2014

TIME-OPTIMAL CONTROL OF AN UNMANNED OCEAN SURFACE VESSEL

Jan Eilbrecht
University of Rhode Island, jamieilenator@gmail.com

TIME–OPTIMAL CONTROL OF AN UNMANNED OCEAN SURFACE VESSEL

BY

JAN EILBRECHT

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

MECHANICAL ENGINEERING AND APPLIED MECHANICS

UNIVERSITY OF RHODE ISLAND

2014

MASTER OF SCIENCE THESIS

OF

JAN EILBRECHT

APPROVED:

Thesis Committee:

Major Professor David Chelidze

Musa Jouaneh

Lutz Hamel

Nasser H. Zawia

DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND

2014

ABSTRACT

Path planning for unmanned ocean vehicles often neglects constraints arising from the dynamics of the vehicle. Therefore, this thesis is concerned with finding a method that can incorporate a dynamical vehicle model into the problem of time-optimal path planning. This requirement makes the path planning task an optimal control problem. Based on a review of optimal control theory and possible solution procedures, sampling–based algorithms and especially the Sparse RRT algorithm are identified as promising means of solving the problem.

At first, several benchmark tests are carried out to demonstrate that Sparse RRT is able to generate reliable results. Then, the algorithm is applied to a dynamic boat model with three degrees of freedom and two control inputs. Constraints on the states of the system as can arise from obstacles or energy constraints are accounted for as well as constraints on the control inputs. Also, the effect of ocean current fields is incorporated.

In all planning scenarios, the application of Sparse RRT yielded plausible results. In conclusion of this thesis, the Sparse RRT algorithm therefore can be rated as a very flexible tool due to its ability to solve optimal control problems like time-optimal path planning for dynamic vehicle models while accounting for several different types of constraints.

However, it also turned out that especially higher-dimensional models require long computation times to ensure good results. In order to make a result applicable to real technical systems, post-processing procedures might be necessary. Also, several parameters of the algorithm have to be chosen carefully.

ACKNOWLEDGMENTS

As John Betts states in the preface of his book on optimal control [1], "solving an optimal control problem is not easy." In my experience, this is also true for writing a thesis. All the more I am indebted to my advisor, Professor David Chelidze, for all discussions and advice as well as for all I was allowed to learn in general while working in the Nonlinear Dynamics Laboratory. I also owe thanks to Dr. Thomas Wettergren and Dr. David Segala from the Naval Undersea Warfare Center for contributing valuable suggestions, as well as I would like to thank my co-workers Shahab and especially Son for all helpful hints. Special thanks also go to Professor Jouaneh and Professor Hamel for serving on my thesis committee and also for teaching their really beneficial classes.

In addition, I am grateful for all the people who made my stay at the University of Rhode Island such a great experience. This, however, would not have been possible without the financial support from the German National Academic Foundation.

Finally, I want to thank my friends near and far and especially my family for all encouragement and support.

TABLE OF CONTENTS

ABSTRACT ii

ACKNOWLEDGMENTS iii

TABLE OF CONTENTS iv

LIST OF FIGURES viii

CHAPTER

1 Introduction 1

2 Literature review 2

2.1 General remarks 2

2.2 Planning without differential constraints 3

2.2.1 Graph searching techniques 3

2.2.2 Level set methods 7

2.2.3 Mixed integer linear programming 7

2.3 Planning under differential constraints 8

2.3.1 Analytical solution 8

2.3.2 Dynamic Programming 9

2.3.3 Classical optimization techniques 11

2.3.4 Metaheuristic algorithm approaches 12

2.3.5 Sampling-based algorithms 15

2.4 Conclusion and implications for the presented thesis 19

3 Optimal control 23

Page

3.1 The general optimal control problem 23

3.2 Necessary conditions for optimality 24

3.2.1 Minimization of a functional 24

3.2.2 Final state and final time 26

3.3 Constraints 29

3.3.1 Differential constraints 29

3.3.2 Control constraints 32

3.3.3 State inequality constraints 32

3.4 Solution procedures 33

3.5 Formulation of the considered problem 34

3.6 Summary 35

4 Analysis of the Sparse RRT algorithm 36

4.1 General functionality 36

4.2 Properties 37

4.3 Implementation 38

4.3.1 Storing states 39

4.3.2 Measuring distances in state space 40

4.3.3 Trajectory propagation 41

4.4 Verification 42

4.4.1 Zermelo's Problem 42

4.4.2 Jet current 47

4.4.3 Energy-optimal control problem 48

4.5 Conceptual testing 50

4.5.1 Obstacles 51

4.5.2 Energy constraints 52

4.6 Influence of the free algorithm parameters 56

4.6.1 "Drain" radius 56

4.6.2 "Best Nearest" radius 58

4.6.3 Goal tolerance 59

4.6.4 Propagation duration 60

4.6.5 Number of iterations 61

4.7 (Post-)Processing of results 62

4.7.1 Filtering 63

4.7.2 Selective sampling 64

4.7.3 Averaging 65

4.8 Summary 66

5 Modeling of and planning for an unmanned boat 67

5.1 Dynamic boat model 67

5.1.1 Degrees of freedom and coordinate systems 67

5.1.2 External forces 68

5.1.3 Equations of motion 71

5.2 Path planning for the dynamic model 72

5.3 Discussion 76

6 Summary and future prospects 79

6.1 Summary 79

6.2 Future prospects 80

Page

LIST OF REFERENCES 81

APPENDIX

Model parameters 85

BIBLIOGRAPHY 86

LIST OF FIGURES

Figure		Page
2.1	General idea of dynamic programming (cf. [2, p.55])	11
4.1	General functionality of Sparse RRT	37
4.2	Zermelo's problem: analytical and numerical solution	47
4.3	Optimal solution and solution generated by Sparse RRT	48
4.4	Energy-optimal control problem with fixed final time	50
4.5	Solutions to Zermelo's problem in the presence of an obstacle	53
4.6	Test result for variable speed with limited energy	55
4.7	Drain radius blocking the goal region in a bounded space	57
4.8	Drain radius blocking narrow passage (cf. [3])	57
4.9	Large r_{near} leading to repeated propagation of the same node	59
4.10	Goal tolerance as an upper limit to the solution precision	60
4.11	Problem of section 4.4 based on a shorter propagation duration	62
4.12	Filtered control sequence and resulting trajectories	63
4.13	Results for derivative-bounded control sequence	64
4.14	Results based on an averaged control sequence	65
5.1	Schematic of the considered boat model (courtesy of [4])	68
5.2	Turning radii for different thrust magnitudes	73
5.3	Path resulting from given initial speeds	74
5.4	Speeds and heading angle	74
5.5	Time-optimal path for a dynamical boat model	75
5.6	Time-optimal path for different final heading direction	77

CHAPTER 1

Introduction

As [5] notes, "planning a path in complex environment is a problem as old as antiquity." However, it was probably Zermelo in 1931 [6] who provided the first mathematical approach to the problem of how to steer a vehicle from one state to another, optimizing a certain criterion. Since then, the problem has experienced steadily growing attention. This is especially true against the background of autonomous underwater vehicles (AUV) that have been under development for the last decades and are now an important mean of data sampling in oceanography [7]. In order to maximize the benefit of a mission, path planning procedures that optimize certain criteria are of great importance.

Recently, the Naval Undersea Warfare Center (NUWC) has shown interest in developing a method to plan paths for an unmanned surface vessel for data gathering purposes over short distances. As chapter 2 shows, existing approaches have shortcomings that make them not appropriate for the solution of this very problem. Therefore, this thesis aims at identifying an adequate solution procedure based on an optimal control approach. This thesis is structured as follows:

In chapter 2, a survey of existing approaches to the considered problem along with a review of benefits and drawbacks of the respective methods is given. Chapter 3 provides the theoretic foundation of optimal control and formulates the problem in terms of an optimal control problem. Chapter 4 addresses the analysis of an algorithm intended for solving this optimal control problem, the Sparse RRT algorithm. Results of the application of this algorithm to a dynamical boat model are shown in chapter 5, while chapter 6 is concerned with a concluding discussion of the results and possible further research.

CHAPTER 2

Literature review

In this chapter, an organized overview of existing approaches as well as literature related to the problem of path planning for unmanned marine vessels is given. The aim is to review these approaches, pointing out both advantages and shortcomings. From this evaluation, the motivation of this thesis will be derived.

2.1 General remarks

In the following, "path planning" will be understood as determining a sequence of states of a system leading from a given start state to a goal state while optimizing a certain criterion. As [5] states, path planning for ocean vehicles is especially challenging due to the "highly dynamic and multiscale system" of the ocean with current fields and complex geometries in vicinity to the coast.

Most research on path planning for unmanned marine vessels has been done considering underwater vessels of the glider type. Nonetheless, in most cases the found results in principle also apply for surface vessels because usually, the vertical degree of freedom is neglected so paths are planned in a plane. Therefore, a differentiation between approaches based on the application is not necessary in the following. In some cases, this can even hold for results obtained for unmanned aerial vehicles (UAV). From a methodological point of view, even procedures from the broad field of path planning for land-based autonomous vehicles can be of importance.

In general, the existing approaches can be divided into two classes: planning without and planning under differential constraints.

In the latter case, following [8], a state transition equation of the form

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}\left(\boldsymbol{x}(t), \boldsymbol{u}(t), t\right) \tag{2.1}$$

constrains how a transition between neighboring states can be made, based on some decisions (or actions/controls) $\boldsymbol{u}(t)$, while planning without these kinds of constraints basically allows arbitrary transitions.

This leads to different types of results: for planning without differential constraints, a result consists of an explicit representation of the found path in terms of the states to be traversed. Planning under differential constraints in contrast leads to a sequence of decisions $\boldsymbol{u}(t)$ from which the actual trajectory can be computed by integration, what actually becomes the solution to an optimal control problem as will be described later.

2.2 Planning without differential constraints

2.2.1 Graph searching techniques

One important class of solution procedures for path finding problems are graph searching methods. Being of great importance in computer science in general and also for path planning for land robots, they are also applied to path planning for unmanned marine vehicles. The general idea of a graph searching technique based solution procedure is to discretize an area, transforming it into a graph consisting of nodes and edges and applying a search strategy like Dijkstra's algorithm and the A* algorithm [8] or modifications of these [9–13]. The nodes usually represent spatial coordinates, while the weighted edges represent the cost of traversing between nodes. Depending on the criterion to be optimized, the edge weights could represent distance, traveling time, energy consumption, exposure to risk and so forth. In the following, Dijkstra's algorithm and the A* algorithm will be described briefly.

Dijkstra's algorithm

As described in [8], Dijkstra's algorithm is based on assigning each node a parent node that minimizes the traveling cost to that node. Given these information, a minimum cost path to a certain node can be constructed by backtracking the parent nodes to the start node. The parent nodes are determined incrementally during the search process:

In each iteration, the costs of traveling to all adjacent nodes of the node with the lowest overall cost are determined, making that node the parent of all neighbors. This node is taken from a priority queue which is maintained to store the traveling cost information for all nodes that have been assigned a cost in this way. As long as a node is in that priority queue, its cost value respectively its parent node can change when a cheaper path to it is found. As soon as the vicinity of a node is examined, the traveling cost to that node cannot change anymore and it is removed from the priority queue. If the surrounding of the goal node is examined, the algorithm can construct the optimal path by backtracking all parent nodes. Dijkstra's algorithm is complete, that is it finds an optimal solution if one exists.

A^* algorithm

The A^* algorithm is an extension of Dijkstra's algorithm that attempts to reduce the number of nodes which has to be considered during computations. This is achieved by using an estimate of the remaining cost of traveling from a given node to the destination node, assigning a higher priority in the queue to nodes with a lower cost estimate. In doing so, the search is guided towards the destination node. Provided that the heuristic never overestimates the remaining cost, this algorithm also finds an optimal solution if one exists.

Applications

An early approach to path planning for unmanned marine vessels using a graph searching method can be found in [14]. There, the A* algorithm is used to generate resource-optimal paths for long-range missions, considering obstacles and current information. This approach is taken on by [15, 16], providing a more detailed description of the algorithm and especially of how to include current fields in the planning process. In order to minimize traveling time, the cost of traversing a graph edge between two nodes $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ is calculated according to

$$t_{ij} = \frac{d_{ij}}{v_{ij}} \tag{2.2}$$

where d_{ij} is the euclidean distance between the two nodes and v_{ij} the ground speed of the vehicle, calculated as sum of the local current velocity and the vehicle speed relative to the current, projected on the straight connection line of the nodes. This projection is necessary because the velocity vector of the vehicle must not necessarily be parallel to that line due to the current field . The heuristic function that is used to estimate the remaining cost from a given node $\boldsymbol{x}_i$ to the destination is chosen as

$$h(\boldsymbol{x}_i) = \frac{d_i}{v_{max}} \tag{2.3}$$

with d_i being the euclidean distance to the destination and v_{max} the sum of the maximum vehicle velocity and the maximum current velocity in the considered area. This ensures that the cost is never underestimated. Also, the impact of different connectivity schemes between adjacent nodes on the shape of the planned path is evaluated: in increasing the number of possible connections to neighboring nodes, a path can adapt better to the features of the current field, but at the same time, the computation time is increased.

In [9], a Dijkstra-like algorithm called "Symbolic Wavefront Expansion" is used to determine the time-optimal path as well as the optimal departure time

in a time-varying current field. This is achieved by not evaluating cost values for each cell, but cost functions in dependency from the departure time.

In [10], algorithms based on Dijsktra's algorithm and the A* algorithm are modified to be able to calculate time-optimal paths in nonstationary current fields. A speed-up of the computation time is achieved by using results from [6] that help to reduce the number of nodes to be searched. In order to make a path feasible for real missions, a smoothing procedure is introduced.

The problem of path smoothing is also considered in [11], where a shortest path is generated for a UAV. After generation, this path is refined using Bézier curves with defined minimum curvature to make the path more traversable for a real vehicle.

A different approach to the same problem is chosen in [12], modifying A* in a way so that the planned path does not directly depend on the grid discretization anymore. This is achieved by integrating a kinematic vehicle model. Starting at one node, the model is integrated for several different heading angles over a specified time interval. The coordinates of the reached end points and the heading angles used to generate the respective trajectories are then stored in the nearest node of the graph. Applying the same heuristic as described in (2.3), the points next to be evaluated are chosen as long as the destination is not reached. This procedure produces more realistic results than A*, even though it is only based on a kinematic and not a dynamic model.

A similar approach to produce continuous paths can be found in [13]. There, the A* algorithm is combined with a fast marching method, resulting in an algorithm that produces continuous paths with constrained curvature to account for vehicle kinematics, considering current fields.

Graph searching techniques generally provide exact solutions against the back-

ground of the graph they are based upon, being robust to local minima solutions. This graph discretization is probably their major drawback, because the produced paths are not smooth enough to serve as a trajectory. Even though some effort has been undertaken to work around this, the basic problem cannot be overcome. Additionally, the time required to calculate a path might grow rapidly, depending on the resolution of the discretization. Algorithms like A* indeed provide means to reduce this time, but in the presence of a current field, the estimate in (2.3) might be to conservative, reducing the efficiency of using a heuristic.

2.2.2 Level set methods

In [5], time-optimal paths for ocean vehicles in continuous dynamic current fields are generated using level set methods. This path planning procedure is developed in [17] and [18]. The idea is to evolve a scalar field the zero level set of which represents a position reachable by a vehicle in a certain time interval. The scalar field is evolved by the current field and the vehicle speed until its zero level set reaches the target position. Then, the shortest path is generated by integrating backwards in time, reversing the velocity components that evolved the level set. It is shown that the generated paths are very exact and in addition, the method can calculate paths for multiple vehicles starting at the same point with only slightly more computational effort than for single vehicle planning. Nonetheless, dynamic effects are neglected and the vehicle's speed is assumed to be constant.

2.2.3 Mixed integer linear programming

In [19], a method based on mixed integer linear programming is used to find a vehicle path that maximizes the line integral of the uncertainty of field estimates along this path. In other words, the objective is to plan a path so that the ocean vehicle can conduct measurements in the regions of greatest uncertainty, max-

imizing the benefit of a measurement. The presented framework is able to plan paths for multiple vehicles, considering collision avoidance with both other vehicles and natural obstacles. Additionally, several modes of communication either with an accompanying ship, radio stations at shore or buoys and the thereby imposed constraints are considered. Since this approach focuses on optimizing the impact of measurements, the influence of ocean currents or the vehicle's dynamics is neglected, even though it is pointed out that further research could be conducted on path smoothing in order to generate more realistic paths.

2.3 Planning under differential constraints

As indicated at the beginning, planning under differential constraints requires the solution of an optimal control problem if the found path is not only required to be feasible, but also to optimize a certain criterion. Therefore, the optimal control problem will be important for the following chapters of the thesis. An introduction to the theoretic foundations of optimal control can be found in chapter 3, so at this point only an overview of existing applications will be given.

The optimal control approach is based on finding control inputs for a dynamic vehicle model which make it follow an optimal trajectory. Unlike the previously described approaches, this will give rise to a path that is as realistic as the vehicle model is. However, this problem is complex, since it is known that planning under differential constraints in general is NP-hard [8].

2.3.1 Analytical solution

For simple kinematic constraints, analytical solutions to the path planning problem can be derived using optimal control theory. A rather famous example in the planning community is the so-called "Dubins car," a kinematic model of a car that can only move forward at constant speed and has a minimum turning

radius [8]. It can be shown that the shortest path between two arbitrary configurations consists of no more than three motion primitives. These primitives are driving straight at maximum speed and turning left or right with minimum turning radius. From all possible combinations of these, only six are possibly optimal (cf. [8]). An extension to a car that can also back up exists, the so-called "Reeds-Shepp car."

In [20], the planar motion of a kinematic model of an airplane subject to constant wind is considered, whereby the model of the airplane is identical to a Dubins car. Even though analytical solutions are advantageous in many senses, in this case they are only available for models too simple for the considered application.

2.3.2 Dynamic Programming

The computational technique of Dynamic Programming was developed by R.E. Bellman and according to [21] has been closely connected to the increasing importance of optimal control. Even though to the knowledge of the author there exists no application of this method to the problem of path planning for autonomous ocean vehicles, a short overview of the method has to be given for completeness.

Following the explanation in [2], it is based on what is called the "Principle of Optimality," stating that "an optimal policy has the property that whatever the initial state and initial decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision." This implies a recursive approach ([2] refers to this as "Dynamic Programming," while [8] differentiates further, calling the following algorithm "Backward Value Iteration"):

Consider a sequence $k_1 ... k_n,\ n \in \mathbb{N}$, where each element indicates a stage of a decision making process at which a decision respectively control input has to be determined. Now, according to the Principle of Optimality, in order to minimize

the cost of getting from an initial state at stage k_1 to a certain state at stage $k + n$, the decision made at k_2 has to result in an optimal trajectory. Given this trajectory for every state admissible at k_2, the problem has been reduced to finding the decision at stage k_1 that minimizes the sum of the cost of the trajectory from the initial state at stage k_1 to some state at k_2 and the cost from there to the final state at stage k_n. This computation of course requires the knowledge of the optimal trajectories from all states at stage k_2 that recursively can be determined in the same way. Figure 2.1 illustrates this: the state at stage k_n can be reached from three states at stage k_{n-1} by several different trajectories. However, as shown, only one trajectory from each state yields optimal cost. The three states at stage k_{n-1} again can be reached by different trajectories from the single state at stage k_{n-2}, but only one trajectory presents the optimal way of reaching a certain state. The optimal sequence from the state at stage k_{n-2} to the final state at stage k_n then can be determined by calculating which combination of optimal paths yields the lowest overall cost.

Now, without differential constraints, Dynamic Programming could also be applicable to the problems described in section 2.2 and actually, according to [8], Dijkstra's algorithm and therefore also the A* algorithm are special forms of Dynamic Programming.

As [8] states, Dynamic Programming is theoretically able to provide optimal controls for any system. Nonetheless, it is agreed upon the fact that computational cost is usually not reasonable if the dimension of the state space is high (where [8] sees a limit for practical use at a state space dimension of six, as it is the case for a rigid body in the plane).

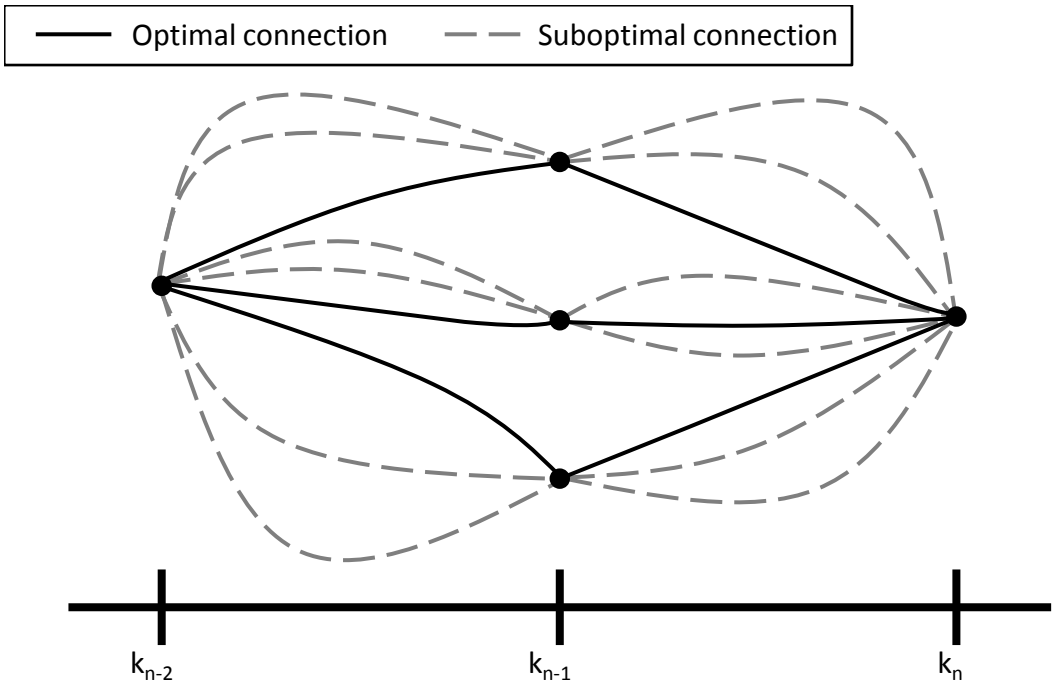

Figure 2.1. General idea of dynamic programming (cf. [2, p.55])

2.3.3 Classical optimization techniques

An early application of optimal control techniques to the field of unmanned underwater vehicles can be found in [22]. There, the problem is solved using Sequential Quadratic Programming to determine piece-wise constant control inputs to a six degree-of-freedom dynamic system. The switching times of the control values are fixed due to convergence problems. The presented approach is able to avoid obstacles, even though those have to be included into the problem formulation as path constraints which limits the flexibility in modeling more complicated obstacles.

In [23] and especially [24], the optimal control problem is solved using the Nonlinear Trajectory Generation (NTG) framework, creating energy- or time-optimal paths. The method is able to account for ocean currents, but obstacles are not accounted for. Calculations are done for kinematic, but also simple dynamic models of an underwater vehicle.

Also a nonlinear optimization approach is chosen in [25], where an extended version of Zermelo's problem (see [6]) is treated. The considered problem is to find a minimum-time path for a kinematic boat model across a river with strong currents and moving obstacles in it by adjusting the heading angle as a control variable. This is achieved by discretizing the control input function, making it piecewise constant. Mapping the switching times on a fixed interval enables the control values and switching times to become decision variables of the nonlinear optimization process, whereby obstacles are included using penalty functions.

2.3.4 Metaheuristic algorithm approaches

As remarked in [26], a unique definition of metaheuristic optimization algorithms does not exist. Nonetheless, several general properties are commonly associated with this class of often nature-inspired algorithms, such as being approximate procedures for solving complicated optimization problems, making use of diversification and intensification schemes to explore the solution space. At this point, only applications to the problem of path planning for ocean vehicles will be listed.

Artificial potential fields

Even though usually not listed among the metaheuristics, artificial potential fields will be classified as such because they also are a problem-unspecific, generally applicable solution procedure.

An early approach to path planning for autonomous underwater vehicles using artificial potential fields is presented in [27]. As stated there, at that time, artificial potential fields together with graph searching techniques were the two main areas of motion planning in robotics. However, to the knowledge of the author, [27] is the only reference in path planning for ocean vehicles that makes essential use of

that technique, while [28] uses it to some extent.

The described algorithm works as follows: a path is represented by straight line segments between nodes the coordinates of which have to be chosen adequately in order to optimize the path in terms of length and obstacle avoidance. An initial guess for a path is provided and subsequently transformed by changing the node coordinates, using a gradient-based searching algorithm like the Simplex method. This method tries to minimize the potential of the path, consisting of two elements: obstacles are modeled using artificial potential fields, so crossing an obstacle drastically increases the potential of a path and therefore will be avoided. Also, the length of path segments is assigned a potential, similar to an elastic chord. Therefore, the algorithm also tries to minimize the path length.

This procedure is described as being relatively fast and easily extensible to higher dimensions. Yet, drawbacks are the fact that the number of nodes the path contains of has to be chosen heuristically in advance. Also, it is possible to obtain only locally optimal solutions, while some scenarios cannot be solved reliably at all, e.g. those containing concave obstacles. In addition, current fields and the vehicle's dynamic are completely neglected.

An extension to this work is provided by [28], making use of artificial potential fields for obstacle avoidance in planning energy-optimal paths for autonomous underwater vehicles. Energy cost and other optimization criteria like traveling time and water depth are included using weighted penalty-function-like terms. The optimization task is carried out using a self-developed Local Random Search algorithm in combination with parallelization and Simulated Annealing. Simulated Annealing is another metaheuristic that accepts worse solutions than the current one with a certain probability in order to escape local minima. This probability decreases gradually, just like the temperature in an annealing process [26].

Genetic algorithms

In [29], a genetic algorithm is employed to plan energy-optimal paths through a current field of high magnitudes and spacial variability. In this genetic algorithm, a randomly generated population of feasible paths, represented by nodal coordinates connected by straight line segments, is transformed by several genetic operators such as crossover and mutation to form new solutions. These genetic operators aim at combining good solutions while also introducing random elements to ensure an ample exploration of the solution space. A multiple start approach to forming the population of initial guesses provides additional improvements. The choice of parameters indicating the influence of random elements appears to be essential for good performance of the algorithm, being probably the major drawback together with the fact that unlike Dijkstra's algorithms or the A*-algorithm, the found solution must not necessarily be the global optimum.

Ant colony optimization

In contrast to the approaches presented so far, [30] applies a metaheuristic to solve an optimal control problem for a relatively detailed model of a surface vessel. This approach is based on an algorithm called "Ant colony optimization" and can account for obstacles, but does not consider a current field.

It is based on a discretization of the surrounding into a grid, but nonetheless generates continuous vehicle trajectories. This is achieved in a way that resembles the procedure presented in [12]: starting at one node, control inputs (heading and velocity) are set to some discrete value and the system's equations are integrated until the trajectory crosses the boundary of a grid cell. Then, the control inputs are changed according to some rules and the integration is continued. Obviously, the choice of the control inputs at the grid cell boundaries determines whether the destination is reached or not and whether the path fulfills the optimality criterion.

Therefore, the core part of the algorithm are the rules to determine these inputs.

The general idea is to continuously attempt to create different paths while the algorithm runs. Each path is evaluated and assigned a fitness value to in a way that allows to determine which control input chosen in which cell has lead to a promising path. These values will be preferred subsequently, so the surrounding of promising paths is more likely to be explored. This scheme does not guarantee an optimal solution and can get caught in local minima, but as the authors show, performs well in several scenarios.

Comparison of metaheuristics

In [31] and [32], a variety of metaheuristic algorithms is applied to the optimal control problem of weighted time-energy path planning for an underwater vehicle, including Genetic algorithms, Memetic algorithms, Particle Swarm Optimization, Ant colony optimization and Shuffled Frog Leap optimization. The obtained results are compared to those computed using a conjugate gradient penalty method that incorporates boundary conditions using penalty functions and then tries to obtain a solution using a gradient descent algorithm. However, the quality of a solution is not quantified. Also, ocean currents are neglected.

2.3.5 Sampling-based algorithms

As [8] states, due to the extreme difficulty of planning under differential constraints "the overwhelming majority of solution techniques [in robotic motion planning] are sampling-based." In the following, algorithms based on Rapidly-exploring Random Trees will be considered.

Rapidly-exploring Random Tree (RRT)

Having been introduced in [33] and further analyzed in [34, 35], Rapidly-exploring Random Tree is a rather young algorithm. However, as [36] remarks,

it has shown to work well in practice and several modifications exist that aim at improving some drawbacks. Even though being applicable to planning without differential constraints, the algorithm was intended for considering these constraints, even in higher-dimensional state spaces, from the beginning on.

As described in [33], the general idea is to construct a search tree that equally covers the state space, whereby the nodes of the tree correspond to states in state space and the edges to trajectories between the adjacent nodes. A sequence of connected edges that reaches from the initial state to the destination state is a solution to the motion planning problem.

Initially, the tree only contains the initial state and is extended incrementally by continuously selecting random points from the state space, based on uniform sampling. Then, the tree is grown towards these points, resulting in new edges and nodes. The way the tree is extended towards the sampled point is crucial: an attempt to exactly connect two states is equivalent to solving a two-point boundary value problem, what might not be possible nor computationally efficient as will be discussed further in chapter 3. Therefore, the original RRT-algorithm selects the point from the existing tree that is closest to the sampled point (measured in an appropriate metric in the state space) and then integrates a state transition equation of the form (2.1) from there over a fixed time interval, using a control input that minimizes the distance between the end point of the found trajectory and the sampled point, but does not necessarily end up at the sampled point. If the edge does not intersect with any obstacles, its end point is added to the graph. In doing so, over time a uniform covering of the state space is achieved, yielding possible trajectories to the destination state.

Due to the random nature of the sampling, the algorithm is not complete as e.g. Dijkstra's algorithm, which always returns a solution in finite time if one exists

or terminates otherwise. However, under very general assumptions, this algorithm is probabilistically complete, i.e. with enough run-time, the probability of finding a solution converges to one [8]. To the knowledge of the author, only one attempt using the RRT-algorithm for path planning for unmanned ocean vehicles was made in [37]. However, this approach does not exploit the ability of the algorithm to account for differential constraints since it is only concerned with large-scale path planning. Therefore, it is more related to the approaches shown in section 2.2.1.

RRT*

Despite of its advantages, the RRT-algorithm suffers from the drawback that it is not optimal, i.e. it achieves to generate some path and also tries to improve this path subsequently, but most likely will never find an even close to optimal path. To address this issue, an improved version, called RRT*, was developed in [38], further analyzed in [39] and extended to be able to handle differential constraints in [36]. The contribution of the RRT* algorithm is to not simply generate a new state and assign the closest existing state of the tree to it as a parent, but to find it a parent node within a certain radius that minimizes its cost.

In detail, this works as follows: having sampled a random state, at first a trajectory from the closest existing node of the tree is extended towards the sample, where the end state of the resulting trajectory becomes a new node. Then, all nodes lying within a radius around this node are determined. From these nodes, it is attempted to generate trajectories to the newly found node, whereby the start node of the trajectory yielding minimum cost becomes the parent of the newly found node. Following that step, it is also examined if making the newly generated node a parent of one of the nearby nodes would reduce the cost of those. If so, the respective node is connected to the new node, while its old connection to the tree is discarded. This procedure is referred to as "rewiring."

It can be proven that this extension makes the algorithm asymptotically optimal when general conditions apply, so it "converges to an optimal solution almost surely as the number of samples approaches infinity [36]." It is also claimed that the computational efficiency of the original RRT-algorithm is preserved.

Even though no application to the problem of path planning for autonomous ocean vehicles could be found, there exist some applications to other dynamic systems:

In [36], three systems are considered, namely a system similar to Dubins car as discussed earlier in section 2.3.1, a double integrator and a combination of these two. All these systems share the property that there exist steering-procedures, i.e. it is not necessary to actually solve a two-point boundary value problem since the solution for connecting two states can be derived analytically in advance, so this result only has to be applied for each trajectory generation. Therefore, these results cannot be generalized to more complex systems for which such a steering procedure is not known.

However, in [40] the algorithm is applied to the problem of calculating time-optimal maneuvers for an off-road vehicle represented by a dynamic system with 8 degrees of freedom. For that purpose, the algorithm is extended such that it can handle more complex dynamical systems and higher-dimensional state spaces. This is achieved by relaxing the requirement to exactly connect two states by a trajectory. Now, trajectories that end in a predefined vicinity of a specific node are accepted as approximate solution to the boundary value problem. If a newly added node becomes the parent of an already existing node as a result of the rewiring procedure described above, the imprecision connected with this approximate solution requires to recalculate the trajectories to the children of that node using the stored control inputs.

In order to handle higher-dimensional state spaces computationally more efficiently, planning is conducted in the so-called "task space." This is a space of lower dimensionality than the original state space, where sampling and distance-checking takes place, while the tree is still grown in the original state space.

Sparse RRT

Even though the results presented in [40] appear promising, [3] attempts to improve several aspects, leading to the Sparse RRT algorithm. More precisely, it avoids the problematic dependency on solving two-point boundary value problems and also maintains a sparse data structure, allowing for faster queries, which will be explained later in chapter 4.

In [3], several dynamic systems are examined and if possible compared to the other, above mentioned versions of RRT-based algorithms. It seems that the presented extensions actually improve the algorithm.

2.4 Conclusion and implications for the presented thesis

In the preceding sections, a variety of different approaches to the problem of path planning for unmanned ocean vehicles and similar applications was presented. As described, each of these methods revealed different advantages as well as drawbacks. Despite of the diversity of the applied methods, a general tendency can be identified:

Often, it is assumed that dynamic effects like inertia or interaction between vehicle and water can be neglected. This is justified by the fact that the length of the calculated path is much longer then the dimensions of the vehicle, so the dynamics would not lead to a drastic large-scale difference between the planned path and the path taken by the vehicle. Only the optimal control based methods [22, 23, 25, 30–32] and the modified A* approach in [12] do not make this

assumption and try to implement at least a kinematic or even a dynamic vehicle model. Other approaches apply path-smoothing techniques to account at least for a turning radius of a vehicle.

In order to plan a path for an unmanned ocean surface vessel which is to operate only within a range of several kilometers or even fewer, it has to be assumed that the vehicle's dynamics will lead to a significant difference in the path taken by the vehicle and the prescribed path, so paths have to be planned considering these aspects.

The use of kinematic models may produce a smoother path than applying e.g. unmodified graph searching techniques, but still might not be a good nominal trajectory that can be followed by a vehicle in a real application without large deviations. Also, the presented kinematic models all rely only on the heading angle as control input, not providing room for energy considerations in planning because those might require to change the vehicle's speed.

The application of path-smoothing might also generate more realistic trajectories, but still is more an approximate approach based on the maximum and not the actual turning radius of a vehicle, completely neglecting changes in the vehicle's velocity.

Thus, only the optimal control approaches considering a dynamic vehicle model in [22, 23, 30–32] can provide a really appropriate solution to the path planning problem. RRT-based algorithms as described in section 2.3.5 might be able to solve the problem, but have not been applied to ocean vehicle path-planning under differential constraints.

In [22], a very detailed dynamical model is used, however, convergence seems to be problematic and the procedure to include obstacles appears cumbersome. In constrast, [23] uses a quite simple dynamic model, but is also based on the

application of sophisticated nonlinear optimization techniques and software. This might not be available to all users. In addition, an obstacle avoidance scheme is not described. In [30], planning is based on a really detailed vehicle model and can avoid obstacles, but neglects currents. The approaches presented in [31, 32] also do not account for currents and in addition seem to make use of solution vectors of fixed length during the optimization process. This introduces the problem that long paths might have a poorer quality than shorter path, since for both, the same number of control inputs is used, complicating it for longer paths to adapt to the environment.

This general impression has been strengthened by tests performed for own implementations of both an A* algorithm similar to the one presented in [10] and also an Ant Colony Optimization algorithm as presented in [30]. It turned out that in addition to neglecting any dynamic effect, the shape of paths generated by the A* algorithm is highly limited by the grid resolution.

The Ant Colony Optimization algorithm, even though capable of accounting for realistic dynamic effects, appeared to be highly depending on the heuristic for the heading angle. On the one hand, this limits the generality of the algorithm because for every scenario, these heuristic values have to be provided by the user. On the other hand, if the algorithm would be extended by current fields, the heuristic would become inappropriate. This is due to the fact that it basically aims at pointing out the direction of the shortest path to the destination, avoiding obstacles. Now, if current fields are present, the shortest path is not necessarily the fastest path.

Based on all these insights, it is desirable to employ a procedure based on an optimal control approach that provides enough flexibility to take into account all necessary restrictions, amongst others dynamic effects, obstacles, currents and en-

ergy limitations. Concluding from the good performance described in section 2.3.5, RRT-based algorithms and especially Sparse RRT as developed in [3] appear to be promising. Therefore, the remaining sections will focus on this algorithm and optimal control theory, to which some theoretic foundations will be described in the following chapter.

CHAPTER 3

Optimal control

In the following chapter, an overview of basic optimal control theory will be given to facilitate the formulation of the considered problem and some benchmarks. Most derivations follow the work of [2, 21]. In addition, common solution procedures will be listed and the implications of several types of constraints will be discussed.

3.1 The general optimal control problem

According to [2, p. 3], the objective of optimal control theory is "to determine the control signals that will cause a process to satisfy the physical constraints and at the same time minimize (or maximize) some performance criterion." In mathematical terms, an optimal control problem can be stated as follows:

Find an optimal control function $\boldsymbol{u}^*(t)$ that causes the dynamic system

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}\left(\boldsymbol{x}(t), \boldsymbol{u}(t), t\right) \tag{3.1}$$

to follow the optimal trajectory $\boldsymbol{x}^*(t)$, minimizing the functional

$$J(\boldsymbol{u}) = h\left(\boldsymbol{x}(t_f), t_f\right) + \int_{t_0}^{t_f} g\left(\boldsymbol{x}(t), \boldsymbol{u}(t), t\right)\, \mathrm{d}t \tag{3.2}$$

while accounting for all constraints. In dependency of the appearance of the functional $J(\boldsymbol{u})$, three different problem types can be distinguished:

Bolza problem: $h \neq 0$ and $g \neq 0$

Mayer problem: $h \neq 0$ and $g = 0$

Lagrange problem: $h = 0$ and $g \neq 0$.

3.2 Necessary conditions for optimality

3.2.1 Minimization of a functional

Subsequently, necessary conditions for the minimization of the functional $J(\boldsymbol{u})$ under different conditions will be derived. Following [2, p. 109], a functional is "a rule of correspondence that assigns to each function $\boldsymbol{x}$ in a certain class $\boldsymbol{\Omega}$ a unique real number. $\boldsymbol{\Omega}$ is called the domain of the functional, and the set of real numbers associated with the functions in $\boldsymbol{\Omega}$ is called the range of the functional."

In order to determine extreme values of a functional, the concept of an increment has to be introduced [2]:
If $\boldsymbol{x}$ and $\boldsymbol{x}+\delta\boldsymbol{x}$ are functions for which the functional $J(\boldsymbol{x}, \boldsymbol{x}+\delta\boldsymbol{x})$ is defined, then the increment of J is denoted by

$$\Delta J(\boldsymbol{x}, \boldsymbol{x}+\delta\boldsymbol{x}) = J(\boldsymbol{x}+\delta\boldsymbol{x}) - J(\boldsymbol{x}) \ , \tag{3.3}$$

where $\delta\boldsymbol{x}$ denotes the variation of the function $\boldsymbol{x}$, being a curve in the vicinity of $\boldsymbol{x}$ as measured by a suitable norm. Thus, the increment of J can be seen as the change in J, caused by the variation $\delta\boldsymbol{x}$.
If the variation $\delta\boldsymbol{x}$ is small, the increment can be written as

$$\begin{aligned} \Delta J(\boldsymbol{x}, \boldsymbol{x}+\delta\boldsymbol{x}) &= J(\boldsymbol{x}+\delta\boldsymbol{x}) - J(\boldsymbol{x}) && (3.4)\\ &\approx J(\boldsymbol{x}) + \frac{\partial}{\partial \boldsymbol{x}} J(\boldsymbol{x})\delta\boldsymbol{x} + \text{t.h.o} - J(\boldsymbol{x}) && (3.5)\\ &\approx \delta J(\boldsymbol{x}, \delta\boldsymbol{x}) \ , && (3.6) \end{aligned}$$

approximating the increment by the variation δJ of J. The variation δJ is the change in J that is linear in the variation $\delta\boldsymbol{x}$, while terms of higher order (t.h.o.) are being discarded.

Now, according to [2], a functional J with domain $\boldsymbol{\Omega}$ has a relative extremum at $\boldsymbol{x}^*$ if there is an $\epsilon > 0$ such that for all functions $\boldsymbol{x}$ in $\boldsymbol{\Omega}$ which satisfy $||\boldsymbol{x}-\boldsymbol{x}^*|| < \epsilon$ the increment of J has the same sign. This leads to the Fundamental Theorem of

the Calculus of Variations for functions that are not constrained on $\boldsymbol{\Omega}$:

If $\boldsymbol{x}^*$ is an extremal, the variation of J must vanish on $\boldsymbol{x}^*$ so that

$$\delta J(\boldsymbol{x}^*, \delta\boldsymbol{x}) = 0 \tag{3.7}$$

for all admissible $\delta\boldsymbol{x}$. A variation is said to be admissible if $\boldsymbol{x} + \delta\boldsymbol{x}$ is a member of the class $\boldsymbol{\Omega}$. For a functional

$$J(\boldsymbol{x}) = \int_{t_0}^{t_f} g\left(\boldsymbol{x}(t), \dot{\boldsymbol{x}}(t), t\right)) \ \mathrm{d}t \tag{3.8}$$

of several independent functions $\boldsymbol{x}(t)$ with fixed start and end points t_0 and t_f as well as states $\boldsymbol{x}_0$ and $\boldsymbol{x}_f$, this implies that

$$\begin{aligned}\delta J\left(\boldsymbol{x}, \delta\boldsymbol{x}, \dot{\boldsymbol{x}}, \delta\dot{\boldsymbol{x}}\right) = \int_{t_0}^{t_f} &\left[\frac{\partial g\left(\boldsymbol{x}(t), \dot{\boldsymbol{x}}(t), t\right)}{\partial \boldsymbol{x}}\right]^T \delta\boldsymbol{x} \\ &+ \left[\frac{\partial g\left(\boldsymbol{x}(t), \dot{\boldsymbol{x}}(t), t\right)}{\partial \dot{\boldsymbol{x}}}\right]^T \delta\dot{\boldsymbol{x}} \ \mathrm{d}t = 0 \ .\end{aligned} \tag{3.9}$$

The second term in the integral can be integrated by parts

$$\begin{aligned}\int_{t_0}^{t_f} \left[\frac{\partial g\left(\boldsymbol{x}(t), \dot{\boldsymbol{x}}(t), t\right)}{\partial \dot{\boldsymbol{x}}}\right]^T \delta\dot{\boldsymbol{x}} \ \mathrm{d}t &= \left[\frac{\partial g\left(\boldsymbol{x}(t), \dot{\boldsymbol{x}}(t), t\right)}{\partial \dot{\boldsymbol{x}}}\delta\boldsymbol{x}\right]_{t_0}^{t_f} \\ &\quad - \int_{t_0}^{t_f} \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial g\left(\boldsymbol{x}(t), \dot{\boldsymbol{x}}(t), t\right)}{\partial \dot{\boldsymbol{x}}}\delta\boldsymbol{x} \ \mathrm{d}t \\ &= -\int_{t_0}^{t_f} \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial g\left(\boldsymbol{x}(t), \dot{\boldsymbol{x}}(t), t\right)}{\partial \dot{\boldsymbol{x}}}\delta\boldsymbol{x} \ \mathrm{d}t \ ,\end{aligned} \tag{3.10}$$

which leads to

$$\int_{t_0}^{t_f} \left(\frac{\partial g\left(\boldsymbol{x}(t), \dot{\boldsymbol{x}}(t), t\right)}{\partial \boldsymbol{x}} - \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial g\left(\boldsymbol{x}(t), \dot{\boldsymbol{x}}(t), t\right)}{\partial \dot{\boldsymbol{x}}}\right)^T \delta\boldsymbol{x} \ \mathrm{d}t = 0 \ . \tag{3.11}$$

The Fundamental Lemma of the Calculus of Variations now states that if a function $h(t)$ is continuous on an interval and

$$\int_{t_0}^{t_f} h(t)\delta x(t) \ \mathrm{d}t = 0 \tag{3.12}$$

for every variation $\delta x(t)$ that is continuous on the interval, then $h(t)$ must be zero. This means that

$$\frac{\partial g\left(\boldsymbol{x}(t), \dot{\boldsymbol{x}}(t), t\right)}{\partial \boldsymbol{x}} - \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial g\left(\boldsymbol{x}(t), \dot{\boldsymbol{x}}(t), t\right)}{\partial \dot{\boldsymbol{x}}} = \mathbf{0} \ . \tag{3.13}$$

Equation 3.13 is usually referred to as Euler equation, being a necessary condition for extremality of a function.

3.2.2 Final state and final time

So far, the final time t_f and the final states $\boldsymbol{x}_f$ were assumed to be fixed. In the following, the effect of free final time and final states will be considered. At first, the increment of the functional J has to be formed, taking into account that now the final time and the states at that time are not fixed, but also subject to variations:

$$\begin{aligned} \Delta J &= \int_{t_0}^{t_f+\delta t_f} g\left(\boldsymbol{x}(t), \dot{\boldsymbol{x}}(t), t\right)\ \mathrm{d}t - \int_{t_0}^{t_f} g\left(\boldsymbol{x}^*(t), \dot{\boldsymbol{x}}^*(t), t\right)\ \mathrm{d}t \qquad (3.14)\\ &= \int_{t_0}^{t_f} g\left(\boldsymbol{x}(t), \dot{\boldsymbol{x}}(t), t\right) - g\left(\boldsymbol{x}^*(t), \dot{\boldsymbol{x}}^*(t), t\right)\ \mathrm{d}t + \int_{t_f}^{t_f+\delta t_f} g\left(\boldsymbol{x}(t), \dot{\boldsymbol{x}}(t), t\right)\ \mathrm{d}t \ . \end{aligned}$$

Assuming that all curves $\boldsymbol{x}, \dot{\boldsymbol{x}}$ contained in the first term of the first integral lie in the vicinity of the extremal, this can be written as

$$\begin{aligned} \Delta J = &\int_{t_0}^{t_f} g\left(\boldsymbol{x}^*(t) + \delta\boldsymbol{x}(t), \dot{\boldsymbol{x}}^*(t) + \delta\dot{\boldsymbol{x}}(t), t\right)\ \mathrm{d}t + \int_{t_f}^{t_f+\delta t_f} g\left(\boldsymbol{x}(t), \dot{\boldsymbol{x}}(t), t\right)\mathrm{d}t \\ &- \int_{t_0}^{t_f} g\left(\boldsymbol{x}^*(t), \dot{\boldsymbol{x}}^*(t), t\right)\ \mathrm{d}t \ . \end{aligned} \tag{3.15}$$

Taylor series expansion leads to cancelation of the last integral term, resulting in

$$\begin{aligned} \Delta J = &\int_{t_0}^{t_f} \left\{ \left[\frac{\partial g\left(\boldsymbol{x}^*(t), \dot{\boldsymbol{x}}^*(t), t\right)}{\partial \boldsymbol{x}}\right]^T \delta\boldsymbol{x}(t) + \left[\frac{\partial g\left(\boldsymbol{x}^*(t), \dot{\boldsymbol{x}}^*(t), t\right)}{\partial \dot{\boldsymbol{x}}}\right]^T \delta\dot{\boldsymbol{x}}(t) \right\} \mathrm{d}t \\ &+ \text{t.h.o.} + \int_{t_f}^{t_f+\delta t_f} g\left(\boldsymbol{x}(t), \dot{\boldsymbol{x}}(t), t\right)\ \mathrm{d}t \ . \end{aligned} \tag{3.16}$$

Integration by parts of the expression containing $\delta\dot{\boldsymbol{x}}(t)$ yields

$$\int_{t_0}^{t_f} \frac{\partial g\left(\boldsymbol{x}^*(t), \dot{\boldsymbol{x}}^*(t), t\right)}{\partial \dot{\boldsymbol{x}}} \delta\dot{\boldsymbol{x}}(t)\ \mathrm{d}t = \left[\frac{\partial g\left(\boldsymbol{x}^*(t), \dot{\boldsymbol{x}}^*(t), t\right)}{\partial \dot{\boldsymbol{x}}} \delta\boldsymbol{x}(t)\right]_{t_0}^{t_f} - \int_{t_0}^{t_f} \frac{\mathrm{d}}{\mathrm{d}t} \frac{\partial g\left(\boldsymbol{x}^*(t), \dot{\boldsymbol{x}}^*(t), t\right)}{\partial \dot{\boldsymbol{x}}} \delta\boldsymbol{x}(t)\ \mathrm{d}t\ . \tag{3.17}$$

Now, $\delta\boldsymbol{x}(t_0) = 0$ since the initial state is fixed. Therefore,

$$\begin{aligned}\Delta J = &\int_{t_0}^{t_f} \left\{ \frac{\partial g\left(\boldsymbol{x}^*(t), \dot{\boldsymbol{x}}^*(t), t\right)}{\partial \boldsymbol{x}} - \left[\frac{\mathrm{d}}{\mathrm{d}t} \frac{\partial g\left(\boldsymbol{x}^*(t), \dot{\boldsymbol{x}}^*(t), t\right)}{\partial \dot{\boldsymbol{x}}}\right] \right\}^T \delta\boldsymbol{x}(t)\ \mathrm{d}t \\ &+ \left[\frac{\partial g\left(\boldsymbol{x}^*(t_f), \dot{\boldsymbol{x}}^*(t_f), t_f\right)}{\partial \dot{\boldsymbol{x}}}\right]^T \delta\boldsymbol{x}(t_f) + \int_{t_f}^{t_f+\delta t_f} g\left(\boldsymbol{x}(t), \dot{\boldsymbol{x}}(t), t\right)\ \mathrm{d}t \\ &+ \text{t.h.o.}\ .\end{aligned} \tag{3.18}$$

At this point, the last integral can be expressed as

$$\int_{t_f}^{t_f+\delta t_f} g\left(\boldsymbol{x}(t), \dot{\boldsymbol{x}}(t), t\right)\ \mathrm{d}t \approx g\left(\boldsymbol{x}(t_f), \dot{\boldsymbol{x}}(t_f), t_f\right) \delta t_f + \text{t.h.o.}\ , \tag{3.19}$$

leading to

$$\begin{aligned}\Delta J = &\int_{t_0}^{t_f} \left\{ \frac{\partial g\left(\boldsymbol{x}^*(t), \dot{\boldsymbol{x}}^*(t), t\right)}{\partial \boldsymbol{x}} - \left[\frac{\mathrm{d}}{\mathrm{d}t} \frac{\partial g\left(\boldsymbol{x}^*(t), \dot{\boldsymbol{x}}^*(t), t\right)}{\partial \dot{\boldsymbol{x}}}\right] \right\}^T \delta\boldsymbol{x}(t)\ \mathrm{d}t \\ &+ \left[\frac{\partial g\left(\boldsymbol{x}^*(t_f), \dot{\boldsymbol{x}}^*(t_f), t_f\right)}{\partial \dot{\boldsymbol{x}}}\right]^T \delta\boldsymbol{x}(t_f) + g\left(\boldsymbol{x}(t_f), \dot{\boldsymbol{x}}(t_f), t_f\right) \delta t_f + \text{t.h.o.}\ .\end{aligned} \tag{3.20}$$

The second to last term again can be expanded to give

$$\begin{aligned}g\left(\boldsymbol{x}(t_f), \dot{\boldsymbol{x}}(t_f), t_f\right) = &\ g\left(\boldsymbol{x}^*(t_f), \dot{\boldsymbol{x}}^*(t_f), t_f\right) \\ &+ \left[\frac{\partial g\left(\boldsymbol{x}^*(t_f), \dot{\boldsymbol{x}}^*(t_f), t_f\right)}{\partial \boldsymbol{x}}\right]^T \delta\boldsymbol{x}(t_f) \\ &+ \left[\frac{\partial g\left(\boldsymbol{x}^*(t_f), \dot{\boldsymbol{x}}^*(t_f), t_f\right)}{\partial \dot{\boldsymbol{x}}}\right]^T \delta\dot{\boldsymbol{x}}(t_f) \\ &+ \text{t.h.o.}\ .\end{aligned} \tag{3.21}$$

Since $\delta t_f \delta\boldsymbol{x}(t_f)$ and $\delta t_f \delta\boldsymbol{x}(t_f)$ are of higher order, the increment finally becomes

$$\begin{aligned}\Delta J = &\int_{t_0}^{t_f} \left\{ \frac{\partial g\left(\boldsymbol{x}^*(t), \dot{\boldsymbol{x}}^*(t), t\right)}{\partial \boldsymbol{x}} - \left[\frac{\mathrm{d}}{\mathrm{d}t} \frac{\partial g\left(\boldsymbol{x}^*(t), \dot{\boldsymbol{x}}^*(t), t\right)}{\partial \dot{\boldsymbol{x}}}\right] \right\}^T \delta\boldsymbol{x}(t)\ \mathrm{d}t \\ &+ \left[\frac{\partial g\left(\boldsymbol{x}^*(t_f), \dot{\boldsymbol{x}}^*(t_f), t_f\right)}{\partial \dot{\boldsymbol{x}}}\right]^T \delta\boldsymbol{x}(t_f) + g\left(\boldsymbol{x}^*(t_f), \dot{\boldsymbol{x}}^*(t_f), t_f\right) \delta t_f + \text{t.h.o.}\ .\end{aligned} \tag{3.22}$$

Now, $\delta \boldsymbol{x}_f$ is not independent but depends on both δt_f and $\delta \boldsymbol{x}_f$ according to

$$\delta \boldsymbol{x}_f = \delta \boldsymbol{x}(t_f) + \dot{\boldsymbol{x}}^*(t_f)\delta t_f \tag{3.23}$$

$$\delta \boldsymbol{x}(t_f) = \delta \boldsymbol{x}_f - \dot{\boldsymbol{x}}^*(t_f)\delta t_f \ . \tag{3.24}$$

Introducing this into (3.22) and discarding all terms of higher order, retaining only those linear in the variations, the variation of J becomes

$$\begin{aligned}\delta J = &\int_{t_0}^{t_f} \left\{ \frac{\partial g\left(\boldsymbol{x}^*(t), \dot{\boldsymbol{x}}^*(t), t\right)}{\partial \boldsymbol{x}} - \left[\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial g\left(\boldsymbol{x}^*(t), \dot{\boldsymbol{x}}^*(t), t\right)}{\partial \dot{\boldsymbol{x}}}\right] \right\}^T \delta \boldsymbol{x}(t)\ \mathrm{d}t \\ &+ \left\{ g\left(\boldsymbol{x}^*(t_f), \dot{\boldsymbol{x}}^*(t_f), t_f\right) - \left[\frac{\partial g\left(\boldsymbol{x}^*(t_f), \dot{\boldsymbol{x}}^*(t_f), t_f\right)}{\partial \dot{\boldsymbol{x}}}\right]^T \dot{\boldsymbol{x}}^*(t_f) \right\} \delta t_f \\ &+ \left[\frac{\partial g\left(\boldsymbol{x}^*(t_f), \dot{\boldsymbol{x}}^*(t_f), t_f\right)}{\partial \dot{\boldsymbol{x}}}\right]^T \delta \boldsymbol{x}_f \ .\end{aligned} \tag{3.25}$$

As derived before, the variation must equal zero in order to be a candidate for an extremum. Also, the Euler equation (3.13) must be satisfied. This can be seen from the following reasoning: supposed one is given a curve $\boldsymbol{x}^*$, being an extremal to the problem with free final time t_f and free final states $\boldsymbol{x}_f$. Then, this curve obviously also is an extremal to a problem where that final time and those final states were given in advance as boundary conditions. Therefore, since the Euler equation has to hold for the problem with fixed final conditions, this is also true for the problem with free final conditions. For (3.25) this implies that

$$\begin{aligned}\delta J = &\left\{ g\left(\boldsymbol{x}^*(t_f), \dot{\boldsymbol{x}}^*(t_f), t_f\right) - \left[\frac{\partial g\left(\boldsymbol{x}^*(t_f), \dot{\boldsymbol{x}}^*(t_f), t_f\right)}{\partial \dot{\boldsymbol{x}}}\right]^T \dot{\boldsymbol{x}}^*(t_f) \right\} \delta t_f \\ &+ \left[\frac{\partial g\left(\boldsymbol{x}^*(t_f), \dot{\boldsymbol{x}}^*(t_f), t_f\right)}{\partial \dot{\boldsymbol{x}}}\right]^T \delta \boldsymbol{x}_f = 0 \ .\end{aligned} \tag{3.26}$$

This is a general necessary condition that has to be fulfilled by an extremal curve. Making appropriate substitutions to this equation, necessary conditions for more specific problems with e.g. fixed final time or fixed final states can be generated. If e.g. the final states $\boldsymbol{x}_f$ are fixed, then there will not be a variation of them, so

$\delta \boldsymbol{x}_f = 0$. It could also occur that only some final states are fixed and some are not. Then, the variation of the vector $\boldsymbol{x}_f$ has to be split up, yielding zero for fixed states and unspecified values for the remaining free states.

3.3 Constraints

The previous section was concerned with general necessary conditions for the optimization of a functional. In the following, necessary conditions will be derived for cases where different constraints apply. Namely, these are differential equation constraints, state and control constraints.

3.3.1 Differential constraints

So far, only necessary conditions for finding an extremum of a functional under different boundary conditions have been derived. In optimal control problems, additional constraints arise, first of all due to the set of differential equations describing the dynamic system to be controlled. In the following, necessary conditions will be derived taking these differential equations into account. In doing so, it will be assumed that one is given a Lagrange problem, that is an extremum of the functional

$$J(\boldsymbol{u}) = \int_{t_0}^{t_f} g(\boldsymbol{x}(t), \boldsymbol{u}(t), t)\ \mathrm{d}t \tag{3.27}$$

subject to

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}\left(\boldsymbol{x}(t), \boldsymbol{u}(t), t\right) \tag{3.28}$$

is to be found. A common approach to take constraints into account is to simply add them to the functional to be minimized using Lagrange multipliers. Because the constraints have to hold at every time $t \in [t_0, t_f]$, the Lagrange multipliers are functions of time. Applying this yields the so-called "augmented functional"

$$J_a(\boldsymbol{u}) = \int_{t_0}^{t_f} \left\{ g(\boldsymbol{x}(t), \boldsymbol{u}(t), t) + \boldsymbol{p}^T(t) \left[\boldsymbol{f}\left(\boldsymbol{x}(t), \boldsymbol{u}(t), t\right) - \dot{\boldsymbol{x}}(t) \right] \right\}\ \mathrm{d}t\ . \tag{3.29}$$

Defining

$$g_a\left(\boldsymbol{x}(t), \dot{\boldsymbol{x}}(t), \boldsymbol{u}(t), \boldsymbol{p}(t), t\right) = g(\boldsymbol{x}(t), \boldsymbol{u}(t), t) + \boldsymbol{p}^T(t)\left[\boldsymbol{f}\left(\boldsymbol{x}(t), \boldsymbol{u}(t), t\right) - \dot{\boldsymbol{x}}(t)\right] \tag{3.30}$$

leads to

$$J_a(\boldsymbol{u}) = \int_{t_0}^{t_f} g_a\left(\boldsymbol{x}(t), \dot{\boldsymbol{x}}(t), \boldsymbol{u}(t), \boldsymbol{p}(t), t\right)\ \mathrm{d}t\ . \tag{3.31}$$

In order to find an extremum of this functional, its variation must vanish. This leads to (3.25), with two additional variations $\delta\boldsymbol{u}$ and $\delta\boldsymbol{p}$, so

$$\begin{aligned}
\delta J_a(\boldsymbol{u}^*) = \Bigg\{ & g_a\left(\boldsymbol{x}^*(t), \dot{\boldsymbol{x}}^*(t), \boldsymbol{u}^*(t), \boldsymbol{p}^*(t), t\right) \\
& - \left[\frac{\partial g_a\left(\boldsymbol{x}^*(t), \dot{\boldsymbol{x}}^*(t), \boldsymbol{u}^*(t), \boldsymbol{p}^*(t), t\right)}{\partial \dot{\boldsymbol{x}}}\right]^T \dot{\boldsymbol{x}}^*(t_f)\Bigg\}\delta t_f \\
& + \left[\frac{\partial g_a\left(\boldsymbol{x}^*(t), \dot{\boldsymbol{x}}^*(t), \boldsymbol{u}^*(t), \boldsymbol{p}^*(t), t\right)}{\partial \dot{\boldsymbol{x}}}\right]^T \delta\boldsymbol{x}_f \\
& + \int_{t_0}^{t_f} \Bigg\{ \Bigg[\frac{\partial g_a\left(\boldsymbol{x}^*(t), \dot{\boldsymbol{x}}^*(t), \boldsymbol{u}^*(t), \boldsymbol{p}^*(t), t\right)}{\partial \boldsymbol{x}} \\
& - \left[\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial g_a\left(\boldsymbol{x}^*(t), \dot{\boldsymbol{x}}^*(t), \boldsymbol{u}^*(t), \boldsymbol{p}^*(t), t\right)}{\partial \dot{\boldsymbol{x}}}\right]\Bigg]^T \delta\boldsymbol{x}(t) \\
& + \left[\frac{\partial g_a\left(\boldsymbol{x}^*(t), \dot{\boldsymbol{x}}^*(t), \boldsymbol{u}^*(t), \boldsymbol{p}^*(t), t\right)}{\partial \boldsymbol{u}}\right]^T \delta\boldsymbol{u}(t) \\
& + \left[\frac{\partial g_a\left(\boldsymbol{x}^*(t), \dot{\boldsymbol{x}}^*(t), \boldsymbol{u}^*(t), \boldsymbol{p}^*(t), t\right)}{\partial \boldsymbol{p}}\right] \delta\boldsymbol{p}(t)\Bigg\}\ \mathrm{d}t = 0\ .
\end{aligned} \tag{3.32}$$

Following the same reasoning as before in section 3.2.2, this must be true regardless of the boundary conditions, that is free or fixed final conditions. Therefore, the integral term must be zero:

$$\begin{aligned}
\int_{t_0}^{t_f} \Bigg\{ & \left[\left[\frac{\partial g\left(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t\right)}{\partial \boldsymbol{x}}\right]^T + \boldsymbol{p}^{*T}(t)\left[\frac{\partial \boldsymbol{f}\left(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t\right)}{\partial \boldsymbol{x}}\right] - \frac{\mathrm{d}}{\mathrm{d}t}\left[-\boldsymbol{p}^{*T}(t)\right]\right]\delta\boldsymbol{x}(t) \\
& + \left[\left[\frac{\partial g\left(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t\right)}{\partial \boldsymbol{u}}\right]^T + \boldsymbol{p}^{*T}(t)\left[\frac{\partial \boldsymbol{f}\left(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t\right)}{\partial \boldsymbol{u}}\right]\right]\delta\boldsymbol{u}(t) \\
& + \left[\left[\boldsymbol{f}\left(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t\right) - \dot{\boldsymbol{x}}^*(t)\right]^T\right]\delta\boldsymbol{p}(t)\Bigg\}\ \mathrm{d}t = 0\ .
\end{aligned} \tag{3.33}$$

Now, as before from (3.12), this holds if the term in front of the variation is zero. This also holds for several independent variations. However, because $\delta\boldsymbol{u}(t)$ determines the future evolution of the system, the variation $\delta\boldsymbol{x}(t)$ is not independent so this must not necessarily be true for the term in front of $\delta\boldsymbol{x}(t)$. But, because the Lagrangian multipliers $\boldsymbol{p}(t)$ are arbitrary, they can be selected so that the term becomes zero, so

$$\dot{\boldsymbol{p}}^*(t) = -\left[\frac{\partial \boldsymbol{f}\left(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t\right)}{\partial \boldsymbol{x}}\right]^T \boldsymbol{p}^*(t) - \frac{\partial g\left(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t\right)}{\partial \boldsymbol{x}} \ . \tag{3.34}$$

This expression is referred to as the "costate equations," with $\boldsymbol{p}(t)$ being the costates. It implies that the costates also depend on the control $\boldsymbol{u}(t)$. Nonetheless, the term in front of $\delta\boldsymbol{p}(t)$ must be zero to fulfill the original differential equation constraint (3.28).

The terms outside of the integral must also add to zero if this holds for the integral term. Evaluating those terms it follows that

$$0 = \left[g\left(\boldsymbol{x}^*(t_f), \boldsymbol{u}^*(t_f), t_f\right) + \boldsymbol{p}^{T*}(t_f)\left[\boldsymbol{f}\left(\boldsymbol{x}^*(t_f), \boldsymbol{u}^*(t_f), t_f\right)\right]^T\right]\delta t_f - \boldsymbol{p}^{T*}(t_f)\delta\boldsymbol{x}_f \ . \tag{3.35}$$

Introducing the so-called Hamiltonian in the style of the one known from mechanics,

$$\mathcal{H}\left(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{p}(t), t\right) = g\left(\boldsymbol{x}(t), \boldsymbol{u}(t), t\right) + \boldsymbol{p}(t)\left[\boldsymbol{f}\left(\boldsymbol{x}(t), \boldsymbol{u}(t), t\right)\right] \ , \tag{3.36}$$

these conditions can be written as

$$\boldsymbol{x}^*(t) = \frac{\partial \mathcal{H}}{\partial \boldsymbol{p}}\left(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), \boldsymbol{p}^*(t), t\right) \tag{3.37}$$

$$\boldsymbol{p}^*(t) = -\frac{\partial \mathcal{H}}{\partial \boldsymbol{x}}\left(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), \boldsymbol{p}^*(t), t\right) \tag{3.38}$$

$$\boldsymbol{0} = \frac{\partial \mathcal{H}}{\partial \boldsymbol{u}}\left(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), \boldsymbol{p}^*(t), t\right) \tag{3.39}$$

with the boundary conditions

$$0 = \mathcal{H}\left(\boldsymbol{x}^*(t_f), \boldsymbol{u}^*(t_f), \boldsymbol{p}^*(t_f), t_f\right)\delta t_f - \boldsymbol{p}^{T*}(t_f)\delta\boldsymbol{x}_f \tag{3.40}$$

which have to be adapted to the specific case. So, generally, provided that the initial states $\boldsymbol{x}_0$ and time t_0 are given, the solution of the optimal control problem requires to solve a two-point boundary value problem.

3.3.2 Control constraints

Regardless of any constraints,

$$\Delta J = J(\boldsymbol{u}) - J(\boldsymbol{u}^*) \geq 0 \tag{3.41}$$

will always be a necessary condition for a relative minimum. Based on that, for constrained control inputs it can be shown that from (3.37)-(3.39) only the third condition (3.39) changes to

$$\mathcal{H}\left(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), \boldsymbol{p}^*(t), t\right) \leq \mathcal{H}\left(\boldsymbol{x}^*(t), \boldsymbol{u}(t), \boldsymbol{p}^*(t), t\right) \ . \tag{3.42}$$

This is known as Pontryagin's Minimum Principle, stating that the control must minimize the Hamiltonian (also referred to as the maximum principle, depending on the application).

3.3.3 State inequality constraints

State inequality constraints of the form

$$\boldsymbol{a}(\boldsymbol{x}(t), t) \geq \boldsymbol{0} \tag{3.43}$$

can be treated in a similar way as differential constraints. One general idea presented in [2] is to transform an inequality constraint into an equality constraint and then adding this to the objective functional using Lagrange multipliers. The transformation is achieved introducing a new state variable x_{n+1} with

$$\dot{x}_{n+1}(t) = \sum_{i=1}^{l} a_i(\boldsymbol{x}(t), t)^2 \cdot \Theta\left(-f_i\left(\boldsymbol{x}(t), t\right)\right) \ , \tag{3.44}$$

where Θ is the Heaviside function with

$$\Theta\left(-a_i\left(\boldsymbol{x}(t), t\right)\right) = \begin{cases} 0, & \text{for } a_i\left(\boldsymbol{x}(t), t\right) \geq 0 \\ 1, & \text{for } a_i\left(\boldsymbol{x}(t), t\right) < 0 \end{cases} \ . \tag{3.45}$$

Therefore, $\dot{x}_{n+1}(t)$ is always greater than zero or exactly equal to zero if all constraints are satisfied. In order to enforce the latter, the boundary conditions $x_{n+1}(t_0) = 0$ and $x_{n+1}(t_f) = 0$ are imposed on

$$x_{n+1} = \int_{t_0}^{t_f} \dot{x}_{n+1}(t)\ \mathrm{d}t + x_{n+1}(t_0)\ . \tag{3.46}$$

Since the integrand is always greater than or equal to zero, the boundary conditions only can be met if the integrand is zero, that is if all constraints are fulfilled. With a Lagrange multiplier, this state can be added to the differential constraints. The necessary conditions (3.37)-(3.39) must also hold for this new state variable.

3.4 Solution procedures

Since most problems cannot be solved analytically, solution procedures based on numerical methods have to be applied. In general, these can be divided into direct and indirect methods [41].

Indirect methods are based on the evaluation of necessary conditions for an optimal solution as derived in section 3.2. As it was shown, the necessary conditions result in a two-point boundary value problem that has to be solved numerically using procedures like (multiple) shooting or collocation methods. Another possibility is the application of gradient methods that attempt to optimize the Hamiltonian (3.42) by iteratively improving an approximation of the optimal control $\boldsymbol{u}^*(t)$ [42].

The indirect approach leads to accurate results if convergence can be achieved, but suffers from a major drawback [41]: in order to achieve global convergence, a good initial guess has to be supplied that might be difficult to determine. This also includes knowledge of the structure of switching times, i.e. about the time intervals where constraints are active or not.

Direct methods use a discretized version of the problem, enabling the application of well-developed nonlinear optimization techniques like Sequential Quadratic

Programming (SQP). This approach is usually sufficiently robust and easier to formulate, however not as accurate as indirect methods. State constraints can be incorporated more easily. If further refinement is needed, solutions obtained by direct methods can be used as initial guesses for indirect methods.

A variety of mostly commercial software exists that provides means of solving optimal control problems. An overview of common software packages can be found in [43], while [1] provides several examples for solving real-world problems using such software.

3.5 Formulation of the considered problem

Taking all this into account, the problem to be considered in this thesis can be formulated in terms of an optimal control problem. The objective is to plan a time-optimal path for an unmanned ocean surface vessel. In time-optimal planning, the performance criterion simply becomes

$$J = \int_{t_0}^{t_f} \mathrm{d}t \tag{3.47}$$

with the Hamiltonian

$$\mathcal{H}\left(\boldsymbol{x}(t), \boldsymbol{u}(t), t\right) = 1 + \boldsymbol{p}^T(t)\left[\boldsymbol{f}\left(\boldsymbol{x}(t), \boldsymbol{u}(t), t\right)\right] \ , \tag{3.48}$$

where the function $\boldsymbol{f}\left(\boldsymbol{x}(t), \boldsymbol{u}(t), t\right)$ also contains information about the local current field. It is assumed that the initial time t_0 and initial states $\boldsymbol{x}_0$ are given. The final time t_f is unknown, while the final states $\boldsymbol{x}_0$ will be partly or completely fixed (the final position of the vessel must be specified, while final velocity or orientation might not be of interest).

Several constraints apply to this problem: the control inputs (for example thrust and thrust direction) only are allowed to lie within a certain interval

$$\boldsymbol{u}_{min} \leq \boldsymbol{u}(t) \leq \boldsymbol{u}_{max} \ . \tag{3.49}$$

This also holds for some of the state variables, since the path is required to lie within a certain area of a given map. Also, obstacles must be avoided, leading to

$$\boldsymbol{x}_{min} \leq \boldsymbol{x}(t) \leq \boldsymbol{x}_{max} \ . \tag{3.50}$$

If energy limitations are to be considered in planning, a constraint of the type

$$\int_{t_0}^{t_f} e(\boldsymbol{u}(t), t)\mathrm{d}t \leq c \tag{3.51}$$

must be accounted for where c represents the total amount of energy available and the function $e(\boldsymbol{u}(t), t)$ models the instantaneous energy consumption.

3.6 Summary

In the preceding sections, a general survey of basic optimal control theory has been given. Many sophisticated software packages exist that provide tools to solve optimal control problems, however, there exist some drawbacks especially against the background of the considered problem. In general, convergence is a critical point, particularly for indirect methods. As outlined, these have to be provided with a good initial guess of controls, costates and switching times that might not be available. Also, as [8, p.788] states, solution procedures for boundary-value problems are not designed to handle state constraints that arise from the presence of obstacles. Direct procedures in general seem to be more likely to provide a solution, but require additional knowledge in nonlinear optimization and sophisticated software packages which may not be available at reasonable cost.

CHAPTER 4

Analysis of the Sparse RRT algorithm

As outlined in chapter 2, the Sparse RRT algorithm will be used throughout this thesis. In the subsequent chapter, the algorithm, its implementation and its properties will be described in more detail. Also, several test cases are run to evaluate its general behavior and to identify capabilities and limitations.

4.1 General functionality

Just like the other two algorithms described in section 2.3.5, Sparse RRT is a sampling-based algorithm that tries to solve path planning problems for dynamic systems by building a search tree in state space. The nodes of the tree thereby correspond to states, while the edges connecting the nodes correspond to trajectories in state space. The general idea of sampling-based algorithms is to sample a state and try to connect it to the existing tree. The Sparse RRT algorithm, however, works slightly differently because it aims at overcoming the drawback of having to solve a two-point boundary value problem from which RRT* suffers when dealing with complex dynamic systems. It is also able to maintain a sparse data structure, allowing for reduced run-time.

The general functionality of Sparse RRT is visualized in figure 4.1: after having randomly sampled a state, the algorithm checks which node within a certain radius has the lowest cost. This node (or, if none is found, the nearest neighbor of the sampled node) is then propagated, but unlike in the case of RRT* or RRT, it is not attempted to make a connection to the sampled node. Instead, the propagation is based on randomly selected control inputs and propagation durations. If obstacles or other constraints are present, it has to be checked if a newly generated state violates any of them. This mechanism of checking for the node with lowest cost

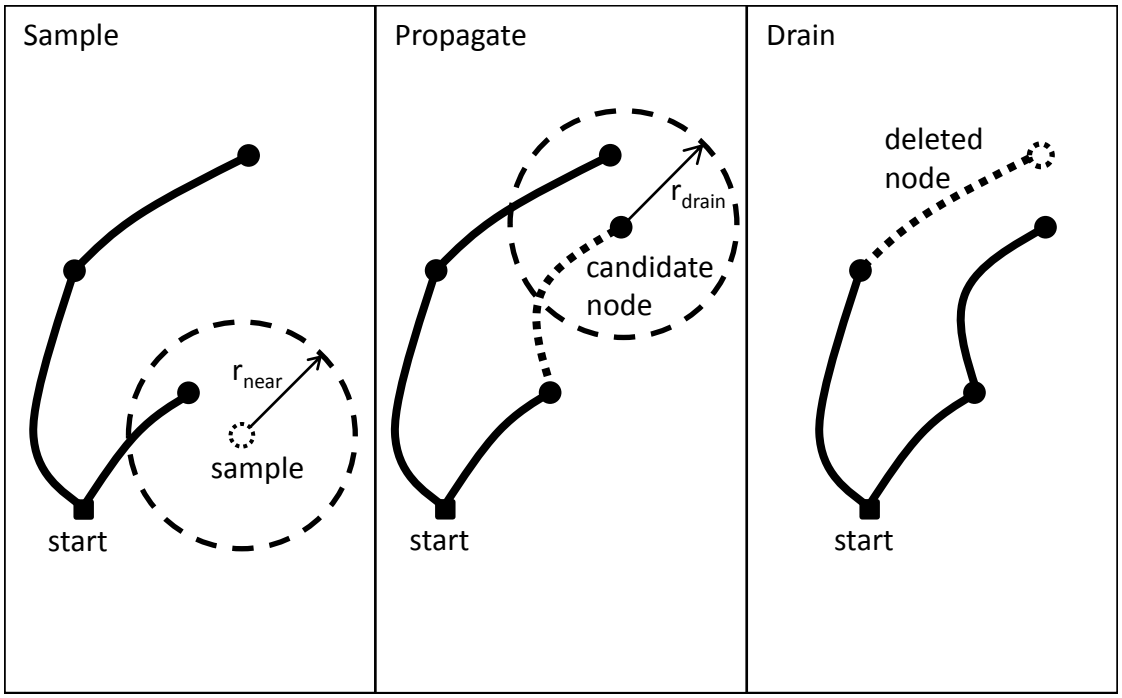

Figure 4.1. General functionality of Sparse RRT

in a defined vicinity, the so-called "Best Nearest" feature, focuses the propagation on low-cost paths.

In order to keep the number of nodes in the tree as low as possible without loosing important information, the algorithm employs what is called "Drain" feature: the cost of adjacent nodes that lie within a certain radius of a newly added node and that are on the so-called "active set" is checked. If no node has lower cost than the newly added one, all other nodes are moved to what is called the "inactive set" or, if they do not have any child nodes, are removed completely. Otherwise, the newly added node will be removed. Being on the inactive set prevents nodes from being selected for future propagation.

4.2 Properties

In general, desirable properties of motion planning algorithms are completeness and optimality, i.e. the guarantee that the algorithm returns the optimal

solution in finite time if one exists. For randomly sampling algorithms, these properties can only hold in an asymptotic sense, that is for an infinite number of samples. RRT was shown to be only probabilistically complete, but almost sure to converge to nonoptimal solutions, while RRT* under general assumptions exhibits both properties. As described in [3], the extensions to RRT that lead to Sparse RRT have two major implications:

Firstly, due to the "Best Nearest" feature, the algorithm becomes asymptotically near-optimal. Not to be mistaken for asymptotical optimality, this means that the probability of finding a solution (if one exists) of which the difference in cost to the optimal solution's cost is bounded approaches one for an infinite number of samples. This bound relies on the number of edges between start and end node, as is proven in [3].

Secondly, the "Drain" feature potentially prevents the algorithm from being probabilistically complete, that is a solution is not always found, even though one exists. The cause for this will be discussed more in detail in section 4.6.

4.3 Implementation

A first version of the algorithm was implemented using MATLAB. However, the unreasonably long run-time turned out to make this approach inapplicable. Therefore, a C# implementation was used which was kindly provided by the authors of [3]. This version is able to solve problems in a more appropriate run-time, most likely due to the speed advantage a compiler language has over an interpreted language as MATLAB. Three aspects which deserve closer attention are the data structure employed by the algorithm for storing the built trajectories, the way the algorithm calculates distances in state space and how the algorithm generates trajectories.

4.3.1 Storing states

For both the "Best Nearest" and the "Drain" feature, Sparse RRT either needs to determine which states lie within a certain vicinity of a selected state or which state is closest to a selected state. Because these types of search have to be performed twice in every iteration, the performance of the algorithm highly hinges on a data structure that allows for fast queries in state space. In the implementation used for this thesis, states are stored in a *kd*-tree.

A *kd*-tree, introduced in [44], is a special form of a binary tree, i.e. a tree-like data structure where starting at a root node, each subsequent node has at most one connection to a parent node and two connections to child nodes. As the name indicates, a *kd*-tree is intended to store k-dimensional data points. The general procedure is to recursively split the available data in two sections along a hyperplane in a different dimension each recursions , whereby each hyperplane is located in a point selected according to a certain strategy. The split is repeated until the bottom of the tree consists of leaves containing only a predefined number of states which are stored in a simple list.

The selection of the split point has a high impact on the balance of the tree, that is the equal distribution of the nodes to the sub trees. This again affects the efficiency of queries. In the implementation of the algorithm at hand, the split point is chosen to lie exactly in the middle between the upper and lower bounds imposed on each dimension. This can be justified against the background of a sampling-based algorithm that eventually will cover the search space evenly and therefore also result in a balanced tree.

The depth of the tree is fixed, i.e. the data is split only a predefined number of times which leads to the fact that the amount of nodes stored in each list at the bottom of the tree must be unlimited if the capacity of the tree is not to be finite.

This, however, will lead to a decrease in efficiency of the algorithm if many nodes are stored due to the exhausting searching of the lists. Therefore, the depth of the tree has to be adapted to the number of nodes which is to be stored.

A *kd*-tree most importantly enables two different query types, the nearest-neighbor search and the range search. While the nearest-neighbor search determines a defined number of nodes respectively states that are closest to the initial point of the query, a range search determines all nodes within a certain range to the reference point.

The general idea for performing a query for the nearest neighbor is presented in [45]: at first, the leave of the tree is accessed that contains the query node. Then, among all other nodes in that leave, the one closest to the query node is determined. Subsequently, adjacent leaves within a radius defined by the closest neighbor found so far have to be searched for closer neighbors.

As described in [46], a range search can be conducted in the following way: at first, it is assumed that for each dimension a range is specified. Then, starting at the root of the tree, for each node the respective range is compared to the split value of the node. Depending on this comparison it can be determined if a node and all its children are within the range or not. By recursively executing this procedure, the set of states within the range can eventually be determined.

4.3.2 Measuring distances in state space

Both the range search and the nearest neighbor search require a way to measure distances in state space, i.e. a metric. As [3] remarks, the best metric would be the cost function to be optimized, however, this value cannot be determined without building a trajectory between two states, leading to a two-point boundary value problem that Sparse RRT was designed to avoid. Therefore, a different metric has to be defined. Being somewhat arbitrary, a choice is justified by the

success it enables. The available implementation of Sparse RRT makes use of the following metric between two k-dimensional states $\boldsymbol{a}$ and $\boldsymbol{b}$:

$$m(\boldsymbol{a}, \boldsymbol{b}) = \sum_{i=1}^{k} w_i \cdot (a_i - b_i)^2 \ , \tag{4.1}$$

where $\boldsymbol{w}$ is a tuple containing the weight for each dimension. This implies that every choice for algorithm parameters which indicate a distance threshold in state space (i.e. $r_{near}, r_{goal}, r_{drain}$) have to be chosen in dependency on the dimensionality k of the state space, since more dimensions lead to (4.1) exceeding a threshold faster.

4.3.3 Trajectory propagation

Propagating a trajectory based on a dynamic model of a system generally is achieved by numerical integration of the system's equations of motion (2.1). For this, the implementation of the algorithm makes use of two integration routines: for simple models, a simple explicit Euler integration is used, that is

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + \dot{\boldsymbol{y}}_n \Delta t \ , \tag{4.2}$$

in which Δt corresponds to the randomly chosen propagation time. Because this might not yield sufficiently exact results for more complex systems, a Runge-Kutta scheme is implemented in addition to the Euler procedure. This generates new states according to

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + \Delta t \left(\frac{1}{6}\dot{\boldsymbol{y}}_n + \frac{1}{3}\dot{\boldsymbol{y}}_a + \frac{1}{3}\dot{\boldsymbol{y}}_b + \frac{1}{6}\dot{\boldsymbol{y}}_c \right) \ , \tag{4.3}$$

where

$$\dot{\boldsymbol{y}}_a = \boldsymbol{f}\left(t_n + \frac{\Delta t}{2}, \boldsymbol{y}_a\right), \qquad \boldsymbol{y}_a = \boldsymbol{y}_n + \frac{\Delta t}{2}\dot{\boldsymbol{y}}_n \tag{4.4}$$

$$\dot{\boldsymbol{y}}_b = \boldsymbol{f}\left(t_n + \frac{\Delta t}{2}, \boldsymbol{y}_b\right), \qquad \boldsymbol{y}_b = \boldsymbol{y}_n + \frac{\Delta t}{2}\dot{\boldsymbol{y}}_a \tag{4.5}$$

$$\dot{\boldsymbol{y}}_c = \boldsymbol{f}\left(t_n + \Delta t, \boldsymbol{y}_c\right), \qquad \boldsymbol{y}_c = \boldsymbol{y}_n + \Delta t \dot{\boldsymbol{y}}_b \ , \tag{4.6}$$

allowing for smaller errors.

4.4 Verification

The subsequent section serves to examine what kind of results can be obtained applying the algorithm. Therefore, several problems to which a solution is known will be considered and solved using the algorithm, making a comparison between the algorithm result and the analytical solution possible.

4.4.1 Zermelo's Problem

A rather popular example for an optimal control problem that can actually be solved analytically is the so-called "Zermelo's Problem." It considers finding the time-optimal path for a simple boat model through a given current field that can be represented analytically. Because the exact solution to this problem is known, it can serve as a benchmark for computer-based solution procedures. Based on the derivations of chapter 3, the solution to this problem will be derived in the following, based on [21, p.77].

The equations of motion of the boat are

$$\dot{x} = V\cos\theta + u(x,y) \tag{4.7}$$

$$\dot{y} = V\sin\theta + v(x,y)\ , \tag{4.8}$$

where $\dot{x}$ and $\dot{y}$ are the ground speed of the boat and V is its velocity relative to the current in the direction indicated by the heading angle θ. The current field is represented by $u(x,y)$ and $v(x,y)$ in x- and y-direction, respectively. The velocity field is assumed to be known and the speed V is constant.

For time-optimal control, the objective functional is reduced to

$$J = \int_{t_0}^{t_f} 1 \ \mathrm{d}t\ . \tag{4.9}$$

Following chapter 3, at first the Hamiltonian

$$\mathcal{H} = 1 + \lambda_x(V\cos\theta + u(x,y)) + \lambda_y(V\sin\theta + v(x,y)) \tag{4.10}$$

is formed. The necessary conditions (3.37) become

$$\dot{\lambda_x} = -\frac{\partial \mathcal{H}}{\partial x} = -\lambda_x \frac{\partial u}{\partial x} - \lambda_y \frac{\partial v}{\partial x} \tag{4.11}$$

$$\dot{\lambda_y} = -\frac{\partial \mathcal{H}}{\partial y} = -\lambda_x \frac{\partial u}{\partial y} - \lambda_y \frac{\partial v}{\partial y} \tag{4.12}$$

$$0 = \frac{\partial \mathcal{H}}{\partial \theta} = V(-\lambda_x \sin\theta + \lambda_y \cos\theta) \ . \tag{4.13}$$

Equation (4.13) establishes a connection between the control input θ and the costates λ_x and λ_y. The costates can be calculated from (4.10) and (4.13) if one considers that the Hamiltonian is not explicitly depending on the time t, what makes it an integral of the system. This again implies that the Hamiltonian has to be constant, and since (3.40) prescribes that it has to be zero at the final state, this holds for all time points. Therefore,

$$0 = 1 + \lambda_x(V\cos\theta + u(x,y)) + \lambda_y(V\sin\theta + v(x,y)) \ . \tag{4.14}$$

From (4.13) it follows that

$$\lambda_x = \lambda_y \cdot \frac{\cos\theta}{\sin\theta} \tag{4.15}$$

which again can be applied to (4.14) leading to

$$\lambda_y = \frac{-\sin\theta}{V + u(x,y)\cos\theta + v(x,y)\sin\theta} \ . \tag{4.16}$$

Using this, (4.15) becomes

$$\lambda_x = \frac{-\cos\theta}{V + u(x,y)\cos\theta + v(x,y)\sin\theta} \ . \tag{4.17}$$

In a next step, the derivatives of the costates with respect to time have to be calculated, that is

$$\dot{\lambda_x} = \frac{\mathrm{d}\lambda_x}{\mathrm{d}\theta}\frac{\mathrm{d}\theta}{\mathrm{d}t} = \frac{V\sin\theta + v(x,y)}{(V + u(x,y)\cos\theta + v(x,y)\sin\theta)^2} \cdot \dot{\theta} \tag{4.18}$$

$$\dot{\lambda_y} = \frac{\mathrm{d}\lambda_y}{\mathrm{d}\theta}\frac{\mathrm{d}\theta}{\mathrm{d}t} = -\frac{V\cos\theta - u(x,y)}{(V + u(x,y)\cos\theta + v(x,y)\sin\theta)^2} \cdot \dot{\theta} \ . \tag{4.19}$$

Applying (4.16), (4.17), (4.18) and (4.19) to (4.11) and (4.12) leads to

$$\frac{V\sin\theta + v(x,y)}{\left(V + u(x,y)\cos\theta + v(x,y)\sin\theta\right)^2}\cdot\dot{\theta} = \frac{\cos\theta}{V + u(x,y)\cos\theta + v(x,y)\sin\theta}\frac{\partial u}{\partial x} + \frac{\sin\theta}{V + u(x,y)\cos\theta + v(x,y)\sin\theta}\frac{\partial v}{\partial x} \tag{4.20}$$

and

$$-\frac{V\cos\theta - u(x,y)}{\left(V + u(x,y)\cos\theta + v(x,y)\sin\theta\right)^2}\cdot\dot{\theta} = \frac{\cos\theta}{V + u(x,y)\cos\theta + v(x,y)\sin\theta}\frac{\partial u}{\partial y} + \frac{\sin\theta}{V + u(x,y)\cos\theta + v(x,y)\sin\theta}\frac{\partial v}{\partial y}\ . \tag{4.21}$$

Multiplying (4.20) by $\sin\theta$ and (4.21) by $\cos\theta$ and adding the resulting equations eventually yields

$$\dot{\theta} = \sin^2\theta\frac{\partial v}{\partial x} + \sin\theta\cos\theta\left(\frac{\partial u}{\partial x} - \frac{\partial v}{\partial y}\right) - \cos^2\theta\frac{\partial u}{\partial y} \tag{4.22}$$

as a relation between the current field and the change in the heading angle. An interesting conclusion can be drawn from this equation: if the current field does not vary in any of the spatial coordinates, i.e.

$$\begin{aligned}\nabla u(x,y) &= \mathbf{0}\\ \nabla v(x,y) &= \mathbf{0}\ ,\end{aligned} \tag{4.23}$$

equation (4.22) becomes

$$\dot{\theta} = 0 \tag{4.24}$$

so the heading angle is constant, making the fastest path a straight line. In [21], the special case of a current field of the form

$$u(x,y) = 0, \qquad v(x,y) = -V\cdot y \tag{4.25}$$

is considered. From this, it follows that

$$\frac{\partial v}{\partial x} = 0\ , \qquad \frac{\partial v}{\partial y} = 0\ , \qquad \frac{\partial u}{\partial x} = 0\ , \qquad \frac{\partial u}{\partial y} = -V\ . \tag{4.26}$$

Applying this to (4.22) results in

$$\dot{\theta} = V \cos^2 \theta \ . \tag{4.27}$$

Next, a relation between the heading angle and the position has to be established. From (4.26) and (4.11) it follows that $\dot{\lambda}_x = 0$ and therefore, $\lambda_x = \text{const}$. This again means that

$$\frac{\cos\theta}{V - V \cdot y \cos\theta} = \frac{\cos\theta_f}{V - V \cdot y_f \cos\theta_f} \ , \tag{4.28}$$

eventually leading to

$$y(\theta) = \sec\theta - \sec\theta_f + y_f \ . \tag{4.29}$$

A relation for x can be found by introducing the inverse of (4.27)

$$\frac{\mathrm{d}t}{\mathrm{d}\theta} = \frac{\sec^2\theta}{V} = \frac{1}{V\cos^2\theta} \tag{4.30}$$

and (4.29) into (4.7), giving

$$\frac{\mathrm{d}x}{\mathrm{d}t}\frac{\mathrm{d}t}{\mathrm{d}\theta} = \frac{V\cos\theta + V\left(\sec\theta_f - \sec\theta - y_f\right)}{V\cos^2\theta} \tag{4.31}$$

$$= \sec\theta - \sec^3\theta + \left(\sec\theta_f - y_f\right) \cdot \sec^2\theta \ . \tag{4.32}$$

The integral of this expression

$$\int_x^{x_f} \mathrm{d}\chi = \int_\theta^{\theta_f} \sec\Theta - \sec^3\Theta + \left(\sec\theta_f - y_f\right) \cdot \sec^2\Theta \ \mathrm{d}\Theta \tag{4.33}$$

becomes

$$x_f - x = \left[\frac{1}{2}\log\left(\frac{\cos\frac{\Theta}{2} + \sin\frac{\Theta}{2}}{\cos\frac{\Theta}{2} - \sin\frac{\Theta}{2}}\right) + \tan\Theta\left(\left(\sec\theta_f - y_f\right) - \frac{1}{2}\sec\Theta\right)\right]_\theta^{\theta_f} \tag{4.34}$$

and making use of

$$\frac{\cos\frac{\Theta}{2} + \sin\frac{\Theta}{2}}{\cos\frac{\Theta}{2} - \sin\frac{\Theta}{2}} = \tan\Theta + \sec\Theta$$

eventually gives

$$x(\theta) = x_f - \frac{1}{2}\left[\sec\theta_f(\tan\theta_f - \tan\theta) - \tan\theta(\sec\theta_f - \sec\theta)\right] \qquad (4.35)$$
$$- \frac{1}{2}\log\left(\frac{\tan\theta_f + \sec\theta_f}{\tan\theta + \sec\theta}\right) + y_f\left(\tan(\theta_f) - \tan(\theta)\right) \ .$$

Based on the system of nonlinear equations (4.29) and (4.35), a time-optimal path from an initial position (x_0, y_0) to an end position (x_f, y_f) can be calculated. If (cf. [21])

$$x_0 = 3.66, \qquad y_0 = -1.86 \ , \qquad (4.36)$$

and

$$x_f = 0, \qquad y_f = 0 \ , \qquad (4.37)$$

(4.29) and (4.35) become

$$-1.86 = \sec\theta_0 - \sec\theta_f \qquad (4.38)$$

and

$$3.66 = -\frac{1}{2}\left[\sec\theta_f(\tan\theta_f - \tan\theta_0) - \tan\theta_0(\sec\theta_f - \sec\theta_0) + \log\left(\frac{\tan\theta_f + \sec\theta_f}{\tan\theta_0 + \sec\theta_0}\right)\right] \ , \qquad (4.39)$$

finally leading to

$$\theta_0 = 105^\circ \qquad \theta_f = 240^\circ \ . \qquad (4.40)$$

Using (4.22), the sequence of heading angles can be calculated from these values. Figure 4.2 shows the path based on this analytical solution as well as the path generated by the algorithm. As one can see, the path generated by Sparse RRT definitely captures the correct shape of the optimal path, however, has some deviations from it due to its random nature.

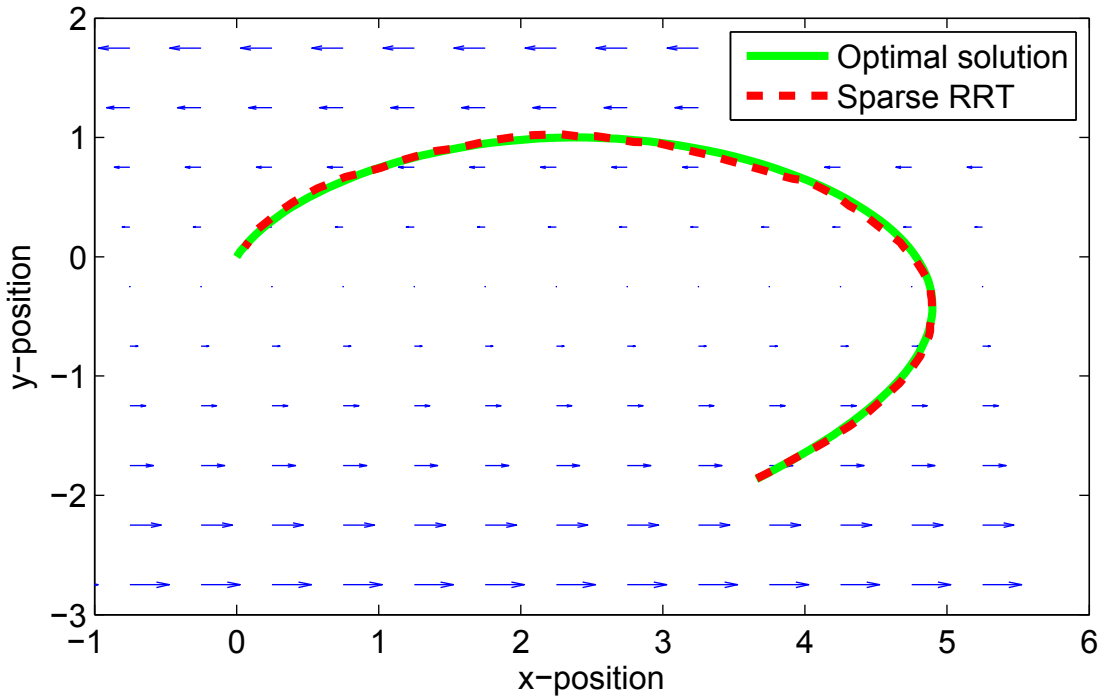

Figure 4.2. Zermelo's problem: analytical and numerical solution

Using (4.30), the traveling time can be calculated by integration, i.e.

$$\int_{t_0}^{t_f} \mathrm{d}t = \int_{\theta_0}^{\theta_f} \frac{\sec^2(\theta)}{V} \tag{4.41}$$

$$t_f = t_0 + \tan(\theta_f) - \tan(\theta_0) \ . \tag{4.42}$$

For the values calculated above, this leads to a dimensionless traveling time of 5.46, while Sparse RRT yields a time of 5.45 for the path shown in figure 4.2. The traveling time is shorter because the goal tolerance accepts a solution within a certain vicinity of the goal as will be discussed later.

4.4.2 Jet current

Another example related to the field of motion planning under the influence of a current field is given in [5]. There, the problem of time-optimal crossing of a jet current is considered for an ocean vehicle modeled by (4.7). The optimal solution, represented by the three different heading angles, can be obtained using

Nonlinear Optimization tools (e.g. fmincon in MATLAB). Figure 4.3 shows both the result obtained using fmincon and Sparse RRT. As one can see, the Sparse RRT solution gets close to the optimal solution that extensively makes use of the current to reduce traveling time.

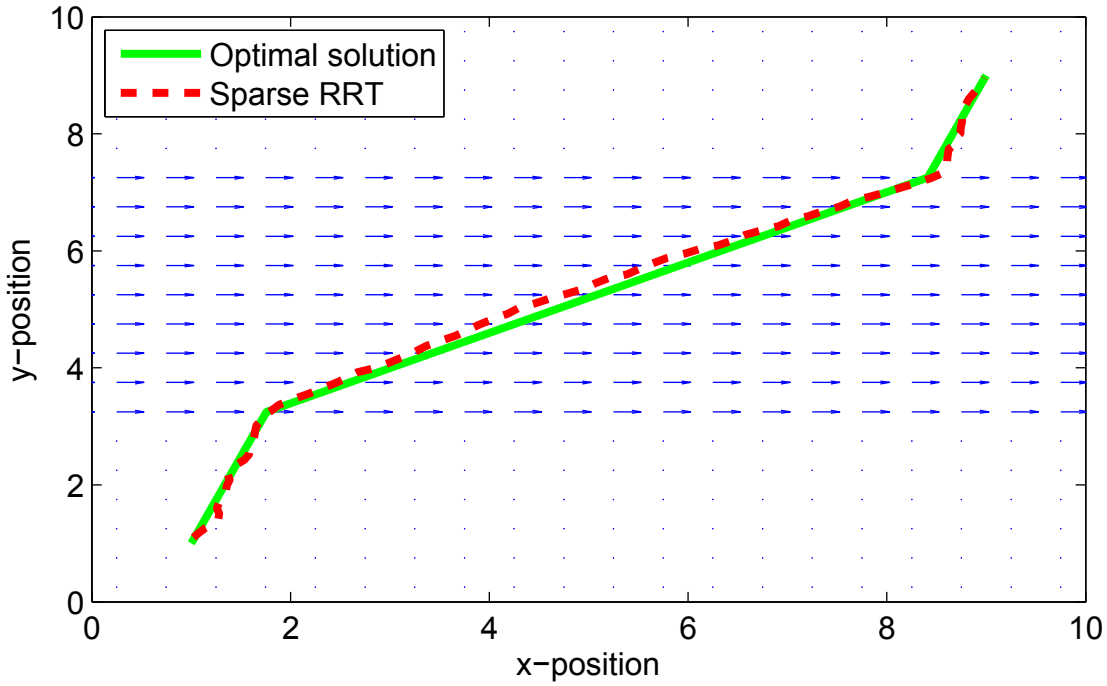

Figure 4.3. Optimal solution and solution generated by Sparse RRT

4.4.3 Energy-optimal control problem

The third example problem to be examined is an energy-optimal control problem given in [2, p.198]. It considers the system

$$\dot{x}_1(t) = x_2(t) \tag{4.43}$$

$$\dot{x}_2(t) = -x_2(t) + u(t) \tag{4.44}$$

and the objective functional

$$J(u) = \int_{t_0}^{t_f} \frac{1}{2} u^2(t)\ \mathrm{d}t\ , \tag{4.45}$$

that is the control effort, i.e. the energy spent, has to be minimized. There exist no constraints on the states or the control. Again, in order to solve the problem analytically, the Hamiltonian is formed as

$$\mathcal{H}\left(\boldsymbol{x}(t), u(t), \boldsymbol{p}(t)\right) = \frac{1}{2}u^2(t) + p_1(t)x_2(t) + p_2(t)\left(-x_2(t) + u(t)\right) \ . \tag{4.46}$$

The necessary conditions (3.37) become

$$\dot{p}_1(t) = -\frac{\partial \mathcal{H}}{\partial x_1} = 0 \tag{4.47}$$

$$\dot{p}_2(t) = -\frac{\partial \mathcal{H}}{\partial x_2} = -p_1(t) + p_2(t) \tag{4.48}$$

$$0 = \frac{\partial \mathcal{H}}{\partial u} = u(t) + p_2(t) \ . \tag{4.49}$$

Now, (4.49) can be solved for $u(t)$ and substituted into (4.44), making (4.43), (4.44), (4.47) and (4.48) a system of four differential equations. Given the following boundary conditions

$$\boldsymbol{x}(0) = \boldsymbol{0} \qquad \boldsymbol{x}(2) = \begin{pmatrix} 5 \\ 2 \end{pmatrix} \ , \tag{4.50}$$

this system can be solved as

$$x_1^*(t) = 7.289t - 6.103 + 6.696e^{-t} - 0.593e^{t} \tag{4.51}$$

$$x_2^*(t) = 7.289 - 6.696e^{-t} - 0.593e^{t} \tag{4.52}$$

$$p_1^*(t) = -7.289 \tag{4.53}$$

$$p_2^*(t) = -7.289\left(1 - e^{t}\right) - 6.103e^{t} \ . \tag{4.54}$$

The optimal control from (4.49) then becomes

$$u^*(t) = -p_2^*(t) = 7.289\left(1 - e^{t}\right) + 6.103e^{t} \ . \tag{4.55}$$

Solving this problem using Sparse RRT offers an additional challenge compared to the two examples analyzed before, to be specific the final time is not subject

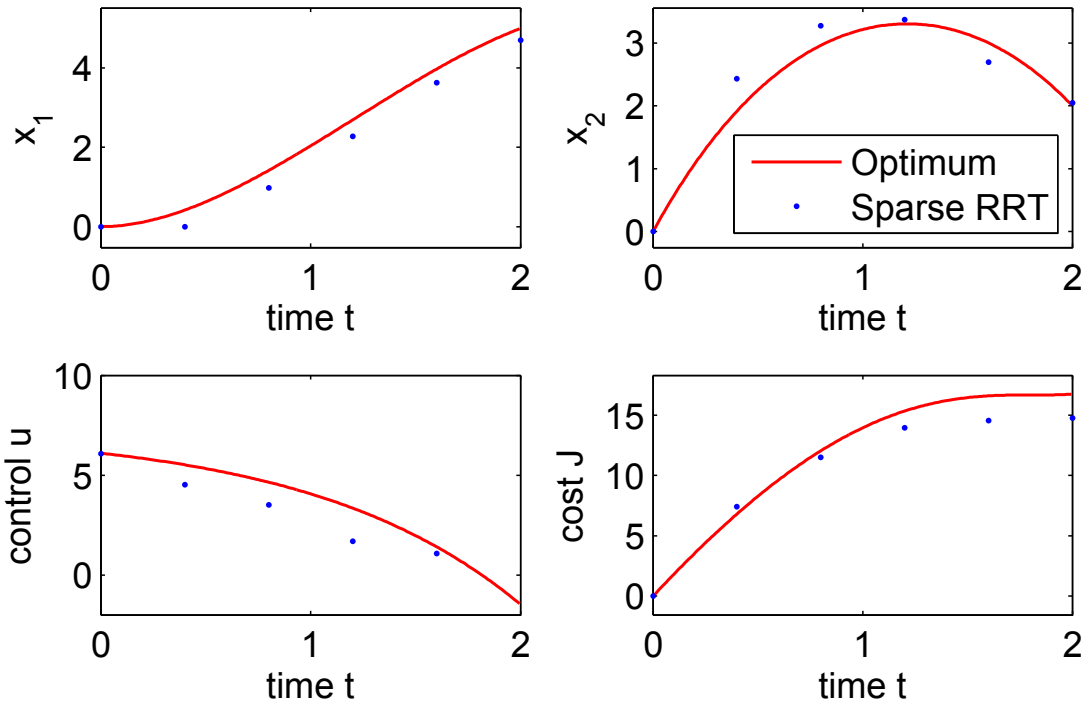

Figure 4.4. Energy-optimal control problem with fixed final time

to optimization but is fixed. This problem can be solved by introducing another state variable τ, and prescribing its value at the final time to the final time, i.e.

$$\tau_f = \int_{t_0}^{t_f} \dot{\tau} \, \mathrm{d}t = t_f \quad , \tag{4.56}$$

where $\dot{\tau} = 1$. This means that in checking whether a node is within the specified tolerance around the goal, the sum of the propagation times has also to be considered. Figure 4.4 shows both the analytical solution and a solution that can be obtained using Sparse RRT. As in the other examples before, the results obtained by Sparse RRT follow the same trend as the analytical solution, but also show a small deviation.

4.5 Conceptual testing

After having analyzed the general ability of the algorithm to produce results close to optimal solutions, this section will serve the purpose of evaluating if the

algorithm can account for typical constraints arising from the considered problem of path planning for an ocean vehicle. In section 2.4, obstacles, ocean currents and energy constraints were identified as such constraints, together with those resulting from the dynamic model of the vehicle.

In the previous section it has already been shown that the algorithm accounts for a current field without any additional effort required. Also, a kinematic vehicle model has been used successfully. Even though the dimensionality of the employed model is lower than of a detailed dynamic model, the general problem is the same, so it can be assumed that the algorithm in principle is capable of accounting for constraints from a dynamical model. This will be evaluated in more detail in chapter 5.

In this section, the focus will be on analyzing the algorithm's behavior in the presence of the two remaining types of constraints, resulting from obstacles and energy constraints.

4.5.1 Obstacles

One advantage of sampling-based algorithms is that they are able to handle obstacles easily. The general idea to make this possible is to check whether a state lies within an obstacle or not. When sampling, the sample is checked in this way. If it lies within an obstacle, it is simply discarded. If it is not within an obstacle, a trajectory is generated. To make sure that this trajectory also does not intersect with obstacles, two possible strategies can be thought of: the first checks for obstacles in each step of the integration, the second only checks the final state. The latter strategy requires that the propagation time is chosen adequately small because otherwise, an obstacle that only lies between the start and end state cannot be detected. In this section, it will be tested if the paths generated by Sparse RRT still tend to yield near-optimal solutions in the presence of obstacles.

In order to do so, a modified version of Zermelo's problem from section 4.4.1 is considered. The derivation in section 4.4.1 allows to calculate the optimal path to an arbitrary point on the map. If obstacles are included, this does not hold anymore, though. However, under some assumptions, an idea of an optimal path can be obtained. If an obstacle is placed in the course of the optimal path shown in figure 4.2, this can be achieved by calculating piecewise optimal paths from the start point to the obstacle and from there to the goal. The fact that this introduces a discontinuity into the heading angle sequence can not be avoided, because forcing continuity would lead to an over-determined system of equation. This, however, does not present a problem, because the kinematic model of the boat is allowed to have discontinuities in the control history.

A piecewise construction of an optimal path in general is admissible under the assumption that the optimal path in the presence of such an obstacle will only deviate slightly from the obstacle-free optimal path. Two deviations are possible: circumnavigating above the obstacle or below it. The cost of the two paths resulting from these possibilities can be calculated by simply adding the costs of the single paths, determining which paths has lower overall cost. If the algorithm would generate a trajectory that follows this path, it can be assumed that it can sufficiently account for obstacles. Figure 4.5 shows both analytical solutions and a trajectory obtained by running the algorithm. As can be seen, it passes below the obstacle which also is the optimal path according to the analytical solution. Therefore, the algorithm seems to be capable of accounting for obstacles in a correct way.

4.5.2 Energy constraints

Another important constraint is imposed by limited energy supplies of a vehicle. Therefore, it is desirable to evaluate how Sparse RRT in principle can account

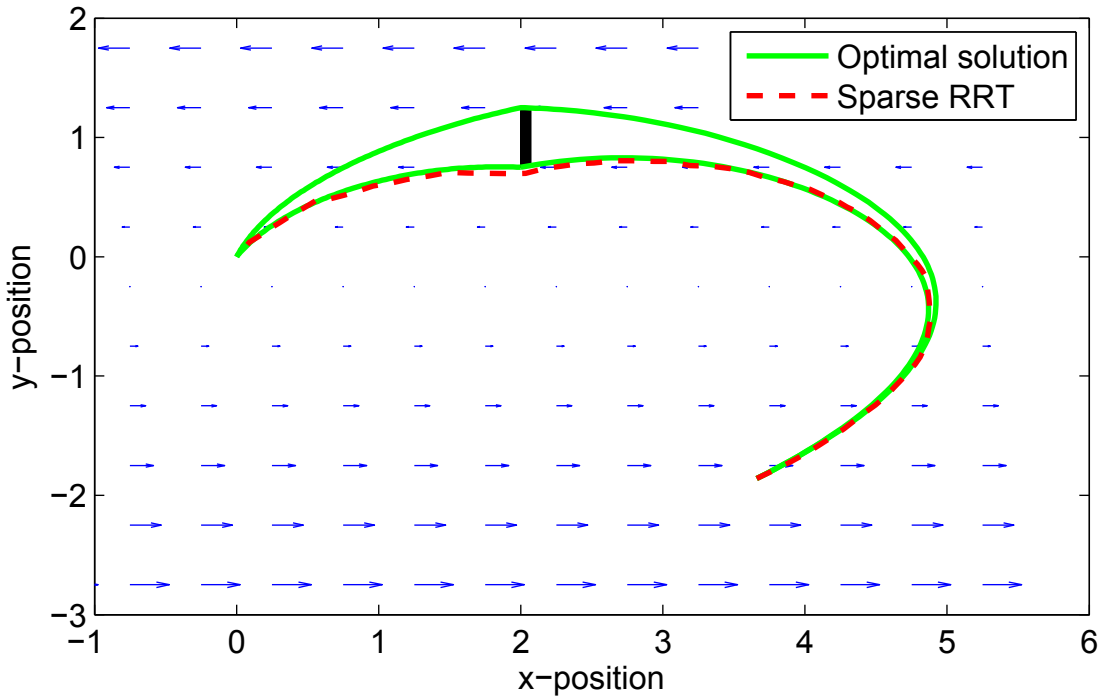

Figure 4.5. Solutions to Zermelo's problem in the presence of an obstacle

for this problem. As introduced in section 3.5, an energy constraint in general can be formulated as

$$\int_{t_0}^{t_f} e\left(\boldsymbol{u}(t), t\right)\, \mathrm{d}t) \leq c\ . \tag{4.57}$$

This kind of constraint can be handled similarly to the fixed time in 4.4.3 by introducing a new state variable $E(\boldsymbol{u}(t), t)$ that indicates the energy which has been spent up to a certain time, that is

$$E\left(\boldsymbol{u}(t), t\right) = \int_{t_0}^{t_f} e\left(\boldsymbol{u}(t), t\right)\, \mathrm{d}t) \leq c\ . \tag{4.58}$$

However, unlike in the case of the fixed final time in 4.4.3, this state is not to be considered in checking whether the goal has been reached or not. This is because the inequality nature of the constraint that does not specify a final value, but only a limit. Therefore, it does not have to be included into the range search of the "Drain" and "Best Nearest" feature. It is only required that the procedure

that checks whether a newly generated state is admissible makes sure that the inequality constraint is not violated.

In order to check the functionality of this method, a simple test setup based on the kinematic boat model (4.7) will be used. However, to incorporate energy considerations, it is necessary to make the thrust and therefore the speed V of the boat variable instead of assuming it to be constant, so the number of control inputs is increased to two. Even though arbitrary in principle for the purpose of demonstration, a simple way to model the (dimensionless) instantaneous energy consumption of such a boat can be

$$e\left(V(t), t\right) = V(t)^2 \tag{4.59}$$

so the consumed energy up to a certain time t_f becomes

$$E\left(V(t), t\right) = \int_{t_0}^{t_f} V(t)^2 \, \mathrm{d}t \ . \tag{4.60}$$

The expected behavior is the following: if no or only a very generous energy limitation is active, the boat will move at full speed. If the energy constraint becomes more restrictive, the boat will be forced to reduce its speed to comply with the constraint.

The following test will be performed: the boat is commanded to move horizontally for 4 units in a calm sea, that is without any external forcing. The dimensionless energy supply is set to 2, the maximum possible speed to 1. Without external forcing, it is reasonable to assume that the speed will be constant the whole time. In total, equation (4.60) becomes

$$2 = \int_{t_0}^{t_f} V^2 \, \mathrm{d}t = V^2 \cdot t_f \ , \tag{4.61}$$

while the time t_f to reach the goal is

$$t_f = \int_0^4 \frac{dx}{V} = \frac{4}{V} \ . \tag{4.62}$$

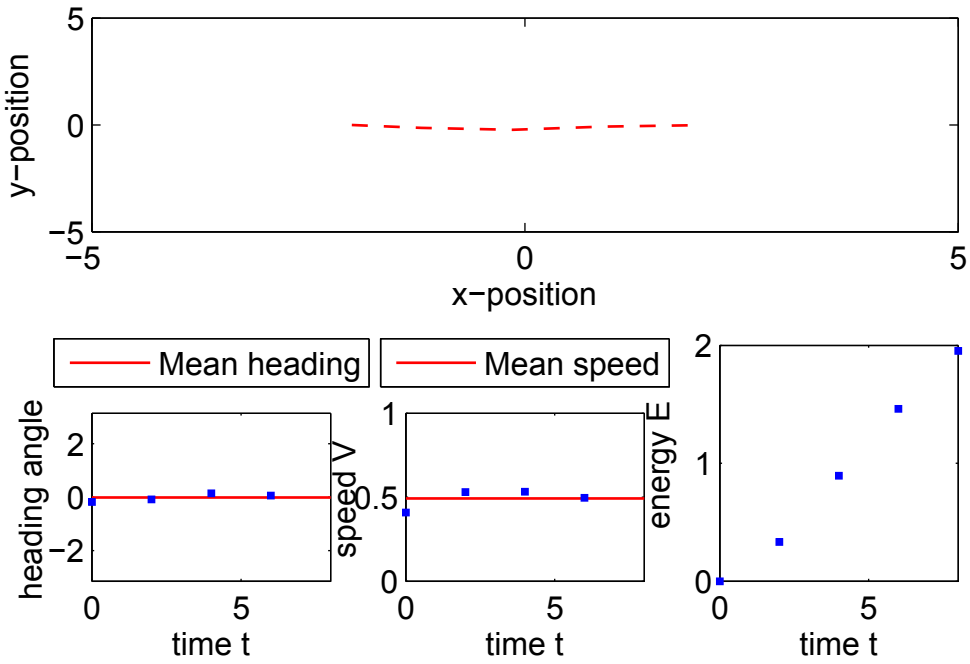

Figure 4.6. Test result for variable speed with limited energy

From these two equations, the speed V and the traveling time t_f can be calculated as $V = 0.5$ and $t_f = 8$.

Figure 4.6 shows the path taken by the boat in a calm sea, i.e. without currents. Also, the heading angles and the speed are shown, as well as the energy consumption. Several observations can be made: firstly, the boat follows a straight line, which was proven in 4.4.1 to be the shortest path if no current field exists. Also, the energy constraint is not violated, but all energy is consumed, what makes sense against the background of a time-optimal control problem. The average boat speed responsible for the energy consumption is definitely below the maximum possible speed of $V = 1$, being close to the analytically obtained value of 0.5. The rugged shape of the control sequences will be discussed later.

4.6 Influence of the free algorithm parameters

Based on the general tendencies depicted in [3] as well as observations during simulations, in this section it will be discussed how a solution can be influenced by those parameters of the algorithm that have to be defined by the user. Primarily, these parameters are the "Drain" radius r_{drain}, the "Best Nearest" radius $r_{nearest}$ and the goal tolerance r_{goal}. Other parameters are the bounds for the propagation time as well as the number of iterations executed.

4.6.1 "Drain" radius

The Sparse RRT algorithm is reported to be computationally efficient [3] since it only grows one trajectory in each iteration, not several as RRT*. Also, the "Drain" feature reduces the cost of queries because the amount of data stored is kept low. As mentioned before, the "Drain" feature aims at limiting the maximum number of nodes that can be stored. In a bounded search space, this number depends on the value chosen for the drain radius.

Apart from this obvious relation, the drain radius has further impact: in section 4.2, it has been mentioned that the drain radius also has influence on whether a solution can be found at all, that is on the completeness of the algorithm. This is due to the fact that a drain radius chosen too large may block the goal or parts of the optimal path. Two examples are shown in figures 4.7 and 4.8. Figure 4.7 shows a scenario where the drain radius leads to a blockade of the goal region, so the algorithm never will return a trajectory because no state can ever enter the goal radius. Figure 4.8 shows an example where the drain radius is chosen too large so that a narrow passage cannot be entered.

Another possibility of preventing finding a solution is to choose an inappropriate ratio of propagation time and drain radius. Since new states are generated by integration of a system of differential equations, a propagation time too small or a

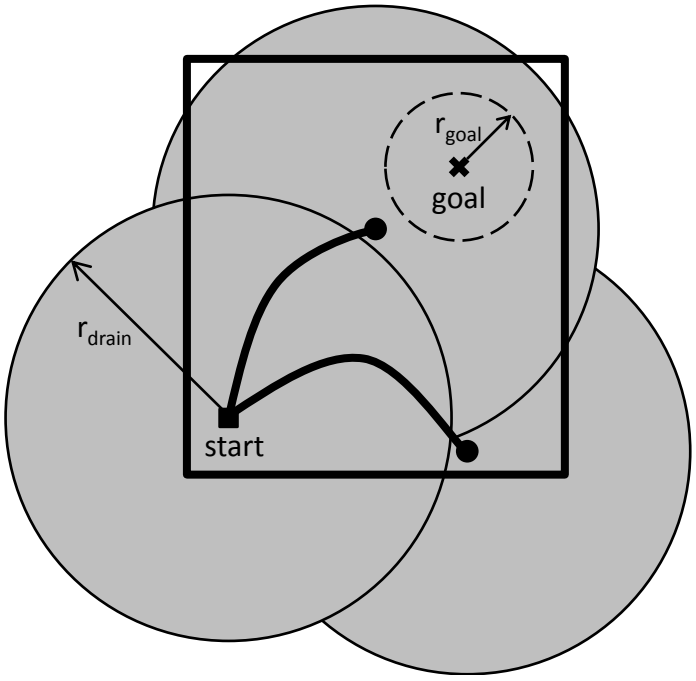

Figure 4.7. Drain radius blocking the goal region in a bounded space

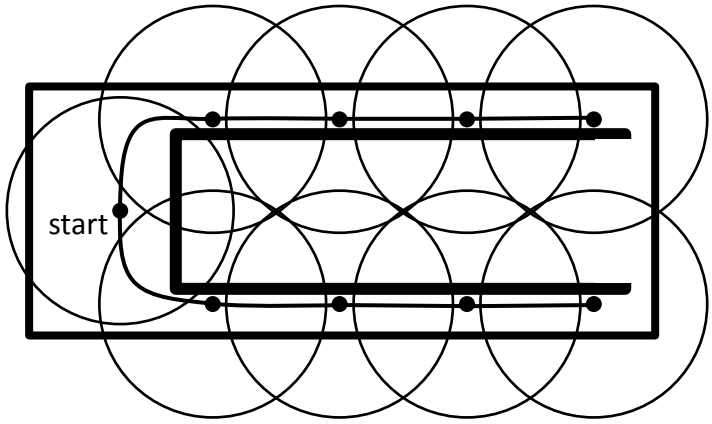

Figure 4.8. Drain radius blocking narrow passage (cf. [3])

drain radius too large might lead to a situation where the newly generated states still fall inside the drain radius of their parent state and therefore are removed. This becomes even more critical if the derivatives of the state variables highly vary in magnitude at different states, i.e. at one state, only a short propagation time would be sufficient to leave the drain radius, while at other states, a longer propagation time is needed. In such a situation it is required to choose the drain radius small enough in order to not exclude some areas of the state space from the search. This again has the drawback that a small drain radius does not improve the computation time as much as a larger radius would do.

An extreme case that can be thought of is a vehicle that is at standstill initially. Given a certain propagation time, in order to move away far enough from its initial state and thereby avoid being discarded, a certain acceleration is required. This demand on the acceleration does not, as it should be, stem from the optimization process, but from the drain radius, i.e. an inappropriate drain radius potentially could drastically influence the result.

4.6.2 "Best Nearest" radius

As described, the "Best Nearest" radius r_{near} is the built-in mechanism to enforce an improvement of feasible solutions during the run-time of the algorithm. This is enabled by determining which node from the vicinity of a sample node as defined by r_{near} has lowest cost. That node then is made the starting point of a trajectory, so the general mechanism strives to propagate nodes with low cost. However, the efficiency of the algorithm depends on the value chosen for r_{near}. Two scenarios are possible:

On the one hand, r_{near} could be chosen too large. In extreme cases, this can lead to a situation similar to the one shown in figure 4.9, where one and the same node is repeatedly selected for propagation, preventing the tree from growing.

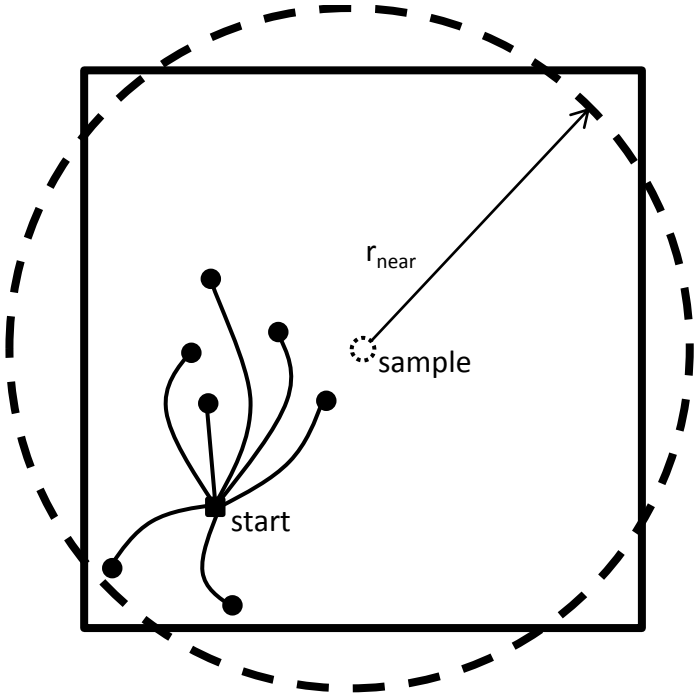

Figure 4.9. Large r_{near} leading to repeated propagation of the same node

On the other hand, if r_{near} is chosen too small, it will only have little effect on improving a solution. Therefore, convergence could be slowed down. This is especially true if r_{near} is chosen smaller than the drain radius r_{drain}: In that case, no vicinity around an arbitrary sample node could contain more than one node of the tree, so no selection process can happen to focus the tree growth on low-cost nodes.

4.6.3 Goal tolerance

The goal tolerance r_{goal} defines a vicinity in state space which to reach will be counted as reaching the goal itself. This is necessary because the likelihood of exactly reaching the desired goal within a limited number of iterations is small. Due to accepting a solution not only upon reaching the goal state, but also a vicinity around it, the goal tolerance at the same time can become an upper bound to how close the goal state will be reached. This can be seen from the following reasoning:

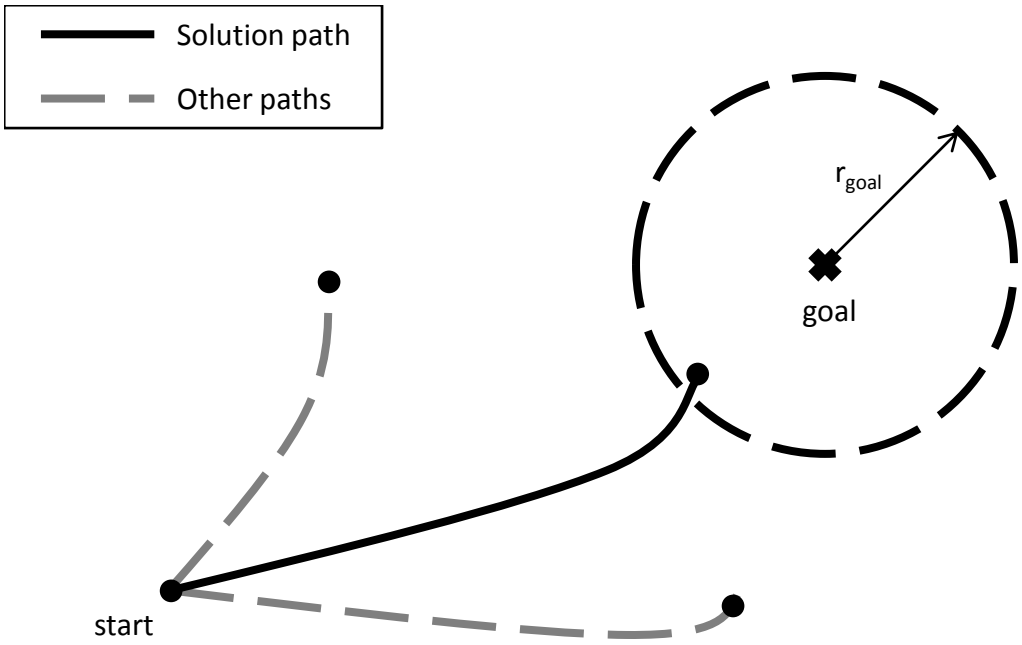

Figure 4.10. Goal tolerance as an upper limit to the solution precision

The algorithm aims at minimizing a cost functional. Therefore, it will eventually generate a path that only reaches the outmost boundary of the vicinity defined by the goal tolerance, because this shortens the path and therefore also the cost functional. Figure 4.10 visualizes this relation for a 2-dimensional state space.

4.6.4 Propagation duration

The propagation duration defines the amount of time for which the system's equations are integrated forward, based on a constant control input having been randomly selected before. The propagation duration is also randomly chosen from a user-defined interval. Apart from the before mentioned relation to the drain radius, the propagation duration has further impact on the results generated by the algorithm.

One important aspect the propagation duration has influence on is the detection of obstacles. The original implementation of the algorithm checks for every

newly generated node whether it lies within an obstacle or not. This is a fast procedure, but can become problematic if the propagation time is chosen too large in relation to the size of the obstacle. In that case, an obstacle could simply be jumped over by the newly generated state, so it would not be detected, but still intersect with the generated trajectory. This issue is solved by checking for obstacles not only at the newly generated node, but during integration, even though this might result in higher run-time requirements.

Another important point influenced by the propagation duration is the smoothness of the generated control signal. While a relatively long propagation duration is likely to generate a control as shown for example in figure 4.4, shorter propagation durations generate sequences as shown in figure 4.11. Clearly, the control sequence $u(t)$ is not smooth, but the resulting state trajectories still match the optimal solution quite well or even better than the results in 4.4.

A simple explanation for this phenomenon is that a longer propagation duration reduces the possible combinations of states, i.e. leads to less single decisions for the optimization process and therefore converges faster to a close-to optimal solution. A shorter duration in contrast increases this number and therefore, given the same run-time, generates more rugged sequences. The drawback of a long propagation duration is as noticed above that the resulting state trajectories might be less precise compared to a short propagation duration. Different means to improve the shape of the control sequence will be discussed subsequently.

4.6.5 Number of iterations

With one iteration being the process of sampling a state, determining which existing node to expand and generating a new state, the number of iterations has high impact on the obtainable results. Obviously, if the number of iterations is too small, the algorithm might not be able to find a solution at all. On the other

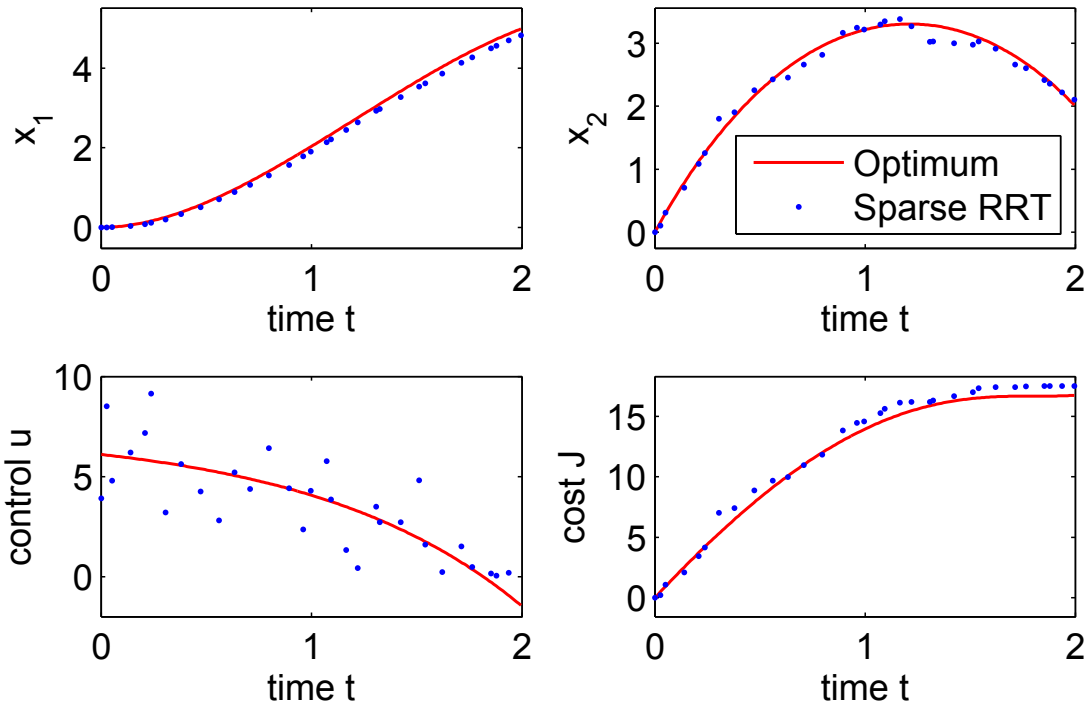

Figure 4.11. Problem of section 4.4 based on a shorter propagation duration

hand, it can be observed and also seen from [3] that once a solution has been found, initially it is often improved quickly. After this initial phase, convergence speed decreases in many cases, but due to the random nature of the algorithm, significant changes can occur, i.e. if only a local minimum had been found before.

4.7 (Post-)Processing of results

As indicated before, the interval chosen for the sampling of the propagation duration has a large impact on the shape of the generated control sequence, potentially making it non-smooth. In this section, three ways to influence this shape towards a smoother appearance will be discussed based on the example of the energy-optimal control problem presented in section 4.4.3. Two of the considered methods are post-processing procedures, while one acts during the run-time of the algorithm. As can be seen, no method leads to drastically different trajectories, but

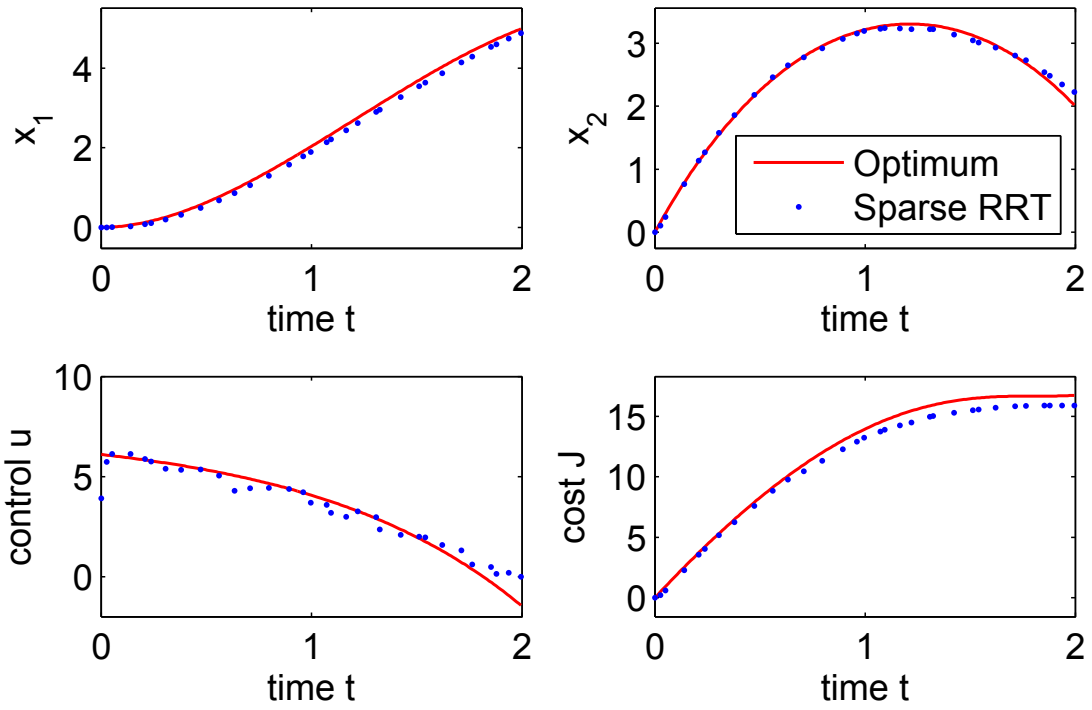

Figure 4.12. Filtered control sequence and resulting trajectories

smoothes the control sequence and therefore makes it more applicable to real-life applications.

4.7.1 Filtering

One possible way to post-process a rugged control sequence is to filter it using a low-pass filter. Applying a sliding-window filter with a range of 9 neighbor points to the data on which figure 4.11 is based can lead to the trajectories shown in figure 4.12. As one can see, the control sequence $u(t)$ is clearly smoothed out, while the trajectories still are very similar to the analytic solutions. However, the control sequence locally deviates from the analytic solution and the cost function even outperforms the analytic solution, which is only possible because the trajectories are allowed to differ from the prescribed goal state by the goal radius as described above, making cost savings possible.

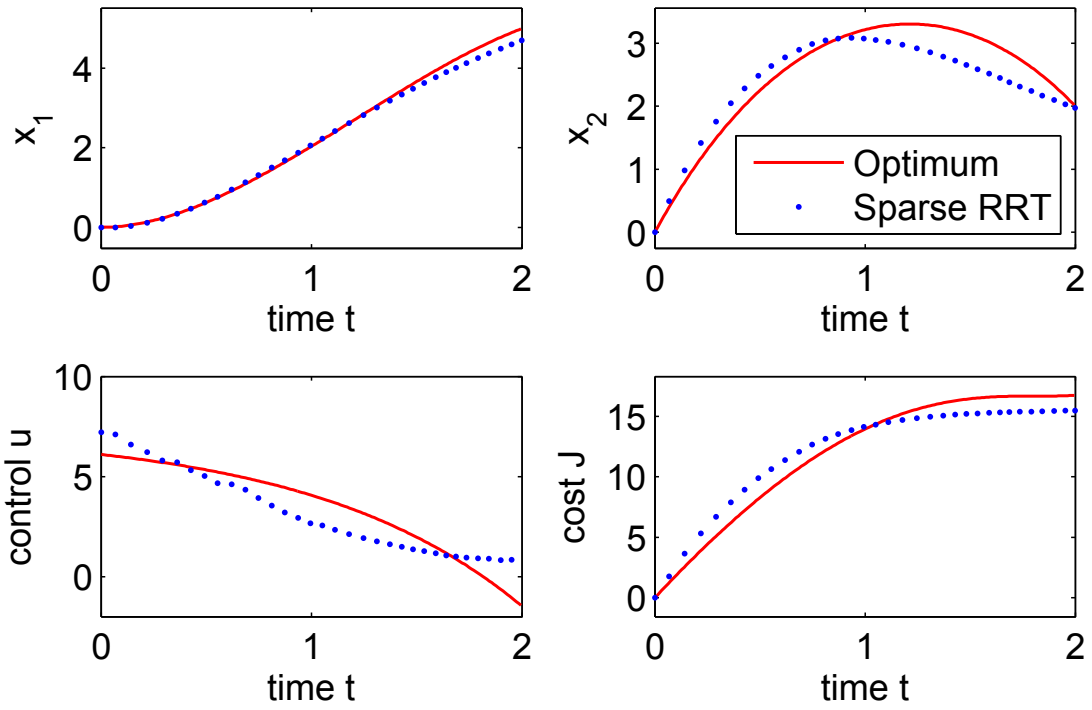

Figure 4.13. Results for derivative-bounded control sequence

4.7.2 Selective sampling

Another way to obtain smoother control sequences is to limit the difference between two consecutive control values, i.e. to forbid large jumps. This procedure is only admissible if no discontinuities in the control history are expected. Applying this technique leads to the curves shown in figure 4.13. Several observations can be made:

Firstly, the generated control sequence definitely is smooth. Also, all boundary conditions for x_1 and x_2 are met, but the resulting trajectories clearly deviate from the analytic solutions. However, the final cost value is only slightly higher than the one calculated by the analytic solution, which implies that the algorithm has converged to a suboptimal solution. This complies with the fact that the algorithm is only asymptotically near-optimal, i.e. likely to yield a solution only close to the optimal one. Nonetheless, the deviation in this example is only acceptably small .

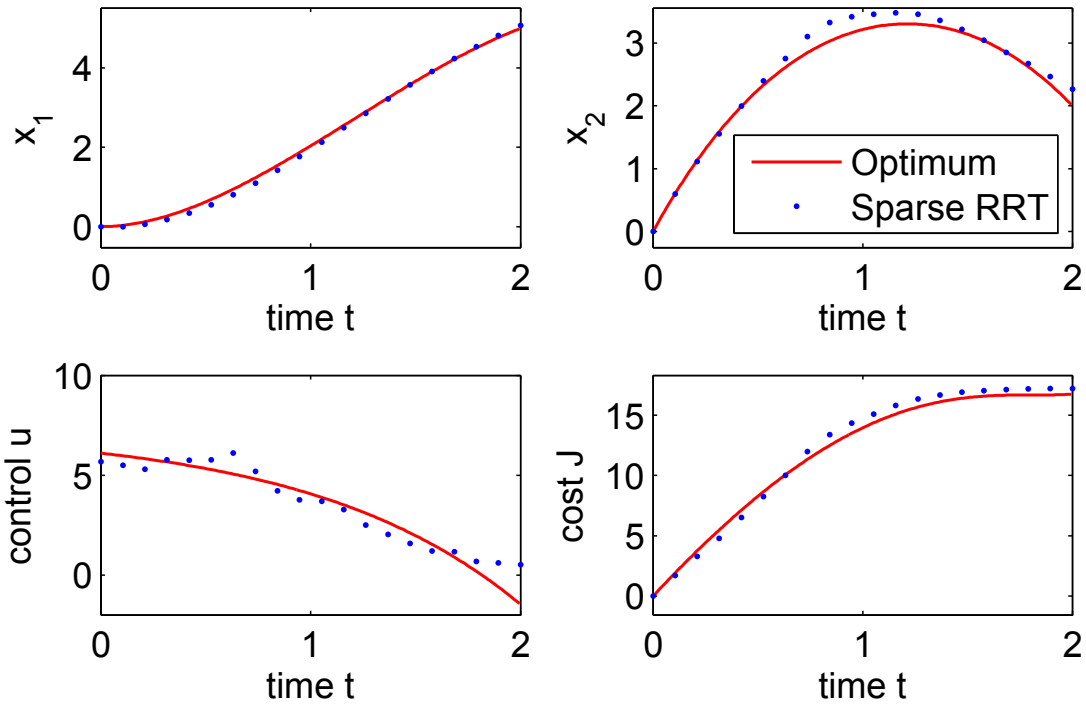

Figure 4.14. Results based on an averaged control sequence

4.7.3 Averaging

Apart from filtering and limiting the changes allowed between to consecutive control values, it is also possible to average the results obtained from several runs of the algorithm. If, as of default, the control values are not applied in constant time intervals, interpolation might be necessary before averaging. Figure 4.14 demonstrates that this method, applied to ten control sequences, also can lead to acceptable results, even though due to averaging and interpolating one boundary condition is not met closely. This method could also be combined with a low-pass filter. However, if the time span of different solutions differs a lot, averaging might not be applicable.

4.8 Summary

In the preceding chapter, the functionality of the Sparse RRT algorithm and its major properties as well as implementational aspects have been outlined. Most importantly, the performance of the algorithm when applied to three example problems has been analyzed. Also, two conceptual tests were executed to evaluate the algorithm's ability to account for obstacles and energy constraints.

The major findings of the preceding chapter are the following: even though being only asymptotically near-optimal, Sparse RRT is capable of finding good solutions. A potential drawback that did not primarily affect the quality of the solution, but the applicability to real technical systems, turned out to be the rugged shape of the generated control sequence. However, several methods were analyzed to alleviate this problem. Eventually, the influence of some of the algorithm's parameters on the solution finding process and the solution itself has been discussed.

CHAPTER 5

Modeling of and planning for an unmanned boat

In chapter 4, the Sparse RRT algorithm and its capabilities and limitations have been analyzed. Based on those findings, in this chapter, the problem of short-distance path planning for an ocean surface vessel, considering simple dynamical effects, obstacles, ocean currents and energy limitations, will be examined.

5.1 Dynamic boat model

In the subsequent section, the simple boat model which will be used for further simulations and its inherent assumptions will be described, based on the one presented in [4] and the derivations in [47].

5.1.1 Degrees of freedom and coordinate systems

In general, a rigid body in space has three translational and three rotational degrees of freedom. However, as [47] remarks, for a ship steering model, it is justified to neglect roll, pitch and heave motions, leading to a three-dimensional model including the position coordinates x, y and the orientation θ in a plane as shown in figure 5.1.

For convenience, the velocity components and later on the external forces will be expressed in a local, body-fixed cartesian coordinate system located in the center of gravity of the boat such that the first axis is oriented from aft to fore and the third axis stands on the plane, with the basis vector tuple

$$\underline{\boldsymbol{e}}_{123} = \{\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3\}^T \ . \tag{5.1}$$

The motions of the boat are referred to as surge u (longitudinal direction), sway v (lateral direction) and yaw rate r (rotation around the third axis), together forming

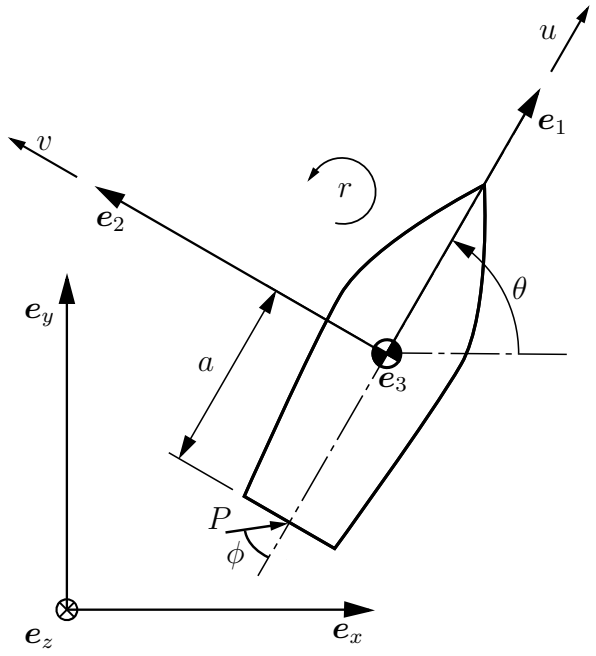

Figure 5.1. Schematic of the considered boat model (courtesy of [4])

the velocity vector

$$\underline{v}^T \underline{\boldsymbol{e}}_{123} = \begin{pmatrix} u \\ v \\ r \end{pmatrix}^T \underline{\boldsymbol{e}}_{123} \ . \tag{5.2}$$

The cartesian, earth-fixed coordinate system with the basis vector tuple

$$\underline{\boldsymbol{e}}_{xyz} = \{\boldsymbol{e}_x, \boldsymbol{e}_y, \boldsymbol{e}_z\} \tag{5.3}$$

is assumed to be an inertial system and can be related to the body-fixed system by the relation

$$\underline{q}^T \underline{\boldsymbol{e}}_{123} = \underline{q'}^T \underline{\boldsymbol{e}}_{xyz} = \boldsymbol{R}\underline{q}\underline{\boldsymbol{e}}_{xyz} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \underline{q}\underline{\boldsymbol{e}}_{xyz} \ , \tag{5.4}$$

where $\underline{q}$ is a tuple of coordinates.

5.1.2 External forces

In [47], a detailed description of possible external forces on an ocean vessel is given, including those due to added mass, buoyancy, hydrodynamic damping,

ocean currents, waves and wind. For the model to be used, however, only some of these will be considered for simplicity and availability of data.

Current field

For the purpose of this thesis, it is assumed that the velocity vector of a given current field $\boldsymbol{v}_{current}$ and the velocity of the boat can be superposed. Also, it is assumed that a given current field is stationary, i.e. not variable with respect to time. If the current field is defined in the earth-fixed reference frame, the relative velocity between the boat and the current, measured in the local coordinate system, will be defined as

$$\begin{aligned} \underline{v}_{rel}^T \underline{\boldsymbol{e}}_{123} = \begin{pmatrix} u_{rel} \\ v_{rel} \\ r \end{pmatrix} \underline{\boldsymbol{e}}_{123} = \underline{v}^T \underline{\boldsymbol{e}}_{123} - \underline{v}_{current}^T \underline{\boldsymbol{e}}_{xyz} \\ = \underline{v}^T \underline{\boldsymbol{e}}_{123} - \boldsymbol{R}^{-1} \underline{v}_{current} \underline{\boldsymbol{e}}_{123} \end{aligned} \tag{5.5}$$

where the current field can be transformed into the body-fixed system using (5.4). Note that a relative yaw rate does not exist due to the fact that the current field is assumed to be rotation free.

Added mass

Forces due to the so-called "added mass" effect occur in an accelerated relative motion between a boat and the surrounding water. As [47] outlines, these forces arise from the acceleration of the fluid particles. Even though in theory the vehicle's acceleration will force the whole fluid to oscillate, the amplitudes of the oscillation will be smaller the greater the distance to the boat gets. Therefore, the influence of this phenomenon is commonly modeled by adding a certain shape- and direction-dependent amount of mass to the mass of the boat. In [4], the forces resulting

from the added mass effect are modeled as

$$\underline{f}^T_{addedMass}\underline{\boldsymbol{e}}_{123} = \begin{pmatrix} 0 & 0 & -Y_{\dot{v}}v_{rel} - Y_{\dot{r}}r \\ 0 & 0 & X_{\dot{u}}u_{rel} \\ Y_{\dot{v}}v_{rel} + Y_{\dot{r}}r & -X_{\dot{u}}u_{rel} & 0 \end{pmatrix} \underline{v}^T_{rel}\underline{\boldsymbol{e}}_{123} + \begin{pmatrix} X_{\dot{u}} & 0 & 0 \\ 0 & Y_{\dot{v}} & Y_{\dot{r}} \\ 0 & Y_{\dot{r}} & N_{\dot{r}} \end{pmatrix} \underline{\dot{v}}^T \underline{\boldsymbol{e}}_{123} \ , \tag{5.6}$$

whereby the parameters, which are specified in appendix A, indicate the amount of added mass for each dimension.

Hydrodynamic damping

Several damping effects result from the interaction between an ocean vehicle and the ambient water. Here, only linear damping and the drag force will be considered, both depending on the relative velocity $\underline{v}^T_{rel}\underline{\boldsymbol{e}}_{123}$ between vehicle and water. For linear damping of low-speed ships, it is assumed that the surge speed can be decoupled from the sway and yaw motions [47]. The linear damping force then can be expressed as

$$\underline{f}^T_{linearDamping}\underline{\boldsymbol{e}}_{123} = \begin{pmatrix} X_u & 0 & 0 \\ 0 & Y_v & Y_r \\ 0 & N_v & N_r \end{pmatrix} \underline{v}^T_{rel}\underline{\boldsymbol{e}}_{123} \ . \tag{5.7}$$

According to [4], the drag force acting on the boat depends quadratically on the relative velocity between boat and water and can be expressed as

$$\underline{f}^T_{drag}\underline{\boldsymbol{e}}_{123} = \begin{pmatrix} X_{u|u|}|u_{rel}| & 0 & 0 \\ 0 & Y_{v|v|}|v_{rel}| & 0 \\ 0 & 0 & N_{r|r|}|r| \end{pmatrix} \underline{v}^T_{rel}\underline{\boldsymbol{e}}_{123} \ . \tag{5.8}$$

Engine thrust

The boat model is equipped with a thruster of variable direction and thrust magnitude. Therefore, the thrust that propels the boat is modeled according to

$$\underline{f}^T_{thrust}\underline{\boldsymbol{e}}_{123} = \begin{pmatrix} P\cos\phi \\ -P\sin\phi \\ aP\sin\phi \end{pmatrix}^T \underline{\boldsymbol{e}}_{123} \ . \tag{5.9}$$

5.1.3 Equations of motion

Following the general Newtonian procedure, the equations of motion can be derived component-wise for the simple boat model. The configuration of the center of gravity of the boat is indicated at any time by the vector

$$\boldsymbol{z}(t) = \underline{z}^T(t)\underline{\boldsymbol{e}}_{xyz} \ , \tag{5.10}$$

expressed in the earth-fixed coordinate system. As mentioned, the velocity of this point is the sum of two components: the surge and sway velocity vector $\underline{v}^T\,\underline{\boldsymbol{e}}_{123}$, given in the body-fixed coordinate system or transformed to the earth-fixed reference frame making use of (5.4), and the current velocity $\underline{v}^T_{current}\underline{\boldsymbol{e}}_{xyz}$, assumed to be given in the earth-fixed coordinate system. That is

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{z}(t) &= \underline{v}^T\,\underline{\boldsymbol{e}}_{123} + \underline{u}^T_{current}\underline{\boldsymbol{e}}_{xyz} \\ &= \left[\begin{pmatrix}\cos\theta & -\sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{pmatrix}\begin{pmatrix}u\\ v\\ r\end{pmatrix} + \underline{v}_{current}\right]^T \underline{\boldsymbol{e}}_{xyz} \ .\end{aligned} \tag{5.11}$$

Assuming that the current field is stationary, the acceleration, expressed in the body-fixed reference frame, then can be calculated using the angular velocity pseudo-vector $\underline{\omega}$ as

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\underline{v}^T\,\underline{\boldsymbol{e}}_{123}\right) = \underline{\dot{v}}^T\,\underline{\boldsymbol{e}}_{123} + \underline{\omega}\times\underline{v}\underline{\boldsymbol{e}}_{123} = \begin{pmatrix}\dot{u}-rv\\ \dot{v}+ru\\ \dot{r}\end{pmatrix}\underline{\boldsymbol{e}}_{123} \ . \tag{5.12}$$

Using Newton's first principle, from (5.12) it then follows that

$$\underline{f}^T\,\underline{\boldsymbol{e}}_{123} = \begin{pmatrix}m & 0 & 0\\ 0 & m & 0\\ 0 & 0 & I_z\end{pmatrix}\begin{pmatrix}\dot{u}-rv\\ \dot{v}+ru\\ \dot{r}\end{pmatrix}\underline{\boldsymbol{e}}_{123} \ , \tag{5.13}$$

with m being the mass of the boat and I_z the moment of inertia with respect to the z-axis. The vector of external forces $\underline{f}^T\,\underline{\boldsymbol{e}}_{123}$ is the sum of all external forces as described in the previous sections, i.e.

$$\underline{f}^T\,\underline{\boldsymbol{e}}_{123} = \left(\underline{f}_{thrust} - \underline{f}_{addedMass} - \underline{f}_{linearDamping} - \underline{f}_{drag}\right)^T\underline{\boldsymbol{e}}_{123} \ . \tag{5.14}$$

Introducing (5.6)-(5.9) to (5.13) eventually yields

$$\begin{aligned}
&\begin{pmatrix} m+X_{\dot{u}} & 0 & 0 \\ 0 & m+Y_{\dot{v}} & Y_{\dot{r}} \\ 0 & Y_{\dot{r}} & I_z+N_{\dot{r}} \end{pmatrix} \begin{pmatrix} \dot{u} \\ \dot{v} \\ \dot{r} \end{pmatrix} \underline{\boldsymbol{e}}_{123} = \left[\begin{pmatrix} P\cos\phi \\ -P\sin\phi \\ a\cdot P\sin\phi \end{pmatrix} \right. \\
&- \begin{pmatrix} X_u + X_{u|u|}|u_{rel}| & 0 & -Y_{\dot{v}}v_{rel} - Y_{\dot{r}}r \\ 0 & Y_v + Y_{v|v|}|v_{rel}| & X_{\dot{u}}u_{rel} + Y_r \\ Y_{\dot{v}}v_{rel} + Y_{\dot{r}}r & -X_{\dot{u}}u_{rel} + N_v & N_r + N_{r|r|}|r| \end{pmatrix} \underline{v}_{rel} \\
&\left. - \begin{pmatrix} 0 & -mr & 0 \\ mr & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} u \\ v \\ r \end{pmatrix} \right] \underline{\boldsymbol{e}}_{123}
\end{aligned} \tag{5.15}$$

which together with (5.11) describe the motion of the system. The model parameters are specified in appendix A. The maximum thrust P is limited to 200 N, allowing for a maximum speed in calm sea of not even 1 m/s as is realistic for a slow unmanned vehicle. The thruster angle ϕ is limited to $\left[-\frac{\pi}{4}, \frac{\pi}{4}\right]$.

Figures 5.2 and 5.3 contain two examples which illustrate the model behavior. Figure 5.2 shows two turning radii of the boat at maximum thruster angle and different thrust magnitudes. As can be seen, the minimum turning radius increases as the thrust decreases. It has to be noted that the boat is not drawn to scale nor shape, but only serves the purpose to indicate the orientation.

Figure 5.3 in contrast shows the behavior of the boat if no thrust is applied, but initial surge, sway and yaw speeds are given. Figure 5.4 shows the development of the speeds over time and gives an impression of the damping in the different directions. The damping in the longitudinal direction of the boat is the lowest, while the lateral and especially the rotational damping are higher. Because the boat lacks the ability to break, the damping is of importance because it represents the only possibility to decelerate.

5.2 Path planning for the dynamic model

Based on the equations of motion presented in (5.15), the Sparse RRT algorithm can be used to plan paths for the boat model. Since the general ability of

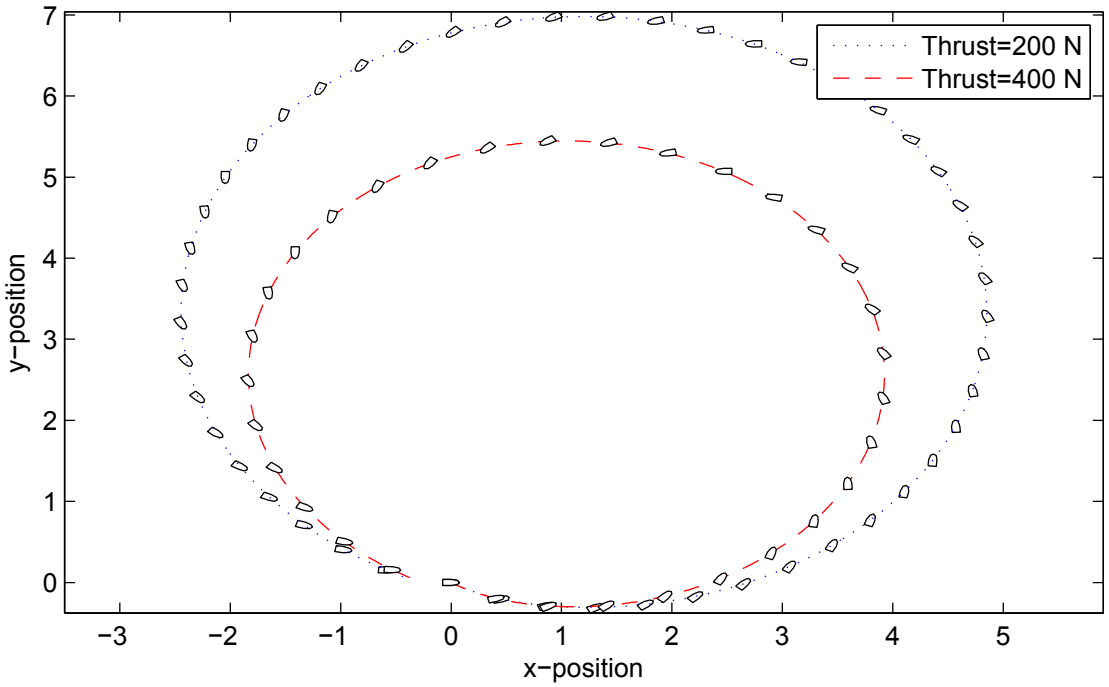

Figure 5.2. Turning radii for different thrust magnitudes

the algorithm to account for obstacles, current fields and energy constraints has been shown before, the focus of this section will be on showing that this still is possible for a sophisticated dynamical model. Therefore, a modified version of the jet current benchmark presented in section 4.4.2 will be used that has been extended by an obstacle blocking the direct path to the goal region. The objective is to plan a path between

$$[x_0,\ y_0,\ \theta_0,\ u_0,\ v_0,\ r_0] = [0,\ 0,\ 0,\ 0,\ 0,\ 0] \tag{5.16}$$

$$[x_f,\ y_f,\ \theta_f,\ u_f,\ v_f,\ r_f] = [21,\ 21,\ -\frac{\pi}{2}, 0, 0, 0]\ . \tag{5.17}$$

The path resulting from these boundary conditions is shown in figure 5.5.

Several observations can be made: firstly, the boat does not completely reach the specified goal position which is due to the specified goal tolerance r_{goal}. Within this tolerance, however, the result is quite precise, especially for the heading angle

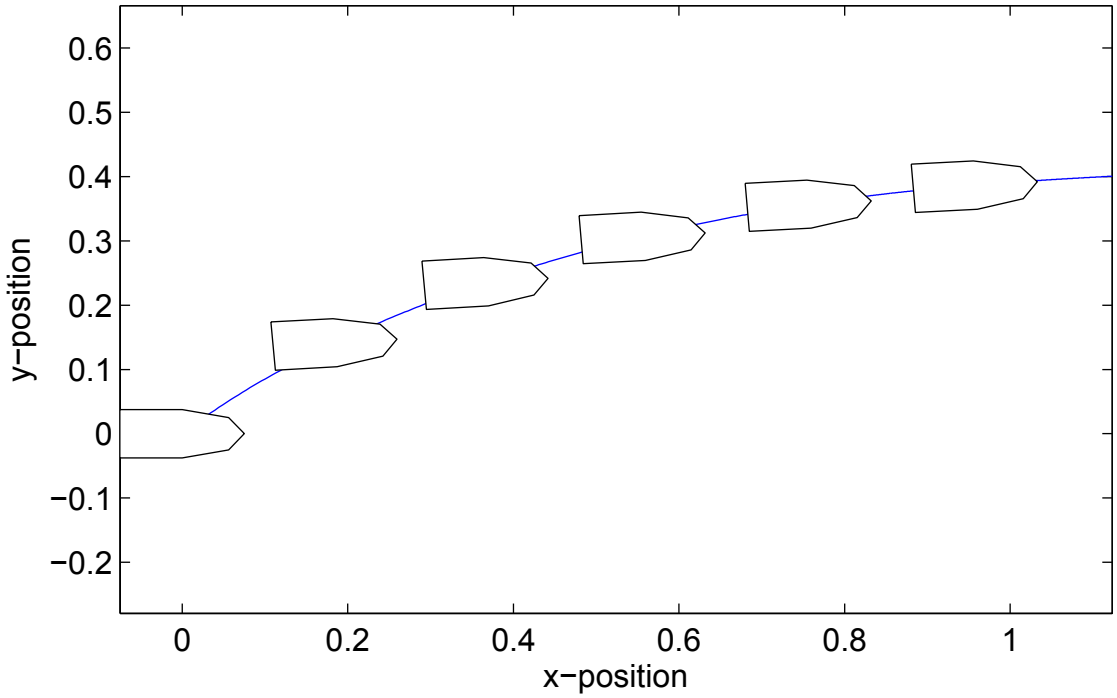

Figure 5.3. Path resulting from given initial speeds

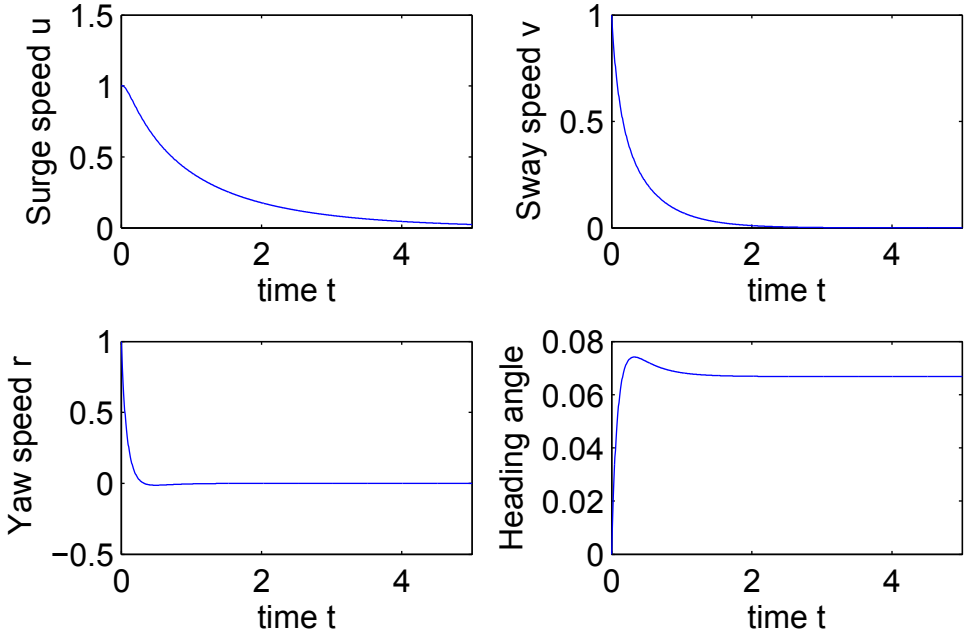

Figure 5.4. Speeds and heading angle

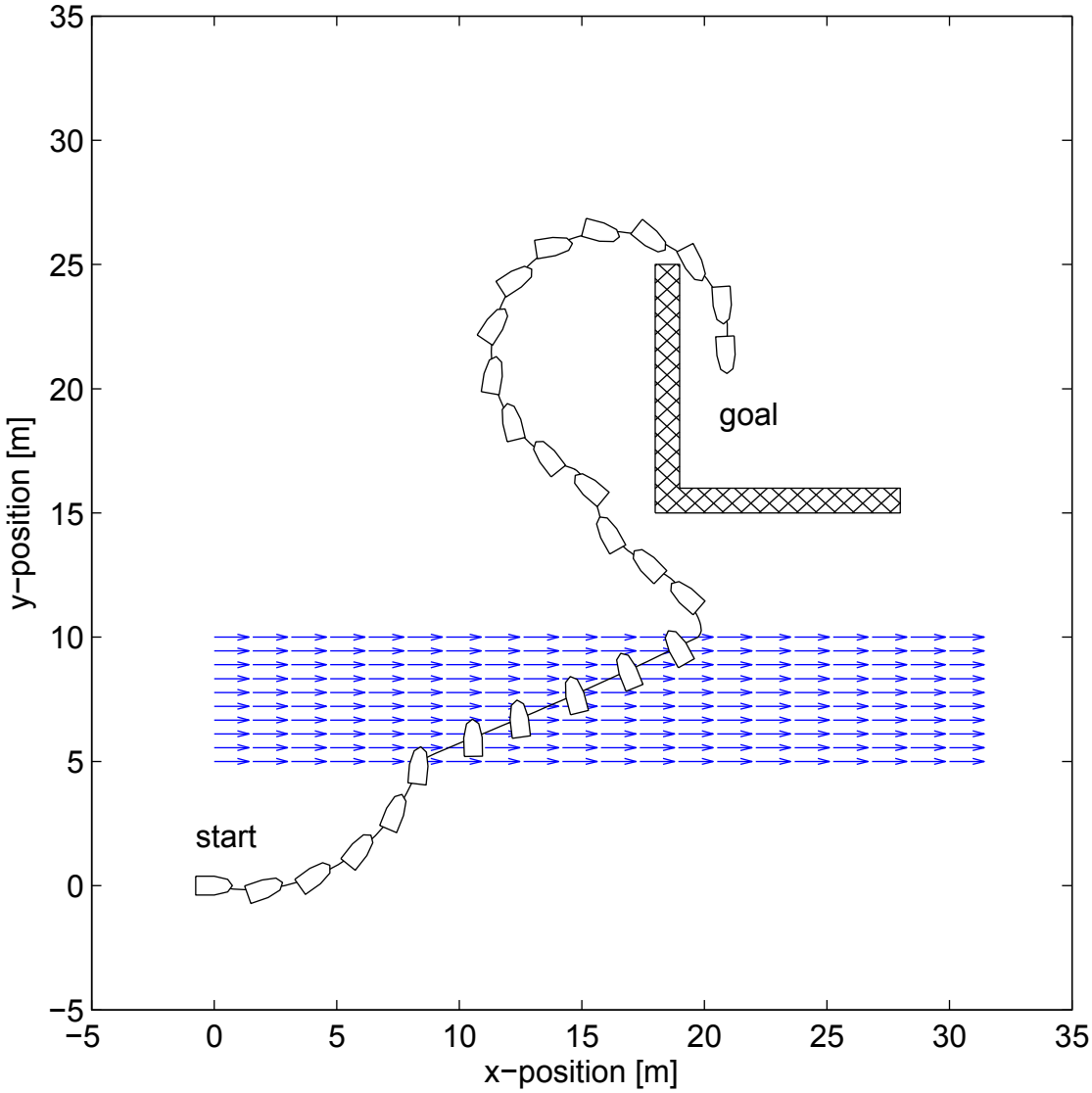

Figure 5.5. Time-optimal path for a dynamical boat model

for which deviations were penalized harder than deviations with respect to the position.

Secondly, the boat's motion is clearly affected by the current field and takes advantage of this. It also avoids the obstacle blocking the direct way to the goal. However, it also has to be noticed that some parts of the found path, especially close to the corner of the obstacle, are not optimal, which is due to the rugged shape of the control sequence. This could be improved by the methods described in 4.7.

Finally, it has to be observed that the dynamics of the boat have a great impact on the shape of the path which can be seen from the large turning radius upon approaching the goal.

The influence of the boat's dynamics is also emphasized by the path shown in figure 5.6, where the final heading angle was set to $\theta_f = \pi$. This change in the final heading angle as opposed to the one specified in (5.17) leads to a completely different path which takes less time than the one shown in figure 5.5 because it can make more use of the current field. However, due to the large turning radius of the boat, following the course around the right corner of the obstacle would have taken longer for a final heading angle of $\theta_f = -\frac{\pi}{2}$ than following the path of figure 5.5.

5.3 Discussion

In the preceding chapter, a dynamical model for a small boat was derived and implemented into the Sparse RRT algorithm in order to plan paths. As it was shown, reasonable results could be obtained. However, some difficulties can arise while planning: in general, the time it takes to obtain good results increases with the dimensionality of the state space, so fast computers are necessary. This is due to the fact that the search has to cover a larger search area and in addition, each

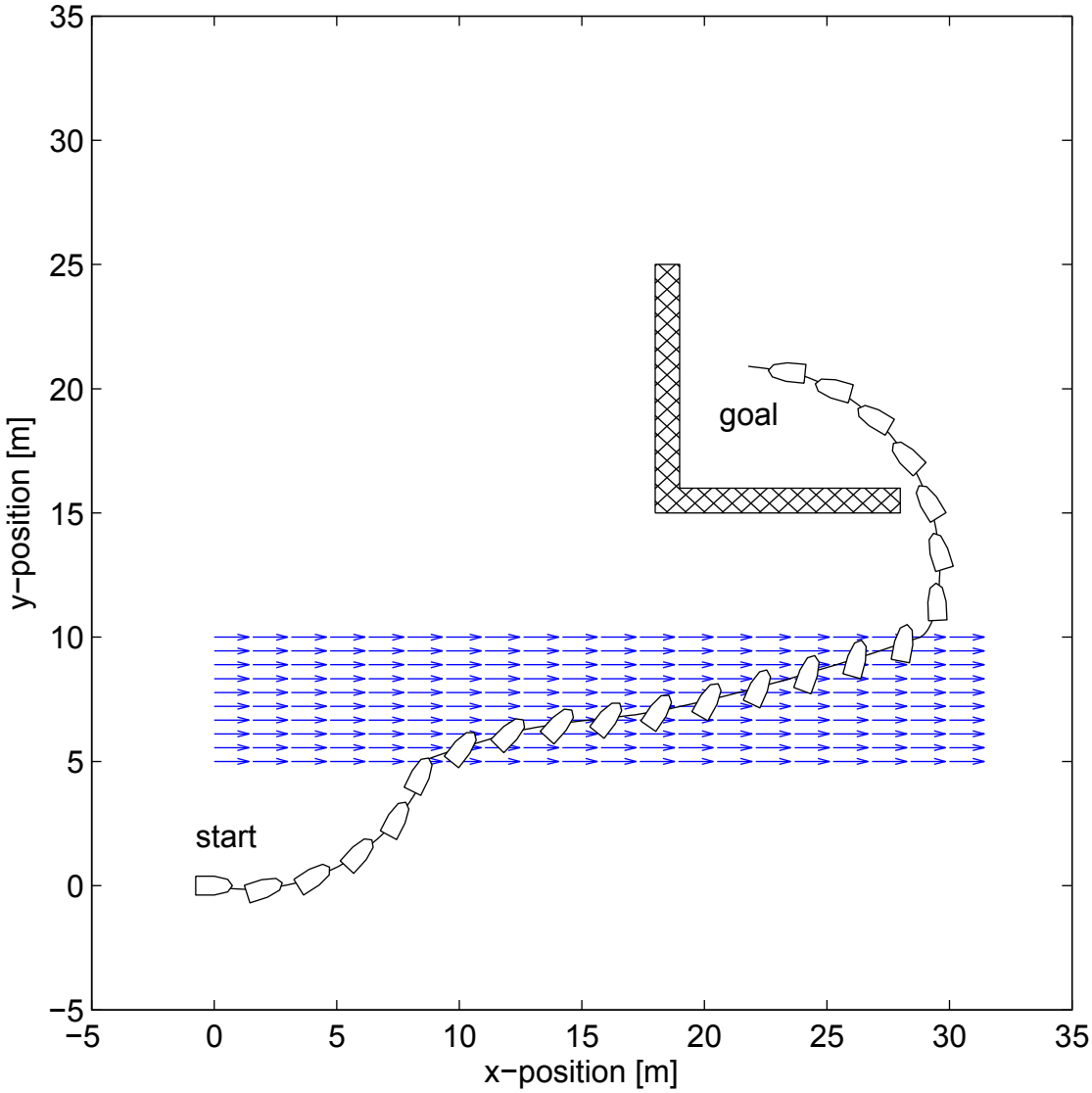

Figure 5.6. Time-optimal path for different final heading direction

added dimension increases the size of the *kd*-tree, slowing the algorithm down.

Most importantly, if a path is to be planned for a highly underactuated model like the one considered, the question of reachability becomes of importance, meaning that many states might not be accessible at all. This can be due to a constrained turning radius, limited thrust or the boat lacking the ability to break. This has to be considered when setting up a planning task. A possible way to alleviate planning is to specify boundary conditions only for some dimensions, for example only for position, heading angle and surge speed, while others are left unspecified.

CHAPTER 6

Summary and future prospects

6.1 Summary

In this thesis, the problem of time-opimal path planning for an unmanned ocean surface vehicle has been considered. Initially, existing approaches to solve this problem have been reviewed, considering both advantages and shortcomings.

In general, these methods could be classified into two groups: planning without differential constraints and planning which takes these into account. While the former are prevalent, the fact that they do not regard dynamical effects makes them suitable only for long-range path planning where these effects can be neglected. For short-range path planning aiming at determining driving maneuvers instead of a set of way points, however, differential constraints have to be taken into account.

Existing attempts to this specific problem were shown to have covered only parts of the required capabilities which are accounting for dynamical effects, obstacles, current fields and energy constraints. While Dynamic Programming as the probably most general tool was discarded due to its limitation to low dimensions and nonlinear optimization-based optimal control software due to potential convergence problems, sampling-based planning algorithms, namely the Sparse RRT algorithm, have been identified as promising solution methods.

This algorithm then was tested in several benchmarks based on optimal control theory and was shown to perform well. Its ability to incorporate all of the above mentioned constraints was demonstrated. Even though a non-negligible dependency of the quality of the results on the parameters of the algorithm was noticed and discussed, in total it turned out to be a flexible tool applicable to the considered problem. Due to its flexibility, the application to other problems as for example energy-optimal planning with time constraints may not be a problem.

The application of the Sparse RRT algorithm to a dynamic boat model with a higher-dimensional state space also led to good results. However, obtaining a solution turned out to be more complicated than for lower-dimensional models, mainly due to increased run-time requirements and the underactuation of the boat.

6.2 Future prospects

Several directions for future inquiries can be pointed out: primarily, gaining a more rigorous understanding of the algorithm parameter's influence would be valuable. A paper by the authors of [3] about this topic, having been under review upon completion of this thesis, may shed more light on this.

Also, techniques for improving the existing algorithm are thinkable of. For example, it could benefit the obtainable results if sampling would be focused towards the best solution known so far as time proceeds, introducing a simulated annealing-like behavior as known from metaheuristic optimization. Even though this has potential to further loosen the probabilistic completeness of the algorithm, in cases without strong local minima it might turn out to be advantageous.

In addition, actual performance comparisons with existing approaches like Dynamic Programming or other sampling-based algorithms could be of interest to determine which is more adequate for a certain problem set.

Besides this, the implications of a newly developed algorithm that samples the control space directly and propagates a randomly-chosen node in configuration space have to be determined. A (however not journal published upon completion of this work) description of this algorithm can be found in [48].

The long-term goal of all research in this field is the application of obtained simulation results to real unmanned boats. This, however, requires more insight into closed-loop control for path-following that accounts for uncertainties, only to mention one challenge.

LIST OF REFERENCES

[1] J. T. Betts, *Practical Methods for Optimal Control Using Nonlinear Programming.* Philadelphia: SIAM, 2001.

[2] D. E. Kirk, *Optimal Control Theory. An Introduction.* Mineola, New York: Dover Publications, 2004.

[3] Z. Littlefield, Y. Li, and K. E. Bekris, "Efficient sampling-based motion planning with asymptotic near-optimality guarantees for systems with dynamics," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on.* IEEE, 2013, pp. 1779–1785.

[4] D. Chelidze, D. B. Segala, and T. Wettergren, "Nominal Trajectory Planning and Optimal Control for Slow Surface Boats," Naval Undersea Warfare Center, Tech. Rep., 2012.

[5] T. Lolla, M. P. Ueckermann, K. Yiğit, P. Haley Jr., and P. F. J. Lermusiaux, "Path planning in time dependent flow fields using level set methods," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, May 2012, pp. 166–173.

[6] E. Zermelo, "Über das Navigationsproblem bei ruhender oder veränderlicher Windverteilung," *Zeitschrift für angewandte Mathematik und Mechanik*, vol. 11, no. 2, pp. 114–124, apr 1931.

[7] R. Bachmayer, N. E. Leonhard, J. Graver, E. Fiorelli, P. Bhatta, and D. Paley, "Underwater Gliders: Recent Developments and Future Applications."

[8] S. M. LaValle, *Planning Algorithms.* Cambridge, U.K.: Cambridge University Press, 2006.

[9] M. Soulignac, P. Taillibert, and M. Rueher, "Time-minimal path planning in dynamic current fields," in *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, May 2009, pp. 2473–2479.

[10] M. Eichhorn, "Optimal routing strategies for autonomous underwater vehicles in time-varying environment," *Robotics and Autonomous Systems*, 2013.

[11] K. Yang and S. Sukkarieh, "3D smooth path planning for a UAV in cluttered natural environments," in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, Sept 2008, pp. 794–800.

[12] E. Fernandez-Perdomo, J. Cabrera-Gamez, D. Hernandez-Sosa, J. Isern-Gonzalez, A. Dominguez-Brito, A. Redondo, J. Coca, A. Ramos, E. Fanjul, and M. Garcia, "Path planning for gliders using Regional Ocean Models: Application of Pinzón path planner with the ESEOAT model and the RU27 trans-Atlantic flight data," in *OCEANS 2010 IEEE - Sydney*, May 2010, pp. 1–10.

[13] Pêtrès, C. and Pailhas, Y. and Patron, P. and Petillot, Y. and Evans, J. and Lane, D., "Path planning for autonomous underwater vehicles," *IEEE Transactions on Robotics*, vol. 23, no. 2, pp. 331–341, April 2007.

[14] K. Carroll, S. McClaran, E. Nelson, D. Barnett, D. Friesen, and G. William, "AUV path planning: an A* approach to path planning with consideration of variable vehicle speeds and multiple, overlapping, time-dependent exclusion zones," in *Autonomous Underwater Vehicle Technology, 1992. AUV '92., Proceedings of the 1992 Symposium on*, Jun 1992, pp. 79–84.

[15] B. Garau, A. Alvarez, and G. Oliver, "Path Planning of Autonomous Underwater Vehicles in Current Fields with Complex Spatial Variability: an A* Approach," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, April 2005, pp. 194–198.

[16] B. Garau, M. Bonet, A. Alvarez, S. Ruiz, and A. Pascual, "Path planning for autonomous underwater vehicles in realistic oceanic current fields: Application to gliders in the western mediterranean sea," *Journal of Maritime Research*, vol. 6, no. 2, pp. 5–22, 2009.

[17] S. V. T. Lolla, "Path Planning in Time Dependent Flows using Level Set Methods," SM thesis, MIT, Dept. of Mechanical Engineering, September 2012.

[18] K. Yiğit, "Path Planning Methods for Autonomous Underwater Vehicles," SM thesis, MIT, Dept. of Mechanical Engineering, June 2011.

[19] N. Yilmaz, C. Evangelinos, P. F. J. Lermusiaux, and N. Patrikalakis, "Path planning of autonomous underwater vehicles for adaptive sampling using mixed integer linear programming," *IEEE Journal of Oceanic Engineering*, vol. 33, no. 4, pp. 522–537, Oct 2008.

[20] T. G. McGee and J. K. Hedrick, "Optimal path planning with a kinematic airplane model," *Journal of guidance, control, and dynamics*, vol. 30, no. 2, pp. 629–633, 2007.

[21] A. E. Bryson and Y.-C. Ho, *Applied Optimal Control*, revised printing ed. Washington, D.C.: Hemisphere Publishing Corporation, 1975.

[22] I. Spangelo and O. Egeland, "Trajectory planning and collision avoidance for underwater vehicles using optimal control," *IEEE Journal of Oceanic Engineering*, vol. 19, no. 4, pp. 502–511, Oct 1994.

[23] T. Inanc, S. C. Shadden, and J. E. Marsden, "Optimal trajectory generation in ocean flows." American Automatic Control Council, 2005.

[24] W. Zhang, T. Inanc, S. Ober-Blöbaum, and J. Marsden, "Optimal trajectory generation for a glider in time-varying 2D ocean flows B-spline model," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, May 2008, pp. 1083–1088.

[25] B. Li, C. Xu, K. L. Teo, and J. Chu, "Time optimal Zermelos navigation problem with moving and fixed obstacles," *Applied Mathematics and Computation*, vol. 224, pp. 866–875, 2013.

[26] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Computing Surveys*, vol. 35, no. 3, pp. 268–308, Sept. 2003.

[27] C. Warren, "A technique for autonomous underwater vehicle route planning," *IEEE Journal of Oceanic Engineering*, vol. 15, no. 3, pp. 199–204, July 1990.

[28] J. Witt and M. Dunbabin, "Go with the flow: Optimal AUV path planning in coastal environments," in *Australian Conference on Robotics and Automation*, vol. 2008, no. 2, 2008.

[29] A. Alvarez, A. Caiti, and R. Onken, "Evolutionary path planning for autonomous underwater vehicles in a variable ocean," *Oceanic Engineering, IEEE Journal of*, vol. 29, no. 2, pp. 418–429, April 2004.

[30] J. B. Escario, J. F. Jimenez, and J. M. Giron-Sierra, "Optimisation of autonomous ship manoeuvres applying Ant Colony Optimisation metaheuristic," *Expert Systems with Applications*, vol. 39, no. 11, pp. 10 120–10 139, 2012.

[31] M. Aghababa, M. Amrollahi, and M. Borjkhani, "Application of GA, PSO, and ACO algorithms to path planning of autonomous underwater vehicles," *Journal of Marine Science and Application*, vol. 11, no. 3, pp. 378–386, 2012.

[32] M. P. Aghababa, "3D path planning for underwater vehicles using five evolutionary optimization algorithms avoiding static and energetic obstacles ," *Applied Ocean Research*, vol. 38, no. 0, pp. 48 – 62, 2012.

[33] S. LaValle, "Rapidly-Exploring Random Trees: A New Tool for Path Planning," 1998.

[34] S. LaValle and J. Kuffner, "Rapidly-Exploring Random Trees: Progress and Prospects," 2000.

[35] S. LaValle and J. Kuffner, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[36] S. Karaman and E. Frazzoli, "Optimal Kinodynamic Motion Planning using Incremental Sampling-based Methods," in *IEEE Conference on Decision and Control (CDC)*, Atlanta, GA, December 2010.

[37] D. Rao and S. Williams, "Large-scale path planning for Underwater Gliders in ocean currents," in *Australasian Conference on Robotics and Automation (ACRA)*, 2009.

[38] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," in *Robotics: Science and Systems (RSS)*, Zaragoza, Spain, June 2010.

[39] S. Karaman and E. Frazzoli, "Sampling-based Algorithms for Optimal Motion Planning," *International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, June 2011.

[40] J. h. Jeon, S. Karaman, and E. Frazzoli, "Anytime computation of time-optimal off-road vehicle maneuvers using the RRT*," in *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on*. IEEE, 2011, pp. 3276–3282.

[41] M. Gerdts, "Optimal Control of Ordinary Differential Equations and Differential-Algebraic Equations," Habilitation thesis, University of Bayreuth, January 2007.

[42] S. Ober-Blöbaum, "Discrete mechanics and optimal control," PhD thesis, University of Paderborn, 2008.

[43] H. S. Rodrigues, M. T. T. Monteiro, and D. F. M. Torres, "Optimal control and numerical software: an overview," 2014, arXiv preprint arXiv:1401.7279.

[44] J. L. Bentley, "Multidimensional Binary Search Trees Used for Associative Searching," *Communications of the Association for Computing Machinery*, vol. 18, no. 9, pp. 509–517, Sept. 1975.

[45] R. F. Sproull, "Refinements to nearest-neighbor searching in k-dimensional trees," *Algorithmica*, vol. 6, no. 1-6, pp. 579–589, 1991.

[46] J. L. Bentley and J. H. Friedman, "Data structures for range searching," *ACM Computing Surveys (CSUR)*, vol. 11, no. 4, pp. 397–409, 1979.

[47] T. I. Fossen, *Guidance and control of ocean vehicles*. Wiley New York, 1994.

[48] G. Papadopoulos, H. Kurniawati, and N. M. Patrikalakis, "Analysis of Asymptotically Optimal Sampling-based Motion Planning Algorithms for Lipschitz Continuous Dynamical Systems," 2014, arXiv:1405.2872v1.

APPENDIX

Model parameters

$$P \in [0, 200\,\mathrm{N}]$$

$$\phi \in \left[-\frac{\pi}{4}, \frac{\pi}{4}\right]$$

$$m = 153.94\,\mathrm{kg}$$

$$I_z = 73.04\,\mathrm{kg\,m^2}$$

$$a = 1.5\,\mathrm{m}$$

$$X_{\dot{u}} = 18.17\,\mathrm{kg} \qquad Y_{\dot{v}} = 124.54\,\mathrm{kg}$$

$$Y_{\dot{r}} = 0\,\mathrm{kg\,m} \qquad N_{\dot{r}} = 36.15\,\mathrm{kg\,m}$$

$$X_u = 107.33\,\mathrm{kg/s} \qquad Y_v = 536.67\,\mathrm{kg/s}$$

$$Y_r = 322\,\mathrm{kg\,m/s} \qquad N_v = 0\,\mathrm{kg\,m/s} \qquad N_r = 1073.33\,\mathrm{kg\,m^2/s}$$

$$X_{u|u|} = 107.33\,\mathrm{kg/m} \qquad Y_{v|v|} = 536.67\,\mathrm{kg/m} \qquad N_{r|r|} = 322\,\mathrm{kg\,m^2}$$

(Courtesy of [4])

BIBLIOGRAPHY

Aghababa, M., Amrollahi, M., and Borjkhani, M., "Application of GA, PSO, and ACO algorithms to path planning of autonomous underwater vehicles," *Journal of Marine Science and Application*, vol. 11, no. 3, pp. 378–386, 2012.

Aghababa, M. P., "3D path planning for underwater vehicles using five evolutionary optimization algorithms avoiding static and energetic obstacles ," *Applied Ocean Research*, vol. 38, no. 0, pp. 48 – 62, 2012.

Alvarez, A., Caiti, A., and Onken, R., "Evolutionary path planning for autonomous underwater vehicles in a variable ocean," *Oceanic Engineering, IEEE Journal of*, vol. 29, no. 2, pp. 418–429, April 2004.

Bachmayer, R., Leonhard, N. E., Graver, J., Fiorelli, E., Bhatta, P., and Paley, D., "Underwater Gliders: Recent Developments and Future Applications."

Bentley, J. L., "Multidimensional Binary Search Trees Used for Associative Searching," *Communications of the Association for Computing Machinery*, vol. 18, no. 9, pp. 509–517, Sept. 1975.

Bentley, J. L. and Friedman, J. H., "Data structures for range searching," *ACM Computing Surveys (CSUR)*, vol. 11, no. 4, pp. 397–409, 1979.

Betts, J. T., *Practical Methods for Optimal Control Using Nonlinear Programming*. Philadelphia: SIAM, 2001.

Blum, C. and Roli, A., "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Computing Surveys*, vol. 35, no. 3, pp. 268–308, Sept. 2003.

Bryson, A. E. and Ho, Y.-C., *Applied Optimal Control*, revised printing ed. Washington, D.C.: Hemisphere Publishing Corporation, 1975.

Carroll, K., McClaran, S., Nelson, E., Barnett, D., Friesen, D., and William, G., "AUV path planning: an A* approach to path planning with consideration of variable vehicle speeds and multiple, overlapping, time-dependent exclusion zones," in *Autonomous Underwater Vehicle Technology, 1992. AUV '92., Proceedings of the 1992 Symposium on*, Jun 1992, pp. 79–84.

Chelidze, D., Segala, D. B., and Wettergren, T., "Nominal Trajectory Planning and Optimal Control for Slow Surface Boats," Naval Undersea Warfare Center, Tech. Rep., 2012.

Eichhorn, M., "Optimal routing strategies for autonomous underwater vehicles in time-varying environment," *Robotics and Autonomous Systems*, 2013.

Escario, J. B., Jimenez, J. F., and Giron-Sierra, J. M., "Optimisation of autonomous ship manoeuvres applying Ant Colony Optimisation metaheuristic," *Expert Systems with Applications*, vol. 39, no. 11, pp. 10 120–10 139, 2012.

Fernandez-Perdomo, E., Cabrera-Gamez, J., Hernandez-Sosa, D., Isern-Gonzalez, J., Dominguez-Brito, A., Redondo, A., Coca, J., Ramos, A., Fanjul, E., and Garcia, M., "Path planning for gliders using Regional Ocean Models: Application of Pinzón path planner with the ESEOAT model and the RU27 trans-Atlantic flight data," in *OCEANS 2010 IEEE - Sydney*, May 2010, pp. 1–10.

Fossen, T. I., *Guidance and control of ocean vehicles*. Wiley New York, 1994.

Garau, B., Alvarez, A., and Oliver, G., "Path Planning of Autonomous Underwater Vehicles in Current Fields with Complex Spatial Variability: an A* Approach," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, April 2005, pp. 194–198.

Garau, B., Bonet, M., Alvarez, A., Ruiz, S., and Pascual, A., "Path planning for autonomous underwater vehicles in realistic oceanic current fields: Application to gliders in the western mediterranean sea," *Journal of Maritime Research*, vol. 6, no. 2, pp. 5–22, 2009.

Gerdts, M., "Optimal Control of Ordinary Differential Equations and Differential-Algebraic Equations," Habilitation thesis, University of Bayreuth, January 2007.

h. Jeon, J., Karaman, S., and Frazzoli, E., "Anytime computation of time-optimal off-road vehicle maneuvers using the RRT*," in *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on*. IEEE, 2011, pp. 3276–3282.

Inanc, T., Shadden, S. C., and Marsden, J. E., "Optimal trajectory generation in ocean flows." American Automatic Control Council, 2005.

Karaman, S. and Frazzoli, E., "Incremental sampling-based algorithms for optimal motion planning," in *Robotics: Science and Systems (RSS)*, Zaragoza, Spain, June 2010.

Karaman, S. and Frazzoli, E., "Optimal Kinodynamic Motion Planning using Incremental Sampling-based Methods," in *IEEE Conference on Decision and Control (CDC)*, Atlanta, GA, December 2010.

Karaman, S. and Frazzoli, E., "Sampling-based Algorithms for Optimal Motion Planning," *International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, June 2011.

Kirk, D. E., *Optimal Control Theory. An Introduction.* Mineola, New York: Dover Publications, 2004.

LaValle, S. M., *Planning Algorithms.* Cambridge, U.K.: Cambridge University Press, 2006.

LaValle, S., "Rapidly-Exploring Random Trees: A New Tool for Path Planning," 1998.

LaValle, S. and Kuffner, J., "Rapidly-Exploring Random Trees: Progress and Prospects," 2000.

LaValle, S. and Kuffner, J., "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

Li, B., Xu, C., Teo, K. L., and Chu, J., "Time optimal Zermelos navigation problem with moving and fixed obstacles," *Applied Mathematics and Computation*, vol. 224, pp. 866–875, 2013.

Littlefield, Z., Li, Y., and Bekris, K. E., "Efficient sampling-based motion planning with asymptotic near-optimality guarantees for systems with dynamics," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on.* IEEE, 2013, pp. 1779–1785.

Lolla, S. V. T., "Path Planning in Time Dependent Flows using Level Set Methods," SM thesis, MIT, Dept. of Mechanical Engineering, September 2012.

Lolla, T., Ueckermann, M. P., Yiğit, K., Haley Jr., P., and Lermusiaux, P. F. J., "Path planning in time dependent flow fields using level set methods," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, May 2012, pp. 166–173.

McGee, T. G. and Hedrick, J. K., "Optimal path planning with a kinematic airplane model," *Journal of guidance, control, and dynamics*, vol. 30, no. 2, pp. 629–633, 2007.

Ober-Blöbaum, S., "Discrete mechanics and optimal control," PhD thesis, University of Paderborn, 2008.

Papadopoulos, G., Kurniawati, H., and Patrikalakis, N. M., "Analysis of Asymptotically Optimal Sampling-based Motion Planning Algorithms for Lipschitz Continuous Dynamical Systems," 2014, arXiv:1405.2872v1.

Pêtrès, C. and Pailhas, Y. and Patron, P. and Petillot, Y. and Evans, J. and Lane, D., "Path planning for autonomous underwater vehicles," *IEEE Transactions on Robotics*, vol. 23, no. 2, pp. 331–341, April 2007.

Rao, D. and Williams, S., "Large-scale path planning for Underwater Gliders in ocean currents," in *Australasian Conference on Robotics and Automation (ACRA)*, 2009.

Rodrigues, H. S., Monteiro, M. T. T., and Torres, D. F. M., "Optimal control and numerical software: an overview," 2014, arXiv preprint arXiv:1401.7279.

Soulignac, M., Taillibert, P., and Rueher, M., "Time-minimal path planning in dynamic current fields," in *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, May 2009, pp. 2473–2479.

Spangelo, I. and Egeland, O., "Trajectory planning and collision avoidance for underwater vehicles using optimal control," *IEEE Journal of Oceanic Engineering*, vol. 19, no. 4, pp. 502–511, Oct 1994.

Sproull, R. F., "Refinements to nearest-neighbor searching in k-dimensional trees," *Algorithmica*, vol. 6, no. 1-6, pp. 579–589, 1991.

Warren, C., "A technique for autonomous underwater vehicle route planning," *IEEE Journal of Oceanic Engineering*, vol. 15, no. 3, pp. 199–204, July 1990.

Witt, J. and Dunbabin, M., "Go with the flow: Optimal AUV path planning in coastal environments," in *Australian Conference on Robotics and Automation*, vol. 2008, no. 2, 2008.

Yang, K. and Sukkarieh, S., "3D smooth path planning for a UAV in cluttered natural environments," in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, Sept 2008, pp. 794–800.

Yiğit, K., "Path Planning Methods for Autonomous Underwater Vehicles," SM thesis, MIT, Dept. of Mechanical Engineering, June 2011.

Yilmaz, N., Evangelinos, C., Lermusiaux, P. F. J., and Patrikalakis, N., "Path planning of autonomous underwater vehicles for adaptive sampling using mixed integer linear programming," *IEEE Journal of Oceanic Engineering*, vol. 33, no. 4, pp. 522–537, Oct 2008.

Zermelo, E., "Über das Navigationsproblem bei ruhender oder veränderlicher Windverteilung," *Zeitschrift für angewandte Mathematik und Mechanik*, vol. 11, no. 2, pp. 114–124, apr 1931.

Zhang, W., Inanc, T., Ober-Blöbaum, S., and Marsden, J., "Optimal trajectory generation for a glider in time-varying 2D ocean flows B-spline model," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, May 2008, pp. 1083–1088.