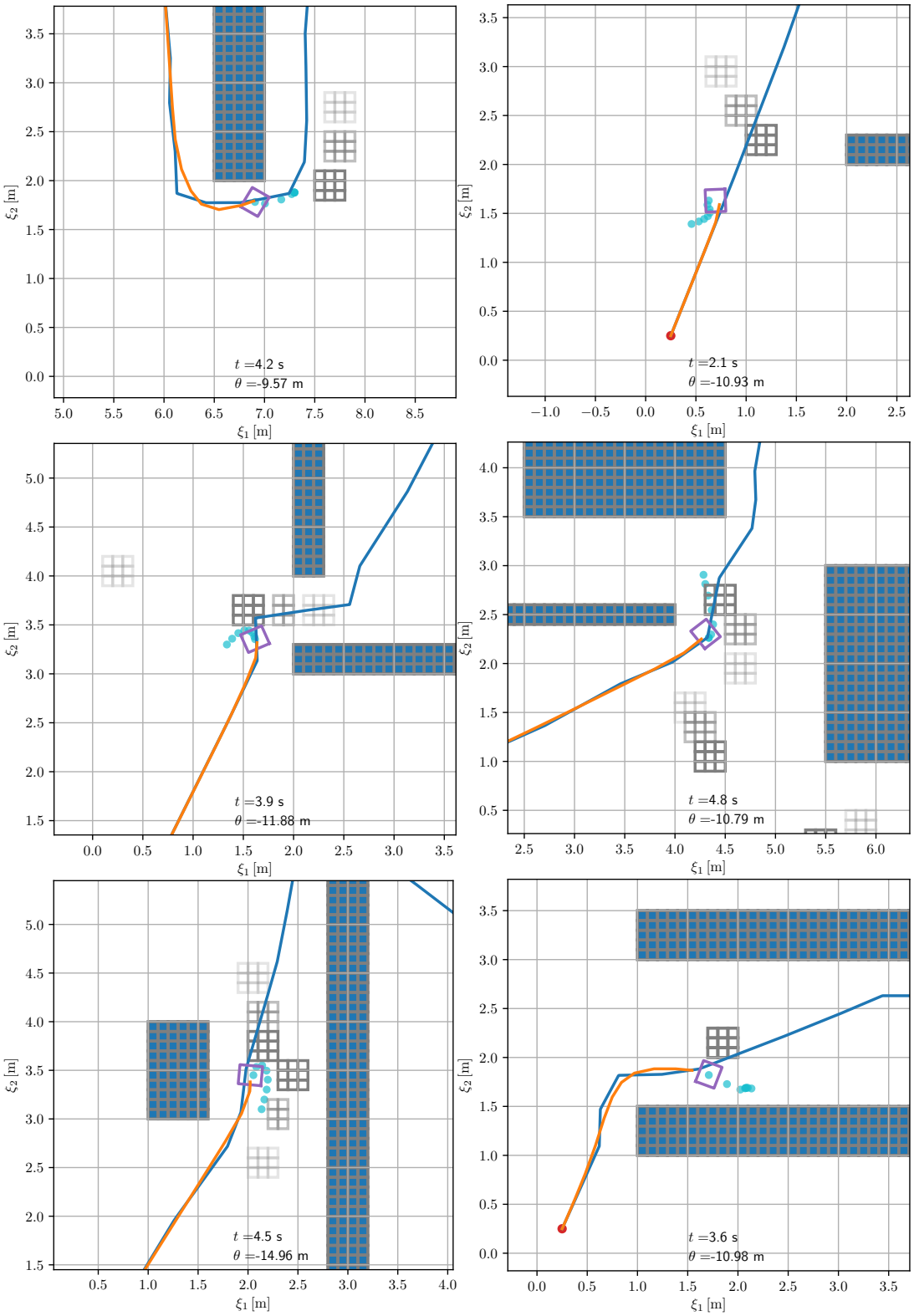


Figure 55: Control results obtained by GT-FSC-MPFC on dynamic environments.

Critical Time Steps – FSC-MPFC

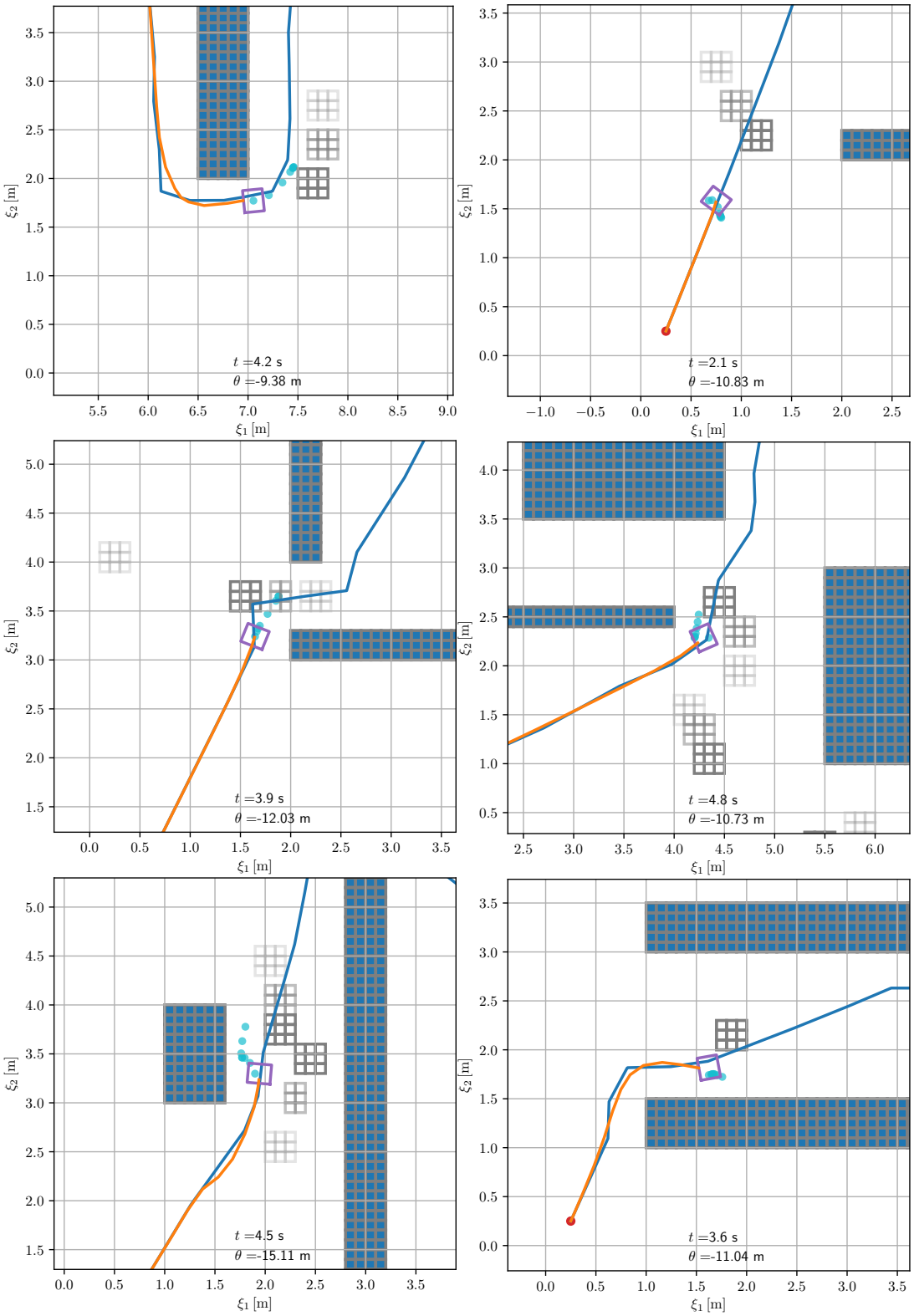


Figure 56: Control results obtained by FSC-MPFC$_{p_0=0.1}$ on dynamic environments.

Critical Time Steps – Conservative FSC-MPFC

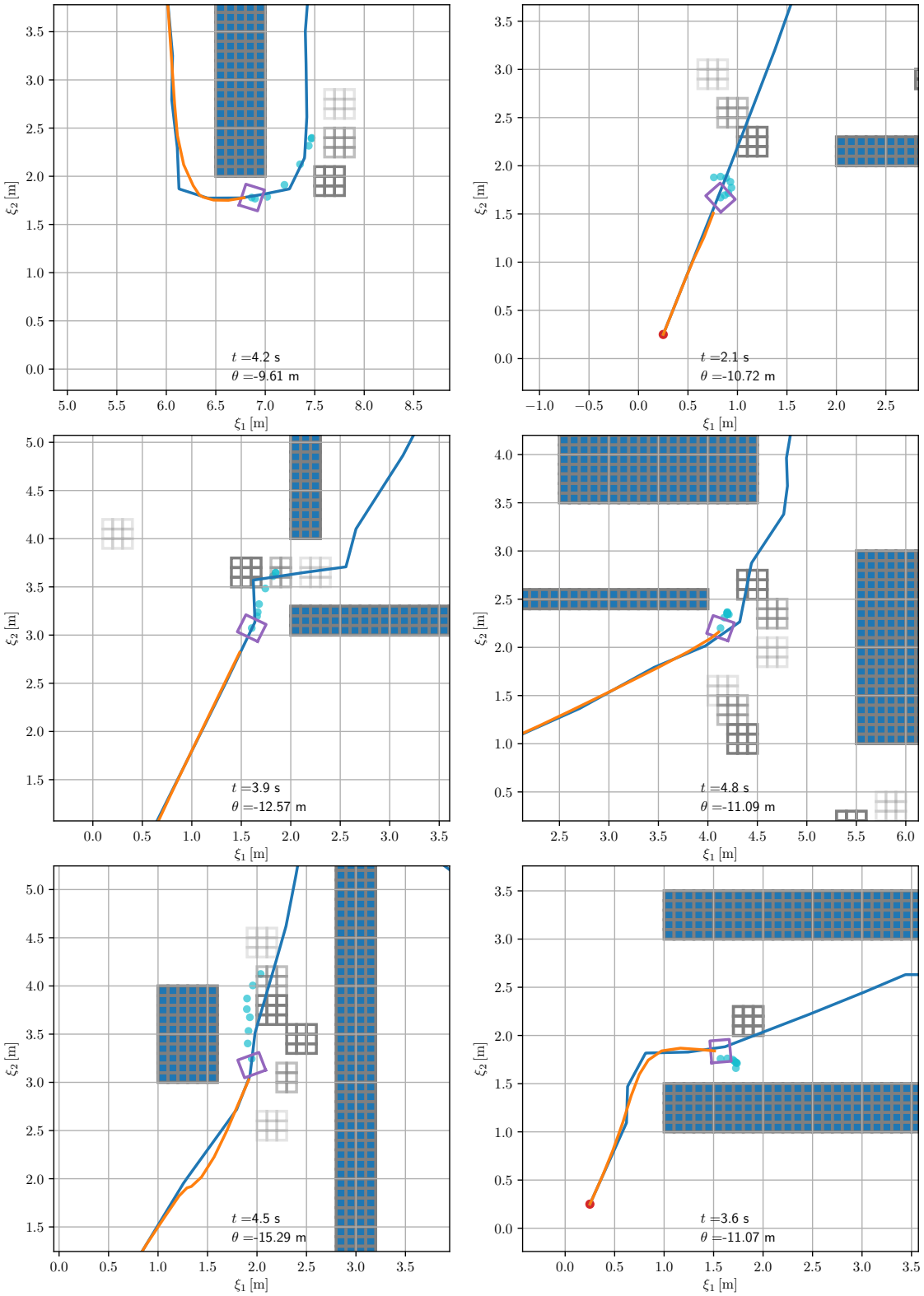


Figure 57: Control results obtained by FSC-MPFC$_{\mathsf{p_0}=0.03}$ on dynamic environments.

Final Path – FSC-MPFC

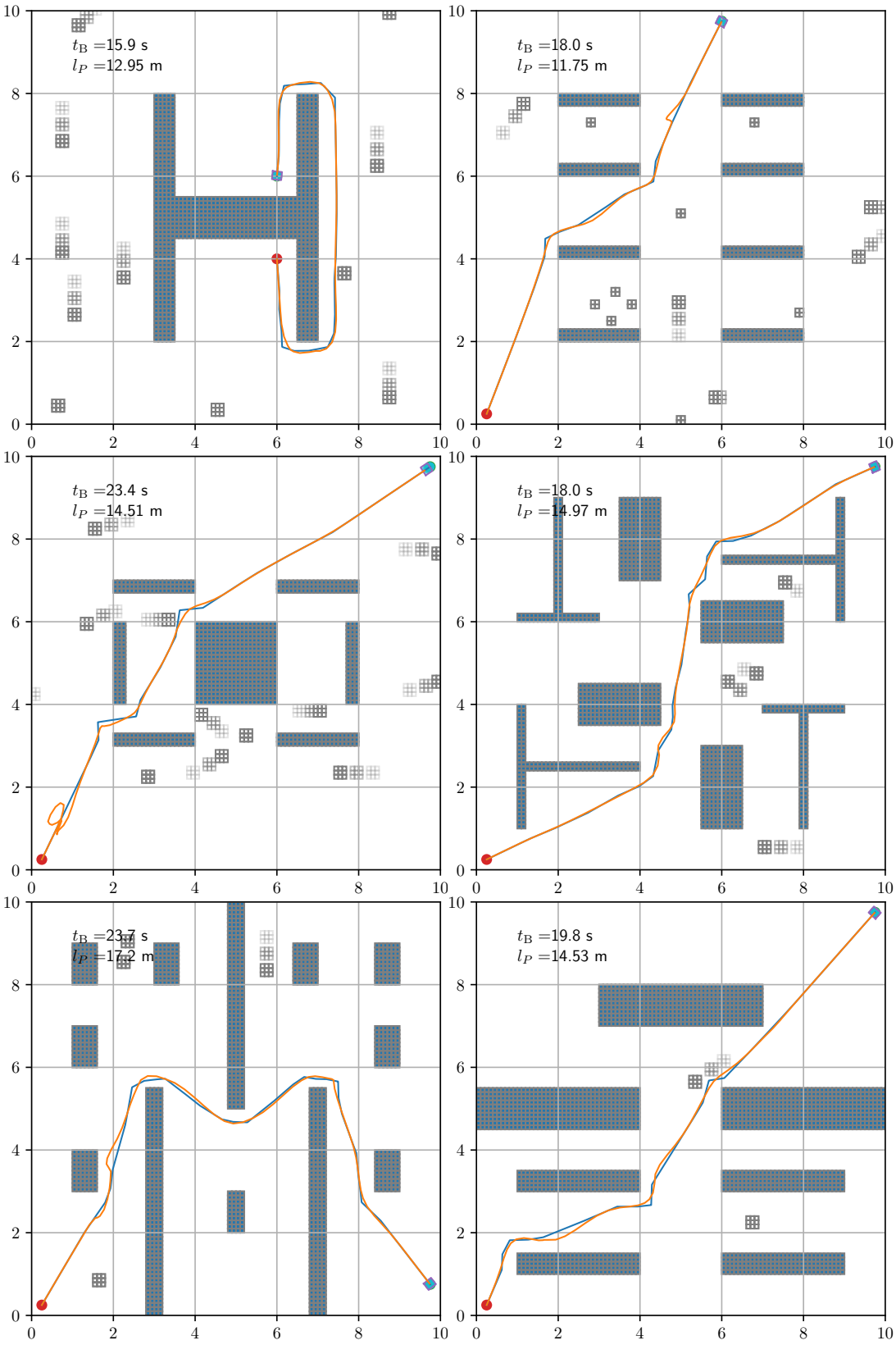


Figure 58: Paths taken by FSC-MPFC$_{p_0=0.1}$ on the GPM20 test environments.