POZNAN UNIVERSITY OF TECHNOLOGY

Deep reinforcement learning for motion planning in man-made environments

by

Piotr Kicki

in the
Institute of Robotics and Machine Intelligence
Faculty of Control, Robotics and Electrical Engineering

Supervisor: Prof. Piotr Skrzypczyński, Ph.D., D.Sc.
Co-supervisor: Krzysztof Walas, Ph.D.

2023

Abstract

Motion planning is a mature area of research in robotics with many well-established methods based on optimization or sampling the state space, suitable for solving a variety of motion planning problems. Recently, a new trend in the field of motion planning emerged, whose key feature is the utilization of the experience in order to improve future motion planning attempts, i.e., machine learning-based motion planning. In this dissertation, we propose advancements in this field, particularly by introducing new solutions to learning-based motion planning for car-like vehicles and constrained kinodynamic motion planning problems. We propose a method for rapid path planning for car-like vehicles that sequentially builds the solution from the locally defined polynomial segments inferred by a neural network, taking into account the geometry of the environment and the non-holonomic constraints. We also introduce another approach that is able to generate, in a single neural network inference, a feasible path that guarantees to reach the goal state exactly. Moreover, we propose a new method for constrained kinodynamic planning that utilizes a neural network to rapidly plan trajectories parametrized using two B-spline curves. Thanks to the proposed parametrization, which allows for imposing boundary constraints on the planned trajectory and its derivatives, and constant planning time, it allows for replaning the robot's motion on-the-fly. The proposed approach allows for incorporating not only boundary constraints but also kinodynamic ones over the whole trajectory, and minimizing some user-defined task loss, thanks to the use of the Lagrangian multipliers-inspired learning procedure. To train our proposed motion planning neural networks, we follow the reinforcement learning paradigm and utilize loss functions that penalize the unwanted behavior of the proposed planners, like a violation of the constraints, collisions with obstacles, long motion times, etc. In the proposed setting, we define the loss functions based on the geometry of the solution, such that they are differentiable and allow for efficient learning of how to plan feasible motions. We propose to use only weak supervision in order to guide the plans outside of the non-convex collision areas, which cannot be done using gradients. Finally, we performed an extensive experimental evaluation of the proposed solutions. All introduced approaches were quantitatively compared with the state-of-the-art motion planning algorithms and showed superb performance in solving local motion planning problems. Moreover, we showed that the proposed approaches also have qualitative advantages over the state-of-the-art algorithms, such as guarantees of boundary constraint satisfaction, constant and short planning time, or the ability to replan the motion on-the-fly. These evaluations were conducted both on the datasets and in a wide range of simulated and several real-world scenarios. This dissertation is concluded with a summary of the contributions made to the field of learning-based robotic motion planning and a list of the limitations of the proposed solutions, as well as the directions of the potential future work.

Streszczenie

Planowanie ruchu jest dojrzałym obszarem badań w robotyce z wieloma ugruntowanymi metodami opartymi na optymalizacji lub próbkowaniu przestrzeni stanów, odpowiednimi do rozwiązywania szerokiej gamy problemów planowania ruchu. Ostatnio pojawił się nowy trend w dziedzinie planowania ruchu, którego kluczową cechą jest wykorzystanie doświadczenia w celu poprawy przyszłych prób planowania ruchu, tj. planowanie ruchu oparte o uczenie maszynowe. W niniejszej rozprawie proponujemy innowacje w tej dziedzinie, w szczególności poprzez zaproponowanie nowych metod planowania ruchu dla pojazdów podobnych do samochodów i systemów z ograniczeniami kinodynamicznymi. Proponujemy metodę szybkiego planowania ścieżki dla pojazdów o kinematyce samochodu, która sekwencyjnie buduje rozwiązanie z lokalnie zdefiniowanych segmentów wielomianowych, wywnioskowanych przez sieć neuronową, która bierze pod uwagę geometrię środowiska i ograniczenia nieholonomiczne. Przedstawiamy również inne podejście, które jest w stanie wygenerować, przy pomocy pojedynczej ewaluacji sieci neuronowej, wykonalną ścieżkę, która gwarantuje dokładne osiągnięcie stanu docelowego. Ponadto proponujemy nową metodę ograniczonego planowania kinodynamicznego, która wykorzystuje sieć neuronową do szybkiego planowania trajektorii sparametryzowanych za pomocą dwóch krzywych B-sklejanych. Dzięki zaproponowanej parametryzacji, która pozwala na nałożenie ograniczeń brzegowych na planowaną trajektorię i jej pochodne, oraz stałemu czasowi planowania, możliwe jest replanowanie ruchu robota w locie. Proponowane podejście pozwala na uwzględnienie nie tylko ograniczeń brzegowych, ale także kinodynamicznych dla całej trajektorii oraz minimalizację funkcji kosztu związanej z zadaniem zdefiniowanym przez użytkownika, dzięki zastosowaniu procedury uczenia inspirowanej mnożnikami Lagrange'a. Aby wytrenować proponowane przez nas algorytmy planowania ruchu oparte o sieci neuronowe, stosujemy paradygmat uczenia ze wzmocnieniem i wykorzystujemy funkcje kosztów, które penalizują niepożądane zachowanie proponowanych planerów, takie jak naruszenie ograniczeń, kolizje z przeszkodami, długie czasy ruchu itp. W niniejszej rozprawie definiujemy funkcje kosztów w oparciu o geometrię rozwiązania, tak aby były one różniczkowalne i pozwalały na efektywne uczenie się planowania dopuszczalnych ruchów. Proponujemy użycie jedynie słabego nadzoru w celu skierowywania planów poza niewypukłe obszary kolizyjne, co nie jest możliwe do osiągnięcia korzytsając z prostych gradientów. Na koniec przeprowadziliśmy obszerną ocenę eksperymentalną proponowanych rozwiązań. Wszystkie wprowadzone podejścia zostały ilościowo porównane z nowoczesnymi algorytmami planowania ruchu i wykazały doskonałą wydajność w rozwiązywaniu lokalnych problemów planowania ruchu. Co więcej, wykazaliśmy, że proponowane podejścia mają również zalety jakościowe w porównaniu do algorytmów wyznaczających aktualny stan wiedzy, takie jak gwarancja spełnienia ograniczeń brzegowych, stały i krótki czas planowania lub możliwość replanowania ruchu w locie. Wspomniane eksperymenty zostały przeprowadzone zarówno na zbiorach danych, jak i w szerokim zakresie symulowanych oraz kilku rzeczywistych scenariuszach. Niniejsza rozprawa zakończona jest podsumowaniem wkładu wniesionego w dziedzinę planowania ruchu robotów opartego na uczeniu maszynowym oraz listą ograniczeń proponowanych rozwiązań, a także kierunkami potencjalnych przyszłych prac.

Acknowledgements

Firstly, I would like to express my sincere gratitude to my supervisor, Piotr Skrzypczyński, for his scientific guidance, constant support, and availability, and to my co-supervisor Krzysztof Walas for introducing me to the world of science and inspiring me to try to enrich it with my ideas and hard work.

I would like to thank Jan Reinhard Peters for a three-month internship at the Intelligent Autonomous Systems Group of the Technische Universitaet Darmstadt that allowed me to work with the best in the field of robotic reinforcement learning, greatly improve my scientific workshop, and test my motion planning algorithms in the wild. I would also like to thank all my colleagues at TU Darmstadt, but especially Davide Tateo and Puze Liu for the inspiring conversations and a lot of honest and hard work we have done together during my stay in Darmstadt.

Last but not least, I would like to thank my family. I would like to thank my beloved wife for her continuous support and overall happiness in my life, and for being extremely patient and understanding. I would also like to thank my son, for being a living and truly inspiring example of how reinforcement learning works at its finest. Finally, I would like to thank my parents for their constant support and everything they managed to teach me.

Abbreviations

ABIT* Advanced Batch Informed Trees.

ADRC Active Disturbance Rejection Control.

AIT* Adaptively Informed Trees.

AQP Anchored Quadratic Programming.

BIT* Batch Informed Trees.

CBiRRT Constrained Bi-directional RRT.

cc-Dubins continous curvature Dubins curve.

CEM Cross Entropy Method.

CMPCB Contextualized Motion Planning Continous Bandit.

CNP-B Constrained Neural motion Planning with B-splines.

iLQR iterative Linear Quadratic Regulator.

LCS Local Coordinate System.

LOAM LiDAR Odometry and Mapping.

LQR Linear Quadratic Regulator.

MDP Markov Decision Process.

MPC Model Predictive Control.

MPC-MPNet Model-Predictive Motion Planning Network.

NLP Nonlinear Programming.

NN Neural Network.

OMPL Open Motion Planning Library.

ReLU Rectified Linear Unit.

RL Reinforcement Learning.

RNEA Recursive Newton-Euler.

RRT Rapidly Exploring Random Trees.

SBMP Sampling-Based Motion Planning.

SL State Lattices.

SLSQP Sequential Least Squares Programming.

SST Stable Sparse RRT.

VFO Vector Field Orientation.

Notation

f	the motion planning function
$\mathbb{N}_+$	the set of natural numbers without 0
$\mathcal{P}$	the set of motion planning problems
$\mathcal{S}$	the set of all solutions to motion planning problems
$\mathfrak{s}$	the random seed
$\aleph$	the symbol of no solution
$\mathcal{P}_f$	the set of solvable motion planning problems
f^*	the ideal motion planning function
F^*	the set of all ideal motion planning functions
f_o^*	the optimal ideal motion planning function
J	the optimality criterion
ζ	the path
$\boldsymbol{q}$	the state of the system
Q	the state space
$\boldsymbol{q}_0$	the initial state of the system
$\boldsymbol{q}_d$	the desired state of the system
Q_d	the set of the desired system states
x	the x coordinate of the robot position
y	the y coordinate of the robot position
θ	the orientation of the robot
β	the angle of the virtual steering wheel
$\mathbb{R}$	the set of real numbers
$\mathbb{S}$	the circle group
p_G	the vehicle's guiding point
ξ	the angular speed of the virtual steering wheel
v	the linear velocity
L	the distance between the front and rear axles
β_{max}	the maximal admissible angle of the virtual steering wheel
Π	the car body
s	the phase variable
$\mathfrak{S}$	the swath of the robot
$\mathfrak{F}$	the free space
$\mathcal{E}$	the environment

κ	the curvature of the path
κ_{max}	the maximal admissible curvature of the path
x'	the derviative of x w.r.t. the phase variable
x''	the second derviative of x w.r.t. the phase variable
$\mathbb{C}^n$	the set of n-times differentiable functions
$\mathfrak{d}$	transformation between the solution space and state space
A	the action space
a	the action
T	the transition function
P	the probability
R	the reward function
r	the reward
t	the time
$\boldsymbol{q}_t$	the state at time t
ζ^i	the i-th subpath of the path ζ
α	the sequence of numbers that partition the interval into subintervals
n_{seg}	number of segments
C	the context
π	the policy
m	the polynomial coefficients
$\mathfrak{L}_i$	the i-th local coordinate system
$X_{\mathfrak{L}_i}$	the x axis of the i-th local coordinate system
$Y_{\mathfrak{L}_i}$	the y axis of the i-th local coordinate system
$x_{\mathfrak{L}_i}$	the x coordinate of the robot expressed in the i-th local coordinate system
$y_{\mathfrak{L}_i}$	the y coordinate of the robot expressed in the i-th local coordinate system
S_ζ	the matrix of the parameters of the segment endpoints
E	the environment representation
$\mathcal{T}$	the task definition
$\mathfrak{F}^C$	the collision space
π^*	the optimal policy
$\mathbb{R}_+$	the set of positive real numbers
π_ϕ	the planning policy
ϕ	the parameters of the planning policy
ζ^r	the reference path
Z^r	the space of all reference paths
$\mathbb{P}(X)$	the power set of the set X
$\mathcal{L}$	the loss function
$\mathcal{L}_{coll}$	the collision loss
$\mathcal{L}_{curv}$	the curvature loss
$\mathcal{L}_{over}$	the overshoot loss
$\mathcal{L}_{tcurv}$	the total curvature loss
ρ	the feasibility indicator function
σ	the collision indicator function
$d(\mathcal{X}, \mathcal{Y})$	the smallest Euclidean distance between elements of the sets $\mathcal{X}$ and $\mathcal{Y}$

$\mathfrak{F}_E^C$	the collision space represented by the map E
$Q_{dX}, Q_{dY}, Q_{d\theta}$	the lengths of the edges of the orthotope of desired configurations $\boldsymbol{q}_d$
l_{ij}	the Euclidean distance between the $(j-1)$-th and the j-th point in the i-th path segment
W	the width of the vehicle
L_B	the distance between the vehicle's guiding point and its rear bumper
L_F	the distance between the vehicle's guiding point and its front bumper
δ_ϕ	the update of the neural network parameters ϕ
γ	the learning rate
$\mathcal{C}$	the space of all contexts
ψ	the neural network output
$\mathcal{B}$	the function that represents the proposed B-spline path construction method
$p(s)$	the B-spline curve
n_p	the number of B-spline control points
$p_1, p_2, \ldots, p_{n_p}$	the B-spline control points
o_p	the B-spline order
d_p	the B-spline degree
$B_{i,j}$	the i-th B-spline basis function of order j
ω	the B-spline basis function weight
$\boldsymbol{u}$	the vector of B-spline knots
$\boldsymbol{u}_{int}$	the vector of B-spline internal knots
$d_{i,j}$	the length of the side of the square between i-th and j-th B-spline control points
$l_{i,j}$	the Euclidean distance between i-th and j-th B-spline control points
l_T	the level of the B-spline control points tree
R_{min}	the minimal vehicle turning radius
l_i	the Euclidean distance between the (i)-th and the $(i+1)$-th path discretization point
$\mathcal{M}$	the constraint manifold
$\boldsymbol{\zeta}$	the trajectory of the system
T	the duration of the trajectory
c, g	the constraint functions
N	the number of equality constriants
M	the number of inequality constraints
$\mathcal{L}_t$	the loss function at time step t
$\mathfrak{f}$	the trajectory parameters
$\boldsymbol{\zeta}_\mathfrak{f}$	the trajectory parametrized by the set of parameters $\mathfrak{f}$
$\boldsymbol{\mu}$	the vector of slack variables
$\mathcal{M}_{\boldsymbol{\mu}}$	the slack variables dependent constraint manifold
$\mathcal{L}_\mathcal{M}$	the manifold loss
$\bar{c}_i$	the desired acceptable violation level of the i-th constraint
$\bar{C}$	the square of the desired acceptable violation level of the i-th constraint
$\boldsymbol{\lambda}$	the vector of Lagrange multipliers
L	the Lagrangian
Λ	the manifold metric
$\mathfrak{m}$	the vector for the manifold metric parameters
$\mathcal{L}_{\mathcal{M},\Lambda}$	the manifold loss under the metric Λ

$\mathfrak{r}$	the time scaling factor
$\mathfrak{t}$	the transformation from phase variable to time
$\boldsymbol{p}(s)$	the configuration B-spline curve
$n_{\mathfrak{r}}$	the number of control points of the time-scaling B-spline curve
$D_p, D_{\mathfrak{r}}$	the degrees of configuration and time-scaling B-splines
$\bar{\dot{\boldsymbol{q}}}$, $\bar{\ddot{\boldsymbol{q}}}$, $\bar{\boldsymbol{\tau}}$	the velocity, acceleration and torque limits
T_{exp}	the expected trajectory duration
ν	the number of neurons in the fully connected layers
n_{bc}	the number of boundary control points
n_{dof}	the number of the degrees of freedom
$\psi^{\boldsymbol{p}}$	the output of the configuration head of the neural network
n_{ibc}, n_{ebc}	the numbers of boundary constraints imposed on the beginning and end of the trajectory
$\boldsymbol{\zeta}_{effort}$	the total effort along the trajectory
$\boldsymbol{\tau}$	the torque
FK	the forward kinematics
SE	the special Euclidean group
$d_{\mathrm{SE(3)}}$	the metric defined over SE(3)
d_{euc}	the Euclidean distance
$\mathcal{M}_{\mathrm{FK}}$	the manifold defined in the task space
n_s	the number of trajectories
n_e	the number of elite trajectoreies
h	the simulation horizon
T_s	the simulation time step
$\widehat{\beta}$	the standard deviation of the steering angle distribution
$\bar{\beta}$	the mean value of the steering angle distribution
$\mathcal{N}$	the normal distribution
$l_i^{z1}, l_i^{z2}, l_i^{z3}$	the extended state observer gains
k_{p_i}, k_{d_i}	the controller proportional and derivative gains
M_{ii}	the i-th diagonal element of the mass matrix
ω_o	the observer bandwidth
ω_c	the controller bandwidth
$\boldsymbol{z}$	the state vector of the extended state observeer
δ_{BN}	the node search radius
$\delta_s = 0.1\,\mathrm{m}$	the witness radius
r_g	the goal radius
(μ_t, σ_t)	the mean and standard deviation of the motion time Gaussian
(μ_c, σ_c)	the mean and standard deviation of the control Gaussian
H	the Huber loss function
$\mathcal{L}^O$	the vertical orientation loss
$\mathcal{L}^{E_r}$	the robot collision loss
FK_{kc}	the set of points in the workspace located along the kinematic chain
$\mathcal{L}^{E_o}$	the object collision loss
FK_o	the set of points that belong to the handled object
$\mathrm{I}(X, Y)$	the indicator function if $X \in Y$

η	the regularization factor

Contents

Chapter 1

Introduction

1.1 Motivation

Since its beginnings, robotics has been trying to match human capabilities to be able to replace or support humans in tedious and hazardous tasks. We can see the abundance of industrial manipulators in factories, which significantly boost the production capacity, or specialized robots built specifically to perform a particular task such as search and rescue, or inspections. However, most robots operate today in specially prepared environments and are unable to deal with the complexity of the world outside the factory. In recent years, robotics research has focused on expanding the ability of robots to cope with other environments, including structured ones, such as those created by humans, but also natural ones, which are characterized by much less structuring and greater unpredictability. However, even man-made environments full of right angles and geometric shapes are challenging for robots. Moreover, we strive to require robots to be flexible and adaptable to many different tasks. We look to them for the ability to adapt to new challenges, similar to the one that characterizes people who can both thread a needle, catch a fast-flying ball, and lift weights, all with the same body.

Nowadays, we can see great advancements in spreading the presence of robots outside of specially prepared environments. Robots like ANYmal are shown to be able to traverse challenging terrains [117] and navigate inside mine [33] or office [65]. But even greater effort is put into the development of self-driving vehicles. Companies like Tesla, Waymo, and Apollo endeavor to make cars autonomously move on the streets of our cities [108]. In turn, developments of Boston Dynamics's Atlas lead to increasing the robot's agility and versatility, which allows it to seamlessly switch between doing parkour and carrying objects [42].

All of these abilities are possible thanks to the recent substantial advances in the fields of control, perception, localization, mapping, and motion planning. However, in the case of some sudden changes in the environment, like a pedestrian intrusion into the roadway, robots still rely on reflexes and lack the ability to rapidly replan. While this may be sufficient in some cases, sometimes we need to deliberately plan the robot's reaction to the dangerous situation to ensure that following the reflexes will not cause more damage than not reacting at all. Moreover,

motion planning is still a big challenge in complex environments that introduce tight constraints, especially if they require a very fast response. Fast motion planning and replanning are key abilities of humans, which on the lowest level are performed unconsciously [128], but provide us with great agility and incredible ability to run, play sports, and drive cars. Therefore, in this thesis, we will focus our attention on motion planning, and assume that the rest of the robot navigation framework, i.e., control, perception, mapping, and localization tasks, are solved.

The two most popular groups of robot motion planning algorithms are the ones based on sampling the state or action space and the ones that utilize optimization. Although Sampling-Based Motion Planning (SBMP) algorithms, such as Rapidly Exploring Random Trees (RRT) [99] or Stable Sparse RRT (SST) [105], are guaranteed to find the solution, this warranty only holds for a time tending to infinity. In practice, they need a significant amount of time to find a feasible solution, which makes them unreliable in the case of the need for rapid motion planning. In turn, optimization-based motion planning algorithms typically requires much less computation to find a solution, but are prone to get stuck in suboptimal local minima or even infeasible solution. Over the last two decades, researchers have been trying to address the shortcomings of the classical planning methods. Nevertheless, they are still far from the speed of planning achieved by humans for practical traffic planning problems [131].

One of the key factors of human success is the ability to learn. We can observe that the motions of infants are rather chaotic and uncoordinated, but during the first years of life, they gain incredible motor skills through learning. Inspired by this observation, we see the necessity of enabling robots to learn in order to reach or even surpass human motor skills. Thanks to the recent advances in machine learning, it seems this is becoming possible. With the use of neural networks, even now, we are able to equip robots with similar abilities to learn and improve their skills, e.g., motion planning. In recent years, a lot of effort was put into incorporating the experience and learning into motion planning [173]. One of the first trials to reuse the motion planning experience was exploiting the memory of previously planned motions and reusing them based on the task similarity [13]. Another approach to utilize the experience is to use it to bias the sampling distribution in SBMP algorithms [28, 73, 186], or to directly predict the next states that should be attached to the search tree [142]. However, even the use of these kinds of methods is not enough to plan complex motions within tens of milliseconds. Moreover, the use of expert demonstrations by these methods usually requires inefficient data collection and strictly limits their performance. Furthermore, we do not engage in sampling of the state space nor in choosing the subsequent states to reach while we are running or playing basketball. Furthermore, we do not engage in sampling the state space nor in choosing the subsequent states to reach while we are running or playing basketball. These observations lead us to conclude that this approach may be insufficient to achieve human-level motion planning performance.

Therefore, we suspect that a new approach to learning how to plan is necessary. Thus, in this thesis, we propose a novel approach to learning how to plan motion that is rooted in the concept of *planning as inference* [19] and extensively uses the Reinforcement Learning (RL) paradigm [162] and differentiable loss functions [122], to efficiently learn based on its own experience. Thanks to this approach, we are able to learn motion planning behavior that takes into account robot

and environment constraints and generates plans at the human or even super-human level, i.e., within milliseconds [131].

While, in general, it is preferable to have motion planning methods that can generate plans fast to increase the responsiveness of the robotic systems, the necessity of rapid motion planning is especially visible in applications that require rapid responses to the changes that take place in the environment. One of the most prominent examples of this is autonomous driving, as the situation on the road can change rapidly due to the relatively high velocities of the vehicles and the unpredictability of the pedestrians. Similarly, in various sports, like football, table tennis, or fencing, the reaction time needs to be minimized to achieve the best results. Moreover, these activities also require very dynamic, complex, and precise movements, that simple reactive policy may not be able to generate. Therefore, we presume that to enable robots to achieve the agility of humans, we need more efficient motion planning approaches that utilize the experience to excel at the considered tasks.

1.2 Problem statement

In the previous section, we have shown the need for efficient robot motion planning for further development of agile robots. By efficient robot motion planning, we understand the broad aspects of making motion planning algorithms feasible to use in practical scenarios, like satisfying the constraints imposed by the robot or environment and finding a solution within tightly limited computation time. In this thesis, we would like to develop motion planning methods that are practical in the above-described sense, allow to rapidly replan the motion, taking into account constantly changing conditions, generate smooth solutions, and are reliable. To achieve this, we strongly utilize the concept of learning for motion planning, as we believe that for complex robotic systems, only the exploitation of the experience enables one to satisfy all of the aforementioned goals of efficient motion planning.

For many years, a significant effort was made to introduce motion planning methods that are complete, as this is a warranty of being able to solve the problem or to report problem infeasibility. Unfortunately, for non-trivial motion planning problems in continuous domains, these kinds of solutions are hard to define. The best we could achieve was probabilistic completeness, which is a very powerful feature of the motion planning algorithms that guarantees the solution will be found with a probability equal to 1 for planning time going to infinity [100]. Yet theoretically sound, this is not very practical as typically, in the robotic use-cases, we cannot wait so long to complete the planning, especially for planning some dynamic motions in rapidly changing environments like playing some sports or driving a car. In this thesis, we would rather like to achieve practical reliability than completeness. One of the key aspects of reliability is the small, deterministic, and known amount of computations needed to generate a plan. By knowing the result of a planning attempt in a very short fixed time, even in the case of failure robot is able to react by replanning the motion, changing the goal, or even stopping its activity and reassessing its goals and their feasibility. Moreover, fixed planning time enables planning the motion exactly from the approximately known state of the robot – the one predicted to be achieved within

planning time. This feature is very important in terms of replanning the motion on the fly, which is a crucial ability needed to achieve human-level agility [131].

Another important aspect of motion planning is the satisfaction of the constraints. Robots themselves can introduce a very severe limitation to the motion plans that are feasible for them, i.e., possible to be followed. For example, car-like robots cannot move sideways and make arbitrarily sharp turns, and basically, all movements of all robots are limited by the power of their motors and the durability of their construction. These limitations make optimization-based motion planning methods struggle due to the nonlinearity of these constraints and significantly slow down the SBMP algorithms by introducing some narrow passages in the search space. While, in general, one may trade off the quality of the solution for the planning time, it may be hazardous in the case of the hard constraints. Similar effects are also visible in terms of the extrinsic constraints stemming from the geometry of the environment. We suppose that an efficient solution to motion planning problems considering these types of constraints is to exploit the structure of the environmental and robot constraints and learn how to deal with them during motion planning. Therefore, in this thesis, we focus our attention on man-made environments as they are characterized by strong structurization and reduced complexity. Moreover, we assume that these constraints are known and well described, which is practical to achieve in the context of planning in man-made environments for known robots.

The pursuit of the aforementioned practical goals allowed us to formulate the following scientific hypothesis of the dissertation:

A wide range of robotic motion planning problems defined in man-made environments can be solved in nearly constant time using deep reinforcement learning.

This general hypothesis is followed by supportive hypotheses, which detail the benefits of using deep reinforcement learning in robot motion planning tasks as proposed in this research:

- Formalization of the path planning problem for a car-like vehicle as the Markov Decision Process (MDP) allows for sequentially building the solution using the sub-paths generated with a neural network trained using reinforcement learning.
- Using the B-spline path representation and the proposed path construction method allows for efficient planning within a single inference of the neural network and introduces an inductive bias to the learning process.
- It is possible to solve constrained kinodynamic motion planning problems using neural networks trained under the reinforcement learning paradigm and B-spline-based trajectory representation.

1.3 Related work

Motion planning is one of the fundamental components of the robotics system and is necessary to achieve the desired level of autonomy. This crucial role explains why motion planning is one of the most mature and important areas of robotic research. Despite tremendous efforts,

known planning algorithms are insufficient for many robotic tasks because they cannot handle task constraints or produce the plan within the specified amount of time. In this section, we present a brief of the most important concepts in robot motion planning and a comprehensive review of the motion planning methods that are the most related to the ones proposed in this dissertation, i.e., motion planning methods for autonomous vehicles, motion planning problems with constraints, kinodynamic motion planning problems, and learning-based motion planning techniques.

1.3.1 Overview of the motion planning methods

Robot motion planning is, in general, a computationally hard problem, specifically PSPACE-hard [145]. Even for relatively low-dimensional spaces, it is hard to find a collision-free path among the obstacles. One of the first approaches to robot motion planning focused on decomposing the state space into a graph of connected cells [100, 111, 127], so that classical graph search techniques can be used, such as breadth-first search, depth-first search, or Dijkstra algorithm [31]. However, these types of methods typically focus on the low-dimensional spaces and obstacles that are characterized by simple geometry, which limits its practical applicability for robot motion planning. More complex shapes of obstacles reduce the feasibility of these search techniques as they cause the graph, which represents the free space, to grow. Similarly, to obtain more fine-grained solutions, one has to increase the resolution of these graphs, which also results in much bigger graphs. To search these types of graphs efficiently, authors of [61] proposed an A* algorithm. It utilizes a heuristic to guide the search through the free-space graph, such that the nodes that are likely to be a part of the solution are considered first.

Nevertheless, even efficient graph search techniques may be insufficient if the free-space graph is large, which is a typical case in robotics. The number of vertices grows exponentially with the number of degrees of freedom, which makes planning particularly hard for high-dimensional systems, such as manipulators or humanoids. Similarly, as the state spaces we typically consider in robotics are continuous, planning precise motion requires one to increase the resolution of the state space discretization, which also causes a fast growth of the graph size even for relatively low-dimensional systems. To overcome these limitations, methods that construct the graph while searching through the continuous state spaces were proposed, called sampling-based motion planning algorithms [100]. Their name originates from the sampling procedure, which tries to identify new nodes in the graph and connects them to the existing search tree. Methods of this kind, starting from Probabilistic Roadmaps [85] and RRT [99], and their more recent successors like RRT* [83], Bidirectional Fast Marching Trees* [154], or Batch Informed Trees (BIT*)[49] were at the core of the motion planning research through last two decades. While these types of methods are characterized by the probabilistic completness [100], early approaches like RRT [99] in practice require significant amounts of time to find a solution path, which in the end can be far from the optimal one. To make SBMPs find optimal solutions, the RRT* algorithm was proposed [83]. In turn, to improve the execution time, informed sampling strategies were proposed as a part of Informed RRT* [50] or BIT*[49] algorithms.

The use of informed sampling strategies is also one of the most popular approaches to the recently constituted field of learning-based motion planning methods [73]. This area of research is motivated by the fact that one can take advantage by exploiting the knowledge of how to plan gained in previous planning attempts or by mimicking the behavior of some planning expert, instead of starting the planning from scratch every time. One of the first works in this field considers memorizing and reusing the already computed solutions [13]. However, nowadays most of the learning-based methods focus on improving the specific parts of the sampling-based motion planning algorithms, such as sampling [28, 142, 186], extending the nodes in a tree [114, 152], collision-checking [34, 184], or learning a guiding heuristics [68, 166]. In turn, the learning-based motion planning methods we propose in this dissertation use machine learning models to directly infer the motion plan. These kinds of approaches, while being less popular, are the topic of several recent papers like [45, 81, 153].

In parallel with sampling-based approaches, optimization-based motion planning methods have been developed. Instead of building a search tree and constructing the solution, these methods begin with an initial solution and modify it to minimize the cost function, taking into account the constraints imposed on the solution. Methods like CHOMP [189], TrajOpt [151] or GPMP2 [123] showed an ability to solve tasks like avoiding obstacles with PR2 robot [123, 151] or planning the walking movements of a quadruped [189]. The machine learning-based motion planning methods we propose in this dissertation draw inspiration from the optimization-based planners, especially [27], but instead of directly optimizing the solution, we optimize the parameters of our motion planning function. This approach allows us to spend more time on motion optimization, due to learning how to plan in an offline stage, while being extremely fast at the online inference phase, which, in fact, is similar to the concept of *planning as inference* [19]. Moreover, in the learning phase, we can use privileged information, like reference paths, to learn how to solve motion planning problems that are difficult to solve online using optimization due to the local minima.

In the next subsections, we will focus on the specific areas of the robotic motion planning research that relate the most to the techniques introduced in this dissertation.

1.3.2 Motion planning methods for autonomous vehicles

. A substantial part of this dissertation considers introducing novel approaches to machine learning-based motion planning in the specific use case of planning paths for car-like vehicles. Therefore, in this subsection, we give an overview of the motion planning methods in the autonomous vehicle context.

Early works on self-driving cars adopted the usage of graph search algorithms, especially A* [61], which was proven to be effective in solving the motion planning for the mobile robot [126]. Variants of A* were used by almost all teams that participated in the Darpa Urban Challenge [22, 37]. While A* may be considered a standalone planner, in the case of car-like vehicles, it is typically used jointly with some post-processing steps such as trajectory optimization [37, 119]. In turn, authors of [22] utilized A* only as a high-level route planner, while for the fine-grained

planning of the local motions, they adopted a sampling-based motion planner that was based on the RRTs [99].

There are several approaches in which RRT-based motion planning algorithms can be used to plan the motion of a car [130]. The one used in [22] relies on sampling the control signals. A similar approach was also used to avoid solving *two-point boundary value problem*, which may be difficult to solve for a car-like vehicle [82], by the asymptotically optimal sampling-based kinodynamic planning algorithm SST proposed in [105]. While approaches of this type are dynamically feasible, they may not be sufficient in terms of planning time. To approach this issue with sampling-based motion planning algorithms, one can focus on two aspects: (i) defining an appropriate search space, and (ii) improving the sample efficiency of the SBMP algorithms. A search space that is particularly relevant for car-like vehicles is the so-called Dubins state space [160], which is a SE(2) state space where distance is measured by the length of Dubins curves [41]. In this space, the distances are easy to compute and it is possible to connect any two states using a feasible Dubins curve, which is particularly important for SBMP algorithms. Regarding recent advances in the sample-efficiency of the SBMP algorithms, we can mention BIT* [51], which is an asymptotically optimal anytime algorithm that quickly finds feasible solutions thanks to the restriction of the search space. Its latest modification, Advanced Batch Informed Trees (ABIT*) [157], sacrifices the resolution optimality to achieve faster initial solution times. In turn, the Adaptively Informed Trees (AIT*) algorithm [158], adapts its search to each problem instance by simultaneously estimating and exploiting a problem-specific heuristic.

Planning in Dubin's state spaces is very practical from the computational point of view but also has some disadvantages. Dubins paths [41] are, in general, not smooth enough to be followed with a continuous steering angle while the vehicle is moving, such that to change between straight segments and arcs, one needs to stop a car. A direct response to this issue is the use of continous curvature Dubins curve (cc-Dubins) [48], which inserts clothoid between the segments to allow for continuous change of the steering angle. Nevertheless, even the use of cc-Dubinss results in paths that are characterized by maximum curvature segments, which are undesirable from both passenger comfort and driving speed points of view. One of the approaches that address these limitations is the State Lattices (SL) algorithm [134], which considers searching for the solution path in the dense graph, created over the state space in which edges can be defined as more sophisticated curves than plain Dubins curves with maximal curvature arcs, using A* algorithm [61]. However, as for the other approaches that utilize predefined graphs, there is a tradeoff between the planning time and the solution spatial resolution.

Another interesting idea in terms of solving motion planning problems is to avoid explicitly planning a solution but transform the problem into the control domain. Khatib in [87] proposed to utilize artificial potential fields to compute the control signals that guide the robot toward the desired state while avoiding obstacles. However, methods that are based on the artificial potential fields are difficult to apply to car-like vehicles due to their limited speed and curvature of motion [180].

An important branch of local motion planning for car-like vehicles is trajectory optimization [130]. Approaches of this type can relatively quickly generate local trajectories that are kinematically or even dynamically feasible [26, 148], however, they are typically focused on solving simple

collision avoidance problems around the reference trajectory, path or sequence of waypoints that need to be supplied by another planning algorithm [26, 58, 107]. Moreover, to maintain short planning time, the considered obstacles are typically simplified to disks, ellipses, or rectangles [36, 107, 148] or the robot dynamics is linearized [159]. These simplifications allow for computing the reference trajectory in real time but, at the same time, reduce the applicability of these trajectory optimization methods to use cases that do not require very precise motions in confined spaces. While being extremely practical, trajectory optimization methods for car-like vehicle motion planning are at a different level of the hierarchy than the planning approaches proposed in this dissertation, as typically, they are used as a lower-level trajectory generator combined with some traditional reference path search techniques, such as SL [58, 107] or Dijkstra over the route graph [36]. In a similar manner, these methods can be considered as an efficient approach to improve the existing solutions [179] that do not meet all of the requirements, e.g., slightly collide with the environment or violate the path curvature constraints. This can be a good complement to the fast path planning methods proposed in this work, as it may happen that the path generated by a neural network violates some of the constraints, which may be quickly fixed with an optimization approach.

The success of deep learning in robotics [161] increased the interest in path and motion planning methods entirely learned from data. While for the autonomous vehicles, most of the deep learning approaches are rather focused on predicting control commands directly [176], some of them consider learning how to plan trajectories, paths, or waypoints that can be then followed using a classical controller [94, 176].

A significant number of neural motion planning methods for autonomous vehicles rely on paths demonstrated by experts and use imitation learning to learn from them. Among these methods, we can find the most straightforward ones that are trained to predict sequences of timed waypoints [9, 80, 136, 146], but also those that predict parameters of the classical motion planners to improve their efficiency [177], or cost maps [68, 185] that can be further used to guide planning algorithms. Most of these methods use human expert demonstrations as a ground truth data [136, 146, 177], which enable them to learn human-like driving behaviors, however, these data is expensive to get and contain mostly very safe and not-interesting, from the efficiency of learning point of view, examples. To address this issue, authors of [68, 80] used RRT* to generate expert plans, while in [9] they propose to augment the data by adding some perturbations and go beyond simple behavior cloning by adding some loss functions that encourage desired behaviors on the planner, which seems like a step towards the reinforcement learning approaches.

While the performance of imitation learning methods is bounded by the quality of the data provided by experts and requires sometimes tedious data collection, one can avoid these limitations by learning from its own experience, following the reinforcement learning paradigm. This is particularly popular in the context of autonomous driving [5, 94], however, these solutions typically focus on generating actions directly, instead of planning the path or trajectory [5, 94]. In [74] Q-learning was used to learn how to plan the sequence of waypoints that solve motion planning problems at simulated road intersections. In turn, authors of [25] trained in a reinforcement learning paradigm efficient extend procedure that can be used by the RRT-style planning algorithm. The most similar approach to the one proposed in this dissertation was presented

in [44], which learns, in a RL paradigm, how to plan the motion of the autonomous car, by choosing y-coordinates of the two intermediate points that are then used to fit the solution path. The method proposed in [44] allows for planning only very simple maneuvers, and differs from our work in terms of the need to estimate the gradients, while the methods we propose allow for learning by directly computing the gradient of the loss functions w.r.t. generated plans, and so neural planner weights, which is more similar to the approaches proposed in [9] and [122]. Moreover, our methods focus on predicting a smooth path that provides the autonomous vehicle with the representation of the motion plan that is interpretable, possible to validate, and easily trackable in a predictable way, which is important from the safety point of view [165].

In this subsection, we discussed the motion planning approaches for autonomous vehicles, which are characterized by a special type of constraint, i.e., non-holonomic ones. In the next subsection, however, we will present the current state of knowledge about more general methods of planning with constraints.

1.3.3 Constrained motion planning

The term *contrained motion planning* can be understood in many ways. For example, in robotic applications, we may require the planning algorithm to find a solution within some specified time budget [76, 84], such that we can claim the system to be able to plan in real-time. Another interesting constraint that stems from the real-world requirements is to finish the movement within some specified time limit, which is typically addressed in the optimization-based motion planners by minimizing the movement time while maintaining the other constraints satisfied [115, 171]. However, most of the robotic research on constrained motion planning is devoted to the constraints stemming from the physics of the robot and the considered task, e.g., the limited effort of the robot motors, the limited velocity of the robot's joints, or the constrained movement of the end-effector. Thus, in this one and the following subsection, we focus on the approaches to planning the robot's motion that consider these types of constraints, with particular emphasis on planning for manipulators. Some of the tasks that we assign to robots require constraining the robot's movements to some manifold embedded in the task space, for example, keeping the cup held by the robot in an upright position or ensuring that the end-effector of a window cleaning robot remains at some constant distance from the window surface. In recent years, two main approaches to this class of problems have been developed: (i) adding task constraints to the motion optimization problem [15], (ii) sampling constraint-satisfying configurations and generating constraint-satisfying motions in the sample-based motion planners [92, 93].

Although the inclusion of additional task constraints in motion optimization is conceptually simple, it typically leads to a much more difficult optimization problem to solve, since the equations governing the kinematics of robots are usually nonlinear. To overcome the computational burden arising from these nonlinearities, the authors of [151] proposed to use the sequential quadratic programming approach that transforms the Nonlinear Programming (NLP) problem into a sequence of its convex approximations around the current solution. In [15], this idea was extended to a non-Euclidean setting, by transforming the manifold-constrained trajectory optimization problem into an equivalent problem defined in a space with a Euclidean structure. Another

general approach to solving constrained motion planning problems was proposed in [39], where authors modified the update rule of CHOMP by projecting it onto the tangent space of the constraint manifold and adding a correction term to move the solution towards the manifold. In turn, a recent approach to constrained optimization-based motion planning [66] tries to achieve faster convergence and numerical robustness by utilizing augmented Lagrangian optimization using an iterative Linear Quadratic Regulator (iLQR), introduced in [104], and systematically updating Lagrangian multipliers, which is conceptually similar to our approach to training neural network-based motion planners for constrained motion planning problems. However, in practice, if we are looking for a solution to a specific motion planning problem, it may be useful to exploit the structure of the constraints to improve planning efficiency. In [110], the problem of fast planning dynamic movements of a robotic arm whose end effector is limited to moving in a plane has been decoupled into two simpler optimization problems: planning the Cartesian trajectory in a 2D plane and then planning the trajectory in the joint space that follows the Cartesian path. However, this kind of decoupling may result in planning suboptimal trajectories as the simplified solution to the first problem is likely to cut out the optimal solution from the search space of the second one. In contrast, in this dissertation, we propose to plan directly in the robot's configuration space, and we show that it is possible to use a general approach to learning-based motion planning, to plan better trajectories than using handcrafted optimization.

In contrast to optimization-based motion planning methods, the adaptation of SBMP algorithms to solve constrained motion planning problems is not so straightforward. Simply discarding samples that are outside the constraint manifold usually results in a severely reduced likelihood of drawing an acceptable state, also connecting two states by a path that lies within the manifold is non-trivial [92, 93]. Therefore, a number of approaches to this class of problems have been proposed, such as constraint relaxation, projection onto the manifold, using the tangent space of the constraint manifold, or the topological atlas. The constraint relaxation approach originated from the assumption that there is some acceptable level of constraint violation, such that we can relax the constraints that define the manifold surface and give it some non-zero volume to allow for sampling [16, 17]. However, this approach circumvents the problem by creating a new one – planning in narrow passages [164]. To mitigate this, Constrained Bi-directional RRT (CBiRRT) algorithm was proposed [12]. This method exploits projections onto the constraint manifold to make the sampled and interpolated states satisfy the constraints. A significant reduction in planning time compared to projection-based methods was achieved by Tangent bundle RRT [89], which, instead of sampling in the configuration space, proposes to sample in the tangent space of the constraint manifold. An alternative to tangent space sampling is the approach that defines charts that parameterize the manifold locally and coordinate them by creating a topological atlas [77]. In this approach, RRT seeks the direction of the atlas expansion, and then sampling is performed only in the space defined by the atlas.

However, sampling-based methods for constrained motion planning methods usually generate only paths, neglecting the dynamics of the movement that is essential for planning the dynamic motion of the robot. Therefore, in the following subsection, we discuss approaches to kinodynamic motion planning.

1.3.4 Kinodynamic motion planning

The key point in kinodynamic motion planning is to include knowledge about the robot dynamics and its limitations in terms of the maximal torque generated by the motors or its maximal velocities, so that planned motions can be executed on the given robotic platform.

It is straightforward to convert motion optimization problems to the kinodynamic setting by including the robot's dynamics in the constraints. However, this introduces more nonlinearities to the optimization problem, slows down the solvers, and increases the likelihood that the optimization converges to a local minimum unless it is properly initialized [178].

In turn, to adjust the sampling-based motion planners to the kinodynamic setting, a significant effort has to be made in terms of both the Extend procedure of the search tree and state cost-to-go estimation. One of the first approaches to address these issues was to linearize the system and utilize Linear Quadratic Regulator (LQR) to connect states and calculate the cost of this connection [132]. To avoid problems that arise from linearization, the authors of [155] proposed using a gradient-based method to solve the NLP problem of connecting any two robot configurations. A similar idea was recently presented in [138], where Model Predictive Control (MPC) was used to perform a search tree extension.

A problem with kinodynamic planning that uses sampling is the need to sample a state space that has a high number of dimensions (as the velocities are included in the state). One of the important issues of the sampling-based approaches to kinodynamic planning is the need to sample a high-dimensional state space, since the state in kinodynamic planning needs to consist not only of the configuration but also of velocities. To mitigate this, in [188], an optimal partial-final-state-free controller was used to connect the states so that the dimensionality of the sampling space is reduced twice. An interesting alternative approach to solving SBMP problems with kinodynamic constraints is to sample controls instead of states [24, 105]. This significantly simplifies the satisfaction of the constraints, but usually leads to relatively slow planning due to the need to simulate the system behavior and efficiently guide the tree expansion. Nevertheless, none of these approaches allows one to simultaneously handle both kinodynamic and task-based constraints. To address motion planning problems of this type, an adaptation of the atlas method [77] with the LQR-based search tree Extend procedure was proposed in [18]. Unfortunately, this algorithm strongly relies on the linearization of the system and still requires a significant amount of time to prepare the plan.

In contrast to the methods mentioned above, the approach proposed in this dissertation, thanks to the use of machine learning, B-spline-based trajectory parameterization, and constraint relaxation technique [16], is capable of planning smooth trajectories that can satisfy, up to the specified constraint violation budget, an arbitrary set of constraints, including both task-related and kinodynamic ones. Moreover, we can guarantee that the solution will be computed in a small constant time and that it can be optimized to meet the execution time requirements.

1.3.5 Learning-based motion planning

In this subsection, we want to present different approaches to learning-based robot motion planning, with a particular emphasis on the methods that consider planning with task and kinodynamic constraints. The overview of the method devoted to motion planning for car-like vehicles is given in Section 1.3.2.

For many real-world robotic applications, conventional planning methods are not fast enough to meet the processing time constraints imposed by real-world dynamics (e.g., for rapid car maneuvers [88]) or produce solutions far from optimal. Moreover, tasks performed by the robots, even though they may require solving different planning problems, seem to reveal some similarities. Therefore, to obtain plans of better quality within tighter time bounds, learning-based approaches to improve robot motion planning were proposed. These types of methods originated from [13], in which, the conventional planner was used to solve the problems. However, generated plans were not only executed but also saved in the library of solved problems. After gathering plans, they can be reused to solve new but similar problems in a shorter time by simply modifying the most appropriate known path.

Most of the learning-based motion planning methods build upon the algorithms from RRT family to use their guaranties of probabilistic completeness [173]. Some of these methods try to modify the sampling distribution based on the experience and the representation of the task. In [186] neural network decides whether the sampled state should be rejected or not, whereas in [67] neural network is used to predict the state-action value function to choose the best nodes to expand. Authors of [121] used a convolutional neural network to predict some crucial regions of the environment to increase the sampling in them. At the same time, in [28], a probability distribution is directly inferred by the neural network in the form of a heatmap. In [142], the sampling distribution is encoded in a stochastic neural network (due to dropout used during inference), which is trained to predict the next state towards the goal. An adaptation of biasing the sampling distribution to comply with a manifold of constraints was presented in [101, 129], where adversarial training was used to learn how to generate data on the constraint manifold. An architecture consisting of a generator and discriminator trained jointly was also used in [141]. This work builds upon [142] and uses a generator to predict the next state, but extends it with a discriminator, which is used to predict deviation from the constraint manifold and to project the predictions on it.

Another approach to learning-based planning is to utilize a neural network to transform the considered planning space into a new one that poses features that are useful from the motion planning perspective, and another one to retrieve the solution in the original space. At first, this concept was proposed in [72], where the higher-dimensional planning space is transformed to the lower one, which should have a positive effect on the planning efficiency of SBMP methods. For efficient planning in the latent space, authors of [72] also propose to learn dynamics and collision-checking networks to mirror the local steering and collision-checking procedures of the traditional SBMP. Notably, these networks can be trained through only raw data of the system's states and actions, along with a supervising collision checker, without the need for expert planner demonstrations. Instead of using the concepts of the SBMP algorithms, authors of [69] proposed

to use a gradient-based optimization in the latent space to navigate from the initial to the desired robot state, taking into account a single obstacle represented by the neural network that predicts collisions with it, while in the recent paper [181] they extended the approach with the scene embeddings to plan in more complex scenarios. In turn, in [4] a collision-free mapping between the state space and latent space is proposed. The idea is to generate a latent space that allows for connecting any two states using linear segments that are collision-free after remapping to the original planning space. The main drawback of the approaches of these types is that we have no direct control over the transients between the points from the planned sequence. Moreover, they still require a sequential process of planning, by sampling or by gradient-based updates, which may not be fast enough for time-critical applications.

Interesting utilization of previous experience in the context of motion planning is proposed in [78] and [8]. Instead of learning how to generate a solution or improving the parts of the SBMP algorithms, they focus on trajectory optimization, which is supported by the knowledge gained in past optimization attempts. Trajectories predicted by both non-parametric [78] and parametric [8] models are used to warm-start the classical trajectory optimization. This kind of approach to learning-based motion planning can be an interesting use case for the fast motion planning methods proposed in this dissertation, as they rapidly generate solutions that can be further optimized using standard trajectory optimization approaches.

Learning-based approaches are also used to improve the feasibility of solving kinodynamic motion planning problems. Authors of [175] emphasized that solving a two-point boundary value problem for a nonlinear dynamical system is NP-hard and proposed to learn a distance metric and steering function to connect two nodes of RRT, based on the examples generated using optimal control. In turn, in [6], the learning of control policy is done indirectly by learning the state trajectory of some optimal motion planner and then using inverse dynamics of the system to determine controls. A lazy approach to limit the number of computationally expensive NLP solver calls was presented in [182], where the neural network, instead of replacing the solver, is trained to predict which nodes of a RRT are steerable to, and what will be the cost of steering. Contrary to these methods, authors of [102] propose to use a MPC extensively to solve local NLP problems and learn only how to determine the next state in a way towards the goal using the experience gained by mimicking demonstration trajectories.

In contrast to these methods, the approach proposed in this thesis offers an extremely fast way of finding near-optimal trajectories that are able to solve constrained kinodynamic motion planning problems. Moreover, our proposed approach is able to smoothly replan the motion on-the-fly, which is not possible with the use of state-of-the-art constrained kinodynamic motion planning algorithms. Furthermore, the solutions proposed in this dissertation, do not require demonstration trajectories, as they learn from their own experience. Thus, their performance is not upper-bounded by the quality of the demonstrations.

1.4 Proposed solution

In the previous section, we have shown a broad perspective on the existing motion planning methods. We intentionally used the terms planner and algorithm interchangeably, because all of the presented solutions are in fact algorithms, as they gradually construct or improve the solution according to some prescribed sequence of instructions and procedures. However, all of these methods can be seen from a slightly different perspective, which focuses on the input-output characteristics disregarding the specific way of processing. In this sense, we can describe any motion planning algorithm as a function

$$f : \mathcal{P} \times \mathbb{N}_+ \rightarrow \mathcal{S} \cup \{\aleph\}, \tag{1.1}$$

that transforms elements of the set of all motion planning problems $\mathcal{P}$, taking into account the value of some random seed $\mathfrak{s} \in \mathbb{N}_+$ (to include probabilistic algorithms), into the elements of the set of all solutions $\mathcal{S}$ or into a symbol of no solution $\aleph$, under some specified motion planning problems and solutions representations. This perspective on motion planning is schematically shown in Figure 1.1, where the representation of the motion planning problem, i.e., environment map and initial and desired configurations is transformed by some motion planning function into the representation of the solution, i.e., a path. In this formalism, for example, we can describe a complete algorithm as a motion planning function that for all elements of the set $\mathcal{P}$ returns a feasible solution or report that it does not exist. Showing the inexistence of the solution for non-trivial motion planning problems is a tough challenge and an active area of research [100, 103]. However, in this thesis, we will focus our attention on the subset of motion planning problems that have solutions, i.e., a set of feasible motion planning problems $\mathcal{P}_f \subset \mathcal{P}$, and functions defined on this set. One very special motion planning function defined on the set of solvable motion planning problems $\mathcal{P}_f$ is an *ideal planning function* defined by

$$f^* : \mathcal{P}_f \rightarrow \mathcal{S}, \tag{1.2}$$

which transforms all feasible motion planning problems into solutions. In fact, we can presume that there exists a big family of these kinds of functions F^* as many motion planning problems have more than one solution. In practice, we would be more interested in solutions that are optimal in some predefined sense, e.g., minimizing the time needed to execute the solution, or energy input required. Therefore, we focus our attention on a specific *ideal planning function*, i.e., *optimal ideal planning function* f_o^* defined by

$$f_o^* = \underset{f^* \in F^*}{\operatorname{argmin}} \forall_{p \in \mathcal{P}_f} J(f^*(p)), \tag{1.3}$$

which for all feasible motion planning problems generates solutions that are optimal w.r.t. a given optimality criterion J.

One of the most important goals of motion planning would be to know the formula of this function or at least be able to evaluate it in some reasonable time. However, for most of the practical motion planning problems, it is extremely hard to provide a closed form of this function, and trials made to replicate its behavior with an algorithm still require tremendous amounts of

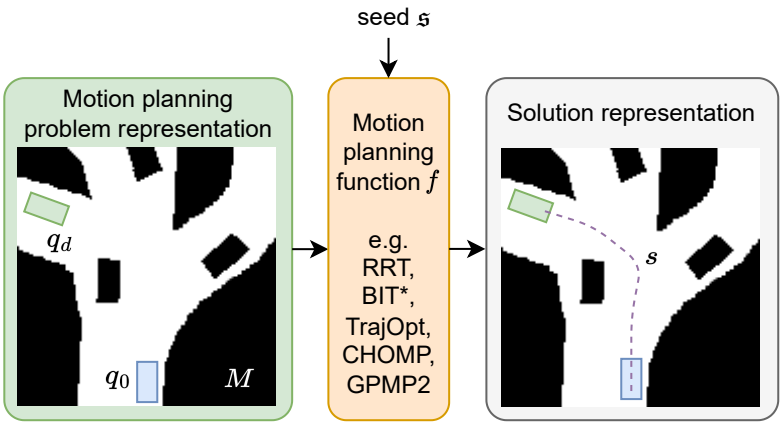

FIGURE 1.1: Scheme of the functional perspective on the robot motion planning. Every planning algorithm can be viewed as a planning function that transforms the problem representation into the representation of the solution.

computation. Therefore, the key point of the solution proposed in this thesis is to exploit the experience and learning to approximate the *optimal ideal planning function*. In particular, to make this approximation feasible we focus only on some very narrow subsets of the set $\mathcal{P}_f$ and utilize a Neural Network (NN) to serve as a universal apprixmator [133]. To train a NN to be able to generate motion plans, as the *optimal ideal planning function*, we need to somehow copy its behavior. One of the most popular approaches to this problem is to utilize behavior cloning [149] and try to directly copy the responses of the function f_o^* to some sample input motion planning problems. Unfortunately, this approach requires spending tremendous amounts of computation to find optimal solutions to the given planning problems using probabilistically-complete motion planning methods. Moreover, it gives no clue, during the learning process, about the feasibility of the solution generated by the NN as it allows only to asses the similarity to the ground truth solution. Having in mind these significant limitations of behavior cloning, we propose to follow a different approach. Instead of blindly copying the behavior, we propose to describe mathematically the properties which the *optimal ideal planning function* should have, i.e., define this function implicitly and try to learn how to satisfy the same properties with the plans generated by the NN. By doing so we can validate any plan generated by the NN against the set of criteria that have to be met by the *optimal ideal planning function*, without the need to collect a specific optimal solution. This approach, is an instance of the RL paradigm, as we do not know the ground truth output the NN should generate, but rely on some function that asses the quality of the NN response directly.

To further improve the quality of the *optimal ideal planning function* approximation, we propose ways to utilize the structure of both problem and solution representations, introduce new ways of constructing the solutions from the neural network outputs, and propose differentiable learning procedures based on the geometrical properties of the solution representation. Thanks to these advances, we are able to efficiently learn how to plan and replan near-optimal motions, within the time scale of at most a few inferences of the NN, for a few very challenging classes of motion planning problems.

1.5 Content of the thesis

The content of the dissertation is divided into two main parts. The machine learning-based approach to path planning for autonomous vehicles is presented in Chapters 2 and 3, while in Chapter 4 we introduce a more general approach to machine learning-based trajectory planning with a particular focus on robotic manipulation. The proposed methods are experimentally evaluated and quantitatively and qualitatively analyzed in chapters 5 and 6. The dissertation is concluded in Chapter 7.

Chapter 2 describes the implementation of the idea of planning by learning how to approximate the *ideal optimal planning function* for planning local maneuvers for car-like vehicles. The chapter formalizes the planning problem as a MDP and proposes a neural network to represent the planning policy.

The core idea of Chapter 2 is extended in Chapter 3. In this chapter, MDP formalism is abandoned in favor of the bandit formulation, making it possible to speed up the planning process. Moreover, in this chapter, we introduce a B-spline path representation and a novel procedure for path construction, which allow us to directly impose the boundary constraints on the solution and introduce an inductive bias to the proposed neural planner.

While the previous chapters focus on path planning for non-holonomic mobile robots, in Chapter 4 we put an emphasis on the kinodynamic motion planning for robotic manipulators. In this chapter, we introduce a general approach to learning how to plan in the close vicinity of the constraint manifold, and a novel B-spline trajectory parametrization. Thanks to these advancements, in the same chapter, we propose a machine learning-based planning algorithm able to solve tedious constrained kinodynamic motion planning problems in several milliseconds and replan the motion of the robot on-the-fly.

In Chapter 5 we present the experimental evaluation of the machine learning-based path planning methods for car-like vehicles introduced in Chapters 2 and 3. The experimental evaluation of their capabilities and comparison with baseline motion planners is performed on the dataset of local motion planning problems introduced in Chapter 2, while the qualitative comparison is done on some predefined scenarios simulated using CARLA [38].

The experiments regarding constrained kinodynamic learning-based motion planning are presented in Chapter 6. The chapter focuses on the trajectory planning for robotic manipulators, presenting the applicability of the general method proposed in Chapter 4 for two challenging tasks: (i) rapid movement of a heavy vertically oriented cuboid, and (ii) fast shooting in the robotic Air Hockey. The proposed approach is compared to state-of-the-art motion planners for both tasks and evaluated on the real robotic setup for the second one. Moreover, in this chapter, we present the abilities of the proposed method to replan the motion on-the-fly and show it in action by performing some trick shots.

The dissertation is summarized in Chapter 7. This chapter also contains conclusions with clearly stated contributions of the dissertation. The final part contains the future work plans.

1.6 Projects and publications

The work presented in the thesis was possible due to funding from several sources. The author of the thesis took part in the following projects as:

- **Researcher** in "Robotic technologies for the manipulation of complex deformable linear objects (REMODEL)" funded by the European Commission in the Horizon 2020 framework under the grant agreement No 870133, 01.2020–10.2023.
- **Researcher** in "Perception and control in the task of robotic manipulation of elastic objects (RoManElO)" funded by the National Centre for Research and Development under the grant No LIDER/3/0183/L-7/15/NCBR/2016, 03.2019–12.2020.

P. Kicki also received the following scholarships:

- **NAWA STER** – 3-month Ph.D. internship at Technische Universität Darmstadt in the group of prof. Jan Peters within the MOBILITY part of the programme of Internationalisation of doctoral schools, grant no. BPI/STE/2021/1/00005 funded by National Agency for Academic Exchange,
- **ICRA Travel Grants** for the participation in the 39th International Conference on Robotics and Automation 2022 in Philadelphia and 40th International Conference on Robotics and Automation 2023 in London,
- **NAWA PROM** – 2 weeks of the research stay at University of Technology Sydney and further remote cooperation within the programme of International scholarship exchange of Ph.D. candidates and academic staff, grant no. PPI/PRO/2018/1/00005 funded by National Agency for Academic Exchange.

The results of the dissertation were already presented in the following journal articles:

1. **P. Kicki**, T. Gawron, K. Ćwian, M. Ozay, P. Skrzypczyński, Learning from experience for rapid generation of local car maneuvers, Engineering Applications of Artificial Intelligence, Volume 105, 2021, 104399, October 2021, **IF_{2021}: 7.802, Quartile: Q1 in Artificial Intelligence**.
2. **P. Kicki**, P. Liu, D. Tateo, H. Bou-Ammar, K. Walas, P. Skrzypczyński, J. Peters, "Fast Kinodynamic Planning on the Constraint Manifold", IEEE Transactions on Robotics (early access), **IF_{2023}: 7.8, Quartile: Q1 in Control and Systems Engineering**.

The thesis also contains results presented during the conferences and workshops:

1. **P. Kicki**, P. Liu, D. Tateo, H. Bou-Ammar, K. Walas, P. Skrzypczyński, J. Peters, "Towards Fast Kinodynamic Planning on the Constraint Manifold", 2023 International Conference on Robotics and Automation, Agile Movements: Animal Behavior, Biomechanics, and Robot Devices workshop, London, Great Britain.

2. **P. Kicki**, P. Liu, D. Tateo, K. Walas, P. Skrzypczyński, J. Peters, "Fast Constrained Kinodynamic Neural Motion Planning", 34th Polish Conference on Artificial Intelligence, 2023, Łódź, Poland.
3. **P. Kicki**, P. Skrzypczyński, "Speeding up deep neural network-based planning of local car maneuvers via efficient B-spline path construction", 2022 International Conference on Robotics and Automation (ICRA), Philadelphia, PA, USA, 2022, pp. 4422-4428.
4. **P. Kicki**, P. Liu, D. Tateo, P. Skrzypczyński, J. Peters, "Fast Motion Planning of a 7-DoF Manipulator Arm for Robotic Air-Hockey ", 3rd Polish Conference on Artificial Intelligence, 2022, Gdynia, Poland.
5. **P. Kicki**, P. Skrzypczyński, "Speeding up DNN-based planning of local maneuvers via efficient B-spline path construction", 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems, Combining Learning and Motion Planning workshop, virtual.
6. **P. Kicki**, T. Gawron, P. Skrzypczyński, "Learning Rapid Maneuver Planning for Car-Like Vehicles Using Gradient-based Policy Search", 2020 Robotics: Science and Systems, Learning (in) Task and Motion Planning workshop, virtual.

Chapter 2

Learning Rapid Maneuver Planning for Car-Like Vehicles Using Gradient-based Policy Search

2.1 Introduction

While most of the robots are statically mounted industrial manipulators, the advent of service robotics brings to the table a massive proliferation of mobile robots. In recent years, there has been intensified work on the development of autonomous cars, as they are believed to significantly change the way we travel and transport goods. To make self-driving cars to be used on public roads, several key enabling technologies must be developed, such as efficient environment perception, vehicle localization, and motion planning, to name a few [63, 79]. While most of these challenges seem to be solved already, a very important aspect of self-driving is the innate need to create systems able to rapidly react to the sudden changes in the environment, which constantly happen in road traffic.

To avoid collisions and move nimbly in traffic, accurate and extremely fast local motion planning systems are needed [30]. However, planning for self-driving cars needs to consider the constraints imposed by the kinematics of the cars and states of scenes perceived by their sensors, which makes it difficult to fit into the time frame. While state-of-the-art path planning methods are considered sufficient for autonomous vehicles [55], their ability to solve highly constrained planning problems comes at a high computational cost. In contrast, lane changing, parking, or overtaking maneuvers are relatively easy for experienced human drivers, who in a short time horizon intuitively generate nearly-optimal paths for their vehicles, exploiting the previous experience in performing these maneuvers [57]. However, conventional path planners [100] do not take advantage of the prior experience that comes from learning from similar cases solved earlier. This seems to waste a huge potential of exploiting the experience gained in the past to create, through machine learning, knowledge useful for solving motion planning problems.

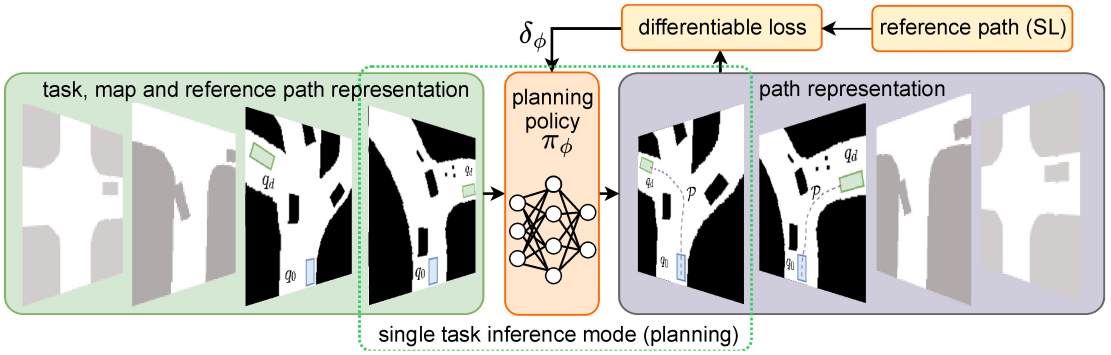

FIGURE 2.1: Conceptual scheme of the rapid path generation system. We use gradient-based policy search to teach the neural network to plan feasible paths.

Therefore, we want to investigate whether is it possible to learn a path planning policy, which is then used to rapidly generate feasible and near-optimal local paths for typical maneuvers of a car-like vehicle subject to severe kinematic constraints. To do so, we propose a motion planning policy represented by a deep neural network, which based on the representation of the motion planning problem, generates a proposition of the solution (Fig. 2.1). To learn how to approximate the *optimal ideal planning function*, we utilize the gradient-based policy search approach [40] in a reinforcement learning framework with a differentiable loss function and weak supervision. While the proposed loss function relies on the reference paths, these paths are used only in the training stage to retreat from the collision areas. Therefore, they do not constitute a performance upper bound for the proposed method (weak supervision), which can find paths that are shorter and/or smoother than the reference ones. Thanks to this property, the reference paths may be easily computed (as they do not have to be optimal) using any complete path planning algorithm able to handle the constraints present in the considered problem.

In this chapter, we present the following contributions to the field of motion planning:

1. An approach for rapid path generation under differential constraints by approximating the *optimal ideal planning function* by a neural network.

2. A novel differentiable loss function that penalizes infeasible paths, due to the violation of the constraints imposed either by the vehicle kinematics or the environment map, and encourages the paths to be optimal with respect to the chosen criterion.

3. We introduce a dataset of local maps of the urban environment (based on real sensory data) and motion planning scenarios that can be used for the training and evaluation of local planners for self-driving cars. Using this dataset, we demonstrate the generalization abilities of the learned model of our network, illustrate its advantages over selected state-of-the-art planning algorithms, and quantitatively analyze the accuracy of the proposed solution, planning times, and properties of the generated paths.

2.2 Problem definition

To present the proposed neural network-based approach to motion planning for car-like vehicles, we need first to define the problem we are trying to solve. In this chapter, we consider the problem of planning a feasible monotonic path ζ, from the initial state of the robot $\boldsymbol{q}_0$ to a subset of desired robot configurations Q_d. To formalize this definition, we need to introduce the notion of the robot state $\boldsymbol{q}$ defined by

$$\boldsymbol{q} = \begin{bmatrix} x \\ y \\ \theta \\ \beta \end{bmatrix} \in Q, \tag{2.1}$$

where θ, x and y define vehicle orientation and position in the global coordinate frame, β is the angle of the virtual steering wheel in the kinematic bicycle model [144], and $Q = \mathbb{R}^2 \times \mathbb{S}^2$ is the state space. To simplify the description of the kinematics of the considered system, we assume that the x and y coordinates describe the position of the velocity guiding point p_G located in the middle of the vehicle's rear axle. Therefore, the kinematics of the car-like vehicle can be written as

$$\dot{\boldsymbol{q}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ \frac{\tan\beta}{L} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \xi \end{bmatrix}, \tag{2.2}$$

where ξ is the rate of change of the steering wheel angle, v is the speed of the vehicle's guiding point p_G, and L is the distance between the front and rear axles. In this thesis, we assume a simplified kinematic car model, i.e., no lateral and longitudinal slip [144], which is subject to the limited steering angle $\beta \in [-\beta_{max}; \beta_{max}]$. The last but not least aspect of the object whose movement we want to plan is its body. We model the car body Π with a rectangle aligned with the robot orientation θ. Finally, we can define the solution path ζ as a parametric curve that maps the parameter $s \in [0; 1]$ into the elements of the state space Q.

Having defined the kinematic car model and the form of a solution path we want to plan, we can focus on explaining the keywords included in the problem definition stated above. Firstly, the monotonicity of the path means that the velocity v along this path cannot change its sign, i.e., we limit our planner to solve only local motion planning problems that do not require switching between moving forward and backward. In turn, the path feasibility we understood as the possibility to be safely followed, i.e., a path has to be collision-free and satisfy the constraints imposed by the car kinematics. We assume path ζ to be collision-free if the swath $\mathfrak{S}(\zeta, \Pi)$ of the car body Π along the path ζ is a subset of the free space $\mathfrak{F}$, i.e.,

$$\mathfrak{S}(\zeta, \Pi) = \bigcup_{\boldsymbol{q} \in \zeta} \Pi(\boldsymbol{q}) \subset \mathfrak{F}. \tag{2.3}$$

In our setup, environment $\mathcal{E}$ is modeled with a 128×128 occupancy grid with resolution of 0.2 m, thus we consider a 25.6×25.6 m local map. This size of the map is sufficient to represent the

robot's nearby environment to enable local traffic planning for a wide range of urban driving scenarios (the maximal allowed velocity in the urban areas in most EU countries is 50km/h, which is equivalent to about 2s of the planning temporal horizon). In turn, the chosen resolution is a trade-off between the ability to represent small obstacles and the size of the environment representation that needs to be processed by the motion planning algorithm. The free space $\mathfrak{F}$ is a union of the empty cells of the above-mentioned grid map. In turn, the satisfaction of the kinematics constraints of a car-like vehicle means that the movement of the vehicle guiding point p_G along a path ζ is aligned with the planned car orientation θ, i.e.,

$$\theta(s) = \tan \frac{y'(s)}{x'(s)}, \tag{2.4}$$

where $x' = \frac{dx}{ds}$, and the curvature κ of the path ζ, defined by

$$\kappa(s) = \frac{x'(s)y''(s) - x''(s)y'(s)}{(x'^2(s) + y'^2(s))^{\frac{3}{2}}}, \tag{2.5}$$

is limited to the following range $\kappa \in [-\kappa_{max}, \kappa_{max}]$ for all $s \in [0; 1]$, where $\kappa_{max} = \frac{1}{L} \tan \beta_{max}$. One can see that due to the differential flatness of the considered system [11, 46], all abovementioned constraints can be defined as functions of the x and y coordinates along the path and their derivatives w.r.t. phase variable s. Thus, instead of defining the path ζ as a mapping into the configuration space Q, we can compactly define it as a twice differentiable mapping

$$\mathbb{C}^2 \ni \zeta : [0; 1] \to \mathbb{R}^2, \tag{2.6}$$

and introduce a transformation $\mathfrak{b}$ between any point on the solution path ζ and state space Q, defined by

$$\mathfrak{b}(\zeta(s)) = \begin{bmatrix} \zeta_1(s) \\ \zeta_2(s) \\ \tan \frac{\zeta_2'(s)}{\zeta_1'(s)} \\ \arctan\left(L \frac{\zeta_1'(s)\zeta_2''(s) - \zeta_1''(s)\zeta_2'(s)}{(\zeta_1'^2(s) + \zeta_2'^2(s))^{\frac{3}{2}}} \right) \end{bmatrix}. \tag{2.7}$$

In Figure 2.2, we present a schematic view of the motion planning problem described above. Red areas represent the complement of the free space $\mathfrak{F}$, i.e., the collision space, the blue rectangle is the body of the vehicle at the initial state $\boldsymbol{q}_0$, while the green rectangle depicts the set of the desired configurations Q_d.

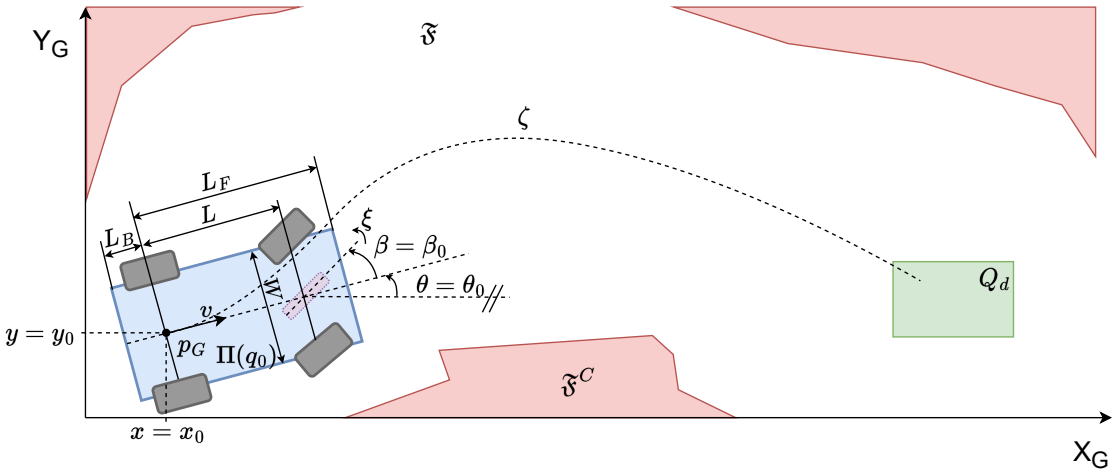

FIGURE 2.2: A scheme of the considered motion planning problem. Planning a monotonic, feasible path from an initial state $\boldsymbol{q}_0 = \begin{bmatrix} x_0 & y_0 & \theta_0 & \beta_0 \end{bmatrix}^T$ to a subset of desired configurations Q_d.

2.3 Proposed solution

2.3.1 Path planning as Markov Decision Process

One of the key ideas of the proposed solution to robotic motion planning is the use of the RL paradigm to learn how to plan. As we mentioned in the Introduction RL enables us to learn based on the evaluations of the generated plan against some loss functions that implicitly define the behavior of the *ideal planning function*. In RL two basic classes of problems are considered, i.e., bandit problems that assume single-state environments, and MDPs that describe sequential decision-making problems [162]. In this thesis, we cover both problem types. However, in this chapter, we focus on the formulation of the motion planning problem as MDP.

MDP is, in general, a discrete-time stochastic control process with the Markov property imposed on the state transitions. We define an MDP as a 4-tuple (Q, A, T, R), where:

- Q is the set of all possible states called state space,
- A is the set of all possible actions called action space,
- T is a transistion function that describes the probability $P(\boldsymbol{q}'|\boldsymbol{q}, a)$ of reaching state $\boldsymbol{q}'$ from state $\boldsymbol{q}$ by taking the action a,
- R is a reward function that assigns a reward r to the given state-action pair $(\boldsymbol{q}, a)$.

At each time step t, the process is in some state $\boldsymbol{q}_t \in Q$, and the actor may choose any action $a \in A$ that is available in state $\boldsymbol{q}$. The process responds by randomly moving into a new state $\boldsymbol{q}_{t+1}$, according to the probabilities defined by the transition function T, and giving a corresponding reward $r = R(\boldsymbol{q}_t, a)$.

Using the MDP formalism introduced above, we can define the process of path planning for a car-like vehicle. State space Q of the MDP is then defined as the state space of the robot (2.1).

Subsequent actions can be represeted with subpaths $\mathbb{C}^2 \ni \zeta^i : [\alpha_i, \alpha_{i+1}] \to \mathbb{R}^2$ of the solution path ζ (2.6), where $[\alpha_i, \alpha_{i+1}] \subset [0; 1]$, defines the i-th subpath domain, and $\bigcup_{i=1,\dots,n_{seg}} [\alpha_i; \alpha_{i+1}] = [0; 1]$, where n_{seg} is the number of subpaths. We assume that in a given state $\boldsymbol{q}_i$, the only possible action-paths are those which start at the state $\boldsymbol{q}_i = \mathfrak{b}(\zeta^i(\alpha_i))$. Transition function T defines the probability of reaching state $\boldsymbol{q}_{i+1}$ by following the subpath ζ^i. While the reward function R assesses the feasibility and quality of the given path ζ^i. As the path ζ^i always contains the state $\boldsymbol{q}$, we can drop the dependency on the state $\boldsymbol{q}$.

Thanks to some properties of the problem of planning local maneuvers for car-like vehicles, we can narrow down some parts of the MDP definition introduced above. For first, we will consider only finite-horizon MDPs, as we expect to reach the goal in a finite sequence of actions. Secondly, we assume that the paths can be followed with arbitrary precision, thus the transition function T degenerates and assigns the probability of reaching the subpath's final state $P(\zeta^i(\alpha_{i+1})|\boldsymbol{q}, \zeta^i)$ equal to 1 and 0 otherwise. By doing so, we no longer have any stochasticity in the process, thus we obtained a deterministic MDP. Moreover, in motion planning problems, the same solution path ζ in different environments or under different task requirements may be considered feasible or infeasible, optimal or nonoptimal. Therefore, we introduce the context C, which contains information about the current task and the state of the robot's environment. We assume that within a planning episode (i.e., a single maneuver), the context remains unchanged.

The part of the MDP definition, that strongly depends on the context is the reward function R. Due to the introduction of the context and the assumption that subpaths begin at the actual state $\boldsymbol{q}_i = \mathfrak{b}(\zeta^i(\alpha_i))$, we can redefine the arguments of the reward function as $R(\zeta^i, C)$. However, as in the proposed learning method we utilize the weak supervision by using the reference paths ζ^r in the learning process, we also need to add it to the arguments of the reward function $R(\zeta^i, C, \zeta^r)$.

To solve the considered problem (see section 2.2), instead of applying a search algorithm to every single problem instance, we utilize the MDP formulation introduced above and aim to find a policy $\pi(\boldsymbol{q}, C)$. The policy $\pi(\boldsymbol{q}, C)$ is a function that specifies the action the agent will choose in state $\boldsymbol{q}$, taking into account the context C. Thanks to framing the problem into the MDP formalism, we know that the optimal policy is Markovian and deterministic [139], therefore we can approximate it with a function of the current state $\boldsymbol{q}$ and the motion planning problem context C.

2.3.2 Action and context definition

In the previous section, we introduced the $\pi(\boldsymbol{q}, C)$, which based on the state $\boldsymbol{q}$ and context C, determines the action a. While the state is relatively straightforward to represent, as it can be described just by 4 numbers (see (2.1)), the definitions of action a and context C require much more detailed description.

2.3.2.1 Action

As we stated in section 2.3.1, actions are in our path planning problem defined as solution subpaths ζ^i. However, to enable the policy π to compute them, we need an appropriate path representation. In the proposed solution, solution path ζ is represented as a spline of the 5th-degree polynomials as they have a sufficient level of smoothness so they may be tracked by a car with a continuous steering angle [59]. Therefore, the subpaths ζ^i are 5th-degree polynomials, which must satisfy, at all connection points, the following continuity conditions

$$\zeta_0^i(\alpha_{i+1}) = \zeta_0^{i+1}(\alpha_{i+1}), \tag{2.8}$$

$$\zeta_1^i(\alpha_{i+1}) = \zeta_1^{i+1}(\alpha_{i+1}), \tag{2.9}$$

$$\frac{d\zeta_0^i}{ds}(\alpha_{i+1}) = \frac{d\zeta_0^{i+1}}{ds}(\alpha_{i+1}) \tag{2.10}$$

$$\frac{d\zeta_1^i}{ds}(\alpha_{i+1}) = \frac{d\zeta_1^{i+1}}{ds}(\alpha_{i+1}) \tag{2.11}$$

$$\frac{d^2\zeta_0^i}{ds^2}(\alpha_{i+1}) = \frac{d^2\zeta_0^{i+1}}{ds^2}(\alpha_{i+1}) \tag{2.12}$$

$$\frac{d^2\zeta_1^i}{ds^2}(\alpha_{i+1}) = \frac{d^2\zeta_1^{i+1}}{ds^2}(\alpha_{i+1}), \tag{2.13}$$

to form a spline. Although we generally allow these subpaths to be polynomial functions of the phase variable s, control algorithms, which in the end will be used to track planned paths, may require some specific representation, e.g., Vector Field Orientation (VFO) [116] control algorithm requires the path to be represented as level-curve [53]. Therefore, we represent the subsequent segments by functions of the form $y = f(x)$. To satisfy this, we redefine the i-th subpath by

$$\zeta^i(s_i) = \begin{bmatrix} x_i \cdot s_i \\ \sum_{j=0}^{5} m_{ij} s_i^j (1 - s_i)^{5-j} \end{bmatrix}, \tag{2.14}$$

where m_{ij} are the polynomial coefficients and x_i is the range of the movement in the x axis, and $s_i = \frac{s-\alpha_i}{\alpha_{i+1}-\alpha_i}$. Unfortunately, this definition is very restrictive, as it limits the orientation along the path to $(-\frac{\pi}{2}; \frac{\pi}{2})$ interval. To alleviate the level-curve requirements and not restrict the representational power of our path representation too much, we choose not to interpret (2.14) in the global coordinate system, but in the local coordinates instead. Starting from the initial state $\boldsymbol{q}_0 = \begin{bmatrix} x_0 & y_0 & \theta_0 & \beta_0 \end{bmatrix}^T$, we define the first Local Coordinate System (LCS) $\mathfrak{L}_1$ at the point (x_0, y_0) with orientatioon defined by θ_0, all described in the global coordinate system. Next, LCSs are constructed in the same way based on the subsequent states $\boldsymbol{q}_i$ for $i \in 1, 2, \ldots, n_{seg}$, but each of which is defined in the LCS that preceded it. The sequence of LCSs, dependencies between them, and defined subpaths ζ^i are schematically presented in Figure 2.3. Our formulation enables us to represent paths that are not functions of the x coordinate. However, it limits the maximum possible turn during the maneuver to $\frac{n_{seg}\pi}{2}$rad, which is enough for most of the typical local car maneuvers. The number of segments n_{seg} affects not only the ability to represent complex maneuvers but also the path computation time, as in the proposed approach amount of computations scales linearly with it.

Assuming that we know the initial state $\boldsymbol{q}_0$, we can define the aforementioned spline using a

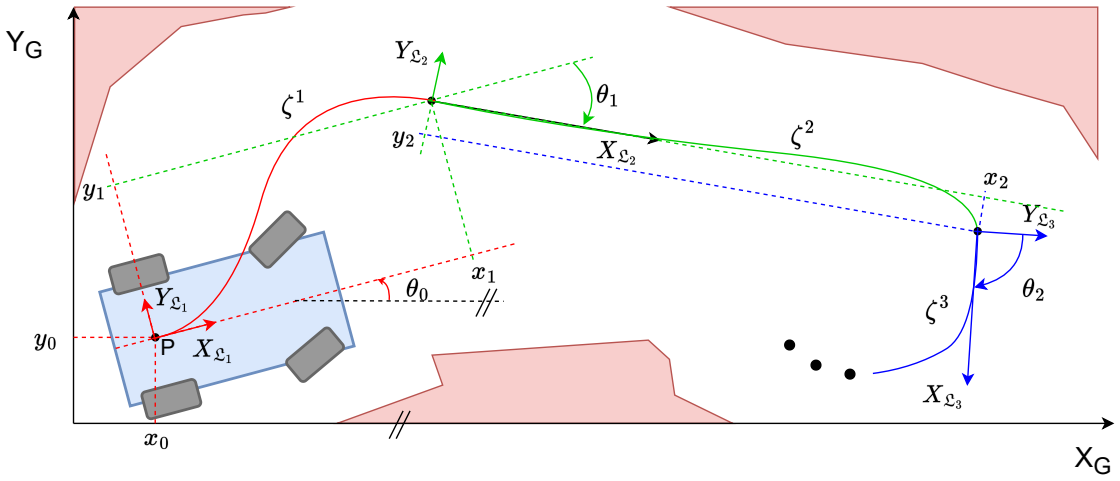

FIGURE 2.3: Scheme of the sequence of LCSs. Each subpath ζ^i as well as each $(i+1)$-th LCS is deifned in the i-th LCS.

$n_{seg} \times 4$ matrix S_ζ of parameters of segment endpoints. Each row of the matrix defines a 2D position (x, y) of an endpoint as well as the first $\frac{dy}{dx}|_{(x,y)}$ and the second $\frac{d^2y}{dx^2}|_{(x,y)}$ derivative of the path at that point. Notice that all parameters in the i-th row of the S_ζ are expressed with respect to the i-th local coordinate system $\mathfrak{L}_i$. Note that for the second derivative, the coordinate system does not matter. However, for brevity, we will assume that all parameters are expressed in i-th LCS.

From the transition point of view, it is important to determine the state $\boldsymbol{q}_i$ of the robot after performing an action S_{ζ_i} in state $\boldsymbol{q}_{i-1}$. Let's observe that each row of the S_ζ matrix unambiguously defines this state by the following transformation

$$\boldsymbol{q}_i = \begin{bmatrix} x_{i-1} + x_{\mathfrak{L}_i} \cos\theta_{i-1} - y_{\mathfrak{L}_i} \sin\theta_{i-1} \\ y_{i-1} + x_{\mathfrak{L}_i} \sin\theta_{i-1} + y_{\mathfrak{L}_i} \cos\theta_{i-1} \\ \theta_{i-1} + \tan\left(\frac{dy}{dx}|_{(x_{\mathfrak{L}i}, y_{\mathfrak{L}i})}\right) \\ \arctan\left(L \frac{\frac{d^2y}{dx^2}|_{(x_{\mathfrak{L}i}, y_{\mathfrak{L}i})}}{(1+(\frac{dy}{dx}|_{(x_{\mathfrak{L}i}, y_{\mathfrak{L}i})})^2)^{\frac{3}{2}}}\right) \end{bmatrix}. \tag{2.15}$$

Moreover, based on the current state $\boldsymbol{q}_{i-1}$ and these parameters one can determine the m_{ij} parameters of the locally defined polynomial (2.14) using following formulas

$$\begin{aligned} m_{i0} &= 0, \\ m_{i1} &= 0, \\ m_{i2} &= \frac{1}{2L} x_{\mathfrak{L}i}^2 \tan\beta_{i-1}, \\ m_{i5} &= y_{\mathfrak{L}i}, \\ m_{i4} &= 5m_{i5} - \frac{dy}{dx}|_{(x_{\mathfrak{L}i}, y_{\mathfrak{L}i})} x_{\mathfrak{L}i}, \\ m_{i3} &= \frac{1}{2}\left(\frac{d^2y}{dx^2}|_{(x_{\mathfrak{L}i}, y_{\mathfrak{L}i})} x_{\mathfrak{L}i}^2 - 20m_{i5} + 8m_{i4}\right). \end{aligned} \tag{2.16}$$

2.3.2.2 Context

A second argument of the policy $\pi(\boldsymbol{q}, C)$ is the context $C = (E, \mathcal{T})$, which we define as a tuple of environment representation E and task definition $\mathcal{T}$. As we mentioned in the problem definition (see 2.2) the information about the robot's environment $\mathcal{E}$ is represented with an occupancy grid map E, which defines the free space $\mathfrak{F}$, and its complement $\mathfrak{F}^C$, that is a crucial information from the perspective of planning and evaluating paths. The second element of the context is the definition of the task $\mathcal{T}$. To define the motion planning task, we need to specify at first the set of goal configurations Q_d, the achievement of which is one of the prerequisites to conclude that the planned path solves the task.

2.3.3 Policy representation

In the previous section, we defined the representation of the action a and C, which together with the state $\boldsymbol{q}$ enables us to define the policy $\pi(\boldsymbol{q}, C))$ that is used to determine subpaths ζ^i. Taking into account that transitions in the considered MDP are deterministic, we can limit ourselves to analyzing only the deterministic policies. Hence, we can define a policy that is meant to solve considered path planning problems as a function defined by

$$\pi(\boldsymbol{q}_i, C) = \zeta^{i+1}, \tag{2.17}$$

which for every considered state $\boldsymbol{q}$ of the robot, and every context C genrates a path segment ζ^{i+1}. We are particularly interested in the functions π, that generate path segments which, in some finite horizon, assuming recursive application of the policy π, lead to the set of goal configurations Q_d. We especially seek policies that can reach the goal in a low number of recursive calls. Moreover, we look for the policies that generate paths that are feasible given the constraints stemming both from the robot kinematics and the geometry of the robot and environment, and at the same time, optimal w.r.t. the considered optimality criterion. Deterministic policies that satisfy the abovementioned properties are fulfilling our definition of the *optimal ideal planning function* presented in Section 1.4, thus we denote them as $\pi^*(\boldsymbol{q}, C)$.

Following the main idea introduced in Section 1.4, instead of searching for the exact and general formulation of the *optimal ideal planning function* $\pi^*(\boldsymbol{q}, C)$, we propose to approximate it for some subset of motion planning problems. To approximate a function, we need to define:

- the signature of a planning function we want to approximate, which we understand as the definition of its inputs and outputs representations,
- the approximator's structure,
- the method to assess the quality of the approximation and update the parameters of the approximator,
- set of problems of our interest on which we want to approximate the behavior of $\pi^*(\boldsymbol{q}, C)$.

In this section, we will address the first two points, while the remaining two are thoroughly discussed in the next sections.

We start defining the signature of the considered planning function from the inputs. In general, our planning policy $\pi(\boldsymbol{q}, C)$ depends on the current state $\boldsymbol{q}$ and context C. While the current state is directly defined by (2.1), context C is more complex and consists of the environment representation $\mathcal{E}$ and task definition $\mathcal{T}$. Following the propsoed problem definition (see Section 2.2), we describe the environment $\mathcal{E}$ using an occupancy grid $E \in \{0,1\}^{128\times128}$, which assigns 1 to the pixels that lie in the free space $\mathfrak{F}$ and 0 otherwise. In turn, the task definition $\mathcal{T}$, introduced in Section 2.3.2.2, consists of the set of goal configurations Q_d and some solution optimality criterion, e.g., path length or accumulated path curvature. While, in general, the optimality criterion can be task-specific, in this thesis, we assume that it is common for all considered motion planning problems. Thus, we do not have to include it in the policy arguments. The opposite is true for the set of goal configurations Q_d, which can be different for different tasks that fit our problem definition, but constant within the planning episode, i.e., a single maneuver. While, in general, these sets may be defined in any way, we decided to restrict our motion planning problems to reach some axis-aligned orthotope[1] centered at some specific configuration $\boldsymbol{q}_d$. Assuming that for the whole considered motion planning problem, the lengths of the orthotope edges remain constant, i.e., we assume the same margins of achieving the desired configuration $\boldsymbol{q}_d$ for all motion planning problem instances. Therefore, to unambiguously define the arguments of the considered motion planning function, we need to specify an occupancy grid E and the desired state of the robot $\boldsymbol{q}_d$. In turn, the output of the considered policy is, in general, defined as a solution path segment ζ^i. However, using the sequential nature of the considered problem and reasoning shown in Section 2.3.2.1, we can greatly simplify the subpath representation and define it unambiguously using just 4 numbers. Therefore, we expect the policy π to return the 2D position (x, y) of a segment endpoint as well as the first $\frac{dy}{dx}|_{(x,y)}$ and the second $\frac{d^2y}{dx^2}|_{(x,y)}$ derivative of the path at that point, all expressed in the LCS defined by the current state $\boldsymbol{q}$. Such a quadruple $(x, y, \frac{dy}{dx}|_{(x,y)}, \frac{d^2y}{dx^2}|_{(x,y)})$ we denote as S_{ζ_i} for i-th segment endpoint parameters. Moreover, because we consider only the monotonic maneuvers, we can put a constraint on the x coordinate of the segment endpoint, i.e., $x > 0$. Summing up the reasoning above, we can clearly define the policy π as a function defined by

$$\pi : Q \times \{0,1\}^{128\times128} \to A, \tag{2.18}$$

where action space $A = \mathbb{R}_+ \times \mathbb{R}^3$.

Having the policy π signature defined, we can think about the structure of the *optimal ideal planning function* approximator. The main idea of the proposed approach to optimal policy approximation is to learn it from data. Therefore, we need to decide which method from the library of machine learning algorithms will best suit the needs of robotic motion planning. Let's note that the argument space defined in (2.18) is huge, consisting of 4 continuous state dimensions times 2^{14} boolean states representing the map. Moreover, we would like to mimic the optimality of π^*. Thus, we presume that the use of non-parametric machine learning methods, such as decision trees or k-Neares Neighbours, would be infeasible, as they would have problems

[1] a generalization of a rectangle for higher dimensions

accurately adjusting their decision boundaries to pursue the optimality and likely cannot generalize to unseen environments [90, 172]. Therefore, we focus our attention on parametric machine learning methods, particularly NNs, as they have shown great approximation and generalization abilities among different domains, including robotics. Moreover, they are proven to be a universal approximator [133] and have a great track record in solving problems that require the processing of image-like structures, which include the maps we are considering. To denote the dependency of the approximation of the policy π on the parameters ϕ of a NN, typically called weights, we write π_ϕ.

To approximate the optimal policy π^* accurately, we propose a NN architecture presented in Figure 2.4. This architecture consists of 3 main functional components: (i) map processor, (ii) configuration processor, and (iii) parameter estimator. The map processor is meant to process the map E representing the robot's environment. We encode map E using one-hot encoding to ease the processing by NN and apply a sequence of 3×3 2D convolution and 2×2 max-pooling layers. In this way, we obtain a $4 \times 4 \times 512$ tensor, which, after being flattened to an 8192-dimensional vector, is then processed by the fully connected layers, finally achieving a size of 256. Due to the assumption that the environment remains unchanged during the planning process, we construct a 256-dimensional embedding of the map E once per episode, thus saving inference time. The second building block of our NN is the configuration processor, which is a sequence of fully connected layers that creates a 256-dimensional embedding of the actual $\boldsymbol{q}$ and desired $\boldsymbol{q}_d$ states. To facilitate the processing of the vehicle orientations in NN, we represent them by their sine and cosine to maintain the topology of the orientation space. Finally, both map and configuration embeddings are concatenated and jointly processed by 4 heads of the parameter estimator, which, by a sequence of 4 fully connected layers, generates the segment endpoint parameters. Each parameter has its own *network head*, and they differ only in terms of the last layer. Heads that estimate $y_{\mathfrak{L}_i}$, $\frac{dy}{dx}|_{(x_{\mathfrak{L}_i}, y_{\mathfrak{L}_i})}$, and $\frac{d^2y}{dx^2}|_{(x_{\mathfrak{L}_i}, y_{\mathfrak{L}_i})}$ do not use activation functions in the last layer, to enable the network to produce arbitrary derivatives and choose displacement in the local y-axis freely. Whereas, for the $x_{\mathfrak{L}_i}$ head, there is a sigmoid at the end, and its output is scaled 10 times to set the maximal length of the single path segment to a reasonable value, relative to the map size, of 10 m and to stabilize gradient propagation. Except for these layers, the rest of the fully connected layers use tanh non-linearity, while convolutional layers utilize Rectified Linear Unit (ReLU) activations, which have been commonly used in fully-connected [113] and convolutional neural networks [120]. Such a network calculates the parameters of a single segment endpoint. Thus, to generate a spline made of n_{seg} polynomial segments, we perform n_{seg} inferences by changing the representation of the actual state $\boldsymbol{q}$ each time, according to the previously determined segment.

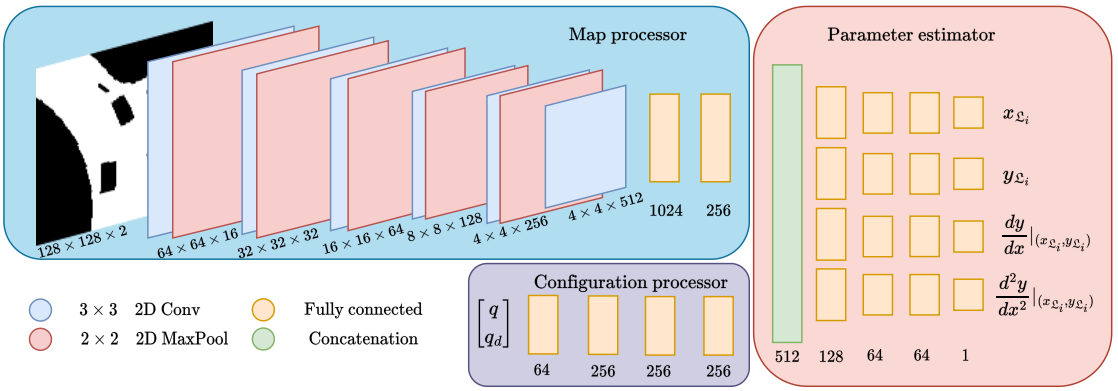

FIGURE 2.4: The proposed architecture of the NN-based optimal policy approximator. The network consists of three branches: (i) map processor, which creates the latent representation of the environment; (ii) configuration processor, which learns representations for the vehicle's actual and desired configurations; and (iii) parameter estimator with 4 heads, which produces parameters of the next segment endpoint using the latent representation of the problem. Numbers under layers represent the shape of the data at its output.

2.3.4 Loss function

To train the proposed neural network, we need to define a loss function, which will drive our model towards some *optimal ideal planning function*. A trivial solution is to use imitation learning to copy the behavior of a complete planning algorithm on multiple path planning tasks [70]. However, this approach has several important drawbacks. First, using a planning algorithm requires a significant amount of time to generate training data. Moreover, the quality of the paths generated by the chosen planner constitutes an upper bound on the performance of the proposed planning policy, which results in paths far from optimal paths or increases the time needed to create the training data. Furthermore, the loss function minimized in supervised learning drives the solutions toward the planned paths without considering their actual feasibility, which may be an issue for models with limited capacity. Such a situation is schematically illustrated in Figure 2.5.

Due to the aforementioned drawbacks of imitation learning, we decided to learn to approximate the optimal policy differently. Instead of directly copying the exact behavior of the *otpimal ideal planning function*, we propose to define what kind of features characterize the optimal policy and to construct mathematically defined functions that implicitly define the behavior of the optimal policy. Having these functions defined, we can evaluate all actions taken by a policy π_ϕ and learn from these evaluations how to plan using the Reinforcement Learning paradigm. In the considered problem, we show that these functions can be defined in such a way that all of them are differentiable and thus allow us to use their gradient directly to optimize the parameters ϕ of the neural network. Thanks to this property, we can use a Gradient-based Policy Search [40] algorithm to train the model to act almost, due to the limited capacity of our model, as an optimal policy π^*, i.e., to generate feasible and near-optimal paths.

These functions that implicitly define the behavior of the optimal policy π^* can be considered in the MDP framework as the components of the reward function R. Thus to properly optimize the policy parameters ϕ we need to define a differentiable reward function R. While the codomain

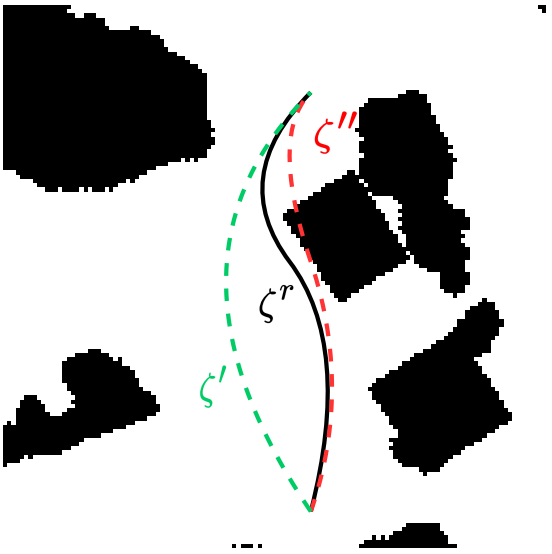

FIGURE 2.5: The path ζ'' has a lower error in terms of imitation learning (lies closer and is more geometrically similar to the reference path ζ^r) than the path ζ'. However, the path ζ'' is infeasible due to the collision with the environment, in contrast to ζ'.

of the reward function is clearly the set of real numbers $\mathbb{R}$, we need to pay more attention to its arguments. To evaluate the reward function R in the considered problem we not only need the current state and action pair $(\boldsymbol{q}, a)$ but also the context C in which this state occurred and action was applied, and a reference path ζ^r to weakly supervise the path generation. Particularly, we need to know the environment geometry E to check for the collisions and the set of the desired states Q_d to evaluate whether the given path leads us to the goal region. Therefore, the reward function R can be redefined by

$$R(\boldsymbol{q}_i, S_{\zeta_{i+1}}, E, Q_d, \zeta^r) : Q \times A \times \mathcal{E} \times \mathbb{P}(Q) \times Z^r \to \mathbb{R}, \tag{2.19}$$

where Z^r is a space of all reference paths ζ^r and $\mathbb{P}(Q)$ is the power set (set of all subsets) of the state space Q. This function allows us to evaluate the quality of the i-th path segment, however, to assess the whole solution path ζ we need to asses the whole rollout from the state $\boldsymbol{q}_0$ using the policy π_ϕ, i.e., a sequence of state action pairs $((\boldsymbol{q}_0, S_{\zeta_1}), (\boldsymbol{q}_1, S_{\zeta_2}), \ldots, (\boldsymbol{q}_{n_{seg}-1}, S_{\zeta_{n_{seg}}}))$. To do so we define a total expected return R_{π_ϕ} by

$$R_{\pi_\phi}(\boldsymbol{q}_0, E, Q_d, \zeta^r) = \sum_{i=1}^{n_{seg}} R(\boldsymbol{q}_{i-1}, S_{\zeta_i}, E, Q_d, \zeta^r), \tag{2.20}$$

where $S_{\zeta_i} = \pi_\phi(\boldsymbol{q}_{i-1}, E)$ and $\boldsymbol{q}_i$ is computed based on $\boldsymbol{q}_{i-1}$ and S_{ζ_i} using (2.15).

Because neural networks are typically trained using the gradient descend method, which requires the gradient of the loss function, from now on we will refer to the loss function

$$\mathcal{L}(\zeta, E, Q_d, \zeta^r) : \mathbb{C}^2 \times \mathcal{E} \times \mathbb{P}(Q) \times Z^r \to \mathbb{R} \tag{2.21}$$

instead to the total expected return function R_{π_ϕ}, having in mind that

$$R_{\pi_\phi}(\boldsymbol{q}_0, E, Q_d, \zeta^r) = -\mathcal{L}(\zeta, E, Q_d, \zeta^r), \tag{2.22}$$

where solution path ζ is obtained by applying the policy π_ϕ to the initial $\boldsymbol{q}_0$ and subsequent states $\boldsymbol{q}_i, \boldsymbol{q}_{i+1}, \ldots, \boldsymbol{q}_{n_{seg}-1}$.

The proposed loss function $\mathcal{L}$ consists of four components:

- Collision loss $\mathcal{L}_{coll}$, to ensure that the planner produces collision-free paths,
- Curvature loss $\mathcal{L}_{curv}$, to ensure that the produced paths are possible to follow by the car with a limited steering angle β,
- Overshoot loss $\mathcal{L}_{over}$, to ensure that the end of the last segment lies within a neighborhood Q_d of the goal configuration $\boldsymbol{q}_d$,
- Total curvature loss $\mathcal{L}_{tcurv}$, to regularize generated paths, such that they have reasonably low total curvature, as it allows for following them with greater velocity,

All listed above losses are summed together to obtain the main loss

$$\mathcal{L} = \mathcal{L}_{coll} + \mathcal{L}_{curv} + \mathcal{L}_{over} + \rho(\mathcal{L}_{coll}, \mathcal{L}_{curv}, \mathcal{L}_{over})\mathcal{L}_{tcurv}, \tag{2.23}$$

where $\rho(\mathcal{L}_{coll}, \mathcal{L}_{curv}, \mathcal{L}_{over})$ is the feasibility indicator function equal to 1 if the path is feasible, that is $\mathcal{L}_{coll} + \mathcal{L}_{curv} + \mathcal{L}_{over} = 0$, and 0 otherwise. Note, that we changed the way the expected return is computed compared to that shown in (2.22). Instead of computing the loss for each segment and then summing these up, we propose to define several functional losses that refer to the different properties of the optimal policy π^* we want to approximate. By doing so, we can assess the whole solution path ζ at once w.r.t. to the given criterion, thus we move the summation over the segments inside each of the abovementioned loss function components.

As all the cost functions outlined above are rather not standard loss functions encountered in training machine learning models, we need to introduce their definitions. For collision, curvature, and total curvature losses, we consider a path ζ as a sequence of n_{seg} segments ζ^i, and we divide each of these segments with 128 points with orientation, equally distant from each other along the x axis of the local coordinate system in which given segment is defined. By doing so we achieve the resolution of configurations along the local x axis greater than 10 cm (as the maximum length of the path segment along the x axis is set to 10 m).

The key idea behind the collision loss is to encourage the neural network to generate paths outside the collision regions in a differentiable way. To do so, we need a reference path, which will serve as a goal of the optimization for the parts of the segments of the produced path that lie inside obstacles, and a way to efficiently detect that the collision has happened. Thus, the loss function consists of two main components: (i) a collision indicator function $\sigma(\Pi_{ij}, \mathfrak{F}_E^C)$ and

(ii) the distance between reference path ζ^r and characteristic points, and it is defined by

$$\mathcal{L}_{coll}(\boldsymbol{q}_0, \zeta, E, \zeta^r) = \sum_{i=1}^{n_{seg}} \sum_{j=1}^{127} \sum_{k=1}^{5} \sigma(\Pi_{ij}, \mathfrak{F}_E^C) d(\zeta^r, \Pi_{ijk}) l_{ij}, \tag{2.24}$$

where $\mathfrak{F}_E^C$ is the collision space represented by the environment map E, Π_{ijk} denotes the five characteristic points on the vehicle body [183], i.e., four corners of the rectangular body of the vehicle and the guiding point in the middle of the rear axle, for the car configuration at j-th point on the i-th segment. Moreover, $d(\mathcal{X}, \mathcal{Y})$ is a function used to compute the smallest Euclidean distance between elements of the sets $\mathcal{X}$ and $\mathcal{Y}$, and l_{ij} is the Euclidean distance between the $(j-1)$-th and the j-th point in the i-th segment. However, those distances are taken into account only if the vehicle periphery Π_{ij} is in collision with the environment, which was denoted in (2.24) by $\sigma(\Pi_{ij}, \mathfrak{F}_E^C)$. Collision $\sigma(\Pi_{ij}, \mathfrak{F}_E^C)$ of the car body Π at j-th point on the i-th segment with the obstacles $\mathfrak{F}_E^C$ is determined by checking if any point on the circumference of the vehicle lie inside the complement of the free space $\mathfrak{F}_E^C$ represented with 0 values on the map E. In our experiments, we computed points from the circumference such that they lie no further than 0.2 m from each other to ensure that we are checking collisions at the same resolution as the resolution of the grid map E.

Similar to collision loss, curvature loss $\mathcal{L}_{curv}$ and total curvature loss $\mathcal{L}_{tcurv}$ are also calculated using the points defined on all segments. Curvature loss is defined by

$$\mathcal{L}_{curv}(\boldsymbol{q}_0, \zeta) = \sum_{i=1}^{n_{seg}} \sum_{j=1}^{127} \max(|\kappa_{ij}| - \kappa_{max}, 0) l_{ij}, \tag{2.25}$$

while total curvature loss by

$$\mathcal{L}_{tcurv}(\boldsymbol{q}_0, \zeta) = \eta \sum_{i=1}^{n_{seg}} \sum_{j=1}^{127} |\kappa_{i(j+1)} - \kappa_{ij}|, \tag{2.26}$$

where $|\cdot|$ denotes the absolute value, κ_{ij} is a curvature of the path at the j-th point in the i-th segment, κ_{max} is maximal admissible path curvature, and η is a regularization strength parameter, which we set to 10^{-4}.

In turn, the overshoot loss $\mathcal{L}_{over}$ is defined as a Manhattan Distance [32] between the final configuration of the planned path $\begin{bmatrix} x_f & y_f & \theta_f & \beta_f \end{bmatrix}^T$ and the set of the desired configurations Q_d, what can be formally written as

$$\mathcal{L}_{over}(\boldsymbol{q}_0, \zeta, Q_d) = \min_{\boldsymbol{q} \in Q_d} \left(|x_f - \boldsymbol{q}_1| + |y_f - \boldsymbol{q}_2| + |\theta_f - \boldsymbol{q}_3| \right), \tag{2.27}$$

where q_i is the i-th element of $\boldsymbol{q} \in Q_d$, for $i \in \{1, 2, 3, 4\}$. However, thanks to the imposed geometry of the set of the desired configurations Q_d (see Section 2.3.3), we can simplify the computation of the overshoot loss and define it by

$$\mathcal{L}_{over}(\boldsymbol{q}_0, \zeta, \boldsymbol{q}_d) = \text{ReLU}(|x_f - x_d| - \frac{Q_{dX}}{2}) + \text{ReLU}(|y_f - y_d| - \frac{Q_{dY}}{2}) + \text{ReLU}(|\theta_f - \theta_d| - \frac{Q_{d\theta}}{2}), \tag{2.28}$$

where x_d, y_d, θ_d are the respective elements of the desired state $\boldsymbol{q}_d$, while $Q_{dX}, Q_{dY}, Q_{d\theta}$ are the lengths of the edges of the orthotope of desired configurations $\boldsymbol{q}_d$ centered at $\boldsymbol{q}_d$, and $\mathrm{ReLU}(\cdot) = \max(\cdot, 0)$.

All parts of the loss function $\mathcal{L}$ are differentiable almost everywhere, which enables us to train it directly using the gradient of the proposed loss function in a Reinforcement Learning paradigm. By almost everywhere, we mean that, due to the use of absolute value, the loss function is not differentiable only at 0, however, it is perfectly fine for learning purposes as for the zero loss we don't have to make any updates. Among loss components, there is only one, which cannot be calculated using the features of the generated path, and thus cannot be trained using Reinforcement Learning methods exclusively – collision loss $\mathcal{L}_{coll}$. To evaluate (2.24), we need a reference path ζ^r, which we obtain from some other planner. However, since the supervision is used only to escape from the forbidden areas of the state space and it is not a performance upper-bound, we call it *weak supervision*. Thus, we called the proposed training procedure *Weakly Supervised Gradient-based Policy Search.*

2.3.5 Dataset

To train our model in a weakly-supervised manner, we need a dataset of local planning problems, of the type described in Section 2.2, with sample solutions. In this work, we make several additional assumptions to further limit the domain of the problems that we want to be able to solve, following the idea of approximating the *optimal ideal planning function* only on some important but relatively small subset of the motion planning problems. These additional assumptions can be summarized as follows:

1. environment maps have to be aligned with the current position and orientation of the ego vehicle,
2. both initial and desired configurations are expressed in the LCS defined by the initial position and orientation of the ego vehicle,
3. vehicle dimensions (see W, L_B, L_F, L in Figure 2.2) as well as maximal steering angle β_{max} are constant,
4. the dimensions of the orthotope that defines Q_d are constant.

These assumptions are thoroughly explained below. By the 1st and 2nd assumptions, we remove the translational and rotational symmetry of the considered problems. By doing so, we express all problems w.r.t. the initial configuration of the ego vehicle and practically maintain the same applicability of the proposed method, while reducing the space of the problems which our learning system has learned how to solve. In turn, 3rd assumption definitely narrows down the range of the problem which can be solved with the proposed approach. Our idea is that the planning policy is specific for a given vehicle on which we want to apply it. Nevertheless, for smaller vehicle models and bigger maximal steering angles, our proposed solution will still work, however, generated solutions will be more conservative. Moreover, we expect that for some deviations in the vehicle

model parameters, for which the feasible path does not change the homotopy class, the transfer of the knowledge from the motion planning neural network pre-trained on different vehicle model parameters to different ones is possible. Finally, the 4th assumption simplifies the problem of planning towards some set of desired configurations Q_d to planning towards specific desired configuration $\boldsymbol{q}_d$ with allowed deviations in each state dimension that from an orthotope in the configuration space Q. For simplicity, we assume that these allowed deviations are constant no matter the desired configuration or environment shape around it. Thus, we don't need to include it in the training data. While this assumption may seem a bit non-practical, it is very popular in the context of robotic motion planning [160].

Given all the assumptions introduced above, we can consider a dataset of motion planning problems completely defined by the initial $\boldsymbol{q}_0$ and desired $\boldsymbol{q}_d$ robot configuration, and the environment representation E in the form of an occupancy grid, as the rest of the parameters are the same for the entire data set. These problems should be solvable using a complete path planner employing our representation of the path. Therefore, the purpose of using an auxiliary planner in our method is twofold:

- if the auxiliary planner finds a path, then we know that the particular problem, i.e., the combination of task and environment, is solvable, thus it is safe to include it in the training data,
- the path planned by the auxiliary planner can be used as a reference path ζ^r to drive the learned solution outside the collision regions.

To generate a dataset of this form, we started from the environments and harvested maps of some real urban environments. Using the Velodyne HDL-64E LiDAR measurements from the KITTI dataset [54] and data we have acquired in town suburbs with a Sick MRS-6124 LiDAR mounted on a bus, we generated a set of occupation grid maps. In all cases, the LiDAR scans were registered using the LiDAR Odometry and Mapping (LOAM) algorithm [187], and the 2D drivable terrain maps were obtained by processing the registered laser scans with an elevation mapping method that handles properly sparse laser data [10]. Finally, 2D occupancy grids were produced from the elevation maps by thresholding the elevation values and setting the drivable (empty) and non-drivable (occupied) cells. In addition, we used the CARLA simulator to gather, much less noisy maps, with multiple obstacles such as pedestrians and different vehicles. We built some maps of Town05 and Town07 [38] to capture both city and rural areas. All the aforementioned maps have the same grid resolution of 0.2 m/px.

From all these maps, we sampled vehicle-centered local maps (128×128 px). We tied the initial configuration of the robot in the middle column and the 120th row of the map image, oriented upwards. To increase the variability of the maps and to simulate the other actors and obstacles on the roads e.f. pedestrians or vehicles, we augmented maps with up to 15 randomly placed rectangular obstacles of different sizes.

For each local map, we generated multiple plans with random final configurations. The plans have been obtained using a modified SL planner [134] utilizing 73 polynomial path primitives. To speed up the generation process, we guided the SL search using a Dubins distance [41] as

a heuristic. Such a lattice-based approach makes it easy to ensure that the time and memory needed to generate a plan are bounded. Furthermore, one can easily tune the search resolution (it corresponds to the number of primitives) and can guarantee the desired accuracy of reaching the final configuration (here 0.2 m in terms of Dubins distance). Sample problems of the proposed dataset together with generated reference paths are visualized in Figure 2.6.

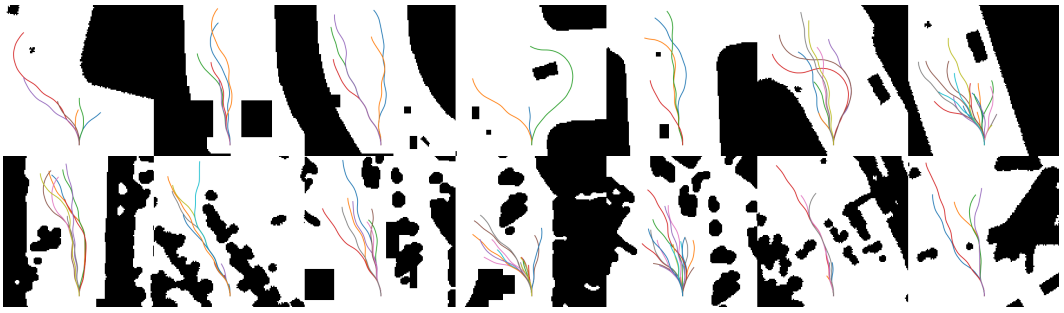

FIGURE 2.6: Visualization of sample elements of the proposed dataset – maps together with exemplary paths planned with the SL algorithm, which connect the actual robot state with some randomly drawn configurations. Scenarios in the first row are obtained from CARLA Town05 and Town07, whereas in the second row, we show maps that are obtained from real LiDAR data.

For training and evaluation purposes we generated training, validation, and test sets. Firstly, the test set was generated entirely from a single map of our own Sick LiDAR dataset. It contains 1014 local maps and 8128 scenarios. Secondly, we collected maps using; (i) another map from our dataset, (ii) maps built from the KITTI dataset, and (iii) maps generated in CARLA. From these maps, we created a validation set by randomly selecting 1996 of them, while the rest were used to create a training set. Finally, on these maps, we randomly drew desired configurations and generated solution paths. By doing so, we obtained 11008 and 115319 scenarios in the validation and training set, respectively. By using data collected from entirely different environments in the training and test sets, we will be able to examine the generalization ability of the proposed method to plan motions in previously unseen environments.

2.3.6 Overall structure of the proposed solution

In the previous sections, we discussed in detail all elements of the proposed solution for the problem of fast motion planning of local monotonic car maneuvers. Having all these elements in mind we can finally draw the big picture of the proposed method. The general scheme of the proposed solution is presented in Figure 2.7. As our method needs to be trained before being used for planning we divided this scheme into two parts. In the upper part of the figure, we can see the behavior of the proposed method in the learning phase. We start from the training set that consists of scenarios defined by the initial and desired robot configurations $(\boldsymbol{q}_0, \boldsymbol{q}_d)$ and map of the environment E, and a sample solution generated with the use of SL motion planning method. Robot desired and initial configurations and a map of the environment are fed into the planning policy π_ϕ represented with a neural network that generates based on this data a solution path ζ. This path, using the full information about the scenario and a reference path, is then assessed by the differentiable loss function $\mathcal{L}$. Based on the loss function $\mathcal{L}$, our proposed gradient-based

policy search method computes the update of the neural network weights ϕ according to the following update rule

$$\phi := \phi + \delta_\phi = \phi - \gamma \frac{\partial \mathcal{L}}{\partial \phi}, \tag{2.29}$$

where δ_ϕ denotes the weights update and γ is the learning rate. After the learning phase is completed, the trained planning policy π_ϕ can be used for planning. Similarly, like in the learning phase, it takes maps E, initial $\boldsymbol{q}_0$, and desired $\boldsymbol{q}_d$ configurations, and plans a path ζ. In the inference phase, no precomputed reference paths are involved and the whole planning process can be realized in less than 50 ms on a medium-class x86 CPU.

Finally, the whole motion planning process can be described procedurally by Algorithm 1. Starting from an initial configuration $\boldsymbol{q}_0$, taking into account environment representation E and desired state $\boldsymbol{q}_d$, the neural network determines the parameters of the segment endpoint S_{ζ_i} (line 3). Based on these parameters, the current vehicle configuration is virtually moved to the end of this segment (line 4), and the new current state is fed to the policy network to determine the next move. After n_{seg} iterations this process ends, and we obtain the whole path ζ that is represented by the sequence of segments ζ^i (line 6).

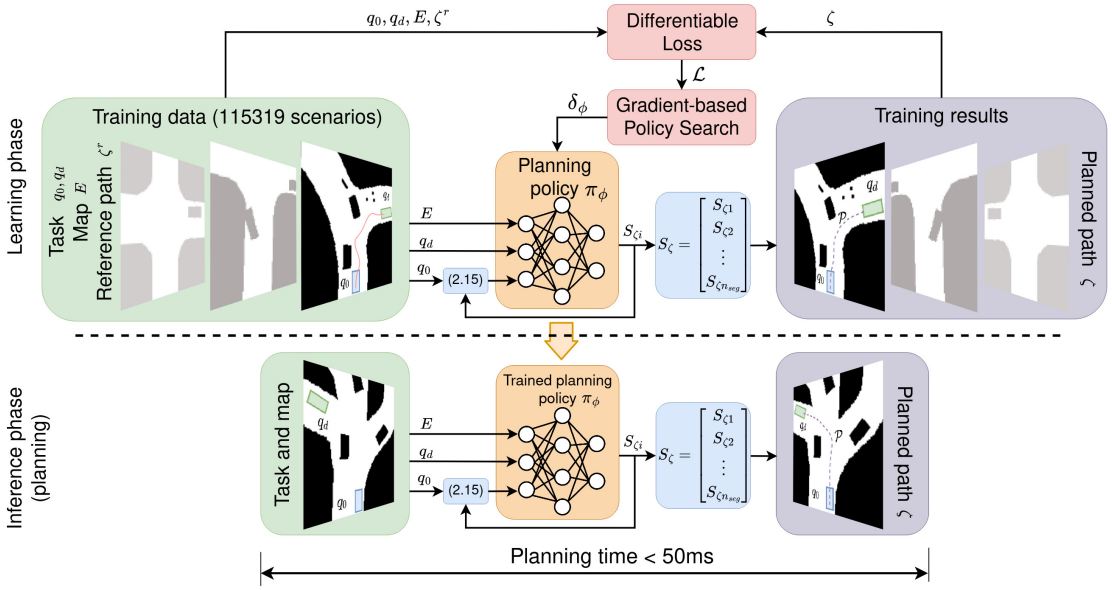

FIGURE 2.7: A conceptual scheme of the proposed approach to rapid path planning. During the inference phase (bottom part of the diagram), task and map representations are processed by a planning policy represented by a neural network to sequentially produce a feasible path that solves the task. In the learning phase (upper part of the diagram), task and map representations, together with reference and planned paths are used to calculate the differentiable loss, and the policy network is improved using the gradient-based policy search algorithm.

Algorithm 1: Our proposed path planning algorithm.

Given: Map E, initial configuration $\boldsymbol{q}_0$, desired configuration $\boldsymbol{q}_d$, planning policy π_ϕ;
for $i \leftarrow 1$ **to** n_{seg} **do**
    Evaluate policy $\pi_\phi(\boldsymbol{q}_{i-1}, E, \boldsymbol{q}_d)$ to obtain parameters S_{ζ_i} of the i-th segment endpoint;
    Perform virtual nominal move along the i-th path segment ζ^i, defined by the current state $\boldsymbol{q}_{i-1}$ and segment parameters S_{ζ_i}, using (2.15), to achieve new vehicle configuration $\boldsymbol{q}_i$;
end
Parameters S_ζ determined by the policy π_ϕ define a path ζ;
Using (2.16) generate a sequence of locally defined polynomial paths of the form defined by (2.14) that can be tracked with VFO controller [53];

Chapter 3

Fast neural network-based planning via efficient B-spline path construction

3.1 Introduction

In this chapter, we will continue our considerations about the path planning problem for a car-like vehicle introduced in Section 2. Therefore, we skip the introduction of the motivations behind solving problems of this type, and, as we will be considering a similar approach to motion planning through learning how to plan, we focus on the possible improvement areas w.r.t. the method introduced in the previous chapter. So, can we do better? The quick response is "for sure", however, let's identify particular areas that are promising in terms of potential amendments.

One of the key aspects of the previous approach was the inspiration on the MDPs and the sequential nature of the planning process so deeply ingrained in motion planning. However, the important question is: "Is it necessary?", especially, because we focus on the local maneuvers. One can argue that in global motion planning, like solving a maze or navigating in a very confined space, even humans tend to sequentially plan some parts of the solution and actively explore the state space searching for the right path. Nevertheless, for local car maneuvers, no driver deliberately plans a sequence of motions, as it takes too much time. The plan for motion that allows for completing the maneuver is somehow intuitively inferred on-the-fly. Inspired by this observation and the pursuit of increasing the planner's reactiveness, we adapt the proposed motion planning policy that was applied sequentially by breaking up with the MDP formalism and fitting the robotic path planning problem into the bandit problem [163] to avoid the sequentiality and to speed-up planning.

Another element of the approach proposed in the previous chapter that we want to focus on is the representation of the path, which also relates to breaking up with sequentiality. Already

introduced path representation seems to be a bit more complicated than the typical path representations and has sequentiality built-in, as each of the segments needs the previous ones to be properly cast to the global coordinate system (see Section 2.3.2.1). While in general, the idea of gradually building the solution from the current robot configuration to the desired one seems natural, for local motion planning we could benefit from a representation that will allow us to determine the solution at once. Moreover, if the desired configuration is given by the definition of the task, which fits our problem definition (see 2.2), it is hard for a sequential path-building approach to reach this state exactly. Obviously, one can imagine that after inferring the last segment the connection to the desired state can be made automatically. However, it may be cumbersome to do so for robots with nonholonomic constraints, which are our particular objects of interest.

A topic strictly related to the solution representation is how to interpret the outputs of the neural network in order to build a solution out of them. Choosing a good way of constructing the path from the neural network outputs may be beneficial in terms of making it easier for a neural network to learn how to plan by structuring the optimization landscape and imposing an inductive bias on the solutions that pose some features that are important to solve the task efficiently.

To address these potential fields of improvement, in this chapter, we propose a novel path parametrization and procedure for its construction. The new parametrization, even though it follows the idea of the approximation of an implicitly defined oracle planning function, breaks up with the Markov Decision Process formalism used in the previous chapter, instead, it follows a bandit problem structure and plans the whole maneuver at once. The path is no longer represented with a sequence of polynomials but with a single 7th-degree 2D B-spline curve. The new representation allows imposing boundary conditions of the solution path. By doing so, we can guarantee that the planned path reaches the goal configuration accurately, unlike the previous approach that usually produced small offsets. The proposed path construction procedure enables our method to compute the solution path within a single inference of a neural network. Moreover, this new parametrization and an interpretation of the neural network outputs introduces an inductive bias [64] to the NN and simplifies the loss function, which significantly speeds up the training.

3.2 Proposed solution

3.2.1 General idea

As the problem we consider is identical to the one described in detail in Section 2.2, we can start directly with introducing the new ideas on improvements to learning how to plan for local car maneuvers. In the previous chapter, we investigated the possibility of using Reinforcement Learning to learn how to plan following a MDP formalism. However, we presume that MDP-based formulation may be not the optimal one from the perspective of the motion planning time, which is crucial for problems that require rapid and reactive planning. Therefore, following the

idea of learning how to approximate the *optimal ideal planning function* using Reinforcement Learning, we want to frame the motion planning problem for a car-like vehicle as a bandit problem.

Bandits are the class of simplified RL problems that consider only one-step episodes, thus avoiding much of the complexity of the full reinforcement learning problem. This assumption degenerates the decision-making process to making a single decision, which in the original n-armed bandit game (a gambling machine with n levers), was to choose the best out of n actions, based on the past experience. This, in general, describes a problem similar to motion planning, in which the main idea is also to choose the best plan for a given scenario, and does not require planning it in several steps. The main deviations from the original n-armed bandit problem are that (i) the set of available actions is replaced with an action chosen from the continuous space of available actions, (ii) the introduction of the context of an episode [20], and (iii) the decoupling the phase of exploring the environment, and learning the optimal policy from the policy application in new episodes. In a typical n-armed bandit problem there is a set number of episodes, e.g., 1000, in which the goal is to explore possible actions, find the best one, and apply it over and over again to maximize the reward. In a slightly more complicated bandit, the reward obtained for each action may fluctuate over time, nevertheless, the idea is still to balance the exploration and exploitation in order to maximize the mean reward for a given number of episodes. Instead, in the proposed motion planning bandit, we want to learn how to plan on a set of episodes, taking into account the context, i.e., the environment and task definitions, such that the learned policy can be used to plan for new scenarios.

Let's describe mathematically our proposed adaptation of the n-armed bandit problem – Contextualized Motion Planning Continous Bandit (CMPCB). The CMPCB problem can be defined as $\{A, \mathcal{C}, R\}$ tuple, where A is the space of available actions, i.e., paths ζ possible to be planned from the current robot configuration, $\mathcal{C}$ is the space of the contexts, which describe the given motion planning task and environment, and finally $R(\zeta, C)$ is a reward function that is meant to assess the quality of the generated plan $\zeta \in A$ in a given context $C \in \mathcal{C}$. One can notice that the considered bandit problem is significantly simplified w.r.t. the MDP-based RL problem formulation, as its definition does not directly contain the state space Q and transition function T, as we will not explore the state space at all. Instead, the single action ζ determined by the planning policy $\pi_\phi(C)$, based on the context C, will be immediately assessed by the reward function R to update the policy parameters ϕ using the gradient of the reward $\frac{dR}{d\phi}$.

The general idea of the proposed solution to the CMPCB problem is schematically shown in Figure 3.1. The idea of this approach is similar to the one presented in the previous chapter (see Figure 2.7). We distinguish 2 phases: the learning phase and the inference phase. In the learning phase, planning policy generates, based on the context C represented by environment E and task $(\boldsymbol{q}_0, \boldsymbol{q}_d)$ description, some output ψ that can be interpreted by the proposed B-spline path construction method. Next, obtained path ζ is evaluated using a differentiable loss function $\mathcal{L}$, based on the current task $(\boldsymbol{q}_0, \boldsymbol{q}_d)$ and environment E representations and reference path ζ^r that is used to escape from the collision areas. Finally, the gradient of this loss function w.r.t. network weights ϕ is determined and used to improve the planning behavior. The inference

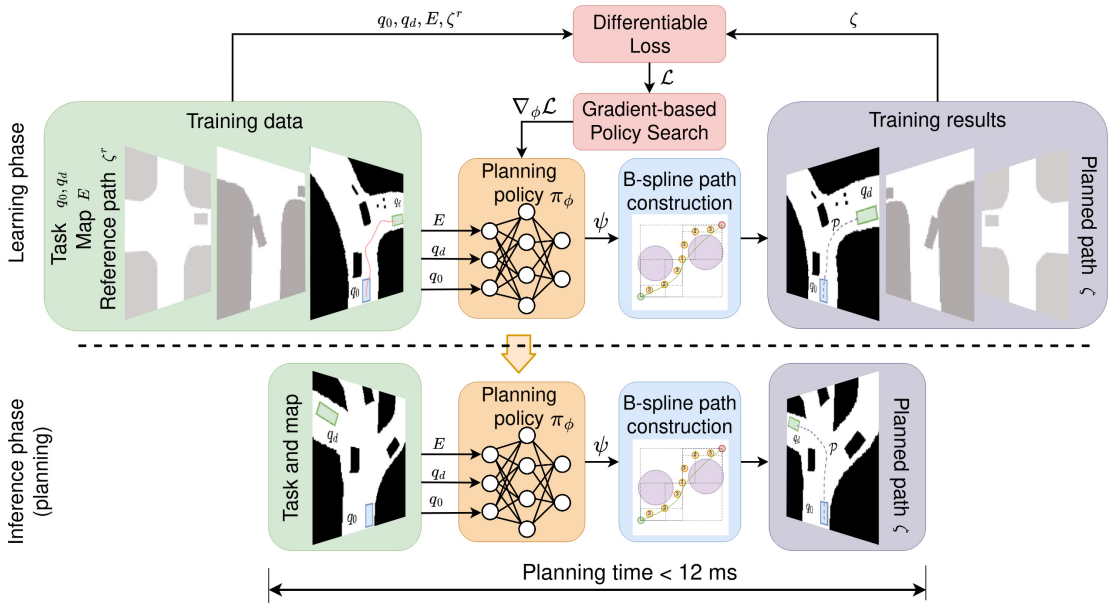

FIGURE 3.1: Conceptual scheme of the rapid B-spline path generation system. The planning policy is represented with a neural network and trained using a gradient-based policy search as in the previous chapter, however, in this case, we are able to generate a plan within a single neural network inference. Neural network outputs are interpreted as B-spline control points by the proposed path construction method.

phase consists of the same steps as the learning phase, except for the path evaluation and policy update. Note, that in this phase there is no reference path.

Although in general planning algorithm proposed in this chapter is similar to the one from the previous one, the policy network is not being queried recursively, but instead, a single forward pass through the planning network is performed. One can see that the proposed approach even better corresponds with the idea of motion planning as a function described in Section 1.4 because in this case, we propose a neural network that directly transforms the representation of the task into the representation of the solution. However, in the proposed approach, the output of the network cannot be directly understood as a solution path, thus we need to add another layer of the interpretation of the neural network outputs. Nevertheless, this approach allows one to generate an entire path using a single inference of a neural network.

As a result, we can describe the planning process as a composition of two functions acting on the task and environment representations

$$\zeta = \mathcal{B}(\pi_\phi(\boldsymbol{q}_0, \boldsymbol{q}_d, E)), \tag{3.1}$$

where $\mathcal{B}$ is a function that represents the proposed B-spline path construction method. Thus, to maintain the differentiability of the whole pipeline we need $\mathcal{B}$ to also be differentiable. Then, the gradient of the loss can be described by

$$\nabla_\phi \mathcal{L} = \frac{\partial \mathcal{L}}{\partial \zeta} \frac{\partial \zeta}{\partial \psi} \frac{\partial \psi}{\partial \phi}. \tag{3.2}$$

In the following sections we will focus on describing main innovations w.r.t. the approach proposed in Chapter 2, i.e., B-spline path representation and the proposed path construction procedure. Moreover, we will highlight the differences in terms of the neural network architecture and loss function.

3.2.2 Path representation

The representation of the solution is one of the crucial decisions one has to make when designing a motion planning method. Typical approaches represent the solution as a polyline, polynomial, or sequence of polynomials. Polylines allow us to easily describe solutions and to connect points in state or task space, however, they result in non-smooth paths, which may be undesirable in terms of the planning of the motion of autonomous cars, or require additional nontrivial smoothing. In contrast, using sequences of polynomials (of order higher than 1) allows us to introduce higher orders of continuity. This enables us to maintain the velocity, accelerations, and higher-order derivatives at the boundaries of segments. Unfortunately, the polynomial-based representation introduced in Chapter 2 does not allow to easily connect arbitrary points in the task space due to the use of the description in local coordinates systems (see Section 2.3.2.1). This approach also has a sequential nature, which we want to avoid in order to improve planning time. Therefore, to enable fast planning of smooth paths that can be forced to connect initial and desired states, we propose to represent paths using B-spline curves.

B-spline curves are parametric curves $p(s)$ described by: (i) a sequence of n_p control points $p_1, p_2, \ldots, p_{n_p}$, (ii) B-spline order o_p defining the degree of polynomials $d_p = o_p - 1$ that constitutes the curve, and (iii) a sequence of $(n_p{+}o_p{+}1)$ nondecreasing knots $\boldsymbol{u} = (u_1, u_2, \ldots, u_{n_p+o_p+1})$ that represents the curve arguments at which the pieces of polynomial meet. The B-spline curve is parametrized with a phase variable $s \in [0; 1]$ and can be defined by

$$p(s) = \sum_{i=1}^{n_p} p_i B_{i,o_p}(s), \tag{3.3}$$

where B_{i,o_p} is a o_p-th order B-spline basis function that can be defined recursively by

$$B_{i,o_p}(s) = \omega_{i,o_p-1}(s) B_{i,o_p-1}(s) + (1 - \omega_{i+1,o_p-1}(s))) B_{i+1,o_p-1}(s), \tag{3.4}$$

where

$$\omega_{i,o_p}(s) = \begin{cases} \frac{s-u_i}{u_{i+o_p}-u_i} & \text{for } u_{i+o_p} \neq u_i \\ 0 & \text{otherwise} \end{cases}, \tag{3.5}$$

and

$$B_{i,1}(s) = \begin{cases} 1 & \text{for } u_i \leq s \leq u_{i+1} \\ 0 & \text{otherwise} \end{cases}. \tag{3.6}$$

Similarly like in Chapter 2, to uniquely define the trajectory to be followed by a car we use the differential flatness property and need to define a trajectory as at least twice differentiable 2-dimensional curve, which defines the path in xy plane, as the rest of the state variables can

be computed based on them. Thus, we define B-spline curve as a function $p(s) : [0; 1] \to \mathbb{R}^2$ that has $x(s) : [0; 1] \to \mathbb{R}$ and $y(s) : [0; 1] \to \mathbb{R}$ components, by setting B-spline control points $p_i = (x_i, y_i) \in \mathbb{R}^2$ to lie on the xy plane. In contrast to the representation defined in (2.15), we no longer need to rely on local coordinate systems and define paths as functions of the form $y = f(x)$. Instead, we can compute the global state of the car-like robot by directly using (2.7), assuming that the solution path ζ from (2.6) is our proposed B-spline curve p. We can do this, as B-spline paths are twice differentiable (assuming $o_p \geq 3$) which is necessary to compute curvature κ of the path. Moreover, thanks to the structure of the B-spline curves computation of their derivatives is very easy. An important property of B-splines is that the derivative of the B-spline basis function can be defined using two B-spline basis functions of the reduced order [35], by

$$\frac{d}{ds}B_{i,o_p}(s) = o_p\left(\frac{B_{i,o_p-1}(s)}{u_{i+o_p} - u_i} - \frac{B_{i+1,o_p-1}(s)}{u_{i+o_p+1} - u_{i+1}}\right), \tag{3.7}$$

and therefore the derivative of the B-spline curve can be also defined as a B-spline curve of the reduced order, i.e.,

$$\frac{d}{ds}p(s) = \sum_{i=1}^{n_p} B_{i,o_p-1}(s)o_p\left(\frac{p_{i+1} - p_i}{u_{i+o_p+1} - u_{i+1}}\right). \tag{3.8}$$

The property we care about a lot is the ability to enforce the boundary conditions on the paths. In the case of the B-splines, this is quite straightforward to implement. As we already mentioned, one of the components needed to define a B-spline is a vector of knots $\boldsymbol{u}$ that determines the ranges of the impact of subsequent control points on the shape of the resultant B-spline curve. To enforce the boundary conditions one can set first and last o_p knots to 0 and 1 respectively. This will eliminate the impact of all of the control points on the curve at its ends, except the first and last ones. Thus if

$$\boldsymbol{u} = (\underbrace{0, \ldots, 0}_{o_p \text{ knots}}, \underbrace{u_{o_p+1}, u_{o_p+2}, \ldots, u_{n_p}, u_{n_p+1}}_{\text{internal knots vector } \boldsymbol{u}_{int}}, \underbrace{1, \ldots, 1}_{o_p \text{ knots}}), \tag{3.9}$$

then

$$p(0) = p_1 \text{ and } p(1) = p_{n_p}. \tag{3.10}$$

Note, that in the case of planning paths for kinematic cars, the positional boundary conditions are not the only ones we need to satisfy. Therefore, we need to address satisfaction of the initial and desired orientation, and, if we would like to avoid turning the steering wheels in place, the initial steering angle (we assume that the desired configuration does not have an imposed desired steering angle). To fix the boundary conditions for higher-order B-spline derivatives we need to define a sequence of internal knots $\boldsymbol{u}_{int}$. To equally distribute the impact of each control point we decided to define them as equidistant to the adjacent knots, thus the knots sequence is defined by

$$\boldsymbol{u} = \left(\underbrace{0, \ldots, 0}_{o_p \text{ knots}}, \underbrace{\frac{1}{n_p - o_p + 2}, \frac{1}{n_p - o_p + 2}, \ldots, \frac{n_p - o_p}{n_p - o_p + 2}, \frac{n_p - o_p + 1}{n_p - o_p + 2}}_{\text{internal knots vector } \boldsymbol{u}_{int}}, \underbrace{1, \ldots, 1}_{o_p \text{ knots}}\right). \tag{3.11}$$

By doing so, we not only enable the computation of the B-spline curve and its derivatives to be done using a single matrix-vector product but also we can represent the first B-spline derivative at the boundaries by

$$\frac{d}{ds}p(0) =(n_p - d_p)d_p(p_2 - p_1), \tag{3.12}$$

$$\frac{d}{ds}p(1) =(n_p - d_p)d_p(p_{n_p} - p_{n_p-1}), \tag{3.13}$$

and second derivative by

$$\frac{d^2}{ds^2}p(0) =(n_p - d_p)^2\frac{d_p(d_p-1)}{2}(2p_1 - 3p_2 + p_3). \tag{3.14}$$

To compute the position of the control points one has to first solve equations (3.12) and (3.13) to determine control point p_2 and p_{n_p-1}, and only then, using the computed value of p_2, solve (3.14). To do so, we need to transform (3.12, 3.13, 3.14) into

$$p_2 =\frac{\frac{d}{ds}p(0)}{(n_p - d_p)d_p} + p_1, \tag{3.15}$$

$$p_{n_p-1} =- \frac{\frac{d}{ds}p(1)}{(n_p - d_p)d_p} + p_{n_p}, \tag{3.16}$$

$$p_3 =\frac{2\frac{d^2}{ds^2}p(0)}{(n_p - d_p)^2 d_p(d_p-1)} - 2p_1 + 3p_2. \tag{3.17}$$

However, note that we do not have straight access to any of $\frac{d}{ds}p(0)$, $\frac{d}{ds}p(1)$, $\frac{d^2}{ds^2}p(0)$, instead our boundary constraints are defined in terms of initial and desired orientation θ_0, θ_d, and initial steering angle β_0. Therefore, to define $\frac{d}{ds}p(0)$, $\frac{d}{ds}p(1)$, $\frac{d^2}{ds^2}p(0)$ in terms of the $\theta_0, \theta_d, \beta_0$ we need to fix the value of derivatives of x or y w.r.t. phase variable s. In our considerations, we assume that

$$\frac{dx}{ds}(0) =x'_0, \tag{3.18}$$

$$\frac{dx}{ds}(1) =x'_d, \tag{3.19}$$

$$\frac{d^2x}{ds^2}(0) =x''_0, \tag{3.20}$$

where x'_0, x'_d and x''_0 are some predefined positive constants. While in general any positive constant satisfies the formal requirements, one should have in mind that choosing too big values of x'_0, x'_d and x''_0 w.r.t. the distance between the initial and end configuration would lead to putting the boundary control points very far away from each other, and thus will reduce the flexibility fo the B-spline representation. Therefore, in our experiments we typically set $x'_0 = \Delta x_2(n_p - d_p)d_p, x'_d = \Delta x_{n_p-1}(n_p - d_p)d_p$ and $x''_0 = \Delta x_3 \frac{1}{2}(n_p - d_p)^2 d_p(d_p - 1)$, where Δx_i is a small constant, e.g., 0.01. By doing so we squeeze the initial control points, as their distance in the x axis will be equal to Δx_i, and thus allow for more sharp maneuvers from the beginning of the motion at the price of possibly higher values of higher-order derivatives.

The last quantities we need to compute are $\frac{dy}{ds}(0), \frac{dy}{ds}(0)$ and $\frac{d^2y}{ds^2}(0)$. To do so, we use the following formulas

$$\frac{dy}{ds}(0) = x'_0 \tan\theta_0, \tag{3.21}$$

$$\frac{dy}{ds}(1) = x'_d \tan\theta_d, \tag{3.22}$$

$$\frac{d^2y}{ds^2}(0) = \frac{1}{x'_0}\left(\frac{\tan\beta_0}{L}\left((x'_0)^2 + \left(\frac{dy}{ds}(0)\right)^2\right)^{\frac{3}{2}} + x''_0\frac{dy}{ds}(0)\right). \tag{3.23}$$

Finally, we can compute boundary control points p_2, p_3 and p_{n_p-1} using (3.15, 3.16, 3.17) with substitutions defined in (3.18-3.23).

3.2.3 Path construction method

In the previous section, we introduced a B-spline path representation. Thanks for fixing the vector of knots, the only remaining parameters that impact the shape of the path are the B-spline control points $(p_1, p_2, \ldots, p_{n_p})$. As proposed in Section 3.2.1, these parameters need to be determined by the neural network. The most straightforward way to achieve this is to just let the neural network planner determine directly the coordinates of $(n_p - 5)$ control points (taking into account that 5 boundary control points can be directly computed using the formulas from Section 3.2.2). It can be also done in a constrained way by restricting the coordinates of the control points to lie in some rectangular area in order to regularize the available solution space and facilitate the training procedure. Unfortunately, direct estimation of the control points has an important drawback. At the very beginning of neural network training, we initialize its parameters ϕ using some random values drawn from the distribution that has an expected value equal to 0. This initialization results in random outputs of the neural network and thus random position of the control points. In turn, the random position of control points may result in a very unexpected and undesirable shape of the path. One of the unwanted effects is a very high path curvature, but even more harmful, from the perspective of further optimization, are self-intersections. The main issue with self-intersections comes from the fact that if we want to optimize the path using gradient-based methods, and we would like to reduce the path length and curvature, then it is extremely hard to untangle the self-intersection, as reducing the size of the loop in the path results in the growth of the curvature. This property may lead to much slower learning or even impede it at all. In Figure 3.2, we present a visualization of a sample path generated by a randomly initialized neural network, whose outputs are directly interpreted as B-spline control points. It is highly unlikely to obtain a path without self-intersections.

We can see that introducing some kind of structure into the process of interpretation of neural network planner predictions is inevitable for efficient learning of how to generate paths possible to follow with a car-like vehicle. To address this observation, we propose a new method of constructing the B-spline path based on the neural network outputs. The main concept of this approach is to implement an ordering of the control points and bias the distributions of the subsequent control points such that a randomly initialized neural network will produce a path without self-intersections. In particular, we propose to insert control points sequentially, based

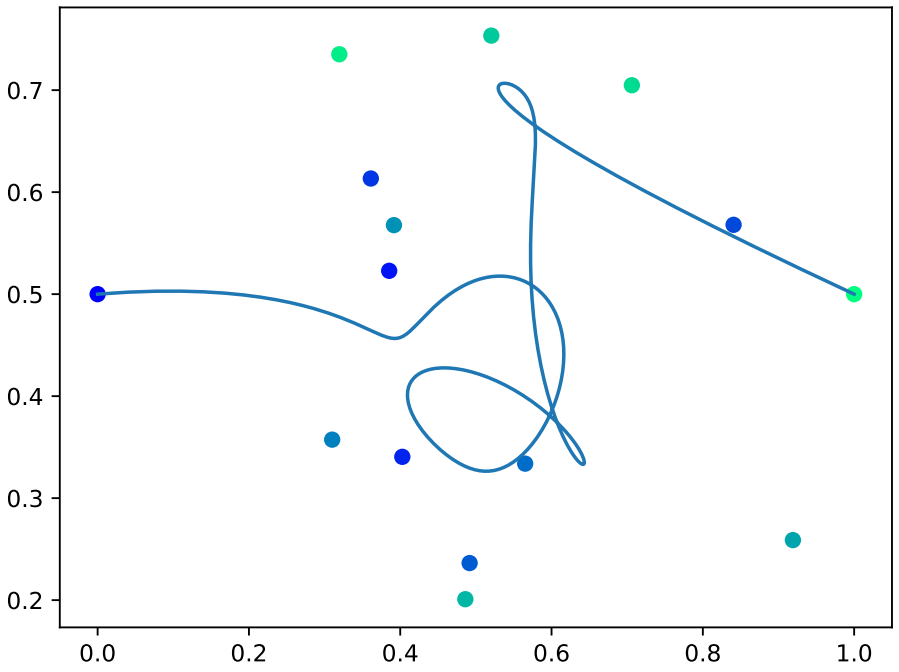

FIGURE 3.2: Sample B-spline path obtained for control points generated by a randomly initialized neural network. The color of the control point represents the ordering (blue is the initial pose and green is the final pose).

on the subsequent predictions of the neural network biased and restricted using a pair of adjacent control points inserted earlier. We start by inserting 5 boundary control points using the method introduced in Section 3.2.2. Next, using the pair of innermost control points (p_3, p_{n_p-1}) and the first two outputs of the neural network $(\psi_1, \psi_2) \in [-1; 1]^2$ we define a new control point $p_{(n_p/2+1)}$ by

$$p_{((n_p-4)/2+3)} = \frac{p_3 + p_{n_p-1}}{2} + \frac{1}{2} d_{3,n_p-1} \cdot (\psi_1, \psi_2), \tag{3.24}$$

where

$$d_{3,n_p-1} = \max\{|x_3 - x_{n_p-1}|, |y_3 - y_{n_p-1}|\}. \tag{3.25}$$

Obtained in this way $p_{(n_p/2+1)}$ constitutes the first level of the tree-like structure that determines how the subsequent control points are computed. In a similar way as for the first level, we can insert another 2 control points on the second level of the tree, 4 control points on the third one, and so on. All of these control points are determined based on the subsequent neural network predictions using (3.24) with different indexes. The general formula for inserting control points can be defined by

$$p_i = \frac{p_{i-a} + p_{i+a}}{2} + \frac{1}{2} d_{i-a,i+a} \cdot (\psi_b, \psi_{b+1}), \tag{3.26}$$

where $b = 2(i-4)+1$, and $a = \log_2(n_p - 4) - l + 1$, where l_T is the level in the tree on which i-th control points is located. To better visualize the dependencies between the control points and indexing scheme, they were illustrated in Figure 3.3. One can see that each control point is defined based on the pair of adjacent control points from higher levels of the tree, which was visualized with the solid black lines. Each level contains the indexes defined by

$$\left\{ \frac{1}{2^l_T}(n_p - 4) + 3, \frac{3}{2^l}(n_p - 4) + 3, \frac{5}{2^l_T}(n_p - 4) + 3, \ldots, \frac{2^l_T - 1}{2^l_T}(n_p - 4) + 3 \right\}. \tag{3.27}$$

Note, that because of the binary structure of the tree, we restrict the number of control points n_p to be equal to $2^l_T + 4$ for $l_T \in \mathbb{N}_+$. We visualize the process of path construction in Figure 3.4.

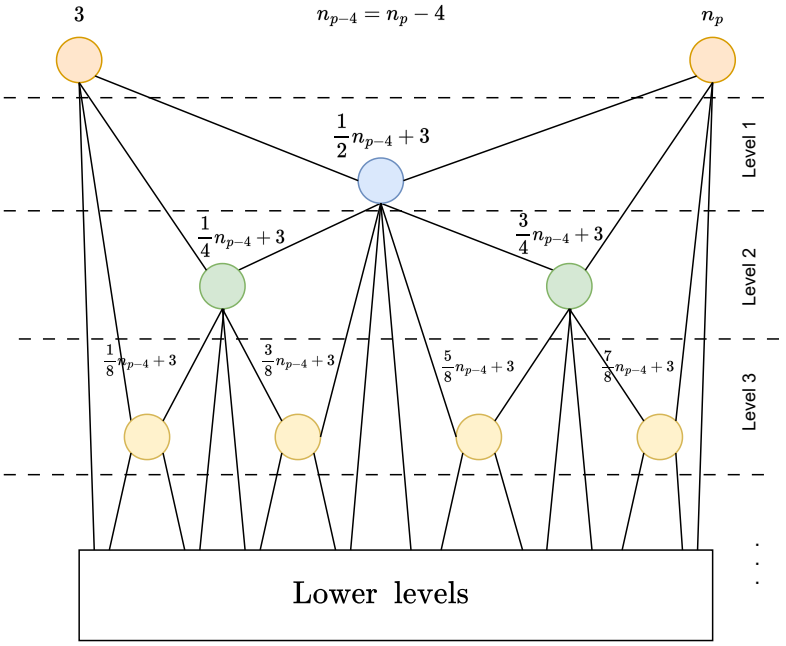

FIGURE 3.3: The diagram of the control points dependencies (denoted with the solid black lines). Starting from the control points defined by the task constraints, we insert internal control points level by level, based on the previously inserted ones.

The proposed way of constructing the path from the neural network outputs is beneficial in terms of the feasibility of the training process and introduces an inductive bias into the inference and learning phase. Due to this representation, the neural network, even randomly initialized, favors paths that have smaller curvature, do not self-intersect, and are ensured to connect initial and goal configurations. Moreover, let's observe that all of the operations performed by the proposed path construction method are differentiable w.r.t. the neural network outputs ψ. Therefore, the introduced path construction procedure fits the framework proposed in (3.2).

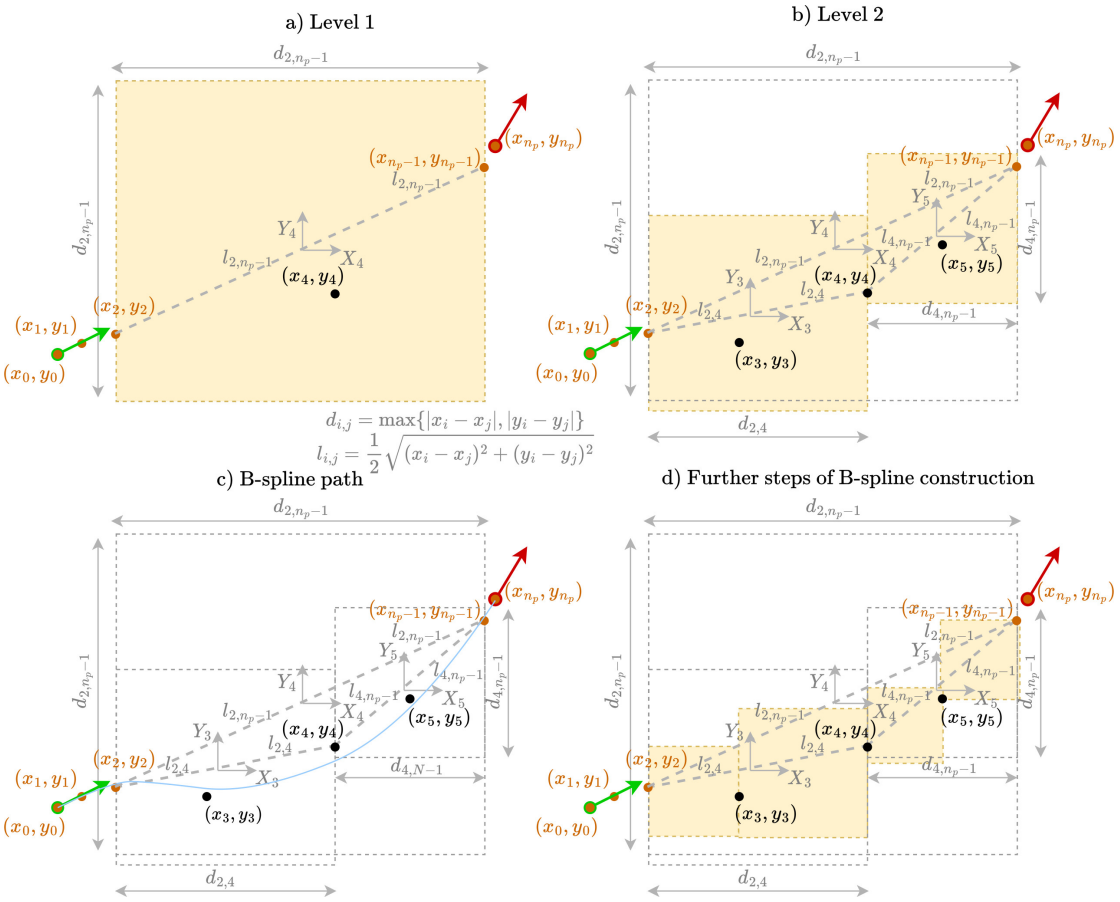

FIGURE 3.4: General idea of defining the B-spline control points. Brown control points stem from the motion planning problem boundary conditions, while black ones are determined by the neural network output and two adjacent control points that define the area (marked in yellow) where the new point can be placed. Subsequent subfigures show (a) the insertion of the control point at the 1st level of the tree, (b) the insertion of the control points at the 2nd level, (c) the generated B-spline path (in blue), and (d) possible further step of B-spline construction, i.e., defining the areas for control points from 3rd level of the tree.

3.2.4 Neural network planner architecture

The significant changes we proposed in this chapter to the solution representation, require us to adjust the architecture of the motion planning neural network-based motion planning policy approximator introduced in Section 2.3.3. For first, we no longer need to repetitively ask the neural network to generate subsequent parts of the solution, but instead, we generate the whole solution at once. Therefore, the input to the *Configuration processor* consists of the initial $\boldsymbol{q}_0$ and desired $\boldsymbol{q}_d$ robot configurations. Moreover, we completely rebuilt the *Parameter estimator*, as thanks to the new path representation and the proposed path construction mechanism, it no longer needs specialized heads. Instead, we have a single head made up of fully connected layers that compute $2(n_p - 5)$ outputs. These outputs are in the range $[-1; 1]$ due to the use of the *tanh* function and are interpreted as x and y coordinates of the control points in the coordinate systems determined by their parents in the tree according to the algorithm introduced in the previous section. The scheme of the proposed neural network architecture is presented in Figure 3.5.

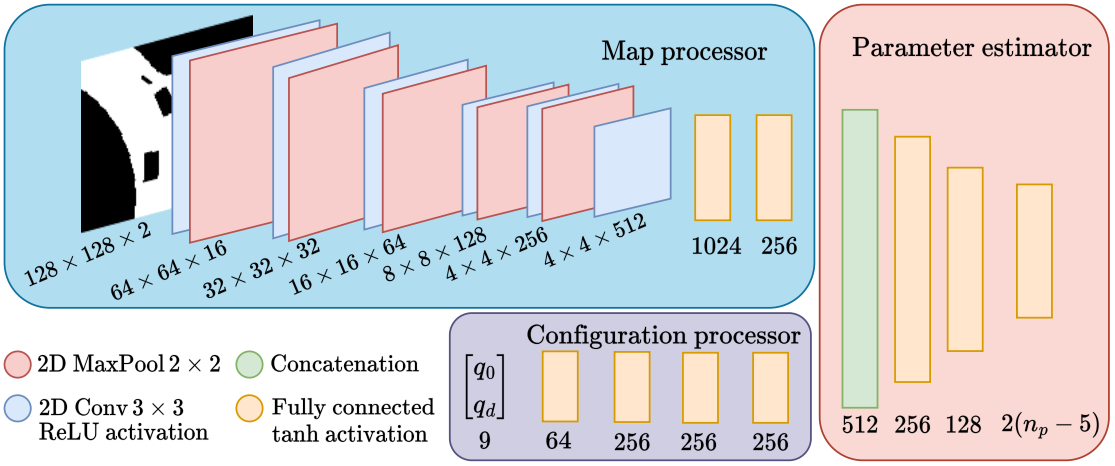

FIGURE 3.5: A general scheme of the proposed neural network-based motion planning policy approximator that generates parameters based on which the B-spline path is constructed. Both *Map* and *Configuration processor* are identical to the ones presented in Figure 2.4, however, we redesigned the *Parameter estimator* to fit the new path representation.

3.2.5 Loss function

The introduced substantial changes in the solution representation, the way of constructing the path, and incorporating the boundary constraints impact the way we compute loss function and its components. The set of features we want to impose on the planned paths is identical to the one introduced in Section 2.3.4, i.e., we want the proposed approach to generate paths that (i) are collision-free, (ii) are of limited and bounded curvature, and (iii) achieve the goal.

Thanks to the proposed path representation and the introduced method of computing the boundary control points of the B-splines, based on the task definition, we ensure that boundary conditions are met. Therefore, the term of the loss function responsible for driving the path towards the goal region is no longer required. Thanks to this, our loss function can be simplified and potentially allows for faster training. Moreover, in contrast to the approach presented in Section 2, we not only have a guarantee of a plan reaching the goal but also reaching the goal state exactly, not the goal region. Nevertheless, the rest of the loss function components are still needed to learn how to plan feasible solutions. Thus, we need to adjust them slightly to match the B-spline path representation.

Firstly, let's observe that having the path represented by a single curve parametrized with a phase variable s, we can easily discretize the path and its derivatives w.r.t. s, which can be easily computed, based on the B-spline control points (see (3.8)), by choosing a sequence of phase variable values, e.g., $\boldsymbol{s} = \{0, \frac{1}{1023}, \frac{2}{1023}, \dots, 1\}$, and evaluating B-spline function and its derivatives at these points. Following this idea we can compute the curvature κ of the path ζ using (2.5) at all discretization points s_i, and therefore compute the discretized curvature values κ_i. Based on them, we define the curvature loss by

$$\mathcal{L}_{curv} = \sum_{i=1}^{1024} \max\left(|\kappa| - \kappa_{max}, 0\right), \tag{3.28}$$

where $\kappa_{max} = \frac{1}{R_{min}}$ is a maximal admissible path curvature that can be followed by the given robot with minimal turning radius R_{min}, and total curvature loss by

$$\mathcal{L}_{tcurv} = \sum_{i=2}^{1024} |\kappa_i - \kappa_{i-1}|. \tag{3.29}$$

Similarly like for curvature, we can use the same discretization to compute the robot configurations along the path and as a result positions of its body. Due to this, we can define the collision loss by

$$\mathcal{L}_{coll}(\boldsymbol{q}_0, \zeta, E, \zeta^r) = \sum_{i=1}^{1023} \sum_{k=1}^{5} \sigma(\Pi_i, \mathfrak{F}_E^C) d(\zeta^r, \Pi_{ik}) l_i, \tag{3.30}$$

where, similarly like for (2.24), $\mathfrak{F}_E^C$ is the collision space represented on the environment map $\mathcal{E}$, Π_{ik} denotes the five characteristic points on the vehicle body [183], i.e., four corners of the rectangular body of the vehicle and the guiding point in the middle of the rear axle, for the car configuration at i-th point on the discretized B-spline path. Moreover, $d(\mathcal{X}, \mathcal{Y})$ is a function used to compute the smallest Euclidean distance between elements of the sets $\mathcal{X}$ and $\mathcal{Y}$, and l_i is the Euclidean distance between the (i)-th and the $(i+1)$-th discretization point. However, those distances are taken into account only if the vehicle periphery Π_i is in collision with the environment, which is denoted in (3.30) by $\sigma(\Pi_i, \mathfrak{F}^C)$. Collision $\sigma(\Pi_i, \mathfrak{F}_E^C)$ of the car body Π at i-th discretization point with the obstacles $\mathfrak{F}_E^C$ is determined by checking if any point on the circumference of the vehicle lie inside the complement of the free space $\mathfrak{F}_E^C$ represented with 0 values on the map E. In our experiments, we computed points from the circumference such that they lie no further than 0.2 m from each other to ensure that we are checking collisions at the same resolution as the resolution of the grid map E.

Note that in the proposed collision loss formulation we are not computing the distance to the free space, which seems to be the most natural way to escape from the forbidden areas. Instead, we propose to use a collision-free reference path as a guidance, and a kind of compact representation of a subset of free space. This compactness and simplicity of the geometric path representation play a key role in gradient-based optimization, as it makes differentiating (3.30) w.r.t. ζ very simple, which is definitely not the case for complex representations like occupancy grids. Nevertheless, we use occupancy grids in the part that does not require differentiability but which defines a mask that denotes the parts of the path that require being pushed out of obstacles.

Having all the loss components defined we can define the general loss function by

$$\mathcal{L} = \mathcal{L}_{curv} + \mathcal{L}_{coll} + \rho(\mathcal{L}_{coll}, \mathcal{L}_{curv}) \gamma \mathcal{L}_{tcurv}, \tag{3.31}$$

where $\rho(\mathcal{L}_{coll}, \mathcal{L}_{curv})$ is an indicator function equal to 1 if the path is feasible, that is $\mathcal{L}_{coll} + \mathcal{L}_{curv} = 0$, and 0 otherwise, and γ is a regularization factor. Note, that (3.31) is very similar to (2.23) but in (3.31) we were able to rule out the dependency on the *overshoot loss* thanks to the B-spline properties. Importantly, all considered losses, similarly to the loss functions defined in Section 2.3.4, are differentiable with respect to the neural network outputs, and, as we shown in Section 3.2.3, our path construction method is also differentiable. Therefore, the gradient of

the proposed loss function can be used to directly optimize the neural network weights using gradient descent algorithms.

Chapter 4

Fast Kinodynamic Planning on the Constraint Manifold with Deep Neural Networks

4.1 Introduction

In previous chapters, we focused on planning the motion for autonomous cars. However, self-driving cars are not the only robots that are in need of reactiveness and rapid motion planning. Similar skills can be essential for other types of robots, such as drones, walking robots, and manipulators. Despite the recent tremendous advancements in robotics we are still far away from human- or animal-level agility. Nowadays we observe that quadrupeds and humanoids are finally capable of not only making steps but also running [29, 42], however, they are still far away from the ability to make sudden turns and react to a rapidly changing environment. In turn, industrial manipulators are typically programmed to perform a given task with great speed and accuracy, however, in most cases, they are limited to performing one repetitive task. Whereas in research applications of multipurpose robotic manipulators, they typically perform very clumsy movements [142], and the full agility potential of the given platform is not exploited [2]. One of the most prominent examples of human agility is sports. However, to beat a human in even a relatively simple sports discipline, like table tennis, billiards, or air hockey, we typically need a specialized robot, while humans are capable of doing all of them using the same multipurpose arms. Even, in our language we often refer to the term *robotic* to the movements that are performed without agility and smoothness, as a sequence of very simple motion primitives. One of the reasons for the above-described shortages of robot agility is the inability to rapidly plan fast and complex robotic movements. The main theme of this chapter is to propose a robot learning framework focused on filling this gap. We pay special attention to the tasks that may be performed with robotic manipulators and require fast and precise movements that have to be computed online due to the dynamics of the task.

Speaking about the dynamic and fast movements, we no longer can plan in the space of geometric paths, as it was done in previous chapters, instead, we are forced to plan trajectories to properly address the above-mentioned agility gap. By introducing the time dependency we have to have in mind not only the robot's kinematics, the geometry of the robot, and the environment but also the innate dynamic limitations of the robotic platform we use, like velocity, acceleration, and torque limits. Moreover, in some cases we would like to impose additional constraints that are related to the task robot is meant to solve, e.g., maintaining a certain orientation of the end-effector, constraining the movement of the end-effector to some subset of the workspace, or satisfying some boundary conditions. Even obstacle avoidance tasks can be described by constraints imposed on the position and orientation of the robot links. Examples of complex, from the robot's point of view, tasks that require a quick computation of a feasible trajectory, i.e., a trajectory solving the task while respecting all kinematic, dynamic, safety and task-related constraints, are ball-in-a-cup [86, 95], table tennis [21, 124], juggling [135], diabolo [170], and air hockey [110, 125].

Although there have been attempts to solve these problems using optimization and sampling-based motion planning algorithms, they all have significant limitations, such as long planning time, computing hard-to-follow plans, or inability to satisfy boundary conditions. One of the approaches to the planning on the constraint manifold is to represent it as a collision-free space, which makes sampling valid states highly improbable [92, 93], resulting in considerably increased planning time. To address this issue, Berenson et al. [12] introduced the concept of projecting the states onto the constraint manifold. However, this approach is restricted to plan paths, which can be hard to follow, as they disregard the system's dynamics. Recent work of Bordalba et al. [18] proposes to solve constrained kinodynamic motion planning problems by building a topological atlas of the constraint manifold and then using LQR to control the system locally. Even though this approach allows for solving complex problems, the motion planning times are prohibitively long for dynamic tasks like the game of air hockey or table tennis. Lengthy planning is also noticeable for optimization-based motion planners, which slow down significantly when subjected to highly non-linear constraints [178] and are prone to get stuck in local minima. An efficient alternative to the above-mentioned methods can be to build a reduced actuator space that keeps the system on states satisfying the constraints [109]. However, this approach may have problems with the satisfaction of boundary constraints.

To address all the above-mentioned issues exhibited by the existing state-of-the-art motion planners and to fill the part of the agility gap between humans and robots, we propose a novel learning-based approach for constrained kinodynamic planning, called Constrained Neural motion Planning with B-splines (CNP-B). Similarly to [12], we frame the problem as planning on a constraint manifold $\mathcal{M}$. Using a similar derivation to [109], we define all the kinematics, dynamics, and safety/task constraints as a single constraint manifold, defined as the zero level set of an arbitrary constraint function $c(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}})$. Our approach does not use an online projection [12] nor continuation [18]. Instead, it exploits the representation power of Deep Neural Networks to learn a motion planning function, indirectly encoding the constraint manifold structure. Implicitly learning the manifold enables us to rapidly plan smooth, dynamically feasible trajectories under arbitrary constraints and highly dynamic motions. Differently from [109], our approach does not require building an abstract action space, making it easy to exploit standard planning

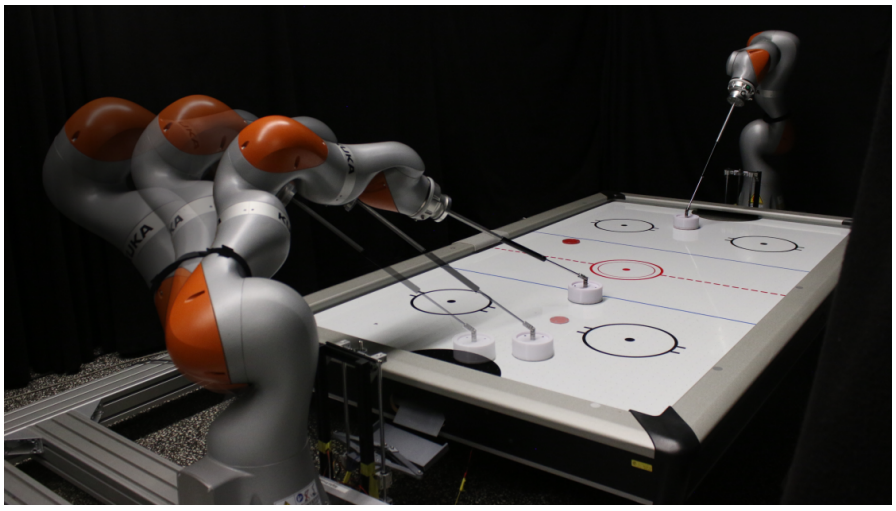

FIGURE 4.1: Proposed learning-based motion planning approach enables rapid planning and replanning of smooth trajectories under complex kinodynamic constraints in dynamic scenarios, e.g., the robotic Air Hockey hitting.

heuristics and improving the interpretability of the plan. In our approach, we frame constraint satisfaction as a manifold learning problem, where the planning function is encouraged to generate trajectories that minimize not only some arbitrary task cost but also the distance from the constraint manifold. Furthermore, violating particular constraints to a certain degree can have only a minor negative impact, while violating other constraints can be unacceptable or dangerous. Hence, we estimate the metric of the constraints manifold, which allows us to attribute different priorities to different constraints.

The proposed approach draws an important inspiration from the previous chapters in terms of learning from experience with a deep neural network and using B-splines for efficient representation of the learned path. In this chapter, we extend these concepts by introducing a method to train a neural network to rapidly plan trajectories, represented using two B-spline curves, that are meant to lie in a close vicinity of the manifold of constraints. To meet this requirement, we extend the learning system architecture by learning the Lagrangian multipliers in the optimization problem, resulting in the learning of the metric of our constraint manifold. This novel approach allows us to weigh each constraint by how much it is important to find a feasible solution to the planning problem. Moreover, thanks to the B-spline representation we show that our method allows us to enforce satisfaction of the boundary constraints, such that the trajectory inferred by the neural network connects precisely two arbitrary robot configurations (positions, velocities, and higher order derivatives). As we will show in simulations and experiments, this new approach is not only much faster than all state-of-the-art motion planning methods we were able to compare as baselines, but also generates trajectories that allow faster and more accurate robot motion while being executed. Furthermore, we show the applicability of the CNP-B in the task of hitting in the game of robotic Air Hockey on a real-world Kuka LBR Iiwa 14 robot (see Figure 4.1). Our proposed general learning for motion planning framework enables the robot to learn how to plan precise dynamic motions outperforming the motion planning algorithm designed specifically for this task [110]. Finally, thanks to the properties of our planner, i.e.,

short deterministic planning time and ability to satisfy boundary constraints, it is able to replan motion on-the-fly and smoothly connect it to the current robot's movement.

4.2 Proposed solution

4.2.1 Problem statement

In previous chapters, we formulated the motion planning problem using Markov Decision Problem and bandit problem formalisms. While the problem we consider in this chapter can be framed as a bandit, due to the lack of sequentiality, we utilize the optimization-based approach, as it is more natural to include the notion of constraints in this case.

Let $\boldsymbol{q}(t) \in \mathbb{R}^n$ be a vector of n generalized coordinates describing a mechanical system at time t. Let $\dot{\boldsymbol{q}}(t)$ and $\ddot{\boldsymbol{q}}(t)$ be the first and second derivative w.r.t. time, i.e., velocity and acceleration of the coordinate vector $\boldsymbol{q}$. Let the tuple $\boldsymbol{\zeta} = \langle \boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}, T \rangle$ be a trajectory of the system for $t \in [0, T]$, with T denoting the duration of the given trajectory. In the following, we assume that $\boldsymbol{\zeta}(t) = \langle \boldsymbol{q}(t), \dot{\boldsymbol{q}}(t), \ddot{\boldsymbol{q}}(t) \rangle$ is a tuple containing the information of the trajectory $\boldsymbol{\zeta}$ at timestep t. We define the constrained planning problem as:

$$\begin{aligned} \underset{\boldsymbol{\zeta}}{\text{argmin}} \quad & \mathcal{L}(\boldsymbol{\zeta}) \\ \text{s.t.} \quad & c_i(\boldsymbol{\zeta}(t), t) = 0 && \forall t, \forall i \in \{1, \dots, N\} \\ & g_j(\boldsymbol{\zeta}(t), t) \leq 0 && \forall t, \forall j \in \{1, \dots, M\}, \end{aligned} \tag{4.1}$$

where N is the number of equality constraints c_i, M is the number of inequality constraints g_j, and $\mathcal{L}$ is an arbitrary loss function describing the task. Typically, to solve a motion planning task, it is enough to generate a trajectory that satisfies some locally defined properties, like limited joint velocity, torque, or ensuring manipulation in free space, at every time step t, and minimize locally defined objectives due to composability. We can exploit this by transforming the objective function of the considered problem into an integral, and describe it by

$$\mathcal{L}(\boldsymbol{\zeta}) = \int_0^T \mathcal{L}_t(\boldsymbol{\zeta}(t), t) dt = \int_0^T \mathcal{L}_t(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t), \ddot{\boldsymbol{q}}(t), t) dt, \tag{4.2}$$

where $\mathcal{L}_t$ is a locally defined loss function.

One of the key design choices in terms of solving problems defined by (4.1) is to specify how the solution trajectory $\boldsymbol{\zeta}$, we are looking for, can be represented. A very popular approach to this, which is very often used in numerical optimization, is to define the components of the trajectory $\boldsymbol{\zeta}$ as polygonal chains and a bunch of new constraints are added to make these components satisfy the differential dependencies between them. This representation, while being straightforward and appropriate for numerical optimization, is not smooth which can pose a problem in case of very fast movements on a thin constraint manifold. One can also decide to define $\boldsymbol{q}(t) \in \mathbb{C}^2$ as at least twice-differentiable function, such that its derivatives w.r.t. time are well defined. However, searching for a solution in the functional space $\mathbb{C}^2$ is not so straightforward and may

be problematic. Therefore, we propose to rely on the parametric representation, such that the generalized coordinates and their derivatives can be represented as at least twice-differentiable function and can be determined based on some set of parameters $\mathfrak{f}$. By doing so we transform the constrained optimization problem (4.1) that seeks optimal trajectory $\boldsymbol{\zeta}$ into the one that looks for the optimal set of parameters $\mathfrak{f}$, i.e.,

$$\begin{aligned} \underset{\mathfrak{f}}{\text{argmin}} \quad & \mathcal{L}(\boldsymbol{\zeta}_{\mathfrak{f}}) \\ \text{s.t.} \quad & c_i(\boldsymbol{\zeta}_{\mathfrak{f}}(t), t) = 0 && \forall t, \forall i \in \{1, \dots, N\} \\ & g_j(\boldsymbol{\zeta}_{\mathfrak{f}}(t), t) \leq 0 && \forall t, \forall j \in \{1, \dots, M\}, \end{aligned} \tag{4.3}$$

where $\boldsymbol{\zeta}_{\mathfrak{f}}$ denotes the trajectory parametrized by the set of parameters $\mathfrak{f}$ in the sense that $\boldsymbol{q}_{\mathfrak{f}}(t)$ is parametrized with $\mathfrak{f}$ and has at least two analytical derivatives w.r.t. time $\dot{\boldsymbol{q}}_{\mathfrak{f}}(t), \ddot{\boldsymbol{q}}_{\mathfrak{f}}(t)$.

4.2.2 Learning how to plan with constraints

The original problem formulation introduced in the previous section (4.3) is a complex constrained optimization problem. The direct application of this formulation makes finding a solution difficult and time-consuming, especially in the presence of complex equality constraints. In fact, while an exact solution may exist, it may be hard to perfectly fit the constraints, due to the use of trajectory parametrization and numerical errors. Moreover, the form presented in (4.3) is not suitable to be used in machine learning-based planning, as it gives no idea how the satisfaction of the constraints should be imposed on the trajectories generated by the machine learning model.

To efficiently solve the considered constrained planning problem and facilitate the use of learning-based solutions, we reformulate it into an unconstrained one by incorporating the constraints into the objective function and scaling them accordingly. We formalize the constraints in the notion of the constraint manifold and allow the solutions to lie close to it, taking into account the acceptable violation budget for each constraint. In our approach, we treat the constraint scaling factors as a metric of the constraint manifold and update it during the learning process.

We base our reformulation on the following steps:

1. defining the constrained manifold as the zero-level curve of an implicit function, unifying equality and inequality constraints,
2. relaxing the original problem by imposing that all solutions should lie in the vicinity of the constraint manifold,
3. transforming this problem into an unconstrained optimization problem by learning the metric of the constraint manifold.

4.2.2.1 Defining the constraint manifold

Our first step towards defining the constraint manifold is to eliminate inequality constraints from (4.3), by introducing slack variables μ_j and M new equality constraints

$$c_{N+j}(\boldsymbol{\zeta}_\mathfrak{f}, t, \mu_j) = g_j(\boldsymbol{\zeta}_\mathfrak{f}, t) + \mu_j^2. \tag{4.4}$$

Thus, we obtain the following formulation of our parametrized constrained optimization problem

$$\begin{aligned} \underset{\mathfrak{f}}{\text{argmin}} \quad & \mathcal{L}(\boldsymbol{\zeta}_\mathfrak{f}) \\ \text{s.t.} \quad & c_i(\boldsymbol{\zeta}_\mathfrak{f}(t), t, [\mu_{i-N}]) = 0 \qquad \forall t, \forall i \in \{1, \dots, N+M\}, \end{aligned} \tag{4.5}$$

where the squared brackets indicate an optional argument. Thanks to dropping inequality constraints we can define a constraint manifold $\mathcal{M}_{\boldsymbol{\mu}}$ by

$$\{(\boldsymbol{\zeta}(t), \mu) \,|\, c_i(\boldsymbol{\zeta}(t), t, [\mu_{i-N}]) = 0, \quad \forall i \in \{1, \dots, N+M\}\}. \tag{4.6}$$

However, in general, we don't want the manifold to be dependent on the slack variables $\boldsymbol{\mu} \in \mathbb{R}^M$. Therefore, to drop this dependency, we first introduce the manifold loss function

$$\mathcal{L}_\mathcal{M}(\boldsymbol{\zeta}(t), t, \boldsymbol{\mu}) = \|\boldsymbol{c}(\boldsymbol{\zeta}(t), t, \boldsymbol{\mu})\|^2 = \sum_{i=1}^{N+M} c_i(\boldsymbol{\zeta}(t), t, [\mu_{i-N}])^2 = \sum_{i=1}^{N+M} \mathcal{L}^i_\mathcal{M}(\boldsymbol{\zeta}(t), t, [\mu_{i-N}]), \tag{4.7}$$

where $\|\cdot\|$ denotes a L2 norm, and $\mathcal{L}^i_\mathcal{M}$ is a manifold loss function associated with i-th constraint. Introduced $\mathcal{L}_\mathcal{M}$ can be interpreted as a distance from the manifold $\mathcal{M}$ defined in a $N+M$-dimensional space, such that the constraint manifold $\mathcal{M}$ can be defined by its 0-level set.

As our goal is to remove the dependency from $\boldsymbol{\mu}$ we focus on the transformed inequality constraints and therefore on the manifold loss function associated with these constraints. Consider i-th manifold loss for $i > N$ defined by

$$\mathcal{L}^i_\mathcal{M}(\boldsymbol{\zeta}_\mathfrak{f}(t), t, \mu_i) = c_i(\boldsymbol{\zeta}_\mathfrak{f}(t), t, \mu_i)^2 = \left(c_i(\boldsymbol{\zeta}_\mathfrak{f}(t), t) + \mu_i^2\right)^2 = c_i(\boldsymbol{\zeta}_\mathfrak{f}(t), t)^2 + 2c_i(\boldsymbol{\zeta}_\mathfrak{f}(t), t)\mu_i^2 + \mu_i^4. \tag{4.8}$$

To eliminate the dependency of μ_i, we need to redefine our loss for inequality constraint as follows

$$\mathcal{L}^i_\mathcal{M}(\boldsymbol{\zeta}_\mathfrak{f}(t), t) = \min_\mu \mathcal{L}^i_\mathcal{M}(\boldsymbol{\zeta}_\mathfrak{f}(t), t, \mu_i). \tag{4.9}$$

We can find a value that minimizes $\mathcal{L}^i_\mathcal{M}(\boldsymbol{\zeta}_\mathfrak{f}(t), \mu_i)$ by taking the derivative of $\mathcal{L}^i_\mathcal{M}(\boldsymbol{\zeta}_\mathfrak{f}(t), \mu_i)$ w.r.t. the slack variable and setting it to zero

$$\frac{d}{d\mu}\mathcal{L}^i_\mathcal{M}(\boldsymbol{\zeta}_\mathfrak{f}(t), t, \mu_i) = 4c_i(\boldsymbol{\zeta}_\mathfrak{f}(t), t)\mu_i + 4\mu_i^3 = 0. \tag{4.10}$$

Thus, we obtain $\mu_i = 0 \vee \mu_i = \pm\sqrt{-c_i(\boldsymbol{\zeta}_\mathfrak{f}(t), t)}$, where the second solution exists only for $c_i(\boldsymbol{\zeta}_\mathfrak{f}(t), t) < 0$. Plugging back the obtained stationary points in (4.8), we obtain a form of

inequality loss that does not depend on μ, and is defined by

$$\mathcal{L}^i_{\mathcal{M}}(\boldsymbol{\zeta}_\mathfrak{f}(t), t) = \begin{cases} c_i(\boldsymbol{\zeta}_\mathfrak{f}(t), t)^2 & c_i(\boldsymbol{\zeta}(ft), t) > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{4.11}$$

This loss can be computed in an equivalent but much more compact form, as

$$\mathcal{L}^i_{\mathcal{M}}(\boldsymbol{\zeta}_\mathfrak{f}(t), t) = \left(\max\left(c_i(\boldsymbol{\zeta}_\mathfrak{f}(t), t), 0\right)\right)^2, \tag{4.12}$$

which is continuous and differentiable everywhere: notice that in 0, the derivative is 0. As a result, we obtain a new definition of the constraint manifold $\mathcal{M}$ that is free from the dependency on $\boldsymbol{\mu}$, and is defined by

$$\mathcal{M} \triangleq \{\boldsymbol{\zeta}(t) \,|\, \mathcal{L}^i_{\mathcal{M}}(\boldsymbol{\zeta}_\mathfrak{f}(t), t) = 0, \quad \forall t, \forall i \in \{1, \dots, N+M\}\}. \tag{4.13}$$

4.2.2.2 Approximated optimization problem

As we mentioned before, our newly defined manifold loss expresses the distance of a given trajectory $\boldsymbol{\zeta}_\mathfrak{f}$ from the manifold $\mathcal{M}$. Ideally, we could use the $\mathcal{L}_{\mathcal{M}}(\boldsymbol{\zeta}_\mathfrak{f}, t) = 0$ as a single constraint of our optimization problem. However, numerical issues, the finite capacity of the used machine learning model, and trajectory parametrization may cause small unavoidable errors. Furthermore, often the task loss $\mathcal{L}$ and manifold constraints $\mathcal{L}^i_{\mathcal{M}}$ will counteract each other, making this optimization particularly difficult and prone to constraint violations imbalances. Indeed, the optimization may favor reducing some constraint violations at the expense of others. When considering real-world tasks, it is often not crucial to have the constraint violation exactly equal to 0, as long as it is below an acceptable threshold. Thus, we simplify the problem by introducing an acceptable level of constraint violations. For simplicity, we bound the elements of the non-negative valued manifold loss function (see (4.7) and (4.12)), i.e., $\mathcal{L}^i_{\mathcal{M}} \leq \bar{C}_i$, where $\bar{C}_i = \bar{c}_i^2$ is a square of the desired acceptable constraint violation level $\bar{c}_i$. Thus, we can write the following optimization problem

$$\begin{aligned} \underset{\mathfrak{f}}{\text{argmin}} \quad & \mathcal{L}(\boldsymbol{\zeta}_\mathfrak{f}) \\ \text{s.t.} \quad & \mathcal{L}^i_{\mathcal{M}}(\boldsymbol{\zeta}_\mathfrak{f}(t), t) \leq \bar{C}_i \qquad \forall t, \forall i \in \{1, \dots, N+M\}, \end{aligned} \tag{4.14}$$

and transform it into the canonical form

$$\begin{aligned} \underset{\mathfrak{f}}{\text{argmin}} \quad & \mathcal{L}(\boldsymbol{\zeta}_\mathfrak{f}) \\ \text{s.t.} \quad & \frac{\mathcal{L}^i_{\mathcal{M}}(\boldsymbol{\zeta}_\mathfrak{f}(t))}{\bar{C}_i} - 1 \leq 0 \qquad \forall t, \forall i \in \{1, \dots, N+M\}. \end{aligned} \tag{4.15}$$

One possible way to solve (4.15) is by applying a Lagrange relaxation technique [14], framing it as an unconstrained optimization of the following function

$$\begin{aligned} \underset{\mathfrak{f}}{\operatorname{argmin}} \max_{\boldsymbol{\lambda}} \quad & L(\boldsymbol{\zeta}_{\mathfrak{f}}, \boldsymbol{\lambda}) \\ \text{s.t.} \quad & \boldsymbol{\lambda} \geq 0, \end{aligned} \tag{4.16}$$

where $L(\boldsymbol{\zeta}_{\mathfrak{f}}(t), \boldsymbol{\lambda})$ is the Lagrangian defined by

$$L(\boldsymbol{\zeta}_{\mathfrak{f}}(t), \boldsymbol{\lambda}) = \mathcal{L}(\boldsymbol{\zeta}_{\mathfrak{f}}) + \sum_{i=1}^{N+M} \lambda_i \left(\frac{\mathcal{L}^i_{\mathcal{M}}(\boldsymbol{\zeta}_{\mathfrak{f}})}{\bar{C}_i} - 1 \right), \tag{4.17}$$

and $\boldsymbol{\lambda}$ is a vector of non-negative Lagrange multipliers.

4.2.2.3 Loss function learning

Unfortunately, (4.16) is not a practical optimization problem, leading to ineffective learning and convergence to local optima. We propose instead an approximate solution, inspired by [156], that updates the Lagrangian multipliers during the minimization of the L. In our approach, we decouple the min-max problem into interleaving minimization of (4.16) w.r.t $\mathfrak{f}$ and updating the values of the multipliers $\boldsymbol{\lambda}$ based on the violation level of the constraints associated with them, i.e., values of $\mathcal{L}_{\mathcal{M}}$.

First, we remove the maximization w.r.t $\boldsymbol{\lambda}$ from (4.16) and write the following minimization problem

$$\underset{\mathfrak{f}}{\operatorname{argmin}} \quad \mathcal{L}(\boldsymbol{\zeta}_{\mathfrak{f}}) + \sum_{i=1}^{N+M} \lambda_i \left(\frac{\mathcal{L}^i_{\mathcal{M}}(\boldsymbol{\zeta}_{\mathfrak{f}})}{\bar{C}_i} - 1 \right). \tag{4.18}$$

Thanks to dropping the maximization w.r.t. $\boldsymbol{\lambda}$, we can remove the -1 term from the Lagrangian, as it is not dependent on $\mathfrak{f}$, such that we obtain

$$\underset{\mathfrak{f}}{\operatorname{argmin}} \quad \mathcal{L}(\boldsymbol{\zeta}_{\mathfrak{f}}) + \sum_{i=1}^{N+M} \lambda_i \frac{\mathcal{L}^i_{\mathcal{M}}(\boldsymbol{\zeta}_{\mathfrak{f}})}{\bar{C}_i}, \tag{4.19}$$

which can be also written down in the form in which the constraints-related terms of the Lagrangian are in diagonal quadratic form, such that we can write

$$\underset{\mathfrak{f}}{\operatorname{argmin}} \quad \mathcal{L}(\boldsymbol{\zeta}_{\mathfrak{f}}) + \mathfrak{c}^T \Lambda \mathfrak{c}, \tag{4.20}$$

where $\mathfrak{c}_i = \sqrt{\mathcal{L}_{\mathcal{M}_i}}$ for $i = 1, 2, \ldots, N+M$ and matrix Λ is a metric of the manifold $\mathcal{M}$ defined by

$$\Lambda = \operatorname{diag}\left(\frac{\lambda_1}{\bar{C}_1}, \frac{\lambda_2}{\bar{C}_2}, \ldots, \frac{\lambda_{N+M}}{\bar{C}_{N+M}} \right). \tag{4.21}$$

By doing so, we framed our extended objective function as a minimization of the sum of task loss $\mathcal{L}(\boldsymbol{\zeta}_{\mathfrak{f}})$ and a squared distance from the manifold scaled by the manifold metric Λ. Therefore, our proposed procedure of interleaving minimization of $\mathcal{L}(\boldsymbol{\zeta}_{\mathfrak{f}}) + \mathfrak{c}^T \Lambda \mathfrak{c}$ and updating $\boldsymbol{\lambda}$ can be seen

as a process of simultaneous learning an appropriate metric of the constraint manifold $\mathcal{M}$ and the parameters $\mathfrak{f}$.

Note, that in the derivations made above we neglected the $\boldsymbol{\lambda} \leq 0$ term from (4.16). Nevertheless, this still needs to be satisfied in order to maintain the negative impact of the constraint violations on the minimization of Lagrangian L. To do so, we simplify the elements of manifold metric Λ, by the following substitution $\mathfrak{m}_i = \log \frac{\lambda_i}{\bar{C}_i}$. Therefore, we can redefine the manifold metric by

$$\Lambda = \operatorname{diag} e^{\mathfrak{m}} = \operatorname{diag} \left(e^{\mathfrak{m}_1}, e^{\mathfrak{m}_2}, \ldots, e^{\mathfrak{m}_{N+M}}\right), \tag{4.22}$$

where $\mathfrak{m} = \begin{bmatrix} \mathfrak{m}_1 & \mathfrak{m}_2 & \ldots & \mathfrak{m}_{N+M-1} \end{bmatrix}$ is a vector of real-valued parameters defining the manifold metric. Now, we can introduce a manifold loss under the metric Λ defined by

$$\mathcal{L}_{\mathcal{M},\Lambda} = \mathfrak{c}^T \Lambda \mathfrak{c}. \tag{4.23}$$

Thus, we can rewrite (4.20) into

$$\underset{\mathfrak{f}}{\operatorname{argmin}}\, \mathcal{L}(\boldsymbol{\zeta}_\mathfrak{f}) + \mathcal{L}_{\mathcal{M},\Lambda}(\boldsymbol{\zeta}_\mathfrak{f}). \tag{4.24}$$

Finally, we introduce the metric learning approach. While we can straightforwardly optimize (4.24) w.r.t. $\mathfrak{f}$ using any gradient optimizer, we need to derive a learning rule for the vector $\mathfrak{m}$. Let $\mathcal{L}^{\neg i}_{\mathcal{M},\Lambda} \triangleq \mathfrak{c}^T_{\neg i} \Lambda_{\neg i} \mathfrak{c}_{\neg i}$ be the complement of the i-th manifold loss element, where index $\neg i$ means all elements except i-th element. Using this notation, we analyze how the $\mathfrak{m}_i$ should change between k-th and $k+1$-th iterations so as not to surpass the desired level of constraint violation $\bar{C}_i$, i.e.,

$$\mathcal{L}^{(k)} + {\mathcal{L}^{\neg i}_{\mathcal{M},\Lambda}}^{(k)} + e^{\bar{\mathfrak{m}}_i^{(k)}} \mathcal{L}^i_\mathcal{M} = \mathcal{L}^{(k+1)} + {\mathcal{L}^{\neg i}_{\mathcal{M},\Lambda}}^{(k+1)} + e^{\mathfrak{m}_i} \bar{C}_i. \tag{4.25}$$

Assuming that the changes of $\mathcal{L}^{\neg i}_{\mathcal{M},\Lambda}$ and $\mathcal{L}$ are small, e.g., by choosing a small learning rate, we obtain

$$e^{\mathfrak{m}_i^{(k)}} \mathcal{L}^i_\mathcal{M} = e^{\bar{\mathfrak{m}}_i} \bar{C}_i \Rightarrow \bar{\mathfrak{m}}_i = \mathfrak{m}_i^{(k)} + \log\left(\frac{\mathcal{L}^i_\mathcal{M}}{\bar{C}_i}\right). \tag{4.26}$$

Finally, we can define our update rule for $\mathfrak{m}_i$ as a small step towards the desired value $\bar{\mathfrak{m}}_i$, i.e.,

$$\Delta \mathfrak{m}_i = \gamma(\bar{\mathfrak{m}}_i - \mathfrak{m}_i) = \gamma \log\left(\frac{\mathcal{L}^i_\mathcal{M}}{\bar{C}_i}\right), \tag{4.27}$$

with the learning rate $\gamma \in \mathbb{R}_+$.

As a result, we transformed a very general form of the optimal trajectory planning problem posed in (4.1) into an alternately solving of an unconstrained optimization problem defined in (4.24) and adaptation of the manifold metric Λ using the update rule defined in (4.27). Moreover, we reformulated the optimization from the general domain of at least twice-differentiable trajectories, into the optimization of the trajectory represented by a set of parameters $\mathfrak{f}$. Nevertheless, our ultimate goal of all of these transformations is to train a machine learning model to plan trajectories. In our considerations, we limit ourselves to parametric models which are

neural networks, parametrized, similarly like in previous chapters using neural network connection weights ϕ. Therefore, we aim not to find the best parameters $\mathfrak{f}$ to minimize the augmented loss $\mathcal{L}(\boldsymbol{\zeta}_\mathfrak{f}) + \mathcal{L}_{\mathcal{M},\Lambda}(\boldsymbol{\zeta}_\mathfrak{f})$ for a particular motion planning problem. Instead, our focus is to find a set of parameters ϕ that allow the neural network π_ϕ to generate, based on the obtained task specifications, sets of function parameters $\mathfrak{f}$ that minimize the augmented loss for a set of motion planning problems drawn from some distribution of considered motion planning problems. Following the typical approach to optimize the neural network weights, we propose to apply stochastic gradient descend methods to optimize $\mathcal{L}(\boldsymbol{\zeta}_\mathfrak{f}) + \mathcal{L}_{\mathcal{M},\Lambda}(\boldsymbol{\zeta}_\mathfrak{f})$ w.r.t. ϕ, i.e.,

$$\nabla_\phi \left(\mathcal{L}(\boldsymbol{\zeta}_\mathfrak{f}) + \mathcal{L}_{\mathcal{M},\Lambda}(\boldsymbol{\zeta}_\mathfrak{f})\right) = \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{\zeta}_\mathfrak{f}} + \frac{\partial \mathcal{L}_{\mathcal{M},\Lambda}}{\partial \boldsymbol{\zeta}_\mathfrak{f}}\right) \frac{\partial \boldsymbol{\zeta}_\mathfrak{f}}{\partial \phi}, \tag{4.28}$$

by assuming that both $\mathcal{L}(\boldsymbol{\zeta}_\mathfrak{f})$ and $\mathcal{L}_{\mathcal{M},\Lambda}(\boldsymbol{\zeta}_\mathfrak{f})$ are differentiable w.r.t. $\boldsymbol{\zeta}_\mathfrak{f}$.

4.2.3 Trajectory parametrization

In previous chapters, we focused on learning how to plan feasible paths. However, to plan dynamic movements we no longer can limit ourselves to paths and have to propose a way to generate trajectories. Therefore, it is a vital question what should be the representation of the trajectory that meets the requirements posed by robotic motion planning? Let's analyze some desirable features we expect a trajectory representation to have:

- smoothness,
- ease of evaluation with minimal computational overhead,
- ease of the computation of the derivatives,
- possibility to impose boundary conditions,
- possibility to represent trajectories of different time lengths.

Although there are many trajectory representations, such as a sequence of timed waypoints, dynamical movement primitives [150], or B-splines [174], we decided to use uniform B-spline curves as they fulfill all the abovementioned requirements. The smoothness is directly controlled by the B-spline degree. In turn, the computational overhead can be minimized easily by assuming the position of knots to be constant. We satisfy this assumption by focusing on uniform B-splines, i.e., B-splines which knots satisfy (3.11). Thanks to this assumption, we can easily pre-compute the values of the B-spline basis functions B for a range of values of the phase variable s and store them in a matrix. In this setting, the computation of the trajectory is a simple matrix-vector product, which can be computed extremely fast even for fine-grained trajectories. Due to B-spline properties, the computation of the derivatives is also a matter of computing matrix-vector products.

While the first 3 points are quite straightforward to satisfy thanks to the B-spline properties, the latter two pose a bit of a challenge. From the considerations made in the previous chapter we know that imposing the boundary conditions on the B-splines is simple, however, now we

need to include the time dependency and satisfy conditions related to velocity and acceleration. A very naive approach to introduce the dependency on the time is to identify the phase variable s with time t, i.e., $s = t$. While simple, this approach does not fulfill the requirement of the possibility of representing trajectories of different time lengths. Nevertheless, this can be fixed easily by introducing some time scaling factor $\mathfrak{r} \in \mathbb{R}_+$, such that $s = \mathfrak{r}t$. Although the use of scaling factor enables planning trajectories of different time-lengths, it is far too constraining. The constant scaling factor directly binds the planned path with the velocity and acceleration with which it is followed, which is highly undesirable as it reduces the representational power of the proposed trajectory parametrization and limits the available spectrum of motions we can represent. The answer to this may be the use of a non-constant monotonically increasing transformation from the phase variable s into the time domain $t = \mathfrak{t}(s)$. We can represent this function by its derivative, i.e., by introducing a variable time-scaling factor $\mathfrak{r}(s) \in \mathbb{R}_+$, which allows for adaptively change the relation between the path and the speed and acceleration along it. In fact, the variable time-scaling factor represents the inverse rate of change of the time t w.r.t. phase variable s, and can be defined by

$$\mathfrak{r}(s) = \left(\frac{dt}{ds}\right)^{-1} = \left(\frac{d\mathfrak{t}(s)}{ds}\right)^{-1}, \tag{4.29}$$

such that

$$\mathfrak{t}(s) = \int \mathfrak{r}^{-1}(s)ds. \tag{4.30}$$

Following this approach, we introduce a trajectory representation that is composed of two B-spline curves: configuration $\boldsymbol{p}(s)$ and time scaling $\mathfrak{r}(s)$. Using these two curves and their derivatives one can define the trajectory $\boldsymbol{\zeta}$ and even higher order derivatives of the configuration vector $\boldsymbol{q}(t)$. Therefore, $\boldsymbol{q}(t)$ can be defined by

$$\boldsymbol{q}(t) \triangleq \boldsymbol{q}\left(\mathfrak{t}(s)\right) = \boldsymbol{q} \circ \mathfrak{t}(s) = \boldsymbol{p}(s), \tag{4.31}$$

where $\circ$ indicates function composition. Whereas, the first derivative of $\boldsymbol{q}$ w.r.t. time can be defined by

$$\dot{\boldsymbol{q}}(t) = \frac{d\boldsymbol{p}(s)}{ds}\frac{ds}{dt} = \frac{d\boldsymbol{p}(s)}{ds}\mathfrak{r}(s) = \dot{\boldsymbol{q}}(s), \tag{4.32}$$

while the second derivative of $\boldsymbol{q}$ w.r.t. time can be defined by

$$\ddot{\boldsymbol{q}}(t) = \frac{d}{dt}\left(\frac{d\boldsymbol{p}(s)}{ds}\right)\mathfrak{r}(s) + \frac{d\boldsymbol{p}(s)}{ds}\frac{d}{dt}\mathfrak{r}(s) = \frac{d^2\boldsymbol{p}(s)}{ds^2}\left(\mathfrak{r}(s)\right)^2 + \frac{d\boldsymbol{p}(s)}{ds}\frac{d\mathfrak{r}(s)}{ds}\mathfrak{r}(s) = \ddot{\boldsymbol{q}}(s). \tag{4.33}$$

This procedure can be continued in order to compute jerk $\dddot{\boldsymbol{q}}(t)$ and higher order derivatives. Note that to compute i-th time derivative of the configuration trajectory $\boldsymbol{q}(t)$, we need i-th derivative of the configuration B-spline $\boldsymbol{p}(s)$ and $(i-1)$-th derivative of the time scaling $\mathfrak{r}(s)$ curve, so the smoothness of the time trajectory depends directly on the smoothness level of B-splines that defines it. Finally, the definition of the trajcetory $\boldsymbol{\zeta}$ contains the trajectory duration T, which can be defined by

$$T = \int_0^1 \mathfrak{r}^{-1}(s)ds. \tag{4.34}$$

4.2.3.1 Boundary conditions

One of the key features of the B-spline representation, from the motion planning point of view, is the ease of imposing boundary conditions. This also applies to the proposed B-spline-based trajectory representation. In fact, we can easily satisfy boundary conditions imposed on the configurations, velocities, accelerations, and higher-order derivatives, which can be defined by

$$\begin{cases} \boldsymbol{q}(0) = \boldsymbol{q}_0, & \dot{\boldsymbol{q}}(0) = \dot{\boldsymbol{q}}_0, & \ddot{\boldsymbol{q}}(0) = \ddot{\boldsymbol{q}}_0, & \dots, \\ \boldsymbol{q}(1) = \boldsymbol{q}_d, & \dot{\boldsymbol{q}}(1) = \dot{\boldsymbol{q}}_d, & \ddot{\boldsymbol{q}}(1) = \ddot{\boldsymbol{q}}_d, & \dots, \end{cases} \tag{4.35}$$

where $\boldsymbol{q}_0, \dot{\boldsymbol{q}}_0, \ddot{\boldsymbol{q}}_0, \boldsymbol{q}_d, \dot{\boldsymbol{q}}_d, \ddot{\boldsymbol{q}}_d$ are the initial and desired configurations and their derivatives given by the task definition. By doing so, we can ensure that the planned motion will start and end at given configurations, but also control for example the end velocity, which may be crucial in the tasks that require hitting some object with the robot in a certain direction, e.g., hitting the ball with the baseball bat. Moreover, by the control of the boundary velocity and acceleration, we can ensure the smoothness of the robot's motion and the continuity of the robot's control signals.

To make our proposed trajectory representation satisfy (4.35), we need to provide formulas that will constrain some of the control points of the configuration and time-scaling B-spline curves. Firstly, we focus on the satisfaction of the boundary configuration constraints. To do so, similarly to what we have done in Section 3.2.2, we have to fix the first and last control points of the configuration B-spline, i.e.,

$$\begin{cases} p_1 = \boldsymbol{q}_0, \\ p_{n_p} = \boldsymbol{q}_d, \end{cases} \tag{4.36}$$

where p_i denotes i-th control point of the configuration B-spline and n_p is the number of configuration B-spline control points. Similarly, we can think about the satisfaction of the velocity constraints, by the definition of the 2nd and $(n_p - 1)$-th configuration control points. However, it also requires the use of the time-scaling B-spline parameters as in the velocity formula (4.32) we see the multiplication by $\mathfrak{r}(s)$. By the analysis of (4.32) at $s = 0$ and $s = 1$ we can see that

$$\begin{cases} \dot{\boldsymbol{q}}(0) = \frac{d\boldsymbol{p}}{ds}(0)\mathfrak{r}(0) = \mathfrak{a}_1^{\boldsymbol{P}}(p_2 - p_1)\mathfrak{r}_1, \\ \dot{\boldsymbol{q}}(T) = \frac{d\boldsymbol{p}}{ds}(1)\mathfrak{r}(1) = \mathfrak{a}_1^{\boldsymbol{P}}(p_{n_p} - p_{n_p-1})\mathfrak{r}_{n_\mathfrak{r}}, \end{cases} \tag{4.37}$$

where $\mathfrak{r}_i$ is an i-th time-scaling B-spline control point, while $n_\mathfrak{r}$ denotes the number of control points of the time-scaling B-spline curve. Therefore, by assuming that we know the 1st and $n_\mathfrak{r}$-th control points of the time-scaling B-spline, we can compute the 2nd and (n_p-1)-th configuration control points using the following formulas

$$\begin{cases} p_2 = \frac{\dot{\boldsymbol{q}}_0}{\mathfrak{a}_1^{\boldsymbol{P}}\mathfrak{r}_1} + p_1, \\ p_{n_p-1} = p_{n_p} - \frac{\dot{\boldsymbol{q}}_d}{\mathfrak{a}_1^{\boldsymbol{P}}\mathfrak{r}_{n_\mathfrak{r}}}. \end{cases} \tag{4.38}$$

By following the same reasoning one can provide formulas for the computation of the next configuration control points. However, for most cases it is enough to provide the formulas up to

the accelerations, thus we give them in the equations below

$$\begin{cases} p_3 = \frac{\aleph - 2\mathfrak{a}_2^{\boldsymbol{p}} p_1 + 3\mathfrak{a}_2^{\boldsymbol{p}} p_2}{\mathfrak{a}_2^{\boldsymbol{p}}}, \quad \text{where} \quad \aleph = \frac{\ddot{\boldsymbol{q}}_0 - \mathfrak{a}_1^{\boldsymbol{p}} \mathfrak{a}_1^{\mathfrak{r}} (p_2 - p_1)(\mathfrak{r}_2 - \mathfrak{r}_1)\mathfrak{r}_1}{\mathfrak{r}_1^2} \\ p_{n_p-2} = \frac{\aleph - 2\mathfrak{a}_2^{\boldsymbol{p}} p_{n_p} + 3\mathfrak{a}_2^{\boldsymbol{p}} p_{n_p-1}}{\mathfrak{a}_2^{\boldsymbol{p}}}, \quad \text{where} \quad \aleph = \frac{\ddot{\boldsymbol{q}}_d - \mathfrak{a}_1^{\boldsymbol{p}} \mathfrak{a}_1^{\mathfrak{r}} (p_{n_p} - p_{n_p-1})(\mathfrak{r}_{n_\mathfrak{r}} - \mathfrak{r}_{n_\mathfrak{r}-1})\mathfrak{r}_{n_\mathfrak{r}}}{\mathfrak{r}_{n_\mathfrak{r}}^2}. \end{cases} \tag{4.39}$$

One can spot, that equations (4.38) and (4.39) rely on the values of $\mathfrak{a}_1^{\boldsymbol{p}}, \mathfrak{a}_2^{\boldsymbol{p}}$ and $\mathfrak{a}_1^{\mathfrak{r}}$. These are some constant quantities, that can be computed based on the parameters of the configuration and time-scaling B-splines. Specifically, given that $D_p, D_\mathfrak{r}$ are the degrees of configuration and time-scaling B-splines respectively, we can compute $\mathfrak{a}_1^{\boldsymbol{p}}, \mathfrak{a}_2^{\boldsymbol{p}}$, and $\mathfrak{a}_1^{\mathfrak{r}}$ by

$$\mathfrak{a}_1^{\boldsymbol{p}} = D_p(n_p - D_p)^2, \tag{4.40}$$

$$\mathfrak{a}_2^{\boldsymbol{p}} = \frac{D_p(D_p - 1)}{2}(n_p - D_p)^2, \tag{4.41}$$

$$\mathfrak{a}_1^{\mathfrak{r}} = D_\mathfrak{r}(n_\mathfrak{r} - D_\mathfrak{r})^2. \tag{4.42}$$

4.2.4 Neural network architecture

Our proposed trajectory parametrization requires us to determine control points of the configuration and time-scaling B-splines. While some of them can be computed analytically based on the definition of the considered task, the majority of them are free parameters that have to be adjusted to satisfy all remaining constraints imposed by the task and optimize some notion of the trajectory quality. In the optimization approach to robotic motion planning these parameters are determined for every motion planning problem instance by some optimization procedure. Instead, in our approach, we propose to optimize the weights of the neural network on a spectrum of motion planning problems, such that it will be able to rapidly generate appropriate B-spline control points for previously unseen tasks by shifting the computational effort from evaluation to the training phase, which can be done offline.

Therefore, we propose a neural network of the architecture presented in Figure 4.2. This architecture is only a proposition, as in general one has a variety of options on how to process the initial and desired states, in order to obtain configuration and time-scale control points. We propose to use a sequence of fully connected layers thanks to its simplicity and short inference time. A minimal input to the proposed network is the initial and desired robot configurations, however, they can be easily augmented with velocities and accelerations. The number of inputs defines the number of boundary conditions n_{bc}, which allows us to compute n_{bc} control points analytically, which is done by the Ω block. In the first layer, we perform a normalization of the inputs, by dividing the configurations $\boldsymbol{q}_0, \boldsymbol{q}_d$ by π, while the velocities $\dot{\boldsymbol{q}}_0, \dot{\boldsymbol{q}}_d$ and accelerations $\ddot{\boldsymbol{q}}_0, \ddot{\boldsymbol{q}}_d$ are divided by the velocity $\bar{\dot{\boldsymbol{q}}}$ and acceleration $\bar{\ddot{\boldsymbol{q}}}$ limits respectively. Then, normalized inputs are processed jointly by a sequence of fully connected layers with ν neurons and *tanh* activation function. Next, the obtained feature vector is split into two heads. Time head consists of a single fully connected layer with $n_\mathfrak{r}$ neurons with exponential activation function to ensure positiveness of outputs, which are scaled by the heuristically chosen expectation of the trajectory duration T_{exp}, and then interpreted as a sequence of time-scaling B-spline control points

$\begin{bmatrix}\mathfrak{r}_1 & \mathfrak{r}_2 & \dots & \mathfrak{r}_{n_\mathfrak{r}}\end{bmatrix}^T$. The idea behind the introduced expectation of the trajectory duration is to bias the time-scaling control points to the estimate of the trajectory duration. This can be done in many different ways, nevertheless, we propose a very simple lower-bound for the real trajectory duration by setting

$$T_{exp} = \max_{i=1,2,\dots,n_{dof}} \frac{|\boldsymbol{q}_{d_i} - \boldsymbol{q}_{0_i}|}{\dot{\boldsymbol{q}}_i}, \tag{4.43}$$

which can be interpreted as the minimal time needed to complete the movement, assuming moving with maximal feasible velocity along the trajectory and infinite acceleration.

A bit more complex than the time-scaling head is the configuration head, which consists of 2 fully connected layers with ν and $(n_p - n_{bc})n_{dof}$ neurons and *tanh* activation. We subtract the number of boundary control points n_{bc} from the total number of control points n_p as they are already defined based on equations (4.36-4.39). One can see, that these equations require only some of the control points of the time-scaling B-spline to be computed, which is already done thanks to the time-scaling head output. However, they are completely independent of the configuration head outputs $\psi^{\boldsymbol{p}} = \begin{bmatrix}\psi_1^{\boldsymbol{p}} & \psi_2^{\boldsymbol{p}} & \dots & \psi_{n_p-n_{bc}}^{\boldsymbol{p}}\end{bmatrix}^T$. In fact, we propose to introduce the opposite dependence, i.e., that the remaining $n_p - n_{bc}$ configuration control points will be computed based on the configuration head outputs $\psi^{\boldsymbol{p}}$ scaled by a factor π, and biased by the linear combination of the inner-most boundary control points. This operation can be defined by

$$p_i = \pi\psi_i^{\boldsymbol{p}} + p_{b_i}, \quad \text{for} \quad i \in \{n_{ibc}+1, n_{ibc}+2, \dots, n_p - n_{ebc} - 2, n_p - n_{ebc} - 1\}, \tag{4.44}$$

where n_{ibc} and n_{ebc} are numbers of boundary constraints imposed on the beginning and end of the trajectory ($n_{bc} = n_{ibc} + n_{ebc}$), while p_{b_i} is the i-th element of the baseline vector, which can be defined by

$$p_{b_i} = \frac{i_{ebc} - (i - n_{ibc})}{i_{ebc}} p_{n_{ibc}} + \frac{i - n_{ibc}}{i_{ebc}} p_{i_{ebc}} \quad \text{for} \quad i \in \{n_{ibc}{+}1, n_{ibc}{+}2, \dots, i_{ebc}{-}2, i_{ebc}{-}1\}, \tag{4.45}$$

where $i_{ebc} = n_p - n_{ebc}$ is the smallest index of the boundary control point that is defined at the end of the trajectory. Thus, to compute the whole vector of the configuration control points we need formulas (4.36-4.45), which are schematically represented in Figure 4.2 as the Ω block.

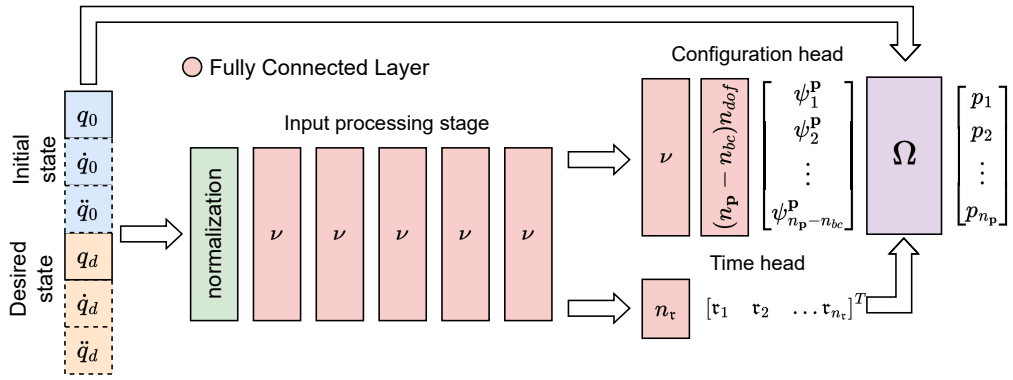

FIGURE 4.2: A neural network architecture that we used to plan trajectories in the experiments presented in this thesis. The dashed line indicates the optional arguments, ν stands for the size of neural network layers, and Ω is a block that performs the computation of the boundary control points of the configuration B-spline, based on the formulas presented in Section 4.2.3.1 and concatenates them with the rest of the configuration control points that are computed based on the last fully connected layer of the configuration head.

4.2.5 Loss functions

In the considered problem we introduced 2 types of losses: (i) task losses, which measure the quality of the trajectory w.r.t. some optimality criteria, and (ii) constraint losses that quantify the violation of the constraints and encourage the trajectories to lie on the constraint manifold. The core idea of the first type of loss we consider is to make neural network models generate trajectories that are close to being optimal. One of the typical values one may want to optimize is the duration of the trajectory, which can be computed easily using (4.34), while another may be the cost of the control, which we can compute using

$$\zeta_{effort} = \sum_{i=1}^{n_{dof}} \int_0^T |\boldsymbol{\tau}_i(t)|\, dt = \sum_{i=1}^{n_{dof}} \int_0^1 |\boldsymbol{\tau}_i(s)|\, \mathfrak{r}^{-1}(s) ds, \tag{4.46}$$

where $\boldsymbol{\tau}_i(s)$ is the torque on the i-th joint, which can be computed based on the trajectory $traj$ using inverse dynamics algorithm, e.g., Recursive Newton-Euler (RNEA) [43]

$$\boldsymbol{\tau}_i(s) = \text{RNEA}(\boldsymbol{q}_i(s), \dot{\boldsymbol{q}}_i(s), \ddot{\boldsymbol{q}}_i(s)), \tag{4.47}$$

assuming no external force, except gravity, is acting on the manipulator.

The second type of losses are the ones stemming from the constraints. The main purpose of them is to penalize trajectories that violate the respective constraint. In typical manipulation tasks, one may encounter the constraints coming from the following sources:

- initial and desired joints position,
- initial and desired joints velocities,
- initial and desired joints accelerations,
- maximal joints velocities,

- maximal joints accelerations,
- maximal joints torques,
- restrictions of the task space, e.g., obstacles, movement of the end-effector restricted to some manifold.

Thanks to the proposed method of boundary constraints satisfaction introduced in the previous section we don't need to make neural network learn how to satisfy them. Therefore, we can omit the first three constraints sources listed above, and focus on the remaining ones. First, violation of the maximal joint's velocities $\bar{\dot{\boldsymbol{q}}}$ can be penalized by

$$\mathcal{L}_{\dot{\boldsymbol{q}}} = \sum_{i=1}^{n_{dof}} \int_0^T \left(\text{ReLU}\left(\dot{\boldsymbol{q}}_i(t) - \bar{\dot{\boldsymbol{q}}}_i\right)\right)^2 dt = \sum_{i=1}^{n_{dof}} \int_0^1 \left(\text{ReLU}\left(\dot{\boldsymbol{q}}_i(s) - \bar{\dot{\boldsymbol{q}}}_i\right)\right)^2 \mathfrak{r}^{-1}(s) ds, \tag{4.48}$$

which is completely in line with the general formula for the i-th component of the manifold loss (see (4.12)). Similarly, we can define loss connected with the maximal joint's accelerations $\bar{\ddot{\boldsymbol{q}}}$ by

$$\mathcal{L}_{\ddot{\boldsymbol{q}}} = \sum_{i=1}^{n_{dof}} \int_0^T \left(\text{ReLU}\left(\ddot{\boldsymbol{q}}_i(t) - \bar{\ddot{\boldsymbol{q}}}_i\right)\right)^2 dt = \sum_{i=1}^{n_{dof}} \int_0^1 \left(\text{ReLU}\left(\ddot{\boldsymbol{q}}_i(s) - \bar{\ddot{\boldsymbol{q}}}_i\right)\right)^2 \mathfrak{r}^{-1}(s) ds, \tag{4.49}$$

and the one related to maximal joint's torques $\bar{\boldsymbol{\tau}}$ by

$$\mathcal{L}_{\boldsymbol{\tau}} = \sum_{i=1}^{n_{dof}} \int_0^T \left(\text{ReLU}\left(\boldsymbol{\tau}_i(t) - \bar{\boldsymbol{\tau}}_i\right)\right)^2 dt = \sum_{i=1}^{n_{dof}} \int_0^1 \left(\text{ReLU}\left(\boldsymbol{\tau}_i(s) - \bar{\boldsymbol{\tau}}_i\right)\right)^2 \mathfrak{r}^{-1}(s) ds, \tag{4.50}$$

where, similarly like in (4.46), $\boldsymbol{\tau}_i(s)$ can be computed using (4.47). In turn, to impose the constraints related to some restrictions of the task space, one has to utilize the forward kinematics of the robot $\text{FK}(\boldsymbol{q}) \in \text{SE}(3)$. While constraints of this type cover a broad range of workspace-related limitations, in this chapter we will give only some sample formulas that are closely related to the constraints we will encounter in the experimental section. In general, one may require the robot's end-effector to remain on some predefined manifold $\mathcal{M}_{\text{FK}}$, i.e., to keep the distance between the robot's end-effector and this manifold close to 0. The loss associated with this requirement can be expressed by

$$\mathcal{L}_{\text{FK}} = \int_0^1 \left(\min_{m \in \mathcal{M}_{\text{FK}}} d_{\text{SE}(3)}\left(\text{FK}(\boldsymbol{q}(s)), m\right)\right)^2 \mathfrak{r}^{-1}(s) ds, \tag{4.51}$$

where $d_{\text{SE}(3)}$ is some distance defined on SE(3). In a similar way, one can describe the collision loss by

$$\mathcal{L}_{\Pi} = \int_0^1 \left(\sum_{b \in \Pi(\boldsymbol{q}(s))} \min_{f \in \mathfrak{F}} d_{euc}\left(b, f\right)\right)^2 \mathfrak{r}^{-1}(s) ds, \tag{4.52}$$

where $d_{euc}\left(b, f\right)$ is an Euclidean distance between an element b of the robot body $\Pi(\boldsymbol{q}(s))$, and an element f of the collision-free space $\mathfrak{F}$.

In (4.28) we stated that the proposed neural network-based motion planner can be trained end-to-end using stochastic gradient descent and backpropagation if the task and constraint losses are differentiable w.r.t. $\boldsymbol{\zeta}_{\mathfrak{f}}$. In fact, due to the differentiability of the functions like forward

kinematics or inverse dynamics, and the perfect knowledge of the used model of the world and robot itself, we can easily compute the gradient w.r.t. $\boldsymbol{\zeta}_\mathbf{f}$ of all loss functions introduced in this section. Moreover, it is relatively straightforward to propose many more loss functions that encourage some desired behaviors and are rooted in the differential geometry of the trajectory generated by the neural network. This approach to learning how to plan may be related to the processes that occur in our brains. Humans often, even subconsciously, generate in their heads some hypothetical problems relevant to the ones they solve in the real world and are trying to solve them using the models of the environment, their bodies, and even other agents. Based on this imagined experience we can improve our future actions without the need to try them in the wild, which is faster than executing actions in the real world but at the same time biased towards the assumed models of the environment and actors. This is the core idea of model-based reinforcement learning, however, in our approach, we focus on the behaviors that can be analytically described by the differentiable loss functions, such that we can directly optimize the parameters of the planner.

Moreover, in (4.2) we introduced a concept of defining the losses locally and using an integral over the duration of the whole trajectory. Note, that all losses introduced in this chapter have the structure shown in (4.2). This may not be obvious at first look, however, the summation over the degrees of freedom can be included in the integral, and for the sake of the implementation we decided to move from the integration in the time domain to the integral of the phase variable s using substitution given in (4.30).

Chapter 5

Experimental verification of the neural network-based path planning for car-like vehicles

5.1 Introduction

In Chapter 2 we introduced a novel approach to learning how to rapidly plan feasible monotonic paths that enable a car-like vehicle to perform local maneuvers. In turn, in Chapter 3 we proposed a new solution to the same class of motion planning problems, which utilizes B-spline path representation to improve the planning speed, as well as the smoothness and accuracy of the generated plans. In this Chapter, we want to verify the abilities of both proposed methods (further referred as *sequential* for the method introduced in Chapter 2 and *episodic* for the one introduced in Chapter 3) to rapidly generate paths that are:

- collision-free when followed with a robot of predefined geometry,
- of limited curvature, such that they can be followed by a car with a limited steering angle,
- short,
- smooth,
- easy-to-be-followed.

Moreover, we want to compare these methods in terms of the above-mentioned criteria and relate them to state-of-the-art path-planning algorithms for car-like vehicles.

To evaluate these abilities and perform comparisons we perform 2 types of experiments:

1. comparison using the motion planning problems from the dataset introduced in Chapter 2.3.5,

2. closed-loop planning and control in challenging scenarios in the CARLA simulator [38].

5.1.1 State-of-the-art path planning algorithms

To obtain a fair perspective on the performance of the proposed motion planning algorithms, we compared them with several state-of-the-art path planning algorithms adjusted to planning for car-like vehicles:

- asymptotically optimal version of RRT – RRT* [83] – a sampling-based motion planning algorithm that builds a tree of robot configurations and modifies it to maintain the optimality of connections,
- BIT* [51] – a sampling-based motion planning algorithm that uses a heuristic to efficiently search a series of increasingly dense implicit random geometric graphs while reusing information from previous batches,
- AIT* [158] – a planner based on BIT* that adapts its search to each problem instance by simultaneously estimating and exploiting a problem-specific heuristic,
- ABIT* [157] – a planner based on BIT* that sacrifices the resolution optimality to achieve faster initial solution times,
- SL [134] – a discrete planner that connects points in the task space using primitives from some predefined set, by doing so it searches for a solution only on a finite lattice of possible moves.

The abovementioned motion planning algorithms propose a general way to efficiently solve motion planning problems. However, the case of monotonic path planning for a car-like vehicle requires making some adjustments and concretizing them.

For SL, we used an implementation of Dynamic Multi-Heuristic A* [75] from SBPL library [106]. SL algorithm was searching on the grid with 0.2m spatial and $\frac{\pi}{16}$rad angular resolution using 11 different motion primitives per orientation, which were made with 3rd-degree polynomials with zero curvature at the boundaries to preserve the continuity of the path curvature.

While the rest of the considered planners differ in terms of the adaptation of the search space and heuristic, at the core of all of them are the procedures of sampling the task space and extending the search tree. The basic versions of these algorithms connect the points in the task space using straight lines, which is infeasible in the case of the car-like robot. To mitigate this issue and not cause a significant drop in their performance we used Dubins curves [41] as an extender and Dubins path length as a cost function, which together constitute a DubinsStateSpace implemented in Open Motion Planning Library (OMPL) [160]. This allows us to rapidly connect any two states in the SE2 state space using paths of limited curvature. The only drawback of this approach is that Dubins curves do not maintain path smoothness and thus the continuity of curvature, which results in the necessity of stopping to turn the steering wheel when changing between straight segments and arcs. To eliminate this issue, one can use cc-Dubins [48] to extend the tree, which connects the segments of the Dubins curves using clothoid.

In the case of the sampling-based algorithms, we used their OMPL [160] implementations and in most of the cases default parameter values for all considered algorithms, as we observed no significant improvement for different sets of the parameters. In particular, for:

- RRT*:
  - maximal motion range – 7.55m,
- BIT*:
  - number of allowed failed samplings = 2,
  - pruning is applied if the number of the samples that can be pruned is bigger than 5% of the number of samples and vertices in the search tree,
- ABIT* – the same as for BIT* and additionally:
  - initial inflation – 10^6,
  - inflation scaling factor – 10,
  - truncation scaling factor – 5,

Besides that, few of the parameters were common, i.e., the probability of choosing the goal state as a goal for tree extension was set to 0.05, the rewiring factor was equal to 1.1, the batch size was set to 100, and all algorithms were using k-nearest search. To interface OMPL with SBPL and cc-Dubins [48] we used Bench-MR library [62].

Our proposed methods also require choosing some parameters. In general, one may search space of neural network architectures, however, we do not aim at finding the very best architecture but rather present the idea and capabilities of proposed neural network-based motion planning. Thus we fixed the architectures of the neural networks to the simple ones presented in Figures 2.4 and 3.5. The most important parameter of our proposed methods is the expressiveness of the path representation understood as the number of its degrees of freedom. In the case of *sequential* planner, this is steered by the number of path segments, which we set in our experiments to 6, which is a balance between the planning time and accuracy. We followed a similar motivation while choosing the depth of the control points tree to 3, which controls the expressiveness of the B-spline paths. The rest of the parameters are related to the learning process of the neural network. In particular, we choose Adam optimizer [91] and set its learning rate to 10^{-4} for *sequential* and $5 \cdot 10^{-4}$ for *episodic* model, while the batch size was set to 128 for both models. We trained all models on NVIDIA GTX1080Ti GPU for 400 epochs after which there were no significant improvements, while for the fairness of the evaluation in the experiments below, we used an Intel Core i7-9750H CPU.

5.2 Exepriments on the dataset

In the first part of the experiments, we focus purely on analyzing the planning performance, without any direct reference to the realization of the planned paths except for the feasibility

validation, which is done purely geometrically. In our experiments, we used a mathematical model of a Kia Rio III [1], whose physical dimensions are the following:

- distance from the rear axle to the rear bumper $L_B = 0.67\,\mathrm{m}$,
- distance from rear axle to front bumper $L_F = 3.375\,\mathrm{m}$,
- distance between axles $L = 2.57\,\mathrm{m}$,
- vehicle width $W = 1.72\,\mathrm{m}$.

Moreover, for the length L and maximal steering angle $\beta_{max} = 0.53\,\mathrm{rad}$, the maximal admissible path curvature is $\kappa_{max} = 0.227\,\frac{1}{\mathrm{m}}$. The subset of the desired states Q_d is defined around a given desired state $\boldsymbol{q}_d$, such that its elements should not lie further than 0.2m from the $\boldsymbol{q}_d$ both in X and Y axes, and the orientation θ should satisfy $0.05\,\mathrm{rad}$.

We evaluate the performance of the considered planners in terms of accuracy, which we define as a ratio of the feasible paths generated by the planner for the test set of path-planning problems (defined in Section 2.3.5) to the cardinality of this set. In our considerations, feasibility is defined by simultaneously zeroing: collision loss $\mathcal{L}_{coll}$, curvature loss $\mathcal{L}_{curv}$, and overshoot loss $\mathcal{L}_{over}$, which ensure that the path is collision-free, possible to follow with a limited steering wheel angle and leads to the close neighborhood of the goal.

5.2.1 Performance

First, we want to qualitatively analyze the planning abilities of the proposed neural network-based motion planners. To do so, we show its performance on several maps from the test set by visualizing the areas that are feasibly reachable with the plans generated by the planner (assuming nominal motion along the path). These visualizations, together with sample planned paths are presented in Figure 5.1. One can see that our proposed planners learned to plan feasible paths that cover wide areas of accessible state space in previously unseen environments. Nevertheless, some areas that seem reachable are not covered by the plans generated by our planners, which illustrates that the proposed planners are not complete. However, they can still perform complex maneuvers in narrow passages that require very precise collision avoidance and judicious use of the available maximum turning angle. We also show that our planners can use both high-quality maps obtained from the CARLA simulator, as well as lower-quality maps, which are obtained from real LiDAR measurements. Speaking about the differences between proposed planners, we can see that they differ in terms of the size and color of the heatmaps. Although sometimes one or another planner does better on individual maps, a general trend can be noticed that the *sequential* planner generates slightly more extensive heatmaps. This may be caused by the fact that the path generated by the *sequential* planner is considered feasible if it reaches a set of configurations around the desired one Q_d (see Sections 2.2 and 2.3.4) but also by the smaller level of its smoothness when compared to the ones generated by the *episodic* planner, which allows for more flexible maneuvering at the price of smoothness of control, and thus driving comfort. This difference in path smoothness is also visible in Figure 5.1, especially thanks to the paths traced by the front corners of the vehicle.

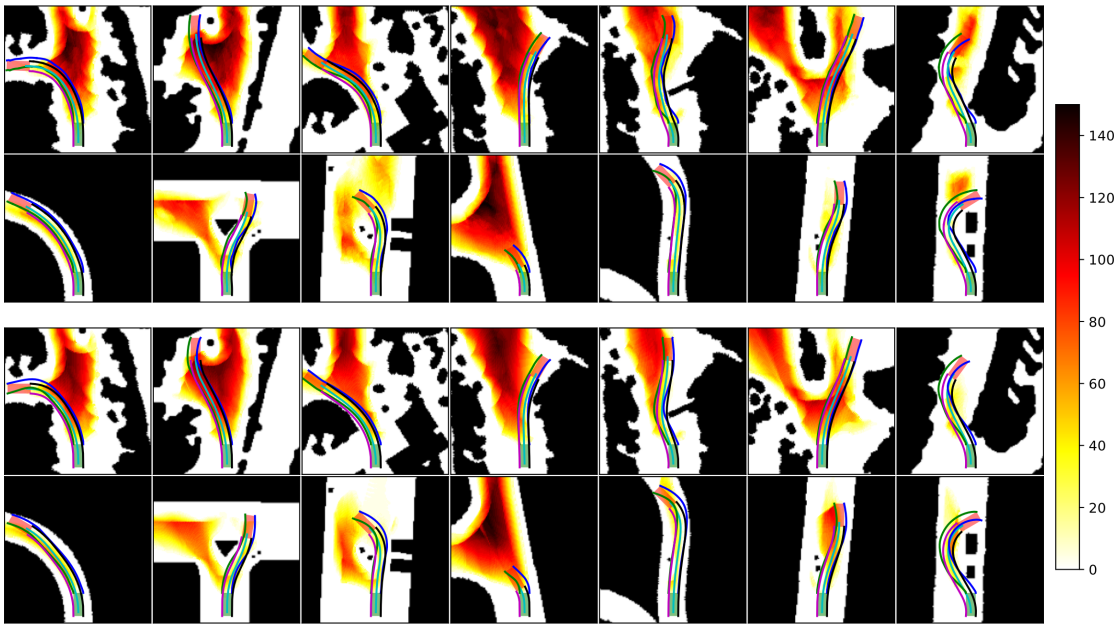

FIGURE 5.1: Visualizations of the sets of the final states (denoted by heatmaps) for which the planner (top rows – *sequential*, bottom rows – *episodic*) generated feasible paths (so-called reachable sets). Darker dots represent wider ranges of possible end orientations of the vehicle, expressed in degrees. Colored lines depict paths drawn by the four corners of the vehicle and the guiding point while moving along the path provided by the network.

Next, we would like to analyze quantitatively the performance of the proposed planners on the test set introduced in Section 2.3.5 and compare it with the state-of-the-art motion planners. In Table 5.1, we compare the proposed methods with the SBMP algorithms in the Dubins state space in terms of accuracy, accumulated vehicle orientation change within an episode, and length of the planned path. The maximal planning time was set to 50ms to match the time scale of the proposed planners, which is also reasonable from the autonomous driving perspective, as rapid planning is necessary for such fast-moving robots. One can see that using appropriate state space and modern SBMP algorithms results in very fast and accurate motion planning. However, when the time budget for the planning is low, as in the considered case, we see that learning-based solutions achieve a better success ratio. Nevertheless, BIT*, AIT*, and ABIT* outperform our methods in terms of the path length and accumulated turn. This is an effect of the use of Dubin's search space, which guarantees the possibility of finding the shortest paths with a minimal amount of turns. However, this comes at a price of curvature discontinuity, which results in the necessity of stopping to change the curvature of motion, and poor comfort for passengers, as the curvature of the motion at the turns is maximal, thus the centrifugal force.

TABLE 5.1: Comparison of the proposed methods with the SBMP algorithms in the Dubins state space. The maximal planning time was set to 50ms. Both accumulated turns and lengths are reported only for paths valid for all planners (except RRT* due to its low accuracy).

Planner	RRT*	BIT*	AIT*	ABIT*	*sequential*	*episodic*
Accuracy [%]	15.97	84.19	84.12	84.15	92.24	90.11
Accum. turn [rad]	—	0.63 ± 0.43	0.63 ± 0.43	0.63 ± 0.43	0.78 ± 0.55	0.94 ± 0.55
Length [m]	–	11.6 ± 6.1	11.6 ± 6.1	11.6 ± 6.1	11.8 ± 6.2	13.9 ± 5.3

To mitigate these phenomena, one has to resort to planners that plan smoother paths. A

comparison of those is presented in Table 5.2. We compared our solutions with the SL planner and a BIT* planner with the cc-Dubins extender, whose segments are guaranteed to maintain the continuity of the curvature. As these methods need more computations to plan, we limit their maximal runtime to 100ms. We can see that by increasing the continuity BIT* lose on the accuracy, also SL solves only about half of the considered motion planning problems, while the proposed methods have a success ratio above 90%. In terms of the planning time, *sequential* planner is comparable to BIT* and about 2 times slower than SL, as it needs to query the neural network multiple times. In contrast, the *episodic* planner performs only a single neural network inference and thus achieves the shortest planning time of 11ms. The *episodic* planner also generates paths of the lowest maximal curvature, which results in smaller centrifugal forces acting on the car following this path and allows for taking these turns with greater velocity. In contrast, the most accurate state-of-the-art approach – cc-Dubins BIT*, still plans paths with very sharp turns due to the Dubins path definition.

When we were analyzing the reachable sets of the configurations we mentioned that the path generated by the *sequential* planner is considered feasible if it reaches a set of configurations around the desired one Q_d, while the *episodic* planner reaches the desired state exactly. Therefore, the comparison of the accuracies of *sequential* and *episodic* planners is biased towards the former one. Therefore, to analyze quantitatively how the performance of the *sequential* planner changes when the size of the set of the allowable end configurations Q_d is reduced, we added to the Table 5.2 *sequential*† planner. In this case, the allowed deviation between the desired and the actually reached configurations is reduced 10 times. The introduction of tighter constraints on the final configuration in the training phase resulted in much harder problems to solve and thus in the reduction of the accuracy parameter to about 80% and significantly higher maximal curvatures.

TABLE 5.2: Comparison of the proposed methods with the planning methods that generate smooth solutions. The *sequential*† planner is a *sequential* planner retrained with 10 times tighter bounds imposed on the end configuration to enable more direct comparison with an *episodic* planner that guarantees the exact reaching of the desired state.

Planner	SL	BIT*	*sequential*	*sequential*†	*episodic*
Accuracy [%]	49.4	72.83	**92.24**	79.25	90.1
Time [ms]	23±27	45±35	43±2	43±2	**11±1**
Mean max κ $[m^{-1}]$	0.179	0.226	0.152	0.192	**0.145**
Continuity	$\mathbb{G}^2$	$\mathbb{G}^2$	$\mathbb{G}^2$	$\mathbb{G}^2$	$\mathbb{G}^5$

5.2.2 Training speed

In Chapter 3 we stated that the proposed B-spline path definition and the proposed construction method introduce an inductive bias on the generated paths. Here, we experimentally analyze their impact on the training of the *episodic* model and refer it to the *sequential* one. In Figure 5.2, we present the plot of epochs mean accuracy achieved by both methods on the training and validation sets. One can see that thanks to the fact that for B-spline path representation, we are able to fix the end configuration to be exactly equal to the desired state, which results in about a 60% success ratio just in the first epoch of training. Fixing the end state relieves us of the need to learn how to reach the goal, which significantly speeds up the training. Moreover, the

speed up in training can be also attributed to the proposed path construction method. Thanks to it, even a randomly initialized neural network can produce reasonable paths, which may solve easier motion planning tasks.

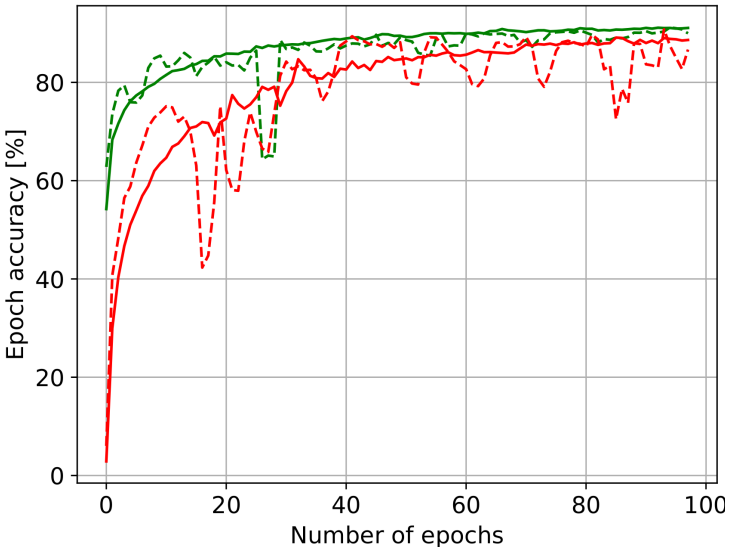

FIGURE 5.2: Accuracy on the training (solid) and validation (dashed) sets obtained by the *sequential* (red) and *episodic* (green) neural network planners throughout the learning process. The *episodic* planner trains much faster thanks to the proposed B-spline path representation and construction method.

5.2.3 Parameters of the methods

The proposed neural network-based motion planning methods don't have many parameters. Particularly, if we assume that the architecture of the network is fixed then the most important parameter is the expressiveness of the path representation understood as the number of its degrees of freedom. In the case of *sequential* planner, this is strictly connected with the number of segments, while for *episodic* with the depth of the control points tree. We show the dependence of the accuracy and planning time on these quantities in Figure 5.3 and 5.4. In both cases, increasing the number of the path's degrees of freedom results in increased planning time. For the *sequential* planner the dependence is linear, as each additional segment results in the additional neural network inference, which is the most computationally expensive part of this algorithm. In turn for the *episodic* planner, growth is non-linear as for the deeper trees the amount of computations needed to interpret the output of the neural network dominates the evaluation of the neural network. Interestingly, in both cases, increasing the expressiveness of the path results in better accuracy only up to some point, after which further increasing leads to the degradation of the performance. For this reason, in the considered dataset, there is no reason to use a number of segments bigger than 8 for the *sequential* planner and a depth bigger than 3 for the *episodic* one.

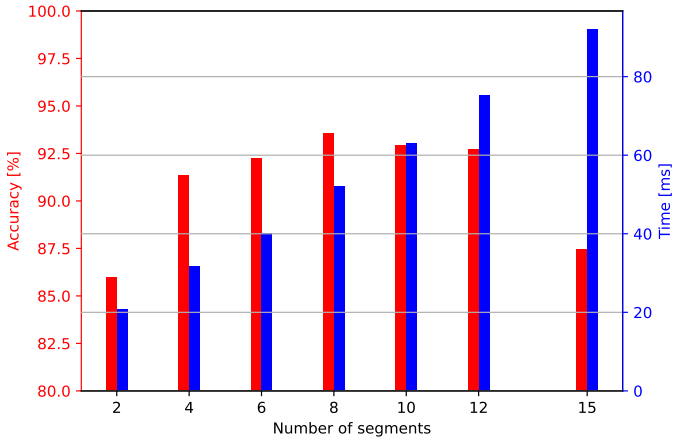

FIGURE 5.3: Accuracy and inference time with respect to the number of segments n_{seg}.

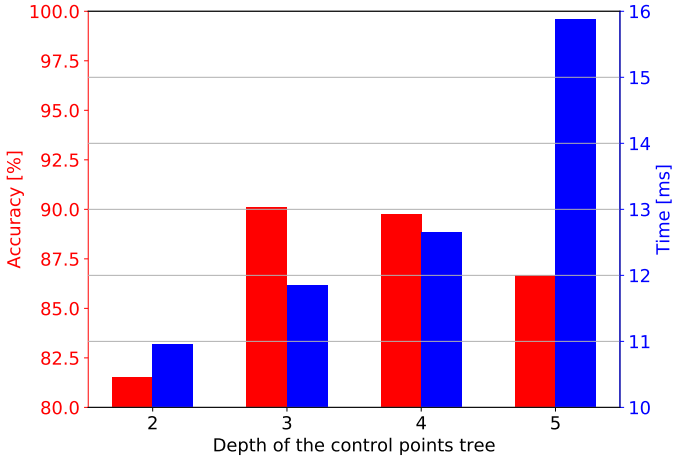

FIGURE 5.4: Accuracy and inference time with respect to the depth of the control points tree.

5.2.4 Ablation studies

In this section, we want to analyze: (i) the impact of the *total curvature loss* that we introduced in Section 2.3.4 to smooth out the paths generated by the neural network-based planners, and (ii) qualitatively the differences between Dubins curves, cc-Dubins and paths generated by our proposed planner

Impact of the *total curvature loss*. In Section 2.3.4 in equations (2.23) and (2.26) we introduced a *total curvature loss* $\mathcal{L}_{tcurv}$ that is meant to smooth out feasible paths generated by the neural network-based planner. The conditional construction of the total loss of the planner (2.23) shows us that *total curvature loss* is the only term in the loss function that is not necessary to obtain a feasible path. Therefore, we analyze the impact of this part of our loss function on the planner accuracy and the quality of generated paths. To conduct this analysis, we trained and tested a new *sequential* model in the same conditions as the one analyzed in previous experiments but without $\mathcal{L}_{tcurv}$. Although the proposed solution can plan feasible paths without this loss, and do it even slightly more frequently than when the loss is added – 92.53% vs. 92.24%, the impact of the $\mathcal{L}_{tcurv}$ is visible in terms of the mean accumulated turn, which is equal to 0.86rad for the model trained without the $\mathcal{L}_{tcurv}$ term, and 0.78rad for the

model trained with it. Note, that such a small difference in terms of the planner's accuracies is very likely caused by the fact that the *total curvature loss* is applied only to the paths that are already feasible, such that it is not affecting the process of improving the infeasible paths.

A more direct view of the impact of the *total curvature loss* is presented in Figure 5.5, in which we compare paths generated by the aforementioned models in several different scenarios. Even though in all cases the maximal admissible curvature is not exceeded, paths in the top row (generated by the model with full loss function) are visually much smoother than those in the second row (generated by the model trained without $\mathcal{L}_{tcurv}$). This smoothness is highly desirable as it limits the sudden and frequent turns, which are usually unnecessary to perform the maneuver, limit maximal allowed velocity, and cause problems in control. Moreover, maneuvers performed smoothly, without redundant turns, are far closer to the expectations of human drivers and give a feeling of safe and comfortable driving.

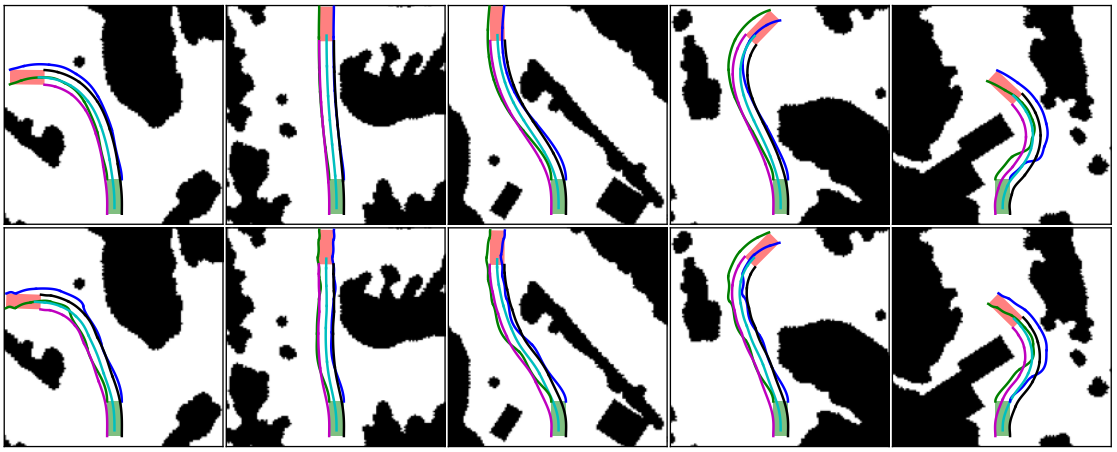

FIGURE 5.5: Sample paths generated by the *sequential* neural network-based motion planner trained with the standard loss function (defined in Section 2.3.4) – first row, and with modified loss function without $\mathcal{L}_{tcurv}$ (defined in (2.26)) – second row. The presence of the total curvature loss term results in much smoother-looking and easier-to-follow paths.

Paths representations. To better visualize the differences between the paths generated by our planner and state-of-the-art competitors, i.e., BIT* with Dubins and cc-Dubins extenders, we showed in Figure 5.6 paths they were able to generate in a scenario of very narrow turn. For the considered problem our planner was able to plan a collision-free path, that satisfies the curvature constraints within 12 ms, in contrast to the BIT* with continuous curvature Dubins paths, which was unable to do so, even when allowed to plan for 100s. The reason it performs so poorly may be due to problems with sampling states that are possible to connect with cc-Dubins curves, and the need to include the clothoid segments between the straight lines and arcs, which is hard to perform when the free space is heavily confined. In turn, BIT* with Dubins extender manages to find a solution, however, finding a collision-free Dubins path in such a turn requires the vehicle to stop 5 times in order to reach the goal. This type of maneuver exposes the problems in planning with the use of curves which enables one to plan fast, but at the same time imposes severe constraints on the curvature of path segments. In contrast, the proposed solution is able to plan extremely fast, while being able to produce smooth paths with a curvature tailored to the specific motion planning problem.

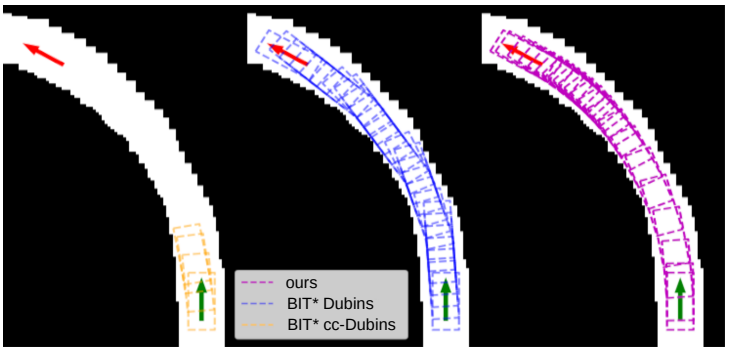

FIGURE 5.6: Paths generated by the proposed planner (purple) and BIT* with continuous curvature Dubins curves (orange) and BIT* with plain Dubins curves (blue) for the turn scenario on the extremely narrow road.

5.3 Experiments in CARLA

In the second part of the experimental evaluation, we focus on the application of the proposed neural network-based motion planning methods in the simulated autonomous vehicle in CARLA simulator [38]. These experiments aim to show not only the pure planning performance, as it was the goal of the previous section, but also the impact of planning time and execution of the generated plans in conditions close to the real robotic platform. In these experiments, we used the same neural network models as in the previous ones, however, due to the fact that Kia Rio III is not modeled in CARLA we applied the planned paths on the Audi A2, whose physical dimensions and steering properties are very similar to the Kia's ones.

5.3.1 Controller

While for the experiments in the dataset, we assumed that the motion of the robot is nominal along the planned path, i.e., ideal path tracking, this is typically not the case when deployed to the robot, even in simulation. Thus, to perform the experiments in a simulated environment in the CARLA simulator, we need a controller able to track the paths generated by the planner. The way of representing the paths by the solution proposed in Chapter 2 allows to use of a specific kinematic controller proposed in [53], however, it is not suitable to be directly used to track the paths represented as B-splines. Therefore, to obtain a fair comparison between the performance of the proposed planners and to bring the conditions closer to those of real autonomous driving systems [7, 47], we decided to implement a MPC style controller that is agnostic to the definition of the solution path.

In these experiments, we used a Cross Entropy Method (CEM) [147] based controller. The core idea of this algorithm is to draw random controls from the normal distribution, simulate the behavior of the system, assess its quality w.r.t. some performance criterion and then update the parameters of the distribution of actions, and repeat the whole process until a satisfactory sequence of actions is found or the time budget is spent. In our considered scenarios, we would like to follow paths, thus we assume that the target velocity is controlled by an internal PID controller in the CARLA simulator and is meant to be constant. Similarly, in CARLA we do not

have control over the steering wheel rotation speed ξ as in (2.2) but need to specify the desired steering angle β instead, which is then achieved using some low-level PID controller. Therefore, we adjust the model introduced in (2.2) to the following form

$$\dot{\boldsymbol{q}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ v\frac{\tan\beta}{L} \end{bmatrix}, \tag{5.1}$$

thus the purpose of our implemented CEM based controller is to adjust the steering angle β to follow the planned path with some predefined velocity v. The pseudocode of CEM in our particular case is presented in Algorithm 2. For a more detailed description of the CEM, we refer the reader to [147].

The proposed algorithm has several parameters, which were set constant throughout all of the experiments, i.e., the number of trajectories n_s was set to 32, the number of elite trajectories $n_e = 4$, simulation horizon h was set to 15 steps with timestep $T_s = 50\,\text{ms}$. Besides that, we initialize all $\widehat{\beta}_i$ to 0.2 rad, while for the mean $\bar{\boldsymbol{\beta}}$ values we use 0 rad for all timesteps at the beginning of the simulation and then we initialize $\bar{\boldsymbol{\beta}}$ with the values obtained at the end of the last iteration of control algorithm. Finally, to choose the best actions we proposed a custom optimality criterion that takes into account the quality of the path tracking in terms of the position and orientation, as well as a penalty factor for too sharp changes in the steering angle to encourage smooth driving. As a result, the criterion J is computed as a sum of (i) the distance between the robot position on the XY plane and the closest point on the path, (ii) the error between the robot orientation and desired orientation at the closest point on the path, and (iii) a sum of absolute differences between the steering wheel orientations in consecutive timesteps. Both orientation-related parts of the criterion are scaled by a factor of 10.

Algorithm 2: CEM based control algorithm for the vehicle steering angle.

Given: number of elite trajectories n_e, number of trajectories n_s, simulation horizon h, vector of means of the steering angles $\bar{\boldsymbol{\beta}} = (\bar{\beta}_1, \bar{\beta}_2, \ldots, \bar{\beta}_h)$, vector of standard deviations of the steering angles $\widehat{\boldsymbol{\beta}} = (\widehat{\beta}_1, \widehat{\beta}_2, \ldots, \widehat{\beta}_h)$, vector of trajectory costs $\mathbf{j}$, optimality criterion J;
$\mathbf{j} = \inf$, actual state $\boldsymbol{q}_0$ and velocity v of the robot, timestep T_s;
while *computatiton time is not exceeded* ***and*** $\min(\mathbf{j}) > \bar{j}$ **do**
  for $i \leftarrow 1$ **to** n_s **do**
    Draw a sequence of steering angles $\boldsymbol{\beta}^i$ from a normal distrubution $\mathcal{N}(\bar{\boldsymbol{\beta}}, \widehat{\boldsymbol{\beta}})$;
    Simulate behavior of the car for T_s using (5.1), its initial state $\boldsymbol{q}_0$ and velocity v, and a sequece of controls $\boldsymbol{\beta}^i$ and store as a trajectory $\boldsymbol{\zeta}$;
    The obtained trajectory $\boldsymbol{\zeta}$ of the vehicle state evolution is then assessed using criterion $j_i = J(\boldsymbol{\zeta})$;
  end
  Find n_e best trajectories and fit the normal distribution parameters $(\bar{\boldsymbol{\beta}}, \widehat{\boldsymbol{\beta}})$ using their sample steering angles;
end
return $\beta_1^{\text{argmin}(\mathbf{j})}$;

5.3.2 Planning typical maneuvers

In our experiments, we want to evaluate the ability of the proposed neural network-based planners to plan feasible paths that are possible to be safely followed with the above-described controller for several practical maneuvers including perpendicular, parallel, and diagonal parking, turning, passing densely parked vehicles and avoiding the collision with the vehicle that suddenly crosses the path of the ego-vehicle. Moreover, we would like to analyze the driving quality, which is affected not only by the quality of the planned path but also by the time spent on the planning, as during this time no new control is computed because both planning and control are calculated in a single thread.

Each of the scenarios is defined as a map, the initial position of all agents, including the ego-vehicle, schedules of controls for other moving agents, and a series of sparse waypoints (located about 15-20m apart from the consecutive ones) that are guiding the vehicle to reach its ultimate goal. The goal of the ego vehicle is to plan and execute a path to the next waypoint on the list. To perform each of the maneuvers in a way that allows for reacting to the changing environment conditions and thanks to the extremely fast motion planning provided by the neural networks, the plan is recomputed every 10 timesteps of the simulation, which framerate is set to 20. To achieve smooth transitions between waypoints, the next one is selected after reaching a distance less than 7m from the actual one. In general, the desired velocity of the ego-vehicle is constant but for the scenarios that should end in the steady-still configuration, we introduce a schedule that after reaching some distance from the ultimate goal linearly reduces the desired velocity down to 0.

Using the proposed neural network-based motion planning algorithms one can plan in a variety of the typical driving scenarios. Here, we present the planning and execution of several typical ones[1].

First, we show 3 maneuvers that were executed successfully using both proposed planners:

- dynamically pass with a velocity of $5\,\frac{\mathrm{m}}{\mathrm{s}}$ several vehicles parked on both lanes of the road avoiding collisions with them (see Figure 5.7),
- go forward with a velocity of $7\,\frac{\mathrm{m}}{\mathrm{s}}$ to find a parking space and perform parallel parking (see Figure 5.8),
- turn left on the crossroad with a velocity of $10\,\frac{\mathrm{m}}{\mathrm{s}}$ (see Figure 5.9).

For all of these maneuvers, our proposed planners were able to plan paths that followed by the controller resulted in successfully completed tasks. Even though both of them are able to solve these challenging tasks there are some notable differences in the planning time and the quality of the planned paths. In particular, we want to analyze the path-tracking errors to find out which ones are easier-to-follow and if there are any issues that result in deviations from the planned paths for a common controller, as even the best-planned path if cannot be followed accurately

[1] the videos presenting the conducted experiments that are described below can be found at `https://chmura.put.poznan.pl/s/PLZPIpRr5C9O0Bn`

may result in dangerous behavior or even an accident. In Table 5.3 we present the integrals of positional (IPE) and orientation errors (IOE) obtained by tracking the paths planned by the *sequential* and *episodic* planners.

TABLE 5.3: Path-tracking performance for both proposed motion planning methods in 3 different scenarios. Results are represented as: integral of position error [m] / integral of orientation error [rad].

	Passing densely parked vehicles	Turning left	Parallel parking
sequential	0.48 / 0.52	0.81 / 0.4	1.54 / 0.62
episodic	0.2 / 0.25	0.56 / 0.18	0.25 / 0.13

FIGURE 5.7: Sequences of frames from the scenario of passing densely parked vehicles executed with the use of the *sequential* (top rows) and *episodic* planners (bottom rows).

FIGURE 5.8: Sequences of frames from the scenario of parallel parking executed with the use of the *sequential* (top rows) and *episodic* planners (bottom rows).

In the second part of our experiments in the CARLA simulator, we want to evaluate further the practical differences in solving scenarios of typical maneuvers when using *sequential* and *episodic* planners.

One of the fundamental differences between *sequential* and *episodic* planners is that the second one guarantees that the desired goal state is reached precisely, at least for the nominal move along the planned path. To evaluate the impact of the guaranteed boundary conditions satisfaction we consider the task of diagonal parking between tightly parked cars. We performed several experiments to identify the smallest gap between cars that allow collision-free parking for both planners. In Figure 5.10 we present a failure scenario, in which the gap is set to 205cm, which

FIGURE 5.9: Sequences of frames from the scenario of left turn executed with the use of the *sequential* (top rows) and *episodic* planners (bottom rows).

is the biggest distance that causes the *sequential* planner to fail. In turn, in Figure 5.11 we present the scenario with the smallest gap that allows for successful parking using *episodic* planner – 175cm. Having in mind that the width of the ego-vehicle is 167.3cm, we can claim that the proposed *episodic* planner allows for precise parking in extremely narrow parking places. Note that the absolute numbers are taken from the datasheets of the considered cars, while the practical collision bodies used in the simulator may be slightly different.

FIGURE 5.10: Sequence of frames from the scenario of diagonal parking in a 205cm wide parking place using *sequential* planner.

FIGURE 5.11: Sequence of frames from the scenario of diagonal parking in a 175cm wide parking place using *episodic* planner. Precise planning and superior path-tracking performance allow for parking in a place that has only about 8cm of theoretical slack.

In all of the above-described experiments, the desired velocity was set to some predefined value. In the next experiment, we want to evaluate how increasing the desired velocity affects the success ratio of the perpendicular parking scenario. This scenario consists of the straight segment in which the ego vehicle needs to avoid collision with the vehicle leaving the parking space, and then perpendicular parking in a relatively wide parking place. A sequence of frames that visualize the scenario is presented in Figure 5.12, while the success ratios of solving this scenario with different velocities are presented in Table 5.4. Wide parking place allows for aggressive parking with relatively high velocity. Both the *sequential* and *episodic* planners are able to successfully park even with the desired velocity up to 9 $\frac{\text{m}}{\text{s}}$. However, the stable performance requires limiting the velocity to 7 $\frac{\text{m}}{\text{s}}$ for *episodic* planner, while the *sequential* one works consistently only up to 5 $\frac{\text{m}}{\text{s}}$.

Finally, we evaluated the proposed *episodic* neural network-based planner on the challenging task of avoiding a collision with a car that suddenly crosses the ego vehicle's trajectory. To make this task even more challenging we set the desired velocity to 15 $\frac{\text{m}}{\text{s}}$ and for responsiveness we replan the path every 5th simulation iteration. Due to such high velocity and replanning frequency,

FIGURE 5.12: Sequence of frames from the task of perpendicular parking using the *episodic* planner.

TABLE 5.4: The number of successfully executed scenarios (out of 5) of perpendicular parking for several different desired velocities. Plans generated by the episodic planner enable successful parking with greater velocities.

	$5\,\frac{\mathrm{m}}{\mathrm{s}}$	$7\,\frac{\mathrm{m}}{\mathrm{s}}$	$9\,\frac{\mathrm{m}}{\mathrm{s}}$
sequential	5	3	1
episodic	5	5	2

we were unable to run the same experiment using the *sequential* planner as the relatively long planning time was introducing a time delay that caused the controller instability.

A sequence of frames that visualize a collision avoidance scenario is presented in Figure 5.13. The ego vehicle is going straight at the speed of $15\,\frac{\mathrm{m}}{\mathrm{s}}$, while suddenly the blue car changes lanes and intersects its trajectory. Thanks to the frequent replanning, the ego vehicle is able to react to the change in the environment and plan a path to smoothly avoid the collision.

FIGURE 5.13: Sequence of frames from the high-speed collision avoidance using the *episodic* planner.

Chapter 6

Experimental verification of the constrained neural kinodynamic motion planning

6.1 Introduction

In Chapter 4 we introduced a novel approach to learning how to rapidly solve kinodynamic motion planning problems using B-spline trajectory representation and neural networks. In this Chapter, we want to verify the abilities of the proposed method to generate plans:

- under very tight planning time limitations,
- minimizing some optimality criterion,
- satisfying the robot's internal limitations, such as maximal joint's velocity, acceleration, and torque,
- satisfying constraints imposed on the position and orientation of the end-effector,
- satisfying the boundary conditions imposed not only on the configuration but also on the velocity and acceleration,
- lying in the collision-free space.

To evaluate these abilities we introduce 2 challenging motion planning tasks:

1. moving a heavy (close to the payload limit) vertically oriented object using KUKA LBR Iiwa 14 robot,
2. high-speed hitting in the robotic Air-Hockey using KUKA LBR Iiwa 14 manipulator.

We chose KUKA LBR Iiwa 14 as this is a widely spread robotic manipulator with 7 degrees of freedom which gives us great flexibility in terms of planning complex movements taking into account the task and robot constraints.

6.1.1 Baseline motion planning algorithms

To obtain a fair perspective on the performance of the proposed motion planning algorithm, we compared it with several State-of-the-Art motion planning algorithms that span the space of different solutions to the considered motion planning problems:

- TrajOpt [151] – a motion optimization algorithm that utilizes Sequential Least Squares Programming (SLSQP),
- CBiRRT [12] – a sampling-based path planning algorithm that uses RRTConnect [98] with the projection of sampled points onto the constraint manifold,
- SST [105] – a sampling-based motion planning algorithm that builds a sparse tree of robot configurations and extends it using random controls,
- Model-Predictive Motion Planning Network (MPC-MPNet) [102] – a sampling-based motion planning algorithm, which uses a neural network to determine the next node in a search tree and CEM to steer towards this configuration.

It is worth noting, however, that the considered motion planning problems are extremely complex as they join the problems of kinodynamic motion planning, with severe constraints imposed not only on the joint's velocities and accelerations but also on the movement of the end-effector, which is highly-nonlinear w.r.t. the movements in joints. The only state-of-the-art approach that matches all of the requirements is trajectory optimization [151]. In turn, the rest of the methods need to be properly adjusted to meet the requirements or we need to accept their limitations. For example, most of the algorithms are not suited to incorporate boundary conditions different than positional ones, therefore, even if they can plan a feasible path to follow, they cannot ensure the right hitting direction in the robotic Air Hockey.

To adjust SST and MPC-MPNet to work on the constrained motion planning problems, we implemented the workspace constraints as obstacles. In the case of the equality constraints, we added margins, for moving a vertically-oriented object, the object's orientation was assumed valid until the product of cosine of roll and pitch angles was bigger than 0.95, while for Air Hockey hitting task, we added an acceptable deviation of 1 cm from the table surface. In turn, to allow CBiRRT to plan in kinodynamic tasks we followed the idea introduced by the authors of [12], which assumes that the movement is quasi-static, such that during the planning only the feasibility of maintaining the robot in a given configuration is checked, while velocities and accelerations are disregarded.

Moreover, to introduce a very competitive benchmark, in the Air Hockey hitting task we extend the baseline by the Anchored Quadratic Programming (AQP) method [110], which is the current state-of-the-art in this area – an algorithm developed specifically to solve the problem of planning in the Air Hockey game.

6.1.2 Evaluation environment

Both the baselines and our method were evaluated in simulation environments developed using ROS1 Noetic [140] and Gazebo [96]. The experiments of real Air Hockey hitting were possible thanks to the in-house constructed physical setup [110] with a Kuka LBR Iiwa 14 robot, full-scale Air-Hockey table, and OptiTrack. All considered motion planning methods were run on the CPUs. For the experiments in simulation, we used an Intel Core i7-9750H CPU, while the real robot experiment uses AMD Ryzen 9 3900x CPU.

6.1.3 Controller

To deploy the proposed motion planning method and fairly compare it with state-of-the-art planners, we need a controller able to track planned trajectories. In particular, as we want to execute fast trajectories on the constraint manifold we need a fast and precise one. Unfortunately, the controller provided by the robot manufacturer has considerable problems with tracking fast trajectories close to the robot's limits. Therefore, to satisfy these requirements we used a decentralized controller based on the Active Disturbance Rejection Control (ADRC) paradigm [52, 60]. Nevertheless, we kept the gravity compensation part of the KUKA controller in the inner control loop on the real robot and mimicked its behavior in simulation. In Figure 6.1 we present the general control scheme of our system.

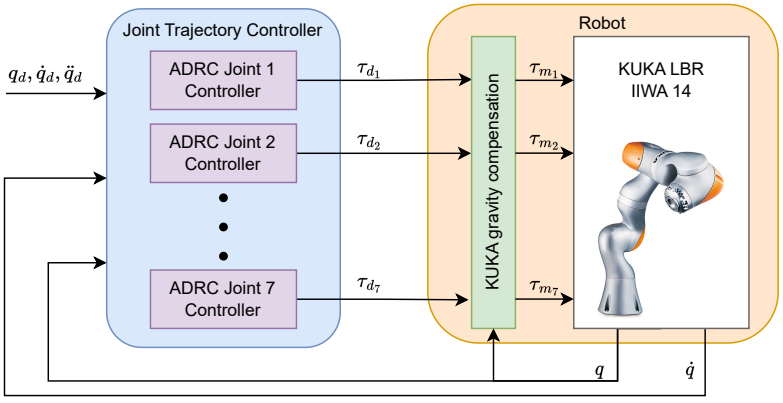

FIGURE 6.1: General scheme of the controller used in our experiments. We utilized the KUKA gravity compensation in the inner loop and the joint trajectory controller in the outer loop.

The crucial parts of the controller we used are the ADRC joint's controllers in the outer loop. ADRC is a paradigm of designing controllers that owes its effectiveness to the rapid online estimation of the *total disturbance*, which is used to decouple the controlled system from the actual perturbation acting on the plant. By *total disturbance* we understand an additional fictitious state variable that represents the effects of the unmodelled parts of the controlled system dynamics as well as external disturbances. In the case of the decentralized manipulator controller, we assume that this disturbance encompasses unmodelled phenomena like stiction and cross-couplings between joints. While in general, cross-coupling is a substantial part of the robot dynamics, its impact is significantly reduced thanks to the robot design and high-ratio gears. Nevertheless, we still need to estimate *total disturbance* to obtain high control performance.

This estimation in the ADRC paradigm is performed using extended state observer [118, 143], which in our case is implemented as typical Luenberger observer [112]. In Figure 6.2, we present schematically a i-th ADRC joint controller we used. In fact, specific joint controllers differ only in terms of the observer gains $l_i^{z_1}, l_i^{z_2}, l_i^{z_3}$, controller proportional and derivative gains k_{p_i}, k_{d_i} and respective diagonal element of the mass matrix M_{ii}. While M_{ii} elements are just taken from the model, the rest of the parameters need to be set. To do so, we utilized the pole placement technique [137], such that we needed to only choose the location of the poles for the controller and observer. One of the most common approaches to place the poles is to locate them at a single value $-\omega$, where ω can be interpreted as a bandwidth, specifically observer bandwidth ω_o and controller bandwidth ω_c. Based on these parameters one can easily compute the observer and controller gains using the following formulas

$$(l_i^{z_1}, l_i^{z_2}, l_i^{z_3}) = (3\omega_{o_i}, 3\omega_{o_i}^2, \omega_{o_i}^3), \tag{6.1}$$

$$(k_{p_i}, k_{d_i}) = (\omega_{c_i}^2, 2\omega_{c_i}). \tag{6.2}$$

Specific values of these bandwidths for all joints, which were used in the experiments we performed, are described in the next section.

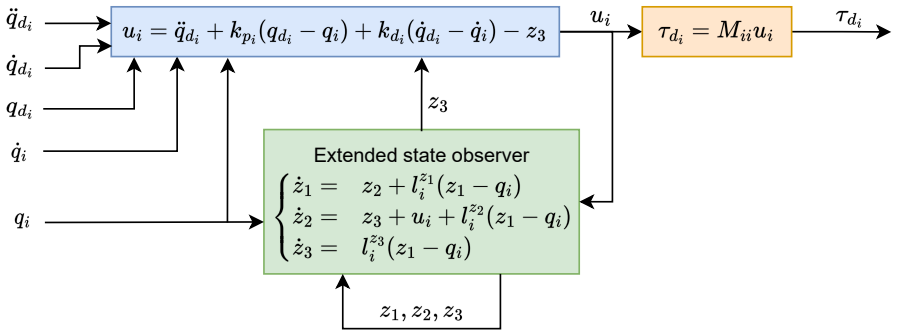

FIGURE 6.2: General scheme of the ADRC-based joint trajectory controller.

6.1.4 Parameters of the algorithms used for evaluation

One of the key features of the research should be its reproducibility [56]. Therefore none of the considered baseline methods was implemented by ourselves, instead, we relied on the following implementations:

- for TrajOpt we employed the SLSQP minimization implemented in SciPy, with constraints and cost function implemented by us, using Pinochchio library [23],
- for MPC-MPNet and SST we used the implementation provided by the authors of [102] with some necessary modifications required to compile and run their code, and our C++ implementation of the heavy object manipulation and robotic Air Hockey systems which also utilizes the Pinocchio library [23],
- for CBiRRT we used the OMPL [160] implementation and our own constraint and distance definitions,

- for AQP we relied on the implementation provided by the authors of [110].

An important part of every motion planning method is its parameters. To make our comparison fair we tried to adjust the parameters of these motion planning algorithms, to maximize their performance. Appropriate parameter tuning is especially important for the sampling-based motion planning algorithms, therefore we performed a random search of the parameters to find the regions of the parameter space, which give the highest success ratios and shortest planning times.

In the case of the SST and MPC-MPNet, some of their parameters are common because they use the same algorithm (RRT*) under the hood, the difference is in the way of selecting the node to expand and in the Expand procedure itself, which is done using MPC in MPC-MPNet algorithm, while in SST random controls are used. Therefore, we set for both algorithms the same parameters of the RRT* algorithm:

- Heavy object manipulation
  - node search radius $\delta_{BN} = 0.2\,\mathrm{m}$,
  - witness radius $\delta_s = 0.1\,\mathrm{m}$,
  - goal radius $r_g = 0.2\,\mathrm{m}$,
- Robotic Air Hockey hitting
  - node search radius $\delta_{BN} = 0.05\,\mathrm{m}$,
  - witness radius $\delta_s = 0.02\,\mathrm{m}$,
  - goal radius $r_g = 0.02\,\mathrm{m}$.

For the specific meaning of these parameters, we refer the reader to the original work on SST* [105]. Regarding the Extend procedure, for both tasks, we set the m following parameters, for SST

- minimum number of steps – 1,
- maximum number of steps – 20,
- integration step – 5 ms,

whereas for MPC-MPNet we set the following parameters of the CEM MPC solver

- number of samples – 64,
- number of elite samples – 4,
- maximal number of iterations – 30,
- convergence radius – 0.02,
- integration step – 10 ms,

- Heavy object manipulation:
  - motion time Gaussian – $(\mu_t, \sigma_t) = (0.05, 0.1)$, clipped at time $t_{max} = 0.1\,\mathrm{s}$,
  - control Gaussian – $(\mu_c, \sigma_c) = (\mathbf{0}, 0.8\bar{\boldsymbol{\tau}})$,
- Robotic Air Hockey hitting:
  - motion time Gaussian – $(\mu_t, \sigma_t) = (0.1, 0.4)$, clipped at time $t_{max} = 0.5\,\mathrm{s}$,
  - control Gaussian – $(\mu_c, \sigma_c) = (\mathbf{0}, 0.5\bar{\boldsymbol{\tau}})$.

Nevertheless, despite the tuning of the above-mentioned parameters, obtained motion planning times were relatively long, which may be caused by the innate difficulty of the considered problem and very tight constraints, which can be viewed as narrow passages in the configuration space [92]. For both SST and MPC-MPNet, we used the Euclidean distance function in the task space. This simplifies the motion planning problem a lot, as it requires the planners to plan only for the position, disregarding the task of reaching the desired velocity. It is also possible to define a distance function that takes into account the distance in the space of the velocities weighted with the distance in the task space, however, this makes the exploration of the states-space inefficient, slowing down further the planning process. Another very important parameter of the MPC-MPNet is the training data. The MPC-MPNet relies on behavior cloning, therefore, we trained it on the solutions generated by AQP in the Air Hockey task, and by our proposed method for lifting a heavy object. For CBiRRT, for both tasks, we set the goal radius to 1 cm, and the allowed tolerance of the constraint to 0.01, while for the range parameter, we used a value automatically determined by OMPL. In the case of the TrajOpt implementation in SciPy [97, 168], there are no parameters that affect the performance of the method, so we only limited the maximum number of iterations to 100, to avoid prolonged optimization.

Our proposed method also has some important parameters that have to be chosen. For our experiments, we set the nominal number of control points of the $\boldsymbol{p}(s)$ and $\mathfrak{r}$ B-splines to 15 and 20 respectively, as it gives enough freedom to plan accurate trajectories that satisfy constraints. To ensure a high level of trajectory smoothness we set the degree of both B-splines to 7, and the number of neurons in each layer of the proposed neural network to 2048. Note that the impact of the size of the neural network and the number of control points is evaluated in the ablation study (see Section 6.5), where we show how robust to these parameters is the proposed approach. Another group of parameters is the one related to the manifold metric optimization. We set the initial values of the constraint scaling factors $\mathfrak{m}^{(0)}$ to zeros for the heavy object manipulation, while for the Air Hockey task, we proposed a different initialization, which was meant to equalize the initial gradients of loss functions and was equal to $\mathfrak{m}^{(0)} = \begin{bmatrix} \mathfrak{m}_{\mathcal{T}}^{(0)} & \mathfrak{m}_{\dot{\boldsymbol{q}}}^{(0)} & \mathfrak{m}_{\ddot{\boldsymbol{q}}}^{(0)} & \mathfrak{m}_{\boldsymbol{\tau}}^{(0)} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 10^{-2} & 10^{-4} \end{bmatrix}$. For the heavy object manipulation task, we defined the following levels of the allowed constraint violations $\bar{C}_E = 10^{-6}$, $\bar{C}_O = 10^{-5}$, $\bar{C}_{\dot{\boldsymbol{q}}} = 6 \cdot 10^{-3}$, $\bar{C}_{\ddot{\boldsymbol{q}}} = \bar{C}_{\boldsymbol{\tau}} = 6 \cdot 10^{-2}$, while for the hittng task $\bar{C}_{\mathcal{T}} = 2 \cdot 10^{-6}$, $\bar{C}_{\dot{\boldsymbol{q}}} = 6 \cdot 10^{-3}$, $\bar{C}_{\ddot{\boldsymbol{q}}} = 6 \cdot 10^{-2}$, $\bar{C}_{\boldsymbol{\tau}} = 6 \cdot 10^{-1}$. Note that the subscripts indicate to which constraint is a given value related, such that $\mathcal{T}$ relates to constraints of the Air-Hockey table geometry, $\dot{\boldsymbol{q}}$ joints maximal velocities, $\ddot{\boldsymbol{q}}$ joints maximal accelerations, $\boldsymbol{\tau}$ joints maximal torques, E collisions with the environment, and O orientation constraints. The update step for the $\mathfrak{m}$ parameters was set to 10^{-2} for both motion planning problems. Finally,

the learning step for neural network parameters was set to $5 \cdot 10^{-5}$, while the batch size was set to 128.

Last but not least, in our experiments, we are not only evaluating the plans themselves but also the effects of their execution by the controller. Due to this reason, we needed to choose the parameters of the used control algorithm. Thanks to the parametrization introduced in Section 6.1.3 to fully define the controller parameters we only need to set observer ω_o and controller ω_c bandwidth for each joint. In Table 6.1 we presented the chosen values that provided good tracking performance.

Table 6.1: Parameters of the ADRC-based controller.

Parameter	Value
$\boldsymbol{\omega_o}$	$\begin{bmatrix}60 & 80 & 100 & 140 & 120 & 120 & 140\end{bmatrix}$
$\boldsymbol{\omega_c}$	$\begin{bmatrix}30 & 35 & 60 & 60 & 60 & 60 & 60\end{bmatrix}$

6.2 Kinodynamic planning for moving a heavy vertically oriented object in simulation

6.2.1 Task description

The goal of this task is to quickly move a heavy cuboid (12 kg, which is close to the payload limit of 14 kg) between two blue boxes using Kuka LBR Iiwa 14, without any collisions, while maintaining the upward orientation of the cuboid. An illustration of this task is shown in Figure 6.3. This task requires planning a joint trajectory between two random configurations, minimizing movement time, and simultaneously satisfying joints' velocity, acceleration, and torque constraints. Moreover, the robot's trajectory has to ensure that both the robot and the handled object will not collide with the environment and that the object will be oriented vertically throughout the whole movement.

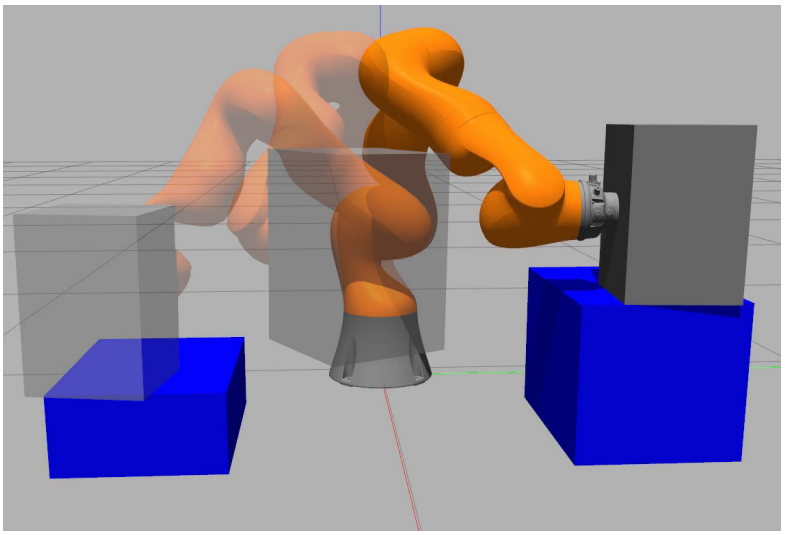

FIGURE 6.3: A Gazebo simulation of the task of kinodynamic planning for moving a heavy vertically oriented object using KUKA LBR Iiwa 14 between two boxes.

6.2.2 Dataset and method adjustments

6.2.2.1 Dataset

To learn how to solve the task, we generated a dataset of 26400 planning problems of this kind, split into training (24000) and validation (2400) subsets. In the dataset, we randomize both the initial and desired position of the object and the initial and desired robot's null space configuration. Both initial and desired velocities are set to 0. Note, that the dataset used in the training and validation contains only motion planning problems, without precomputed solutions, thanks to the use of a loss function that does not require any supervision. Therefore, the generation of the dataset does not require a significant amount of computation, and the performance of the method trained on this dataset is not bounded by the quality of the solutions included in the dataset.

The dataset for this experiment was generated in the following way:

1. draw random initial and desired position of the heavy object,
2. assume that the pedestal boxes have fixed dimensions and are located just beneath the objects,
3. draw initial guess configuration of the robot,
4. starting from this configuration, optimize the robot's initial configuration, such that its end-effector position matches the initial position of the heavy object, and the orientation is vertical,
5. validate if the robot in the initial configuration does not collide with the environment and if it does not violate the torque constraints,

6. starting from the initial configuration, optimize the robot desired configuration, such that its end-effector position matches the desired position of the heavy object,

7. validate if the robot in the desired configuration does not collide with the environment and if it does not violate the torque constraints.

The specific parameters values and ranges are shown in Table 6.2, where z_0, z_d represents the object's initial and desired position along z-axis, whereas $o_h = 0.15\,\mathrm{m}$ is the fixed height of the object.

TABLE 6.2: Parameters of the data generation procedure for heavy object manipulation task.

Parameter	Value
Initial object position	$(x_0, y_0, z_0) \in [0.2; 0.6] \times [-0.6; -0.3] \times [0.2; 0.5]$
Desired object position	$(x_d, y_d, z_d) \in [0.2; 0.6] \times [0.3; 0.6] \times [0.2; 0.5]$
Pedestal 1	$\{(x, y, z) \,\vert\, 0.2 \leq x \leq 0.6, -0.6 \leq y \leq -0.3, z \leq z_0 - o_h\}$
Pedestal 2	$\{(x, y, z) \,\vert\, 0.2 \leq x \leq 0.6, 0.3 \leq y \leq 0.6, z \leq z_d - o_h\}$
Initial guess robot configuration	$q \in [-\frac{\pi}{2}; \frac{\pi}{2}] \times [0; \frac{\pi}{2}] \times [-\frac{\pi}{2}; \frac{\pi}{2}] \times [\frac{\pi}{2}; 0] \times [-\frac{\pi}{2}; \frac{\pi}{2}]^2 \times [-\pi; \pi]$

6.2.2.2 Loss functions

Each of the tasks we place before the robots may require the use of different loss functions. However, based on the task requirements one can easily decide what kind of loss functions are necessary to learn how to solve a particular task. In the case of the task we consider in this Section, the optimality criterion is to minimize the time of the movement. Thus, we can use the task loss function based on (4.34):

$$\mathcal{L}(\boldsymbol{\zeta}_\dagger) = \int_0^T dt = \int_0^1 \mathfrak{r}^{-1}(s) ds. \tag{6.3}$$

In terms of the constraint losses, some of them are rather generic and were already defined in Section 4.2.5. Particularly, in this task we want to satisfy the constraints related to the maximal joint's velocity, acceleration, and torques, which were defined in (4.48-4.50). The only adjustment we made w.r.t. the nominal form of these losses, is the use of Huber loss function H, instead of the square of the internal loss. In practice, we observed that principal squaring may introduce too big updates, especially at the beginning of the training, and cause instabilities. These observations were rather general, therefore in all constraint losses we followed the same idea of using a Huber loss that is much more robust to outliers while having the same characteristics for

the values close to constraint satisfaction. Moreover, to satisfy the specific constraints imposed on this task, we add three additional loss terms, i.e., vertical orientation loss, robot collision loss, and object collision loss. The vertical orientation loss is defined by

$$\mathcal{L}^{O}(s) = \int_0^1 H(1 - \mathrm{R}_{2,2}(\boldsymbol{q}(s)))\mathfrak{r}^{-1}(s)ds, \tag{6.4}$$

where $\mathrm{R}_{2,2}$ is the element of the end-effector rotation matrix with an index of $(2, 2)$, robot collision loss defined by

$$\mathcal{L}^{E_r}(s) = \int_0^1 H\left(\sum_{p\in \mathrm{FK}_{kc}(\boldsymbol{q}(s))} \mathrm{ReLU}(0.15 - d_{euc}(p, E))\right)\mathfrak{r}^{-1}(s)ds, \tag{6.5}$$

where E represents the set of the collision objects in the environment (pedestals), FK_{kc} is a set of points in the workspace located along the kinematic chain (representation of the robot geometry), and $d_{euc}(X, Y)$ is a Euclidean distance between X and Y, and finally, object collision loss

$$\mathcal{L}^{E_o}(s) = \int_0^1 H\left(\sum_{p\in \mathrm{FK}_{o}(\boldsymbol{q}(s))} \mathrm{ReLU}(d_{euc}(p, E) \cdot \mathrm{I}(p, E))\right)\mathfrak{r}^{-1}(s)ds, \tag{6.6}$$

where FK_o represents the set of points that belong to the handled object and $\mathrm{I}(X, Y)$ is an indicator function, which is equal to 1 if $X \in Y$ and 0 otherwise. In our experiments, we defined environment E as two cuboids defined in Table 6.2. The heavy object handled by the robot is a cuboid with dimensions $0.2 \times 0.2 \times 0.3$ m, which for collision-checking purposes is represented by its corners. The robot itself is represented by the positions of the joints in the workspace and points linearly interpolated between them, such that no point lies further than 10 cm from its neighbors. We assume that there is no collision if the obstacles are at least 0.15 m away from the point of the kinematic chain.

6.2.3 Quantitative comparison with state-of-the-art

Finally, after preparing a training dataset and appropriate loss functions we can train our proposed neural network-based motion planner and compare its performance with several state-of-the-art motion planners. To perform the evaluation we generated 2400 random test tasks, using the same procedure that was used to generate the training set but ensuring that the generated data points are different from the ones used for training. For all these tasks, we asked all planners to plan the solution movements, which were then executed in simulation using the control algorithm described in Section 6.1.3.

The results of these experiments are presented in Figure 6.4. The first two bar plots show that our planner reaches the goal in all scenarios, and in nearly 97% of them, it does not violate any of the constraints. In contrast, comparable results are obtained only by CBiRRT [12], which reaches the goal in 86.5% of cases, and only about 66% are valid. The reason for this may be that CBiRRT is not able to accurately validate the feasibility of motions due to the quasi-static assumption. Moreover, this method needs more than 19 times more time to compute the solution (in terms of median values), and generated solutions are almost 3 times longer. The

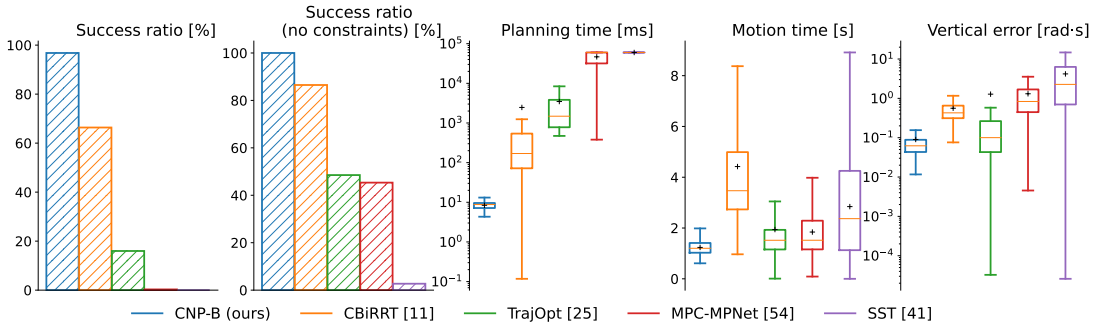

FIGURE 6.4: Planners statistics on the task of rapid movement of a heavy object with orientation constraints and collision avoidance.

last plot shows the error of maintaining the object in an upward position, which we compute as an integral along the trajectory of the sum of the absolute values of roll and pitch angles. The smallest deviation from the orientation constraint is achieved by our proposed solution, while the highest violations are generated by executing the trajectories planned using SST [105]. The only statistic in which our planner is not outperforming significantly other planners is the time of the planned motion. However, this metric cannot be considered without the success ratios, due to the fact that computing a short path that is not solving the given task is not particularly challenging. Last but not least, the result of the MPC-MPNet [102] deserves special attention, as it is the state-of-the-art learning-based solution for kinodynamic motion planning. We trained it using the plans generated by our planner, however, it was unable even to come close to the results achieved by the TrajOpt [151] and CBiRRT [12], not to mention our proposed planner. This behavior may be assigned to the very specific way of introducing randomness in this method, which relies on a strong dropout used at the inference phase. Thanks to this, MPC-MPNet relatively easily violated very strict constraints that were present in the considered motion planning problem.

6.3 Planning high-speed hitting movements in the simulated robotic Air-Hockey

6.3.1 Task description

The goal of this task is to score a goal in the game of robotic Air Hockey from a steady-still puck. This should be done by moving the Kuka LBR Iiwa 14 robot handling the mallet from some predefined initial configuration to the position of the puck and achieving the end-velocity vector pointing towards the middle of the goal of the opponent. An illustration of this task is shown in Figure 6.5. The desired configuration of the robot and velocities in its joints are determined using the optimization algorithm proposed in [110] for a given puck position and desired velocity direction. To achieve feasible and dynamic hitting movement we require the planner to generate a joint trajectory that satisfies the robot joint's velocity, acceleration, and torque constraints and minimize movement time. We also impose task space constraints, such that the mallet

handled by the robot remains on the table surface and between the bands throughout the whole movement.

This task challenges the planner in several important ways. First, to use a planner for generating the movements in the robotic Air Hockey it needs to be extremely fast, as the game could be very dynamic. Also, the movement of the robot needs to be high-speed to catch up with the pace of the game but at the same time, the mallet moved by the robot has to obey a strict constraint of moving only on the table plane. Finally, it is extremely important to plan trajectories that satisfy velocity, and acceleration constraints such that the planned motion is feasible to be tracked accurately, and boundary conditions, such that the planned motion results in hitting the puck towards the goal.

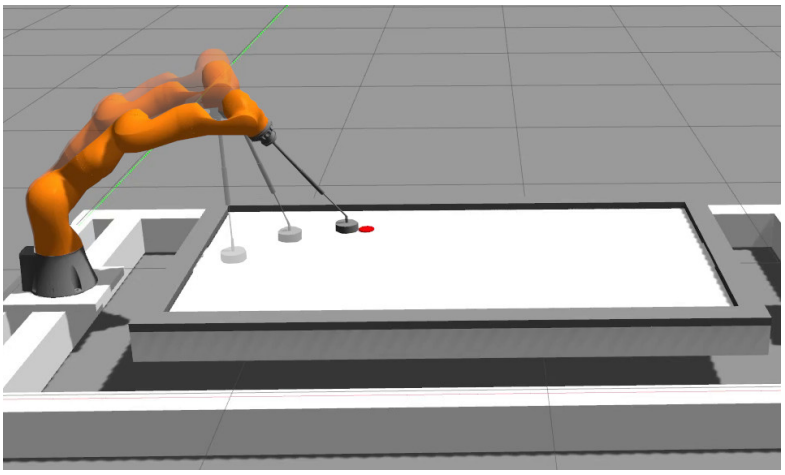

FIGURE 6.5: A Gazebo simulation of the task of fast hitting in the simulated game of robotic Air Hockey with a KUKA LBR Iiwa 14.

6.3.2 Dataset and method adjustments

6.3.2.1 Dataset

To learn how to solve the considered task, we generated a dataset of 19800 Air Hockey hitting planning problems, split into 2 subsets: training (18000) and validation (1800), while the test set was defined separately. The generation of a single data point of these datasets can be described with the following steps:

1. draw random initial and desired position of the mallet, such that they are at least 10 cm apart and fit into the sets defined in Table 6.3,
2. starting from the base configuration (defined in Table 6.3), optimize the robot's initial configuration, such that the position of its end-effector matches the initial position of the mallet,
3. using the desired mallet position and the position of the goal, define the desired hitting angle, that points towards the middle of the goal, and add uniform noise of a range ± 0.3 rad to it,

TABLE 6.3: Parameters of the data generation procedure for Air Hockey hitting task.

Parameter	Value
Initial mallet position	$(x, y, z) \in [0.6; 0.7] \times [-0.05; 0.05] \times [0.155; 0.165]$
Desired mallet position	$(x, y, z) \in [0.65; 1.3] \times [-0.45; 0.45] \times \{0.16\}$
Base robot configuration	$q_0 = [0 \quad 0.697 \quad 0 \quad -0.505 \quad 0 \quad 1.93]$

4. compute the desired joint velocity of the robot that maximizes the manipulability along the hitting direction using the optimization procedure proposed in [110],
5. in half of the cases randomly scale the magnitude of the desired joint velocity, and in the other half set it to a maximal one,
6. validate the possibility of performing the hit, by analyzing if it is possible to avoid a collision after the hit, i.e., if the point defined by $p_h = p_d + v_h \cdot 50\,\text{ms}$ lies between the table bands, where p_d is the desired hitting point, and v_h is the hitting velocity in the workspace.

Note, that similarly to the previous task, the generated dataset does not contain any precomputed solution, only motion planning problems.

While the above-described dataset is useful for learning how to plan for a very challenging problem of high-speed robotic Air-Hockey hitting, using our proposed motion planning method we can solve even more difficult ones, namely rapid replanning of the trajectory that smoothly connects to the current robot's motion. To learn this behavior, we need to create a dataset that is similar to the one for the steady-still hitting, however, far more diversified. To learn how to replan, we need to know how to plan between any two feasible configurations in the workspace. Moreover, to make our planner more versatile and to enable it to plan from any initial state we also randomized the desired hitting direction and initial velocity. Finally, to ensure a smooth connection to the previous trajectory, also at the level of controls, we randomized the initial joint's acceleration. The data generation procedure scheme is similar to the one shown above and differs only in the following steps

1. Initial and desired mallet positions range defined by $(x, y, z) \in [0.6; 1.3] \times [-0.45; 0.45] \times \{0.16\}$
3. draw a hitting angle which differs from the direction of the line connecting initial and desired positions no more than $\frac{2}{3}\pi$,
5. in 80% of the cases randomly scale the magnitude of the desired joint velocity, and set to maximal in the rest,
7. compute random initial joint velocity constrained to the table manifold, or set it to 0 in 20% of cases,
8. compute random initial joint acceleration constrained to the table manifold.

The created dataset consists of 120 000 samples, from which 112 000 belong to the training set and the rest to the validation set.

6.3.2.2 Loss functions

To learn how to plan robot motions for high-speed robotic Air Hockey hitting we need to define appropriate loss functions. The goal in this case is also to move as fast as possible, thus the task loss can be defined in the same way as in (6.3). However, we observed that when the trajectory curvature in the workspace and the robot's joint velocities are high, then the robot controller has problems with accurate trajectory tracking. Therefore, we introduce an additional centrifugal-forces-related regularization term to the typical time-minimization task loss and define the task loss function by

$$\mathcal{L}(\boldsymbol{\zeta}_{\mathfrak{f}}) = \int_0^1 \left(1 + \eta \kappa_{ee}(s) v_{ee}^2(s)\right) \mathfrak{r}^{-1}(s) ds, \tag{6.7}$$

where $\eta = 0.01$ is an experimentally chosen regularization factor, while κ_{ee} and v_{ee} are respectively the curvature and velocity of the end-effector trajectory.

In turn, in the case of the constraints function we used standard ones for penalizing the violations of the joint's velocity, acceleration, and torque limits (see (4.48-4.50)), and introduced an additional constraint manifold loss term that penalizes the displacement of the robot end-effector from the Air Hockey table surface. We define this loss function as the integral over the sum of the losses in x, y, z directions

$$\mathcal{L}^{\mathcal{T}}(\boldsymbol{\zeta}_{\mathfrak{f}}) = \int_0^1 \left(\mathcal{L}^{\mathcal{T}_x}(s) + \mathcal{L}^{\mathcal{T}_y}(s) + \mathcal{L}^{\mathcal{T}_z}(s)\right) \mathfrak{r}^{-1}(s) ds, \tag{6.8}$$

where specific local losses are defined by

$$\mathcal{L}^{\mathcal{T}_x}(s) = H(\mathrm{ReLU}(\mathcal{T}_{\underline{x}} - \mathrm{FK}_x(\boldsymbol{q}(s)))) + H(\mathrm{ReLU}(\mathrm{FK}_x(\boldsymbol{q}(s)) - \mathcal{T}_{\overline{x}})), \tag{6.9}$$

$$\mathcal{L}^{\mathcal{T}_y}(s) = H(\mathrm{ReLU}(\mathcal{T}_{\underline{y}} - \mathrm{FK}_y(\boldsymbol{q}(s)))) + H(\mathrm{ReLU}(\mathrm{FK}_y(\boldsymbol{q}(s)) - \mathcal{T}_{\overline{y}})), \tag{6.10}$$

$$\mathcal{L}^{\mathcal{T}_z}(s) = H(FK_z(\boldsymbol{q}(s)) - \mathcal{T}_z), \tag{6.11}$$

where $\mathcal{T}_{\underline{x}}, \mathcal{T}_{\overline{x}}, \mathcal{T}_{\underline{y}}, \mathcal{T}_{\overline{y}}$ are the lower and upper boundaries of the table in the x and y directions, while $\mathcal{T}_z$ is the table surface height. In turn, FK_x, FK_y, FK_z stands for the x, y and z components of the mallet position determined using forward kinematics FK.

6.3.3 Quantitative comparison with state-of-the-art

Finally, we trained our proposed neural network-based motion planner on the prepared dataset using introduced loss functions and compared its performance with several state-of- the-art motion planners on the set of 1681 hitting scenarios. These hitting problems were not present in the training and validation sets, and they were prepared by setting the initial robot configuration to the base one and locating the puck on a 41×41 uniform grid of locations in the range of the puck positions in the training data. For all these tasks, we asked all planners to plan the solution

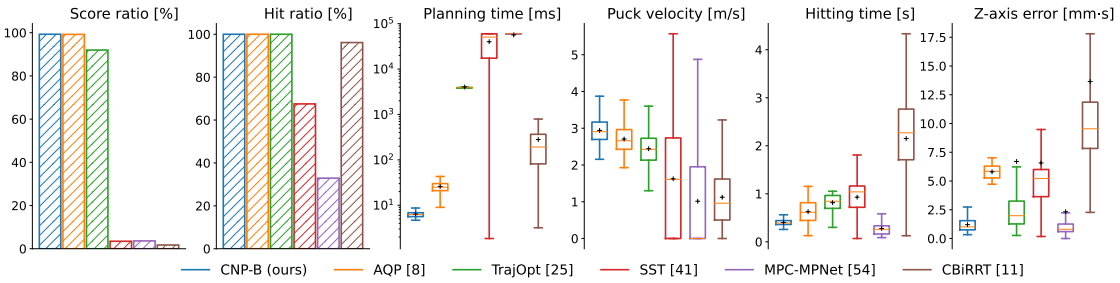

FIGURE 6.6: Planners statistics on the task of hitting in the simulated robotic Air Hockey.

movements, which were then executed in simulation using the control algorithm described in Section 6.1.3.

One of the key challenges of this task is that to score a goal the desired configuration has to be reached with a given velocity. Satisfaction of this constraint is non-trivial for many of the motion planning algorithms, especially sampling-based ones. Therefore for CBiRRT, MPC-MPNet and SST algorithms, we simplified the task, such that their goal was to hit the puck just by reaching the puck position disregarding the desired end velocity direction, i.e., we computed the distance function only for positional coordinates. Moreover, to train the learning-based baseline planner MPC-MPNet we used the trajectories generated by the CNP-B algorithm.

In Figure 6.6 we present the statistical evaluation of all considered planners on the task of the robotic Air Hockey hitting from the steady-still puck. Even though we simplified the task for MPC-MPNet and SST planners, they cannot plan trajectories within a reasonable time. The only sampling-based algorithm that is able to reach the target in almost all scenarios is CBiRRT, however, it produces plans that are very hard to follow (see z-axis error chart in Figure 6.6). Far better performance is achieved by optimization-based planners (TrajOpt and AQP), which always hit the puck and score in 91.97% and 94.88% of scenarios respectively. The main limitation of TrajOpt is that it requires nearly 4s on average to compute the plan, whereas the AQP median planning time is 26ms. A similar planning time scale is achieved only by our proposed method, which enables one to plan within 6.5ms on average. Moreover, our solution achieves a 99.28% scoring ratio, plans the fastest trajectories (except for MPC-MPNet which plans shorter trajectories but rarely hits the puck not to mention scoring), obtains on average highest puck velocities, and despite this, it generates the trajectories that are possible to follow with the accumulated z-axis deviation smaller than 3mm·s.Interestingly, AQP is a state-of-the-art solution tailored specifically to planning for robotic Air Hockey, and yet its performance is dominated in terms of all considered criteria by the solution trained on automatically generated data using our general framework.

6.3.4 Qualitative results for replanning

In the previous section, we have quantified how the proposed solution compares to state-of-the-art solutions in the task of hitting a steady-still puck using a manipulator starting its motion from a steady state. However, our proposed planner can do even more. Due to the short and deterministic planning time, and the ability to satisfy boundary conditions, our proposed

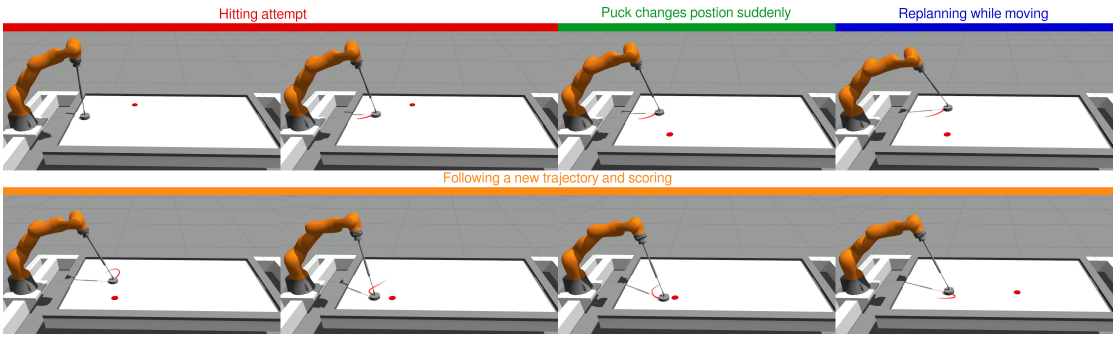

FIGURE 6.7: Sequence of frames form the replanning scenario. The robot starts the hitting motion with the puck located in the upper part of the table, but after 300ms puck is moved to the lower part. In response to this, CNP-B immediately replans the trajectory and scores the goal.

approach allows for solving tasks, that are impossible to solve using state-of-the-art planners, i.e., replanning the robot moves on the fly and smoothly connecting the new motion to the old one.

In this case, we consider a situation when the robot is performing some previously computed plan, and in the meantime, the goal of the movement changes, e.g., the desired hitting direction or the puck's expected position has changed. For this kind of task, the planning time of almost all state-of-the-art methods is too long to react on time. Moreover, classical motion planning methods do not give any practical guarantee about the maximal planning time for such a task. Unlike these approaches, our solution needs a small constant amount of computation to plan the motion. Therefore, we can predict a robot configuration located forward in time along the current trajectory, and plan from this configuration, taking into account the smoothness of the motion and continuity of control, by imposing the boundary conditions on the planned motion.

To visualize on-the-fly replanning we prepared a scenario where the robot tries to hit the puck, but after about 300ms from the beginning of the motion, the puck position changes suddenly. In response to this, the robot replans the trajectory from the point on the actual trajectory that is located a few tens of milliseconds in the future (to compensate for the non-real-time operating system and nondeterministic communication times). Then it waits until the vicinity of this point is reached and switches to the new plan. In Figure 6.7 we show a sequence of frames that visualize the above-described replanning procedure in action. One can see that the robot smoothly changes between plans and is able to score the goal with the replanned trajectory.

6.4 Planning high-speed hitting movements on the real robotic Air-Hockey setup

The most important validation step of the proposed motion planning solution is an experimental evaluation on a real robot. This is especially important in robotics, because of the well-known problem of the *reality gap*, which is common in systems that use machine learning in simulation, learn from a dataset of simulated examples [71]. Moreover, in the development of a planning method typically some theoretical assumptions are made. Thus to reliably deploy the system we

need to verify if they hold in a real-world experiment or are violated to an acceptable degree. To evaluate if the *reality gap* exists in our solution, i.e., if our model trained in simulation is able to generalize to the planning on the real manipulator, we used exactly the same neural network as in the simulation, without any additional learning [1].

6.4.1 Quantitative comparison with state-of-the-art

Similarly, like in simulation, we test our proposed neural network-based motion planner on the task of high-speed hitting in robotic Air Hockey starting from a steady-still manipulator in a base configuration. In the real-robot experiment, we reduced the number of test scenarios to 110, which were defined by the puck positions located on the 10×11 grid. To reduce the *reality gap* the whole setup is done exactly the same as it was in the simulation, such that the only differences stem from the inaccurately or incompletely modeled physics of the robotic system. Particularly, in the simulation we did not consider any friction, air bearing of the table, limited stiffness of the end-effector elements, backlash in the Cardan joint, etc.

The results of the experiments we performed in simulation show us clearly that AQP is the only baseline able to compete with the CNP-B, as it is able to compute safe-to-follow plans in a reasonably short time. Moreover, in the real-world experiment, we need to plan not only the hitting movement but also the safe slowing down movement of the robot after performing a hit. Thus, to ensure safety in real robot experiments, we limited the set of baselines to AQP only. Statistical comparison between AQP and our proposed planner is presented in Figure 6.8. The hitting movement times correspond with the ones obtained in the simulation, and we can see that CNP-B is characterized by a much smaller mean and variance. The puck velocity magnitude is higher for the proposed solution, due to the fact that AQP method scales down the hitting velocity if it cannot find a feasible plan. The biggest advantage of the proposed planner is visible in terms of the z-axis error, as the generated plans are much closer to the table surface. Also, the trajectory tracking errors are smaller for CNP-B, despite significantly faster trajectories.

Nevertheless, from the task point of view, the most important metric (besides safety) is the ratio of scored goals to all attempts. In terms of this metric, CNP-B outperforms AQP, by achieving a ratio of 78.2% compared to 52.7%. In Figure 6.9 we present the grid of puck initial positions and indicate the scored goal from this position with green squares and miss with red dots. One can see that AQP has problems with scoring for the puck close to the corners of the table, while CNP-B errors seem not to show any particular correlation with the position of the puck w.r.t. robot. We hypothesize that these few unsuccessful hits planned by CNP-B are related to some features of the mechanical setup that were not modeled in the simulation, particularly the backlashes in the Cardan joint and the elasticity of the end-effector construction, which are part of the *reality gap*.

[1] The videos of the performed experiments can be found at `https://sites.google.com/view/constrained-neural-planning/`

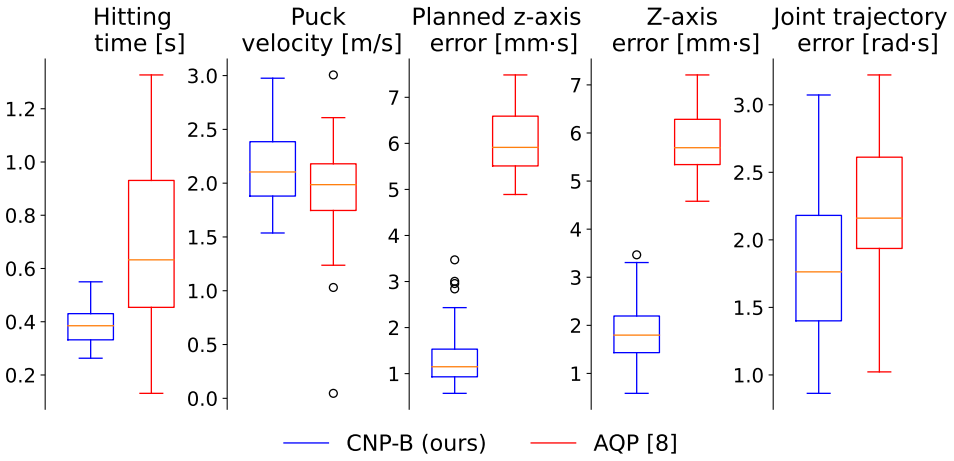

FIGURE 6.8: Planners statistics on the task of hitting in the real robotic Air Hockey.

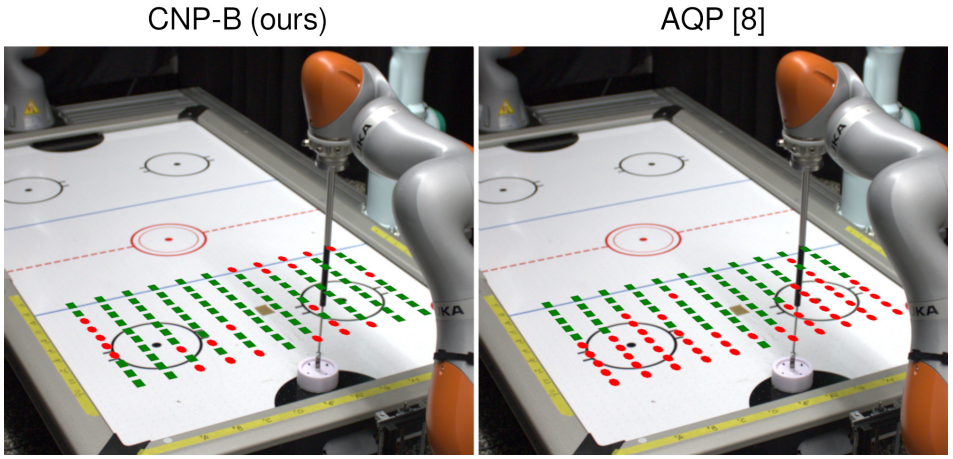

FIGURE 6.9: Visualization of the puck positions in the scenarios on which CNP-B and AQP planners were evaluated. The green squares represent scored goals (success), while the red dots missed shots (failure).

6.4.2 Trick shots

The ability of the proposed neural motion planner to replan robot motions on-the-fly, which we have shown in simulation, is also easily transferable to the real robot without any further learning. In Figure 6.10 we present a sequence of robot movements that were planned by the proposed approach. First, the robot is making feinting movements to confuse the opponent. Next, at some randomly chosen moment, the manipulator is asked to replan its motion to score the goal. To achieve this it queries CNP-B to compute a new trajectory, starting from non-zero velocity and acceleration, and leading to hitting the puck and scoring a goal. This kind of dynamic replanning behavior is possible only because our proposed solution plans within a very short and almost constant time, and is able to plan from non-zero boundary conditions imposed on velocity and acceleration. Nevertheless, on the real robotic platform, there can be some non-deterministic delays stemming for example from the communication between the PC and robot controller. To account for this one can try to estimate the mean delay and plan from the point on the current trajectory located further in the future. However, for the systems with non-negligible delays of high variance, this may be not enough. Interestingly, planning as a single inference of a neural network extends a helping hand to us, namely batch processing. Having an estimate of the possible delay one can plan with a neural network from many possible initial states and finally choose a generated trajectory that starts from the state that is the closest to the actual one. Note that using the same approach may also compensate for different types of errors of replanning from the wrong state, like inaccurate trajectory tracking. Thanks to batch planning we can ensure a smoother transition between planned trajectories, which may be crucial for high-speed movements of the real robot on the constraint manifold.

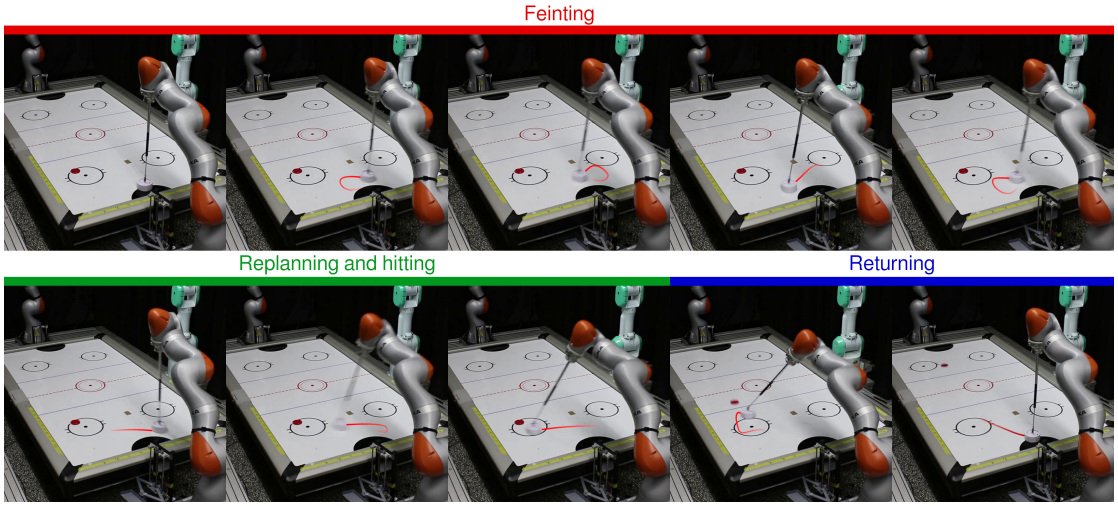

FIGURE 6.10: Fast on-the-fly motion replanning can be used to smoothly change the robot's behavior from feinting to striking almost instantaneously.

6.5 Ablation studies

The proposed motion planning method has multiple parameters that may affect its performance. While in section 6.1.4 we introduced some exemplary values of these parameters that were used

in all above-described experiments, in this section we analyze what is the impact of the changes of these parameters and the experiment's conditions on the performance of CNP-B. Particularly, we want to analyze how the performance of our proposed planner is affected by the size of the dataset used for training, the size of the neural network, and number of the control points of the configuration B-spline. Finally, we want to analyze the generalization abilities of CNP-B to the modification of some crucial parameters of the considered tasks, i.e., the mass of the cuboid handled by the robot and the height of the Air Hockey table.

6.5.1 Training set size

First, we analyze the impact of the training set size. The quality of the machine learning models depends heavily on the training data. To quantitatively assess this dependence in our particular case, we trained CNP-B for the task of simulated Air Hockey hitting on several different-sized subsets of the training dataset (see section 6.3.2.1), and analyzed its planning performance. In Figure 6.11 we present the results achieved by the models trained on fractions of the base dataset. As expected, the best result is obtained by the planner trained on the whole dataset. However, comparable success ratios and mean puck velocities are also achieved by the models trained on 10% and 1% of data. Nevertheless, the reduced number of training planning problems results in significant deterioration of other metrics, i.e., increased hitting time and deviation from the table plane. Time to hit is on average about 50% greater for models trained on 10% and 1% of the training set, and more than 2 times longer for the one trained with 0.1% of data. It is clearly visible, that if the space of the possible motion planning problems is covered more sparsely, planners are not pushing the performance so much to the limits and prefer safer but sub-optimal solutions. In terms of the violations of the table surface constraint, using less training data leads to remarkably worse performance only for the model trained on the smallest fraction of training data. However, for the models trained on 10% and 1% of the training data we observe a notable shift of the error distributions, while their medians stay at a similar level as for the model trained on the whole dataset. It seems that for both 1% and 10% of data, the movement time was traded off for a table constraint satisfaction. Nevertheless, when the space of motion planning problems is not so densely populated, in some of the areas the planned motions may not be accurate enough to maintain the end-effector right on the table plane.

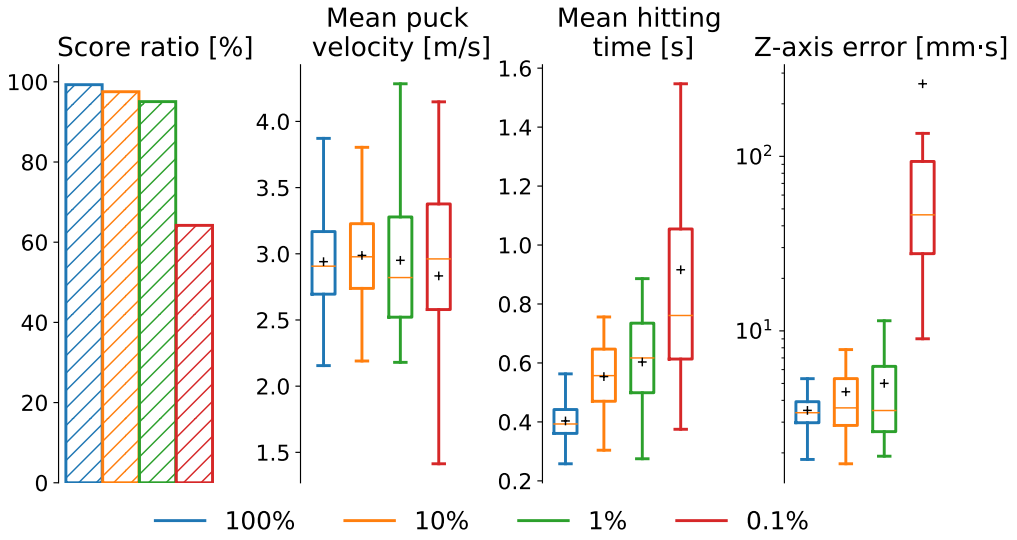

FIGURE 6.11: Comparison of the proposed neural planner trained on 100%, 10%, 1% and 0.1% samples of the training dataset.

6.5.2 Size of the neural network

In addition to the training data, the quality of the machine learning-based motion planner is also affected by the model used to approximate the planning policy. Particularly, the size of the model impacts its expressiveness and the number of computations needed to evaluate it. To assess its impact on the performance of the CNP-B in the task of moving a vertically oriented heavy object between boxes, we evaluated several neural networks of the architecture presented in Figure 4.2 with different numbers of neurons in each layer. Results of this experiment are presented in Figure 6.12. In general, it is clearly visible that almost all analyzed measures are nearly constant for all considered sizes of neural networks. There are only two exceptions. First, scaling the neural network down to 16 neurons caused a notable growth of the motion time and a significant increase in the orientation error. Second, in general, reducing the number of neurons leads to a decrease in the planning time and reaching the possibility to plan at the frequency of 1kHz. Although this finding seems rather obvious, as scaling down the number of computations should reduce the time needed to execute them, surprisingly, the timings for 16, 32, and 64 neurons are almost constant. We presume that this phenomenon may be related to the neural network size fitting the cache of the CPU.

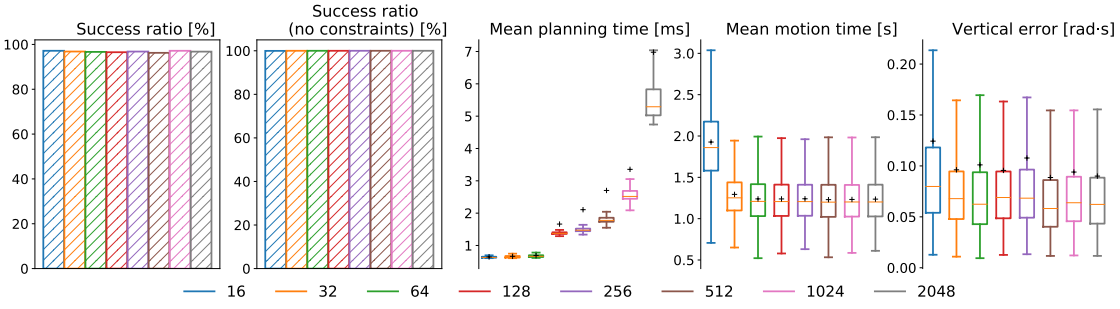

FIGURE 6.12: Comparison of CNP-B with different neural network sizes (numbers of neurons in each layer) in a task of moving a heavy vertically oriented object.

6.5.3 Number of B-spline control points

One of the most important parameters of the proposed method is the number of B-spline control points as it has a direct impact on the expressiveness and flexibility of the solution representation. In Figure 6.13 we present the results of the simulated Air Hockey hitting experiments with different numbers of configuration B-spline control points. We observe that in general, our proposed neural network-based planner is quite robust to the number of control points within a range between 9 and 21. Nevertheless, one can note that reducing the number of control points causes the score ratio and mean puck velocity to decrease slightly, which may be caused by the reduced flexibility of the solution representation. The statistics of deviations from the table plane in the z axis are also quite interesting. It seems that both extreme values lead to slightly increased errors. This may be caused by two different reasons. For 9 control points, by the insufficient expressiveness to plan the movement closer to the constraint manifold, while for 21 points, by the over-parametrization that led to a slightly worse generalization.

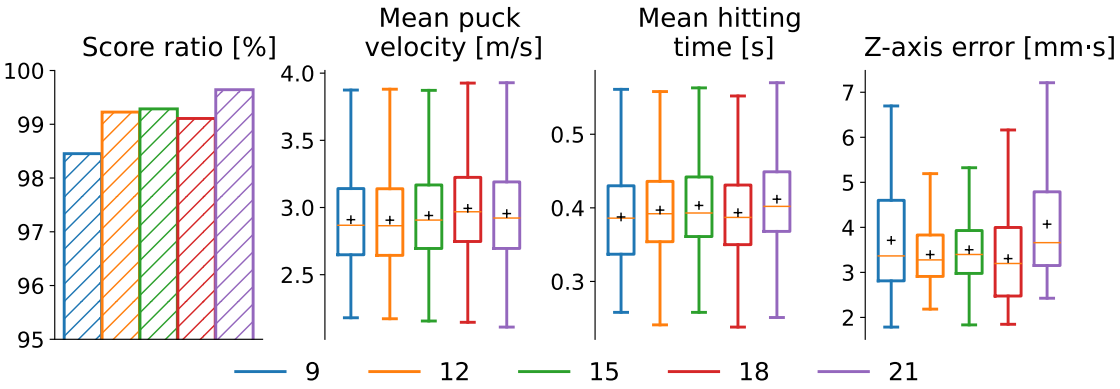

FIGURE 6.13: Comparison of CNP-B performance in simulated Air Hockey hitting for different numbers of B-spline control points.

6.5.4 Generalization abilities

One of the most desired properties of machine learning-based systems is the ability to generalize. We already tested it to some extent, by showing that the proposed planning method is able to solve motion planning problems that were not included in the training dataset. However, in all previous experiments, these test problems belonged to the training data distribution, as the manifold on which we would like to plan was constant for all considered motion planning problem instances. Therefore, to thoroughly test the generalization abilities we need to validate the abilities of CNP-B to plan for the problems from a different data distribution than the one from which the training dataset was drawn. Problems of this kind are likely to happen as they may correspond to a situation where the mass of the object we want to move with the robot is different from the one on which our planner was trained, or the Air Hockey table height has changed. In general, there are two ways of handling these types of situations. The first one, called *domain randomization* [167] focuses on training the machine learning model to be robust to different environmental conditions by training it on the data that comes from a variety of possible environments. While being general, in the sense of not assuming any

knowledge about the test environment, this approach typically trades off the performance for robustness. However, in some cases, we may expect that some of the crucial parameters of the test environment are accessible to the planner, e.g., by the system identification. In this case, we may handle the out-of-distribution test samples by parametrizing our planner with these crucial parameters and training them to include knowledge about them in the planning process. As we don't want to trade off the performance, in our experiments we focus on two aspects of the generalization: (i) the ability to generalize outside the conditions of the training set without any *domain randomization*, and (ii) the possibility to learn how to plan on parametrized manifolds and how it will affect the generalization abilities of CNP-B.

To evaluate the first issue, we tested a neural network-based planner, which was trained to solve the task of moving a vertically oriented object that weighs exactly 12kg (see Section 6.2), on the task of moving lighter and heavier ones. In Figure 6.14 we present the results of this evaluation. As expected, moving objects of smaller mass pose no problem for the proposed planner. However, due to the reduced mass, the generated trajectories are presumably not optimal, as the robot motion with lighter objects could be faster. Nevertheless, the ability to generalize to smaller masses is an important feature of the proposed solution, which is possible because we plan the whole trajectory that is then tracked with a controller. Conversely, typical reinforcement-learning approaches to learning these kinds of skills generate the actions directly in the space of points torques, which may lead to violations of the velocity limits if applied to much lighter objects. In turn, planning for much heavier objects is much more challenging, as the increased mass simply leads to violations of the maximal joint's torques, since the maximal robot's payload is 14kg. This has a major impact on the success rate, by making it impossible to solve some tasks. Infeasible torque commands also lead to inaccurate trajectory tracking, causing collisions with obstacles and increased orientation error. Nevertheless, our method shows reasonably high success rates when the object mass is close to the one used in training.

To further investigate the generalization abilities of the proposed method, and address the second issue that refers to the manifold parametrization, we performed an experiment of simulated Air Hockey hitting with different table heights. In Figure 6.15 we present scoring ratios and errors of maintaining the robot's end-effector on the table plane for a wide range of table heights. The blue color denotes the results for the model that was trained only on the table height equal to 16 cm. It generalizes very well for the heights between 14 and 20 cm, but for greater height differences, the z-axis errors grow significantly. However, it is still able to score for the vast majority of planning problems for a wide range of the considered heights. As a second model, we tested a parametrized one. It was modified slightly to include the height of the table in the input to the neural network and trained on motion planning problems with the tables of heights randomly drawn from the range between 10 and 20 cm (marked with green dashed lines). While in the training range, the score ratio of this model is similar to that obtained by the non-parametrized one, we observe a huge difference outside this region. The parametrized model achieves very low z-axis errors on all heights between -20 and 30 cm showing outstanding generalization properties. Moreover, it achieves a score ratio higher than 75% on almost the whole range of tested heights.

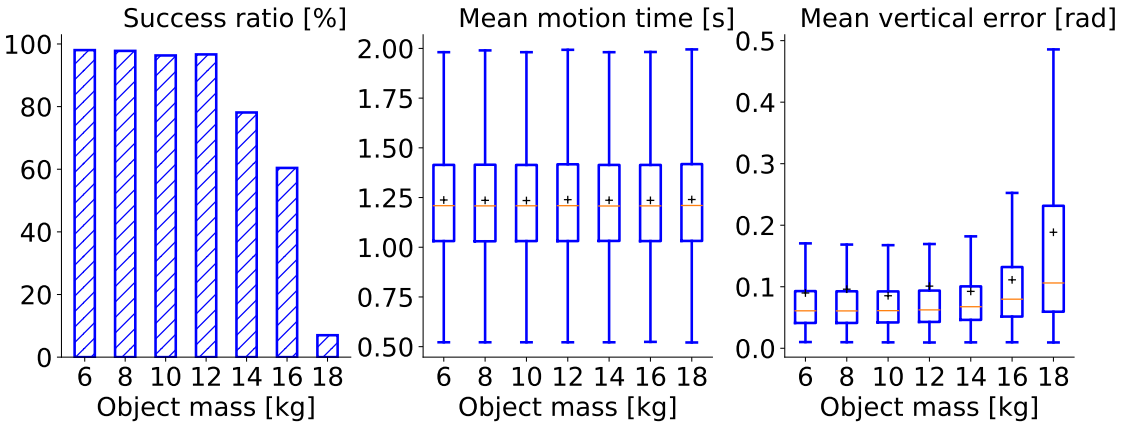

FIGURE 6.14: Analysis of the generalization abilities of CNP-B for different object mass in the task of moving a vertically oriented heavy object.

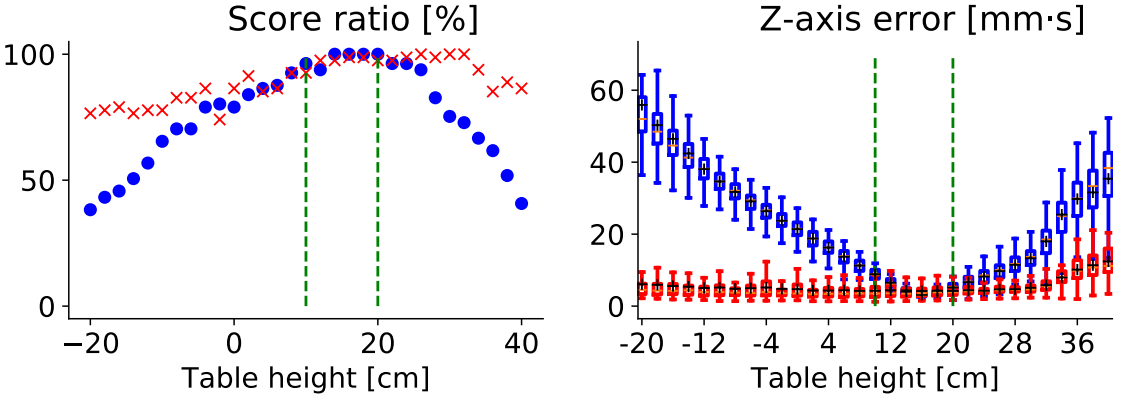

FIGURE 6.15: Analysis of the generalization abilities of CNP-B for different table heights in simulated Air Hockey hitting task. Blue denotes the model trained only on height equal to 16 cm, while the red one is a parametrized model trained on random table height from the range denoted by dashed green lines.

6.6 Discussion

In this section, we conclude the conducted experiments and analyze the strengths and weaknesses of our approach. Our experiments have shown that the proposed methodology effectively outperforms all considered state-of-the-art methods in terms of all considered metrics. We believe that this superb performance can be attributed to three crucial aspects of our approach: (i) the way of handling constraints, (ii) the use of flexible trajectory parameterization, (iii) the learning setup.

First, instead of requiring hard satisfaction of all constraints at all times, we include them in the loss function directly with adaptable weighting factors. This allows us to accept small violations both during training and deployment. While in some tasks, even small trajectory violations can be a major problem, these violations are acceptable for most practical motion planning problems. Unfortunately, our handling of the constraints does not guarantee the planned trajectory will satisfy all the constraints. However, it is fairly easy to verify that a planned trajectory complies with the requirements and abort the plan if necessary. Moreover, in the case of inequality

constraints, during the training phase, one can easily modify the set of constraint-satisfying solutions, such that the learned solutions are pushed towards safety at the price of sub-optimality.

Second, thanks to the introduced B-spline parameterization, we can easily ensure that the plans will always start from the current state, which can be represented by the configuration and its derivatives, and reach the planned one in every trajectory computed by the network. While this property could cause issues in the learning process and plan generation, our B-spline parametrization decouples the geometrical path and the speed of traversing it by learning a spline for the time coordinate. This decoupling gives the proposed planner great flexibility and simplifies its learning process. Using parametrized trajectories simplifies the structure of the neural network, avoiding complex learning procedures required for recurrent networks. Moreover, our proposed method is robust to the choice of the number of B-spline control points, which can be used to control the maximum allowed complexity of the path.

Finally, our learning setup is simple and effective. It does not require any dataset of expert demonstration or tedious estimation of the loss gradient. Instead, it relies on the models of the robot and the environment to compute the differentiable loss function that is used to improve the plans generated by CNP-B. Furthermore, based on the performed generalization experiments we presume that our approach can be generalized to different tasks and settings. In principle, if we provide a sufficiently large set of planning problems, this approach could process all relevant information to solve the task, allowing for example for advanced obstacle avoidance or feasible dynamic movements. Nevertheless, we foresee that there is a huge potential for future research in terms of generating informative planning problems to effectively train versatile skills, limiting the required amounts of computations performed during training.

The most important limitation of the proposed approach is the lack of guarantees that the generated plans do not violate any of the constraints. This can be potentially seen as a drawback w.r.t. other methods, as some of them guarantee that in time going to infinity they will find a solution if one exists. Nevertheless, we argue that from a practical point of view in many robotics applications, it is more important to have a verifiable, but possibly wrong, plan within milliseconds than to have a certifiable planner that needs plenty of time to compute a solution. Moreover, generated solutions, even if infeasible or sub-optimal, can be further improved using optimization techniques, especially because the planning time of our method is negligible w.r.t. to the basic trajectory optimization.

Another important gain from the short planning time, combined with a constant amount of computation and boundary conditions satisfaction thanks to B-spline trajectory representation, is the ability to feasible and smooth replanning on the fly. To the best of our knowledge, it is the first time this kind of dynamic smooth replanning is presented for problems of this complexity, along with a demonstration on a real robot.

Chapter 7

Conclusions

7.1 Summary

This dissertation deals with a reinforcement learning-based approach to local robotic motion planning. We showed that this approach may be successfully applied both for path planning for autonomous cars and trajectory planning for robotic manipulators. Each of these application areas challenges the motion planning algorithms in a different way. In the case of planning for an autonomous car, one should generate paths that are: (i) collision-free, (ii) possible to be followed by a car with a limited steering angle, i.e., with limited curvature, (iii) smooth enough to ensure passenger comfort, (iv) minimizing the overall curvature to facilitate negotiating them at higher speeds without skidding, (v) considering not only the initial position of the vehicle guidance point but also vehicle orientation and steering angle, and (vi) achieve the desired goal precisely. In turn, in the case of robotic manipulation, we focus on the tasks that require the solution trajectory to (i) satisfy the internal constraints of the used robotic platform, such as limited velocities, accelerations, and torques, (ii) satisfy the constraints stemming from the given task, such as the movement of the end-effector limited to some subset of the task space, (iii) optimize some optimality criterion, such as minimization of its duration or energy consumption during its execution, and (iv) allow for imposing the boundary constraints, e.g., to set its beginning and end to arbitrary states, taking into account not only position but also velocity and acceleration. The common requirement in both tasks is to be able to compute the motion plans as fast as possible, to give the robots the ability to adapt to rapidly changing situations and replan the movement to the goal within at most tens of milliseconds. In this dissertation, we showed that the proposed approach to reinforcement learning-based local robotic motion planning addresses all the challenges and requirements listed above.

The core of the proposed approach is the observation that any motion planning algorithm can be viewed as a motion planning function that transforms the representation of the task into the representation of the solution. Our aim was to approximate in a narrow range of its parameters the behavior of a special function of this kind, i.e., *optimal ideal motion planning function* that generates a feasible and optimal, w.r.t. some criterion, motion plans. To do so, we employed

neural networks that directly generate the solution to a considered motion planning problem and are trained using the differentiable loss function that is developed based on the implicit definition of the *optimal ideal motion planning function*, i.e., by defining the features it has to poses in the form of differentiable functions. These two features distinguish our approach from the most popular machine learning-based motion planning algorithms, such as [13, 73, 142], or [81].

In Chapter 2 we introduced a sequential approach to planning a path for a car-like vehicle. We trained a neural network to generate the subsequent segment end-points, such that they can be connected using 5th-degree polynomials and form a solution path. It does so based on the representation of the task that consists of a local map of the environment, and the initial and desired state of the vehicle. We proposed a loss function that is differentiable and penalizes the paths that: (i) do not reach the goal state neighborhood, (ii) violate the maximal allowed curvature constraint, (iii) collide with the environment, and (iv) perform unnecessary turns. Especially hard to obtain is the differentiability of the collisions with the environment. To overcome this difficulty, we proposed to use, during the training phase, the solution path, precomputed with a SL algorithm, as a proxy that attracts the parts of the solution that are colliding with the obstacles. By doing so we ensured that no performance upper bound is put on the paths generated by the proposed method, in contrast to typical learning-based motion planning approaches that utilize behavior cloning [28, 142, 186]. Moreover, we do not have to ensure that the generated reference path is optimal, which significantly reduces the time needed to create a training dataset. To train and evaluate the proposed approach we introduced a dataset of 115319 training, 11008 validation, and 8128 testing motion planning problems, which utilize maps generated from the maps from CARLA simulator [38] and the ones created using real LiDAR scans and LOAM algorithm [187]. The experiments performed on this dataset have shown that the proposed approach is able to generate, in almost constant time of 43 ms, paths that in 92.24% of the test motion planning problems are feasible. It outperforms state-of-the-art planners like BIT*, AIT*, and ABIT*, even though they were solving a simplified problem of searching for a path in a Dubin's state space.

In Chapter 3 we proposed a different way to see the motion planning problems from a learning-based perspective. Instead of considering sequential decision-making and building the solution path from segments, we reframed the path planning problem for a car-like vehicle to the contextual bandits, which even better suits our proposition of treating the motion planning algorithm as a planning function because of the lack of sequentiality. Thanks to this and the utilized B-spline path representation, we were able to generate the solution path in a single inference of the neural network, which resulted in the decrease of the planning time to 11 ms. To achieve this and enable efficient training, we proposed a novel way of interpreting the neural network outputs and constructing the solution path. This new representation, despite allowing for computing the path within a single forward pass through the neural network, introduces several advantages to the motion planner that utilizes it. It enables enforcing boundary conditions, such that we can guarantee that the solution path will reach the goal precisely, not only in terms of the position but also in higher-order derivatives. Moreover, the proposed way of constructing the subsequent control points from the neural network output introduces an inductive bias such that a randomly initialized neural network generates intuitive paths to the goal configuration, which, together with a simplified loss function that no longer needs to encourage reaching the

goal, significantly speeds up the training. The experiments performed on the test set introduced in Chapter 2 have shown that our approach that utilizes B-spline paths reaches an accuracy of 90.1%, which in fact is a result slightly worse than the sequential approach introduced in Chapter 2. However, if we reduce the size of the set of allowed end configurations, the innately precise approach from Chapter 3 outperforms the sequential one. Moreover, the generated solutions have a higher level of smoothness and are characterized by significantly lower maximal path curvature, which makes these paths easier to follow with higher velocities.

To further compare the proposed neural network-based motion planning approaches we performed several experiments in the CARLA simulator, which extends the test of the motion planning algorithms on the dataset to a more realistic setting. In these experiments, we tested the abilities of these planners to online plan the path to perform typical maneuvers, such as perpendicular, diagonal, and parallel parking, turning, passing densely parked vehicles, and avoiding a collision with a vehicle that suddenly crosses the path of the ego-vehicle. In general, the paths generated by the non-sequential neural network-based path planning algorithm were significantly easier to follow, which manifested in more than 3 times lower position and orientation tracking errors. Moreover, the proposed B-spline path planning approach allowed for a more robust execution of the parallel parking scenario with higher velocities than the proposed sequential planner. Furthermore, the tests on diagonal parking task with an extremely narrow parking spot showed that imposing the boundary conditions on the solution allowed for very precise parking with only a few centimeters of slack. Finally, we showed that thanks to the very short planning time the B-spline-based neural network motion planner allows for rapid reactions to the sudden lane intrusion and collision avoidance while moving at $15\,\frac{\mathrm{m}}{\mathrm{s}}$.

In turn, in Chapter 4 we focused on planning dynamic trajectories for the robotic manipulator under different types of constraints including kinodynamic ones. Similarly, as for the path planning, we used a neural network to generate a B-spline path in the joint space, however, to obtain a trajectory the same network also generates a B-spline that represents the rate of change of the phase variable. The proposed solution representation allows for generating variable-length smooth trajectories and enables imposing the boundary conditions not only on the configuration but also on its derivatives w.r.t. time. The latter property enables planning for the tasks that require, e.g., reaching a certain velocity at the end of the movement or which begins from non-zero velocities and accelerations. Thanks to this, and due to the constant amount of computations needed to generate a trajectory, one can use the proposed approach to replan the dynamic motion on-the-fly and seamlessly connect the new plan to the old one ensuring the continuity of the velocity and control signals. Our focus was not only on the satisfaction of the boundary constraints but also on the ones that were imposed on the whole trajectory. In this dissertation, we considered the limits of the manipulator joint's velocities, accelerations, and torques, as well as limiting the movement of the end-effector to a plane in the workspace, or fixing its orientation during the movement. To learn how to plan trajectories that satisfy these constraints we introduced a formulation of the constraint manifold and proposed a learning procedure to update the neural network weights in a way that optimizes them w.r.t. some optimality criterion and at the same time minimizes a distance to the aforementioned constraint manifold.As a result, we proposed an interleaving neural network learning and adaptation of the constraint manifold metric to ensure that the constraints will not exceed the violation budget. All of these allow us to

solve very challenging motion-planning tasks, such as planning the fast collision-free movement of a heavy (close to the robot's maximal payload) object while maintaining its vertical orientation throughout the whole movement, and high-velocity hitting movement in the game of robotic Air Hockey, with reaching some predefined end joint's velocity and maintaining the position of the end-effector on the table plane. These examples showed not only an overwhelming dominance of the proposed method over both classical motion planning approaches and machine learning-based ones, in terms of task success ratio, motion planning time, planned motion time, and constraint violations. Moreover, we showed its ability to rapidly (within several milliseconds) and successfully plan and replan the movements of the real robot. We presented these abilities in the Air Hockey hitting task from the steady puck but also by performing some trick shots, such as feinting the opponent and suddenly scoring, and bouncing the puck against the wall, and then hitting from the moving puck. Finally, we analyzed the generalization abilities of the proposed motion planning method and showed in the example of the Air Hockey table height that it can successfully plan for a wide range of the constraint manifold parameter.

7.2 Conclusions and thesis contribution

The concepts presented in the dissertation and their experimental verification make several contributions to the current state of the art in robotics and machine learning. The main contributions may be summarized as:

1. **A novel approach for rapid path generation under nonholonomic constraints by approximating the implicitly defined optimal ideal planning function using a neural network.** We proposed a novel concept of learning how to plan the motion of a car-like vehicle with nonholonomic constraints that emerges from the idea of approximating *optimal ideal planning function* using a neural network. The experiments suggest that this approach is more effective in rapid motion planning for a car-like vehicle than state-of-the-art motion planners even if they are allowed to plan in a simplified setting.

2. **A novel differentiable loss function which penalizes infeasible paths, since they violate constraints imposed by the vehicle kinematics and environment maps.** To train the proposed neural network-based motion planner we propose a novel differentiable loss function. Each of the features that *optimal ideal planning function* should have is defined as a differentiable loss function that zeroes out in the optimal case. By doing so, we can train a neural network to plan feasible paths in an efficient way, thanks to the analytic form of the loss function gradient, without imposing on its performance any upper bound stemming from the data as is the case with behavior cloning. Due to this, we can speed up the data generation procedure as we no longer need to generate optimal feasible reference paths.

3. **A B-spline path parametrization and a novel procedure of its construction using neural network outputs.** Instead of building the solution path in a sequential way, we propose to learn how to infer the whole solution at once, which is crucial for speeding up planning. To do so, we proposed to use a uniform B-spline path representation and propose

a novel procedure for its construction, which interprets the subsequent neural network outputs as the B-spline control points in a recursive way, using already defined ones. This approach introduces several important advantages over the previous one, such as (i) it allows for imposing the boundary constraints on the generated path, (ii) it introduces an inductive bias to the proposed motion planner, which significantly speeds up the training, and (iii) it produces smoother paths, with lower maximal path curvature, which allows one to traverse the path faster or with lower centrifugal forces, which affects the ride comfort. In the experiments, we show that thanks to the proposed representation enables one to learn faster and that the planned motions are smoother, of lower curvature, and more accurate.

4. **A new machine learning-based approach to robotic arm motion planning that enables planning dynamic trajectories under constraints.** We proposed a new approach to learn how to plan dynamic trajectories that minimize some predefined optimality criterion while satisfying multiple constraints. We formalize these constraints as a constraint manifold and extend the learning system architecture to learn Lagrangian multipliers in the optimization problem, learning simultaneously a metric of our constraint manifold. This novel approach allows us to weigh each constraint by how much it is important to find a feasible solution to the planning problem. We demonstrate in simulations and experiments that this new approach is not only much faster than all state-of-the-art methods we were able to compare as baselines, but also generates trajectories that allow faster and more accurate robot motion while being executed.

5. **A new B-spline-based trajectory representation and a technique to enforce satisfaction of the boundary constraints.** We proposed a new B-spline-based trajectory representation that consists of two B-spline curves, one that defines the solution path as a function of the phase variable, and another one that defines the rate of change of the phase variable w.r.t. time. Using these two functions, one can compute the arbitrary time derivatives of the trajectory, and therefore it is possible to enforce the satisfaction of the boundary constraints that include not only the configurations but also their derivatives w.r.t. time, which allows to connect precisely two arbitrary robot states. The performed experiments confirmed that the proposed representation allows for the effective solving of complex motion-planning problems that require precise and rapid motions.

6. **An ability of the learning-based motion planner to replan on-the-fly motion plans that seamlessly connect with the currently executed trajectory.** Thanks to the proposed B-spline-based trajectory representation, which is able to satisfy arbitrary boundary constraints, and due to the ability of the proposed neural network-based planner to compute the trajectory within a single neural network inference, i.e., in a constant time, we are able to replan the motion of the robot on-the-fly. Specifically, thanks to the small constant planning time one can accurately anticipate what will be the robot configuration in several milliseconds, and due to the boundary constraints satisfaction, plan the new motion right from this state. this allows for a seamless connection with the currently executed trajectory and ensures the continuity of the velocities and torques.

When considering the supportive hypotheses stated at the beginning of the dissertation, it is possible to conclude that:

1. The experiments presented in Sections 5.2.1 and 5.3.2, and particularly in Table 5.1, support the first supportive hypothesis, i.e., "Formalization of the path planning problem for a car-like vehicle as the MDP allows for sequentially building the solution using the sub-paths generated with a neural network trained using reinforcement learning".

2. Sections 5.2.1, 5.3.2 and 5.2.2, and particularly results presented in Table 5.2 and Figure 5.2, support the second supportive hypothesis, i.e., "The B-spline path representation and the proposed path construction method allows for efficient planning within a single inference of the neural network and introduces an inductive bias to the learning process".

3. The experiments presented in Sections 6.2.3, 6.3.3, and 6.4.1, and particularly in Figures 6.4, 6.6, and 6.8, support the third supportive hypothesis, i.e., "Solving constrained kinodynamic motion planning problems is possible with the use of neural networks trained under reinforcement learning paradigm and B-spline-based trajectory representation".

   The supportive hypotheses are proven experimentally, therefore it is possible to affirm the veracity of the main hypothesis of the dissertation.

7.3 Limitations

Every method and approach has some limitations and the ones proposed in this dissertation are no exception. Thus in this section, we want to list and discuss the most prominent limitations of the introduced approaches:

(a) The proposed approach addresses the problem of local motion planning and thus is not intended to be a standalone global planner. However, we suppose that it could be successfully used by some global planners as a subroutine that is efficient at solving subtasks thanks to the utilization of the experience. In this dissertation, we focused on planning the motion of the robot, not how to solve the whole task. We rather want to equip the robot with agility in performing basic movements and by doing so enable the robot to schedule them in order to complete high-level tasks, which can be done either algorithmically or used by a higher-level learning algorithm to learn how to conduct a task, without the need to specify low-level commands such as joint's torques.

(b) The proposed machine learning-based approach to motion planning gives no warranties about the feasibility of the generated plan. However, this problem is greatly mitigated by the fact that checking the feasibility of the plan is often faster than generating a feasible trajectory, which allows for preventing the execution of dangerous motions [169]. Moreover, from our empirical results, our planner has a much higher success rate than the other state-of-the-art planners on all the tasks considered in this dissertation for a limited computation time.

(c) In this work, we focused on describing the behavior of the *optimal ideal planning function* using the differentiable loss functions, which not always may be possible. However, in principle, it is possible to extend this approach by using estimated derivatives and turning the problem into a typical Reinforcement Learning one in which derivatives of the objective w.r.t. policy parameters are not possible to be computed analytically.

(d) In this dissertation, we did not consider planning for dynamic obstacles. However, one may presume that an approach that allows for planning trajectories very quickly may be extended to handle this problem.

(e) Similarly to most planning algorithms, we assume perfect knowledge of system dynamics such that the model mismatch is handled by the tracking algorithm only.

(f) In the case of car-like vehicles, we showed the path planning methods that neglect the car dynamics which is definitely present in the real system. This may excessively limit the maximum feasible velocity along the planned paths.

(g) In the case of non-convex obstacle avoidance, the proposed method may in principle not be able to generate solutions in a different homotopy class than the one to which the reference path belongs.

7.4 Future work

Taking into account the limitations mentioned above, one can propose natural paths for future research in the area of machine learning for motion planning. First, it would be interesting to confirm that the proposed planning module can be used by a higher-level reinforcement learning agent or motion planning algorithm, to generate a low-level motion after being trained to solve local motion planning problems. The use of high-quality low-level skills planning may facilitate the learning or planning of complex behaviors, such as playing an Air Hockey game or car racing. Secondly, to fully take advantage of rapid planning in the case of car-like vehicles one may consider planning trajectories instead of paths and using more complex models of the car dynamics to enable accurate planning for higher velocities and sharper maneuvers. Thirdly, a very important research direction would be to incorporate the dynamically moving obstacles into the considered motion planning problems, as this kind of situation is extremely common in mobile robotics. Moreover, as we presented the solutions only for two types of robots, i.e., car-like and manipulators, it would be interesting to validate whether the proposed methods can be generalized for planning also for other types, such as drones, or walking robots. Furthermore, in this dissertation, we focused our attention on the *optimal ideal planning functions* which features may be described implicitly using differentiable functions that assess their geometry. However, this in general does not have to be always possible, e.g., if we cannot use reference solutions for non-convex collision avoidance. Therefore it would be interesting to check if similar motion planning behaviors are possible to be learned using typical Reinforcement Learning approaches that estimate the gradient of the loss function, instead of using the analytical one. Finally, one of the most important shortcomings of the proposed approach

to motion planning is the fact that it can generate infeasible solutions. An interesting way to address this issue is to employ standard optimization techniques to fix trajectories that slightly violate the constraints. This direction of research is in line with the recently proposed amortized optimization [3], which proposes to use learning-based techniques to improve the efficiency of solving optimization problems, and a more mature idea of *trajectory prediction* [78], which propose to utilize the experience for more informed initialization of trajectory optimization. In fact, one can view machine learning-based motion planning as a subroutine of generating high-quality initial guesses for the optimization-based motion planner [8, 78].

Besides addressing the current limitations of the proposed approach, future work can consider the adaptation of the proposed methods to work outside the man-made environments. This seems doable, as we already have shown that using LiDAR-based 2D occupancy grid one can learn how to avoid obstacles, however, adapting these methods to work with high-dimensional measurements of the unstructured environment and high-dimensional environment representations may be an interesting challenge.

Bibliography

[1] Kia Rio III specification. https://autodata24.com/kia/rio/rio-iii-sedan/details. Accessed: 2023-09-12.

[2] Saminda Wishwajith Abeyruwan, Laura Graesser, David B D'Ambrosio, Avi Singh, Anish Shankar, Alex Bewley, Deepali Jain, Krzysztof Marcin Choromanski, and Pannag R Sanketi. i-sim2real: Reinforcement learning of robotic policies in tight human-robot interaction loops. In Karen Liu, Dana Kulic, and Jeff Ichnowski, editors, *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pages 212–224. PMLR, 14–18 Dec 2023. URL https://proceedings.mlr.press/v205/abeyruwan23a.html.

[3] Brandon Amos. Tutorial on amortized optimization for learning to optimize over continuous domains. *CoRR*, abs/2202.00665, 2022. URL https://arxiv.org/abs/2202.00665.

[4] Tomoki Ando, Hiroto Iino, Hiroki Mori, Ryota Torishima, Kuniyuki Takahashi, Shoichiro Yamaguchi, Daisuke Okanohara, and Tetsuya Ogata. Learning-based collision-free planning on arbitrary optimization criteria in the latent space through cgans. *Advanced Robotics*, 37(10):621–633, 2023. doi: 10.1080/01691864.2023.2180327. URL https://doi.org/10.1080/01691864.2023.2180327.

[5] Szilárd Aradi. Survey of deep reinforcement learning for motion planning of autonomous vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 23(2): 740–759, 2022. doi: 10.1109/TITS.2020.3024655.

[6] Pranav Atreya and Joydeep Biswa. State supervised steering function for sampling-based kinodynamic planning. In *2022 International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 35–43, 2022.

[7] Baidu. Apolloauto. https://github.com/ApolloAuto/apollo, 2023.

[8] Somrita Banerjee, Thomas Lew, Riccardo Bonalli, Abdulaziz Alfaadhel, Ibrahim Abdulaziz Alomar, Hesham M Shageer, and Marco Pavone. Learning-based warm-starting for fast sequential convex programming and trajectory optimization. In *2020 IEEE Aerospace Conference*, pages 1–8, 2020. doi: 10.1109/AERO47225.2020.9172293.

[9] Mayank Bansal, Alex Krizhevsky, and Abhijit Ogale. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. In *Proceedings of Robotics: Science and Systems*, FreiburgimBreisgau, Germany, June 2019. doi: 10.15607/RSS.2019.XV.031.

[10] D. Belter, P. Labęcki, and P. Skrzypczyński. Estimating terrain elevation maps from sparse and uncertain multi-sensor data. In *IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 715–722, 2012.

[11] Atallah Benalia, Mohamed Djemai, and J-P Barbot. Control of the kinematic car using trajectory generation and the high order sliding mode control. In *SMC'03 Conference Proceedings. 2003 IEEE International Conference on Systems, Man and Cybernetics. Conference Theme-System Security and Assurance (Cat. No. 03CH37483)*, volume 3, pages 2455–2460. IEEE, 2003.

[12] Dmitry Berenson, Siddhartha S. Srinivasa, Dave Ferguson, and James J. Kuffner. Manipulation planning on constraint manifolds. In *2009 IEEE International Conference on Robotics and Automation*, pages 625–632, 2009. doi: 10.1109/ROBOT.2009.5152399.

[13] Dmitry Berenson, Pieter Abbeel, and Ken Goldberg. A robot path planning framework that learns from experience. In *2012 IEEE International Conference on Robotics and Automation*, pages 3671–3678, 2012. doi: 10.1109/ICRA.2012.6224742.

[14] D.P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 1999.

[15] Riccardo Bonalli, Abhishek Cauligi, Andrew Bylard, Thomas Lew, and Marco Pavone. Trajectory optimization on manifolds: A theoretically-guaranteed embedded sequential convex programming approach. In *Proceedings of Robotics: Science and Systems*, Freiburg, Germany, June 2019. doi: 10.15607/RSS.2019.XV.078.

[16] Manuel Bonilla, Edoardo Farnioli, L. Pallottino, and Antonio Bicchi. Sample-based motion planning for robot manipulators with closed kinematic chains. *Proceedings - IEEE International Conference on Robotics and Automation*, 2015:2522–2527, 06 2015. doi: 10.1109/ICRA.2015.7139537.

[17] Manuel Bonilla, Lucia Pallottino, and Antonio Bicchi. Noninteracting constrained motion planning and control for robot manipulators. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4038–4043, 2017. doi: 10.1109/ICRA.2017.7989463.

[18] Ricard Bordalba, Lluís Ros, and Josep M. Porta. A randomized kinodynamic planner for closed-chain robotic systems. *IEEE Transactions on Robotics*, 37(1):99–115, 2021. doi: 10.1109/TRO.2020.3010628.

[19] Matthew Botvinick and Marc Toussaint. Planning as inference. *Trends in Cognitive Sciences*, 16(10):485–488, 2012. ISSN 1364-6613. doi: https://doi.org/10.1016/j.tics.2012.08.006. URL `https://www.sciencedirect.com/science/article/pii/S1364661312001957`.

[20] Djallel Bouneffouf, Irina Rish, and Charu Aggarwal. Survey on applications of multi-armed and contextual bandits. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, 2020. doi: 10.1109/CEC48606.2020.9185782.

[21] Dieter Büchler, Simon Guist, Roberto Calandra, Vincent Berenz, Bernhard Schölkopf, and Jan Peters. Learning to play table tennis from scratch using muscular robots. *IEEE Transactions on Robotics*, 2022.

[22] Martin Buehler, Karl Iagnemma, and Sanjiv Singh. *The DARPA urban challenge: autonomous vehicles in city traffic*, volume 56. Springer, 2009.

[23] Justin Carpentier, Guilhem Saurel, Gabriele Buondonno, Joseph Mirabel, Florent Lamiraux, Olivier Stasse, and Nicolas Mansard. The Pinocchio c++ library – a fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives. In *IEEE International Symposium on System Integrations (SII)*, 2019.

[24] Massimo Cefalo and Giuseppe Oriolo. Dynamically feasible task-constrained motion planning with moving obstacles. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2045–2050, 2014. doi: 10.1109/ICRA.2014.6907130.

[25] Binghong Chen, Bo Dai, Qinjie Lin, Guo Ye, Han Liu, and Le Song. Learning to plan in high dimensions via neural exploration-exploitation trees. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

[26] Jianyu Chen, Wei Zhan, and Masayoshi Tomizuka. Autonomous driving motion planning with constrained iterative lqr. *IEEE Transactions on Intelligent Vehicles*, 4(2):244–254, 2019. doi: 10.1109/TIV.2019.2904385.

[27] Yao-Chon Chen. Solving robot trajectory planning problems with uniform cubic b-splines. *Optimal Control Applications and Methods*, 12(4):247–262, 1991. doi: https://doi.org/10.1002/oca.4660120404. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/oca.4660120404`.

[28] Richard Cheng, Krishna Shankar, and Joel W. Burdick. Learning an optimal sampling distribution for efficient motion planning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7485–7492, 2020. doi: 10.1109/IROS45743.2020.9341245.

[29] Suyoung Choi, Gwanghyeon Ji, Jeongsoo Park, Hyeongjun Kim, Juhyeok Mun, Jeong Hyun Lee, and Jemin Hwangbo. Learning quadrupedal locomotion on deformable terrain. *Science Robotics*, 8(74):eade2256, 2023. doi: 10.1126/scirobotics.ade2256. URL `https://www.science.org/doi/abs/10.1126/scirobotics.ade2256`.

[30] Laurène Claussmann, Marc Revilloud, Dominique Gruyer, and Sébastien Glaser. A review of motion planning for highway autonomous driving. *IEEE Transactions on Intelligent Transportation Systems*, 21(5):1826–1848, 2020. doi: 10.1109/TITS.2019.2913998.

[31] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009. ISBN 0262033844.

[32] Susan Craw. Manhattan distance. In Claude Sammut and Geoffrey I. Webb, editors, *Encyclopedia of Machine Learning and Data Mining*, pages 790–791, Boston, 2017. Springer.

[33] Tung Dang, Marco Tranzatto, Shehryar Khattak, Frank Mascarich, Kostas Alexis, and Marco Hutter. Graph-based subterranean exploration path planning using aerial and legged robots. *Journal of Field Robotics*, 37(8):1363–1388, 2020. doi: https://doi.org/10.1002/rob.21993.

[34] Nikhil Das and Michael Yip. Learning-based proxy collision detection for robot motion planning applications. *IEEE Transactions on Robotics*, 36(4):1096–1114, 2020. doi: 10.1109/TRO.2020.2974094.

[35] Carl de Boor. *A Practical Guide to Splines*. Springer Verlag, New York, 1978.

[36] Rowan Dempster, Mohammad Al-Sharman, Derek Rayside, and William Melek. Real-time unified trajectory planning and optimal control for urban autonomous driving under static and dynamic obstacle constraints. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10139–10145, 2023. doi: 10.1109/ICRA48891.2023.10160577.

[37] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. Path planning for autonomous vehicles in unknown semi-structured environments. *The International Journal of Robotics Research*, 29(5):485–501, 2010. doi: 10.1177/0278364909359210. URL `https://doi.org/10.1177/0278364909359210`.

[38] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[39] Anca D. Dragan, Nathan D. Ratliff, and Siddhartha S. Srinivasa. Manipulation planning with goal sets using constrained trajectory optimization. In *2011 IEEE International Conference on Robotics and Automation*, pages 4582–4588, 2011. doi: 10.1109/ICRA.2011.5980538.

[40] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. In M. F. Balcan and K. Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1329–1338, New York, USA, 2016. PMLR.

[41] Lester E. Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, 79:497, 1957.

[42] Boston Dynamics. Picking up momentum, 2023. URL `https://bostondynamics.com/blog/picking-up-momentum/`.

[43] Roy Featherstone. *Rigid body dynamics algorithms*. Springer, 2014.

[44] Árpád Fehér, Szilárd Aradi, Ferenc Hegedüs, Tamás Bécsi, and Péter Gáspár. Hybrid ddpg approach for vehicle motion planning. In *International Conference on Informatics in Control, Automation and Robotics*, 2019. URL `https://api.semanticscholar.org/CorpusID:202094527`.

[45] Adam Fishman, Adithyavairavan Murali, Clemens Eppner, Bryan Peele, Byron Boots, and Dieter Fox. Motion policy networks. In *Proceedings of the 6th Conference on Robot Learning (CoRL)*, 2022.

[46] MICHEL FLIESS, JEAN LÉVINE, PHILIPPE MARTIN, and PIERRE ROUCHON. Flatness and defect of non-linear systems: introductory theory and examples. *International Journal of Control*, 61(6):1327–1361, 1995. doi: 10.1080/00207179508921959. URL `https://doi.org/10.1080/00207179508921959`.

[47] Autoware Foundation. Autoware universe. `https://github.com/autowarefoundation/autoware.universe`, 2023.

[48] T. Fraichard and A. Scheuer. From Reeds and Shepp's to continuous-curvature paths. *IEEE Transactions on Robotics*, 20(6):1025–1035, 2004.

[49] 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. In *IEEE International Conference on Robotics and Automation*, pages 3067–3074, 2015.

[50] Jonathan D. Gammell, Siddhartha S. Srinivasa, and Timothy D. Barfoot. Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2997–3004, 2014. doi: 10.1109/IROS.2014.6942976.

[51] Jonathan D Gammell, Timothy D Barfoot, and Siddhartha S Srinivasa. Batch informed trees (BIT*): Informed asymptotically optimal anytime search. *The International Journal of Robotics Research*, 39(5):543–567, 2020. doi: 10.1177/0278364919890396.

[52] Zhiqiang Gao, Yi Huang, and Jingqing Han. An alternative paradigm for control system design. In *Proceedings of the 40th IEEE Conference on Decision and Control (Cat. No.01CH37228)*, volume 5, pages 4578–4585 vol.5, 2001. doi: 10.1109/CDC.2001.980926.

[53] Tomasz Gawron and Maciej Marcin Michałek. A g3-continuous extend procedure for path planning of mobile robots with limited motion curvature and state constraints. *Applied Sciences*, 8(11), 2018. ISSN 2076-3417. doi: 10.3390/app8112127. URL `https://www.mdpi.com/2076-3417/8/11/2127`.

[54] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the KITTI vision benchmark suite. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361, 2012.

[55] D. González, J. Pírez, V. Milanís, and F. Nashashibi. A review of motion planning techniques for automated vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 17(4):1135–1145, 2016.

[56] Steven N. Goodman, Daniele Fanelli, and John P. A. Ioannidis. What does research reproducibility mean? *Science Translational Medicine*, 8(341):341ps12–341ps12, 2016. doi: 10.1126/scitranslmed.aaf5027. URL `https://www.science.org/doi/abs/10.1126/scitranslmed.aaf5027`.

[57] J. A. Groeger. *Understanding Driving: Applying Cognitive Psychology to a Complex Everyday Task*. Psychology Press, East Sussex, 2000.

[58] Tianyu Gu, Jarrod Snider, John M. Dolan, and Jin-woo Lee. Focused trajectory planning for autonomous on-road driving. In *2013 IEEE Intelligent Vehicles Symposium (IV)*, pages 547–552, 2013. doi: 10.1109/IVS.2013.6629524.

[59] Corrado Guarino Lo Bianco and Oscar Gerelli. Generation of paths with minimum curvature derivative with η^3 -splines. *IEEE Transactions on Automation Science and Engineering*, 7(2):249–256, 2010. doi: 10.1109/TASE.2009.2023206.

[60] Jingqing Han. From pid to active disturbance rejection control. *IEEE Transactions on Industrial Electronics*, 56(3):900–906, 2009. doi: 10.1109/TIE.2008.2011621.

[61] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968. doi: 10.1109/TSSC.1968.300136.

[62] Eric Heiden, Luigi Palmieri, Leonard Bruns, Kai O. Arras, Gaurav S. Sukhatme, and Sven Koenig. Bench-MR: A motion planning benchmark for wheeled mobile robots. *IEEE Robotics and Automation Letters*, 6(3):4536–4543, 2021.

[63] Kersten Heineke, Philipp Kampshoff, Armen Mkrtchyan, and Emily Shao. Self-driving car technology: When will the robots hit the road?, 2017. URL `https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/self-driving-car-technology-when-will-the-robots-hit-the-road`.

[64] Matteo Hessel, Hado van Hasselt, Joseph Modayil, and David Silver. On inductive biases in deep reinforcement learning, 2019.

[65] David Hoeller, Lorenz Wellhausen, Farbod Farshidian, and Marco Hutter. Learning a state representation and navigation in cluttered and dynamic environments. *IEEE Robotics and Automation Letters*, 6(3):5081–5088, 2021. doi: 10.1109/LRA.2021.3068639.

[66] Taylor A. Howell, Brian E. Jackson, and Zachary Manchester. ALTRO: A fast solver for constrained trajectory optimization. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7674–7679, 2019. doi: 10.1109/IROS40897.2019.8967788.

[67] Jinwook Huh and Daniel D. Lee. Efficient sampling with q-learning to guide rapidly exploring random trees. *IEEE Robotics and Automation Letters*, 3(4):3868–3875, 2018. doi: 10.1109/LRA.2018.2856927.

[68] Jinwook Huh, Daniel D. Lee, and Volkan Isler. Learning continuous cost-to-go functions for non-holonomic systems. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5772–5779, 2021. doi: 10.1109/IROS51168.2021.9636139.

[69] Chia-Man Hung, Shaohong Zhong, Walter Goodwin, Oiwi Parker Jones, Martin Engelcke, Ioannis Havoutis, and Ingmar Posner. Reaching through latent space: From joint statistics to path planning in manipulation. *IEEE Robotics and Automation Letters*, pages 1–1, 2022. doi: 10.1109/LRA.2022.3152697.

[70] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Comput. Surv.*, 50(2), 2017.

[71] Sebastian Höfer, Kostas Bekris, Ankur Handa, Juan Higuera, Florian Golemo, Melissa Mozifian, Chris Atkeson, Dieter Fox, Ken Goldberg, John Leonard, C. Liu, Jan Peters, Shuran Song, Peter Welinder, and Martha White. Perspectives on sim2real transfer for robotics: A summary of the r:ss 2020 workshop, 12 2020.

[72] Brian Ichter and Marco Pavone. Robot motion planning in learned latent spaces. *IEEE Robotics and Automation Letters*, 4(3):2407–2414, 2019. doi: 10.1109/LRA.2019.2901898.

[73] Brian Ichter, James Harrison, and Marco Pavone. Learning sampling distributions for robot motion planning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7087–7094, 2018. doi: 10.1109/ICRA.2018.8460730.

[74] David Isele, Reza Rahimi, Akansel Cosgun, Kaushik Subramanian, and Kikuo Fujimura. Navigating occluded intersections with autonomous vehicles using deep reinforcement learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2034–2039, 2018. doi: 10.1109/ICRA.2018.8461233.

[75] Fahad Islam, Venkatraman Narayanan, and Maxim Likhachev. Dynamic multi-heuristic a*. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2376–2382, 2015. doi: 10.1109/ICRA.2015.7139515.

[76] Fahad Islam, Oren Salzman, Aditya Agarwal, and Maxim Likhachev. Provably constant-time planning and replanning for real-time grasping objects off a conveyor belt. *The International Journal of Robotics Research*, 40(12-14):1370–1384, 2021. doi: 10.1177/02783649211027194. URL `https://doi.org/10.1177/02783649211027194`.

[77] Léonard Jaillet and Josep M. Porta. Path planning under kinematic constraints by rapidly exploring manifolds. *IEEE Transactions on Robotics*, 29(1):105–117, 2013. doi: 10.1109/TRO.2012.2222272.

[78] Nikolay Jetchev and Marc Toussaint. Fast motion planning from experience: trajectory prediction for speeding up movement generation. *Autonomous Robots*, 34 (1):111–127, Jan 2013. ISSN 1573-7527. doi: 10.1007/s10514-012-9315-y. URL `https://doi.org/10.1007/s10514-012-9315-y`.

[79] Kichun Jo, Junsoo Kim, Dongchul Kim, Chulhoon Jang, and Myoungho Sunwoo. Development of autonomous car—part ii: A case study on the implementation of an autonomous driving system based on distributed architecture. *IEEE Transactions on Industrial Electronics*, 62(8):5119–5132, 2015. doi: 10.1109/TIE.2015.2410258.

[80] Jacob J. Johnson, Linjun Li, Fei Liu, Ahmed H. Qureshi, and Michael C. Yip. Dynamically constrained motion planning networks for non-holonomic robots. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6937–6943, 2020. doi: 10.1109/IROS45743.2020.9341283.

[81] Tom Jurgenson and Aviv Tamar. Harnessing reinforcement learning for neural motion planning. In *Proceedings of Robotics: Science and Systems*, FreiburgimBreisgau, Germany, June 2019. doi: 10.15607/RSS.2019.XV.026.

[82] S. Karaman and E. Frazzoli. Sampling-based optimal motion planning for non-holonomic dynamical systems. In *IEEE International Conference on Robotics and Automation*, pages 5041–5047, Karlsruhe, 2013.

[83] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011. doi: 10.1177/0278364911406761.

[84] Christos Katrakazas, Mohammed Quddus, Wen-Hua Chen, and Lipika Deka. Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions. *Transportation Research Part C: Emerging Technologies*, 60:416–442, 2015. ISSN 0968-090X. doi: https://doi.org/10.1016/j.trc.2015.09.011. URL `https://www.sciencedirect.com/science/article/pii/S0968090X15003447`.

[85] L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996. doi: 10.1109/70.508439.

[86] Mitsuo Kawato, Francesca Gandolfo, Hiroaki Gomi, and Yasuhiro Wada. Teaching by showing in kendama based on optimization principle. In *International Conference on Artificial Neural Networks*, pages 601–606. Springer, 1994.

[87] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, volume 2, pages 500–505, 1985. doi: 10.1109/ROBOT.1985.1087247.

[88] Piotr Kicki, Tomasz Gawron, Krzysztof Ćwian, Mete Ozay, and Piotr Skrzypczyński. Learning from experience for rapid generation of local car maneuvers. *Engineering Applications of Artificial Intelligence*, 105:104399, 2021.

[89] Beobkyoon Kim, Terry Um, Chansu Suh, and F. Park. Tangent bundle RRT: A randomized algorithm for constrained motion planning. *Robotica*, 34:202–225, 01 2016. doi: 10.1017/S0263574714001234.

[90] Jinho Kim, Byung-Soo Kim, and Silvio Savarese. Comparing image classification methods: K-nearest-neighbor and support-vector-machines. In *Proceedings of the 6th WSEAS International Conference on Computer Engineering and Applications, and Proceedings of the 2012 American Conference on Applied Mathematics*, AMERICAN-MATH'12/CEA'12, page 133–138, Stevens Point, Wisconsin, USA, 2012. World Scientific and Engineering Academy and Society (WSEAS). ISBN 9781618040640.

[91] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL `http://arxiv.org/abs/1412.6980`.

[92] Zachary Kingston, Mark Moll, and Lydia E. Kavraki. Sampling-based methods for motion planning with constraints. *Annual Review of Control, Robotics, and Autonomous Systems*, 1(1):159–185, 2018. doi: 10.1146/annurev-control-060117-105226.

[93] Zachary Kingston, Mark Moll, and Lydia E. Kavraki. Exploring implicit spaces for constrained sampling-based planning. *The International Journal of Robotics Research*, 38(10-11):1151–1178, 2019. doi: 10.1177/0278364919868530.

[94] B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A. Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 23(6): 4909–4926, 2022. doi: 10.1109/TITS.2021.3054625.

[95] Jens Kober and Jan Peters. Policy search for motor primitives in robotics. *Advances in neural information processing systems*, 21, 2008.

[96] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154 vol.3, 2004. doi: 10.1109/IROS.2004.1389727.

[97] Dieter Kraft. A software package for sequential quadratic programming. Technical Report DFVLR-FB 88-28, DLR German Aerospace Center – Institute for Flight Mechanics, Koln, Germany, 1988.

[98] J.J. Kuffner and S.M. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, volume 2, pages 995–1001 vol.2, 2000. doi: 10.1109/ROBOT.2000.844730.

[99] Steven M. LaValle. Rapidly-exploring random trees: a new tool for path planning. *The annual research report*, 1998.

[100] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[101] Teguh Santoso Lembono, Emmanuel Pignat, Julius Jankowski, and Sylvain Calinon. Learning constrained distributions of robot configurations with generative adversarial network. *IEEE Robotics and Automation Letters*, 6(2):4233–4240, 2021. doi: 10.1109/LRA.2021.3068671.

[102] Linjun Li, Yinglong Miao, Ahmed H. Qureshi, and Michael C. Yip. MPC-MPNet: Model-predictive motion planning networks for fast, near-optimal planning under kinodynamic constraints. *IEEE Robotics and Automation Letters*, 6(3):4496–4503, 2021. doi: 10.1109/LRA.2021.3067847.

[103] Sihui Li and Neil T. Dantam. A sampling and learning framework to prove motion planning infeasibility. *The International Journal of Robotics Research*, 0 (0), 2023. doi: 10.1177/02783649231154674. URL `https://doi.org/10.1177/02783649231154674`.

[104] Weiwei Li and Emanuel Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *Proceedings of the 1st International Conference on Informatics in Control, Automation and Robotics, (ICINCO 2004)*, volume 1, pages 222–229, 01 2004.

[105] Yanbo Li, Zakary Littlefield, and Kostas E. Bekris. Asymptotically optimal sampling-based kinodynamic planning. *The International Journal of Robotics Research*, 35(5): 528–564, 2016. doi: 10.1177/0278364915614386. URL `https://doi.org/10.1177/0278364915614386`.

[106] Maxim Likhachev and Dave Ferguson. Planning long dynamically feasible maneuvers for autonomous vehicles. *The International Journal of Robotics Research*, 28(8):933–945, 2009.

[107] Wonteak Lim, Seongjin Lee, Myoungho Sunwoo, and Kichun Jo. Hierarchical trajectory planning of an autonomous car based on the integration of a sampling and an optimization method. *IEEE Transactions on Intelligent Transportation Systems*, 19 (2):613–626, 2018. doi: 10.1109/TITS.2017.2756099.

[108] Todd Litman. Autonomous vehicle implementation predictions: Implications for transport planning, 2023.

[109] Puze Liu, Davide Tateo, Haitham Bou Ammar, and Jan Peters. Robot reinforcement learning on the constraint manifold. In *5th Annual Conference on Robot Learning*, 2021. URL `https://openreview.net/forum?id=zwo1-MdMl1P`.

[110] Puze Liu, Davide Tateo, Haitham Bou-Ammar, and Jan Peters. Efficient and reactive planning for high speed robot air hockey. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 586–593. IEEE, 2021.

[111] Tomás Lozano-Pérez and Michael A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Commun. ACM*, 22(10):560–570, oct 1979. ISSN 0001-0782. doi: 10.1145/359156.359164. URL `https://doi.org/10.1145/359156.359164`.

[112] D. Luenberger. An introduction to observers. *IEEE Transactions on Automatic Control*, 16(6):596–602, 1971. doi: 10.1109/TAC.1971.1099826.

[113] Franco Manessi and Alessandro Rozza. Learning combinations of activation functions. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 61–66, 2018. doi: 10.1109/ICPR.2018.8545362.

[114] Troy McMahon, Aravind Sivaramakrishnan, Kushal Kedia, Edgar Granados, and Kostas E. Bekris. Terrain-aware learned controllers for sampling-based kinodynamic planning over physically simulated terrains. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2925–2930, 2022. doi: 10.1109/IROS47612.2022.9982136.

[115] Tim Mercy, Wannes Van Loock, and Goele Pipeleers. Real-time motion planning in the presence of moving obstacles. In *2016 European Control Conference (ECC)*, pages 1586–1591, 2016. doi: 10.1109/ECC.2016.7810517.

[116] Maciej Marcin Michałek and Tomasz Gawron. Vfo path following control with guarantees of positionally constrained transients for unicycle-like robots with constrained control input. *Journal of Intelligent & Robotic Systems*, 89(1):191–210, Jan 2018.

[117] Takahiro Miki, Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning robust perceptive locomotion for quadrupedal robots in the wild. *Science Robotics*, 7(62):eabk2822, 2022. doi: 10.1126/scirobotics.abk2822.

[118] R. Miklosovic, A. Radke, and Zhiqiang Gao. Discrete implementation and generalization of the extended state observer. In *2006 American Control Conference*, pages 6 pp.–, 2006. doi: 10.1109/ACC.2006.1656547.

[119] Isaac Miller, Mark Campbell, Dan Huttenlocher, Frank-Robert Kline, Aaron Nathan, Sergei Lupashin, Jason Catlin, Brian Schimpf, Pete Moran, Noah Zych, Ephrahim Garcia, Mike Kurdziel, and Hikaru Fujishima. Team cornell's skynet: Robust perception and planning in an urban environment. *Journal of Field Robotics*, 25(8):493–527, 2008. doi: https://doi.org/10.1002/rob.20253. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.20253`.

[120] Dmytro Mishkin, Nikolay Sergievskiy, and Jiri Matas. Systematic evaluation of convolution neural network advances on the imagenet. *Computer Vision and Image Understanding*, 161:11–19, 2017. ISSN 1077-3142. doi: https://doi.org/10.1016/j.cviu.2017.05.007. URL `https://www.sciencedirect.com/science/article/pii/S1077314217300814`.

[121] Daniel Molina, Kislay Kumar, and Siddharth Srivastava. Learn and link: Learning critical regions for efficient planning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10605–10611, 2020. doi: 10.1109/ICRA40945.2020.9196833.

[122] Miguel Angel Zamora Mora, Momchil Peychev, Sehoon Ha, Martin Vechev, and Stelian Coros. Pods: Policy optimization via differentiable simulation. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 7805–7817. PMLR, 18–24 Jul 2021.

[123] Mustafa Mukadam, Jing Dong, Xinyan Yan, Frank Dellaert, and Byron Boots. Continuous-time gaussian process motion planning via probabilistic inference. *The International Journal of Robotics Research*, 37(11):1319–1340, Sep 2018. ISSN

1741-3176. doi: 10.1177/0278364918790369. URL http://dx.doi.org/10.1177/0278364918790369.

[124] Katharina Mülling, Jens Kober, and Jan Peters. A biomimetic approach to robot table tennis. *Adaptive Behavior*, 19(5):359–376, 2011.

[125] Akio Namiki, Sakyo Matsushita, Takahiro Ozeki, and Kenzo Nonami. Hierarchical processing architecture for an air-hockey robot system. In *2013 IEEE International Conference on Robotics and Automation*, pages 1187–1192. IEEE, 2013.

[126] Nils J. Nilsson. Shakey the robot. Technical report, SRI International, 1984.

[127] Colm ó'Dúnlaing, Micha Sharir, and Chee K. Yap. Retraction: A new approach to motion-planning. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC '83, page 207–220, New York, NY, USA, 1983. Association for Computing Machinery. ISBN 0897910990. doi: 10.1145/800061.808750. URL https://doi.org/10.1145/800061.808750.

[128] Mohsen Omrani, Matthew T Kaufman, Nicholas G Hatsopoulos, and Paul D Cheney. Perspectives on classical controversies about the motor cortex. *J Neurophysiol*, 118 (3):1828–1848, June 2017.

[129] Joaquim Ortiz-Haro, Jung-Su Ha, Danny Driess, and Marc Toussaint. Structured deep generative models for sampling on constraint manifolds in sequential manipulation. In Aleksandra Faust, David Hsu, and Gerhard Neumann, editors, *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 213–223. PMLR, 08–11 Nov 2022. URL https://proceedings.mlr.press/v164/ortiz-haro22a.html.

[130] Brian Paden, Michal Čáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles*, 1(1):33–55, 2016. doi: 10.1109/TIV.2016.2578706.

[131] Joseph J. Paton and Dean V. Buonomano. The neural basis of timing: Distributed mechanisms for diverse functions. *Neuron*, 98(4):687–705, May 2018. doi: 10.1016/j.neuron.2018.03.045. URL https://doi.org/10.1016/j.neuron.2018.03.045.

[132] Alejandro Perez, Robert Platt, George Konidaris, Leslie Kaelbling, and Tomas Lozano-Perez. LQR-RRT*: Optimal sampling-based motion planning with automatically derived extension heuristics. In *2012 IEEE International Conference on Robotics and Automation*, pages 2537–2542, 2012. doi: 10.1109/ICRA.2012.6225177.

[133] Allan Pinkus. Approximation theory of the MLP model in neural networks. *Acta Numerica*, 8:143–195, 1999. doi: 10.1017/S0962492900002919.

[134] Mihail Pivtoraiko, Ross A. Knepper, and Alonzo Kelly. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, 26(3):308–333, 2009. doi: https://doi.org/10.1002/rob.20285. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.20285.

[135] Kai Ploeger, Michael Lutter, and Jan Peters. High acceleration reinforcement learning for real-world juggling with binary rewards. In *Conference on Robot Learning*, pages 642–653. PMLR, 2021.

[136] Ashwini Pokle, Roberto Martín-Martín, Patrick Goebel, Vincent Chow, Hans M. Ewald, Junwei Yang, Zhenkai Wang, Amir Sadeghian, Dorsa Sadigh, Silvio Savarese,

and Marynel Vázquez. Deep local trajectory replanning and control for robot navigation. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 5815–5822, 2019. doi: 10.1109/ICRA.2019.8794062.

[137] Jan Willem Polderman and Jan C. Willems. *Pole Placement by State Feedback*, pages 311–339. Springer New York, New York, NY, 1998. ISBN 978-1-4757-2953-5. doi: 10.1007/978-1-4757-2953-5_9. URL https://doi.org/10.1007/978-1-4757-2953-5_9.

[138] Stefano Primatesta, Abdalla Osman, and Alessandro Rizzo. MP-RRT#: a model predictive sampling-based motion planning algorithm for unmanned aircraft systems. *Journal of Intelligent & Robotic Systems*, 103(4):59, Nov 2021. ISSN 1573-0409. doi: 10.1007/s10846-021-01501-3. URL https://doi.org/10.1007/s10846-021-01501-3.

[139] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[140] Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Ng. Ros: an open-source robot operating system. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan, may 2009.

[141] Ahmed Hussain Qureshi, Jiangeng Dong, Asfiya Baig, and Michael C. Yip. Constrained motion planning networks x. *IEEE Transactions on Robotics*, pages 1–19, 2021. doi: 10.1109/TRO.2021.3096070.

[142] Ahmed Hussain Qureshi, Yinglong Miao, Anthony Simeonov, and Michael C. Yip. Motion planning networks: Bridging the gap between learning-based and classical motion planners. *IEEE Transactions on Robotics*, 37(1):48–66, 2021. doi: 10.1109/TRO.2020.3006716.

[143] A. Radke and Zhiqiang Gao. A survey of state and disturbance observers for practitioners. In *2006 American Control Conference*, pages 6 pp.–, 2006. doi: 10.1109/ACC.2006.1657545.

[144] Rajesh Rajamani. *Vehicle Dynamics and Control*. Springer, 2012.

[145] John H. Reif. Complexity of the mover's problem and generalizations. In *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, pages 421–427, 1979. doi: 10.1109/SFCS.1979.10.

[146] Nicholas Rhinehart, Rowan McAllister, and Sergey Levine. Deep imitative models for flexible inference, planning, and control. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=Skl4mRNYDr.

[147] Reuven Y. Rubinstein and Dirk P. Kroese. *The Cross Entropy Method: A Unified Approach To Combinatorial Optimization, Monte-Carlo Simulation (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2004. ISBN 038721240X.

[148] Christoph Rösmann, Frank Hoffmann, and Torsten Bertram. Kinodynamic trajectory optimization and control for car-like robots. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5681–5686, 2017. doi: 10.1109/IROS.2017.8206458.

[149] Caude Sammut. *Behavioral Cloning*, pages 93–97. Springer US, Boston, MA, 2010. ISBN 978-0-387-30164-8. doi: 10.1007/978-0-387-30164-8_69. URL https://doi.org/10.1007/978-0-387-30164-8_69.

[150] Stefan Schaal. *Dynamic Movement Primitives -A Framework for Motor Control in Humans and Humanoid Robotics*, pages 261–280. Springer Tokyo, Tokyo, 2006. ISBN 978-4-431-31381-6. doi: 10.1007/4-431-31381-8_23. URL https://doi.org/10.1007/4-431-31381-8_23.

[151] John Schulman, Jonathan Ho, Alex X. Lee, Ibrahim Awwal, Henry Bradlow, and P. Abbeel. Finding locally optimal, collision-free trajectories with sequential convex optimization. *Robotics: Science and Systems IX*, 2013.

[152] Aravind Sivaramakrishnan, Edgar Granados, Seth Karten, Troy McMahon, and Kostas E. Bekris. Improving kinodynamic planners for vehicular navigation with learned goal-reaching controllers. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9038–9043, 2021. doi: 10.1109/IROS51168.2021.9636657.

[153] Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Universal planning networks: Learning generalizable representations for visuomotor control. In *International Conference on Machine Learning*, pages 4732–4741. PMLR, 2018.

[154] Joseph A. Starek, Javier V. Gomez, Edward Schmerling, Lucas Janson, Luis Moreno, and Marco Pavone. An asymptotically-optimal sampling-based algorithm for bi-directional motion planning. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2072–2078, 2015. doi: 10.1109/IROS.2015.7353652.

[155] Samantha Stoneman and Roberto Lampariello. Embedding nonlinear optimization in RRT for optimal kinodynamic planning. In *53rd IEEE Conference on Decision and Control*, pages 3737–3744, 2014. doi: 10.1109/CDC.2014.7039971.

[156] Adam Stooke, Joshua Achiam, and Pieter Abbeel. Responsive safety in reinforcement learning by pid lagrangian methods. In *International Conference on Machine Learning*, pages 9133–9143. PMLR, 2020.

[157] Marlin P. Strub and Jonathan D. Gammell. Advanced bit* (abit*): Sampling-based planning with advanced graph-search techniques. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 130–136, 2020. doi: 10.1109/ICRA40945.2020.9196580.

[158] Marlin P Strub and Jonathan D Gammell. Adaptively Informed Trees (AIT*) and Effort Informed Trees (EIT*): Asymmetric bidirectional sampling-based path planning. *The International Journal of Robotics Research (IJRR)*, 41(4):390–417, 2022.

[159] John K. Subosits and J. Christian Gerdes. From the racetrack to the road: Real-time trajectory replanning for autonomous driving. *IEEE Transactions on Intelligent Vehicles*, 4(2):309–320, 2019. doi: 10.1109/TIV.2019.2904390.

[160] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. doi: 10.1109/MRA.2012.2205651. https://ompl.kavrakilab.org.

[161] Niko Sünderhauf, Oliver Brock, Walter Scheirer, Raia Hadsell, Dieter Fox, Jürgen Leitner, Ben Upcroft, Pieter Abbeel, Wolfram Burgard, Michael Milford, and Peter Corke. The limits and potentials of deep learning for robotics. *The International Journal of Robotics Research*, 37(4-5):405–420, 2018.

[162] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, 1998.

[163] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL `http://incompleteideas.net/book/the-book-2nd.html`.

[164] Jakub Szkandera, Ivana Kolingerová, and Martin Maňák. Narrow passage problem solution for motion planning. In Valeria V. Krzhizhanovskaya, Gábor Závodszky, Michael H. Lees, Jack J. Dongarra, Peter M. A. Sloot, Sérgio Brissos, and João Teixeira, editors, *Computational Science – ICCS 2020*, pages 459–470, Cham, 2020. Springer International Publishing.

[165] Siyu Teng, Xuemin Hu, Peng Deng, Bai Li, Yuchen Li, Yunfeng Ai, Dongsheng Yang, Lingxi Li, Zhe Xuanyuan, Fenghua Zhu, and Long Chen. Motion planning for autonomous driving: The state of the art and future perspectives. *IEEE Transactions on Intelligent Vehicles*, 8(6):3692–3711, 2023. doi: 10.1109/TIV.2023.3274536.

[166] Ryo Terasawa, Yuka Ariki, Takuya Narihira, Toshimitsu Tsuboi, and Kenichiro Nagasaka. 3d-cnn based heuristic guided task-space planner for faster motion planning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9548–9554, 2020. doi: 10.1109/ICRA40945.2020.9196883.

[167] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30, 2017. doi: 10.1109/IROS.2017.8202133.

[168] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

[169] Matt Vitelli, Yan Chang, Yawei Ye, Ana Ferreira, Maciej Wołczyk, Błażej Osiński, Moritz Niendorf, Hugo Grimmett, Qiangui Huang, Ashesh Jain, and Peter Ondruska. Safetynet: Safe planning for real-world self-driving vehicles using machine-learned policies. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 897–904, 2022. doi: 10.1109/ICRA46639.2022.9811576.

[170] Felix von Drigalski, Devwrat Joshi, Takayuki Murooka, Kazutoshi Tanaka, Masashi Hamaya, and Yoshihisa Ijiri. An analytical diabolo model for robotic learning and control. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4055–4061. IEEE, 2021.

[171] José L. Vázquez, Marius Brühlmeier, Alexander Liniger, Alisa Rupenyan, and John Lygeros. Optimization-based hierarchical motion planning for autonomous racing. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2397–2403, 2020. doi: 10.1109/IROS45743.2020.9341731.

[172] Alvin Wan. Making decision trees accurate again: Explaining what explainable ai did not, 2020. URL `https://bair.berkeley.edu/blog/2020/04/23/decisions/#fnref:data`.

[173] Jiankun Wang, Tianyi Zhang, Nachuan Ma, Zhaoting Li, Han Ma, Fei Meng, and Max Q.-H. Meng. A survey of learning-based robot motion planning. *IET Cyber-Systems and Robotics*, 2021. URL `https://api.semanticscholar.org/CorpusID:236381976`.

[174] Kesheng Wang. B-splines joint trajectory planning. *Computers in Industry*, 10(2):113–122, 1988. ISSN 0166-3615. doi: https://doi.org/10.1016/0166-3615(88)90016-4. URL `https://www.sciencedirect.com/science/article/pii/0166361588900164`.

[175] Wouter J. Wolfslag, Mukunda Bharatheesha, Thomas M. Moerland, and Martijn Wisse. RRT-CoLearn: Towards kinodynamic planning without numerical trajectory optimization. *IEEE Robotics and Automation Letters*, 3(3):1655–1662, 2018. doi: 10.1109/LRA.2018.2801470.

[176] Xuesu Xiao, Bo Liu, Garrett Warnell, and Peter Stone. Motion planning and control for mobile robot navigation using machine learning: a survey. *Autonomous Robots*, 46(5):569–597, Jun 2022. ISSN 1573-7527. doi: 10.1007/s10514-022-10039-8. URL `https://doi.org/10.1007/s10514-022-10039-8`.

[177] Xuesu Xiao, Zizhao Wang, Zifan Xu, Bo Liu, Garrett Warnell, Gauraang Dhamankar, Anirudh Nair, and Peter Stone. Appl: Adaptive planner parameter learning. *Robotics and Autonomous Systems*, 154:104132, 2022. ISSN 0921-8890. doi: https://doi.org/10.1016/j.robot.2022.104132. URL `https://www.sciencedirect.com/science/article/pii/S0921889022000744`.

[178] Mandy Xie and Frank Dellaert. Batch and incremental kinodynamic motion planning using dynamic factor graphs, 2020. URL `https://arxiv.org/abs/2005.12514`.

[179] Wenda Xu, Junqing Wei, John M. Dolan, Huijing Zhao, and Hongbin Zha. A real-time motion planner with trajectory optimization for autonomous vehicles. In *2012 IEEE International Conference on Robotics and Automation*, pages 2061–2067, 2012. doi: 10.1109/ICRA.2012.6225063.

[180] Zhihao Xu, Robin Hess, and Klaus Schilling. Constraints of potential field for obstacle avoidance on car-like mobile robots. *IFAC Proceedings Volumes*, 45(4):169–175, 2012. ISSN 1474-6670. doi: https://doi.org/10.3182/20120403-3-DE-3010.00077. URL `https://www.sciencedirect.com/science/article/pii/S1474667015404616`. 1st IFAC Conference on Embedded Systems, Computational Intelligence and Telematics in Control.

[181] Jun Yamada, Chia-Man Hung, Jack Collins, Ioannis Havoutis, and Ingmar Posner. Leveraging scene embeddings for gradient-based motion planning in latent space. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023.

[182] Mohammadreza Yavari, Kamal Gupta, and Mehran Mehrandezh. Lazy steering RRT: An optimal constrained kinodynamic neural network based planner with no in-exploration steering. In *2019 19th International Conference on Advanced Robotics (ICAR)*, pages 400–407, 2019. doi: 10.1109/ICAR46387.2019.8981551.

[183] Sangyol Yoon, Dasol Lee, Jiwon Jung, and David Shim. Spline-based rrt* using piecewise continuous collision-checking algorithm for car-like vehicles. *Journal of Intelligent & Robotic Systems*, 90:1–13, 09 2017.

[184] Chenning Yu and Sicun Gao. Reducing collision checking for sampling-based motion planning using graph neural networks. *Advances in Neural Information Processing Systems*, 34:4274–4289, 2021.

[185] Wenyuan Zeng, Wenjie Luo, Simon Suo, Abbas Sadat, Bin Yang, Sergio Casas, and Raquel Urtasun. End-to-end interpretable neural motion planner. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8652–8661, 2019. doi: 10.1109/CVPR.2019.00886.

[186] Clark Zhang, Jinwook Huh, and Daniel D. Lee. Learning implicit sampling distributions for motion planning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3654–3661, 2018. doi: 10.1109/IROS.2018.8594028.

[187] Ji Zhang and Sanjiv Singh. Low-drift and real-time lidar odometry and mapping. *Autonomous Robots*, 41:401–416, 2017.

[188] Dongliang Zheng and Panagiotis Tsiotras. Accelerating kinodynamic RRT through dimensionality reduction. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3674–3680, 2021. doi: 10.1109/IROS51168.2021.9636754.

[189] Matt Zucker, Nathan Ratliff, Anca D. Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M. Dellin, J. Andrew Bagnell, and Siddhartha S. Srinivasa. CHOMP: Covariant hamiltonian optimization for motion planning. *The International Journal of Robotics Research*, 32(9-10):1164–1193, 2013. doi: 10.1177/0278364913488805.