

Optimal Coverage and Probabilistic Robust Path Planning for Nonholonomic Mobile Robots

(非ホロノミック移動ロボットの最適領域被覆と確率的ロバスト動作軌道計画)

January, 2021

Doctor of Philosophy (Engineering)

Schae fle Tobias Rainer

シェーフレ トビアス ライナー

Toyohashi University of Technology

Declaration of Authorship

I, Tobias Rainer Schaeffe, declare that this thesis titled, “Optimal Coverage and Probabilistic Robust Path Planning for Nonholonomic Mobile Robots” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given.⁹ With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:



Date:

06.01.2021.

Abstract

In recent years, much attention is given to autonomous vehicles and mobile robots. In industry and households, mobile robots are deployed for a variety of tasks. A growing interest is in mobile robots in households, where it is used for lawn mowing and vacuum cleaning. Mobile robots are also commonly used for space exploration, specifically on Mars, Mars rovers like curiosity are used to explore the environment, and for cleaning train stations and industrial facilities. The above mentioned mobile robots are using path planning algorithms to fulfill their tasks. In general, these path planning tasks are divided into coverage path planning (CPP) and point-to-point (PTP) path planning. The former generates paths with an objective of moving everywhere in a preassigned area and the latter generates paths from a starting position to a goal position while avoiding obstacles. Challenges in these areas differ, CPP is in general used in known bounded environments, while PTP is often used in unknown, uncertain, dynamic environments.

Generally, mobile robots have a limit power supply due to the fact that they have to carry their own batteries. Thus, CPP is a power demanding problem for mobile robots. Therefore, one of the objectives of this dissertation is an algorithm that generates optimal CPP paths in known bounded environments, where the objective of the cost function is either traveling time, repetitive visits or energy consumption. The second objective of this dissertation is a probabilistic robust PTP path planner that generates probabilistic safe paths in real-time under state and environment uncertainties.

Part I of this dissertation presents new optimal offline approaches to solve the CPP problem. A novel hybrid genetic algorithm (HGA), which uses the turn-away starting point (TASP) and backtracking spiral algorithms (BSA) for performing local searches, is proposed for grid-based environmental representations. Three different variations of the HGA are proposed based on the underlying local search algorithm: HGA/BSA using the BSA for local search, HGA/TASP using the TASP algorithm and HGA/Both which uses both algorithms for the local search procedure. The HGA algorithms are validated using the following three different fitness functions: the number of cell visits, traveling time, and a new energy fitness function based on experimentally acquired energy values of fundamental motions. Computational results show that compared to conventional methods, HGA improves paths up to 38%; moreover, HGAs have a consistent fitness for different starting positions in an environment. Furthermore, experimental results prove the validity of the fitness function. It was shown, that the HGA/BSA finds good solutions for the number of visited cells, whereas the HGA/TASP provides better solutions for the traveling time. Both variations showed good results using energy consumption as fitness function. Furthermore, HGA/TASP has the fastest computation time among the HGAs.

Part II of the dissertation focuses on control approaches for parking and generating paths to a specific goal pose, where the focus is in particular on probabilistic robust path planning. A hybrid systems approach to garage parking of a differential drive mobile robot is proposed in this dissertation. The proposed system consists of three system states, in which each system state converges to its desired control value and then switches to the next state. Two system states use input-output linearization for multiple input multiple output (MIMO) systems to conduct exact linearization of the plant, which can then be controlled by a linear controller. Simulation and experimental results show that the robot converges to its desired states and safely parks at the final position in a reasonable time.

To guarantee safe motion planning, the underlying path planning algorithm must consider motion uncertainties and uncertain state information related to static, and dynamic obstacles. This dissertation proposes novel hybrid A* (HA*) algorithms that consider the uncertainty in the motion of a mobile robot, position uncertainty of static obstacles, and position and velocity uncertainty of dynamic obstacles. A variant of the HA* algorithm is proposed in this dissertation that uses a soft constraint in the cost function instead of chance constraints for probability guarantees, this algorithm offers a trade-off between the traveling distance and safety of the path without pruning any additional nodes. Furthermore, this dissertation introduces a method for considering the shape of a mobile robot for probabilistic safe path planning. The performance of the algorithms was evaluated using the Monte Carlo simulation. The results showed that safety can be improved without significantly increasing travel distance. The results also showed that dynamic obstacles were safely avoided, which is in contrast to the conventional HA* algorithm that has a high probability of collision. Furthermore, considering the shape of the robot in the proposed probabilistic approach led to safer paths overall. Additionally, the HA* algorithm that used the chance constraints was very conservative as such, exhibited a very low probability of collision. However, with an increasing number of obstacles, the algorithm may fail to find a solution due to pruning all the expanded nodes. In addition, a probabilistic robust planner that avoids the use of the Gaussian error function or inverse Gaussian error function is proposed. The resulting planner generates a confidence ellipse of the current state and covers the resulting ellipse with two circles of equal radius. The radius of the circles is used to inflate the obstacles and collision with obstacles can be detected with deterministic approaches. The planner was able to find probabilistic robust paths with a smaller computation time than existing probabilistic robust path planners. Lastly, a planner is proposed that receives measurement feedback from the environment and selects the linear velocity at each node according to the current probability of collision. The resulting paths successfully avoid static and dynamic obstacles and had a shorter travelling time than existing planners.

Acknowledgement

I would like to express my thanks to Prof. Naoki Uchiyama for his advice, guidance and for always taking his time to answer my questions. Without his help I would have not decided to take my PhD. I will always be grateful for his support along my academic journey.

Sincere thanks are given to my reviewers Prof. Jun Miura and Prof. Moeto Nagai. I am very thankful for their fruitful feedback and time they spend for my thesis.

Many thanks go to the Amano Institute of Technology for providing valuable financial support for my PhD. Without their help I could not go to Toyohashi even Japan for my studies.

I would also like to thank all the members of the systems engineering laboratory and all my friends in Toyohashi. It is impossible to name them all but I will always cherish the moments and memories that I shared with you.

Many thanks go also to some of my closer laboratory colleagues with whom I could discuss my research. They always provided help when I was stuck with an equation or an algorithm.

Many thanks go to my mother, father and brother in Germany for their continuous support. I was always grateful and happy when I received a package from Germany.

Last but not least, I would like to thank my wife for her patients and support. Especially the last months of my PhD were very hard and I am very thankful for having her.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Fundamentals	2
1.3	Literature review	9
1.4	Outline and summary of contributions	13
1.4.1	Outline	13
1.4.2	Contribution	13
2	Optimal Offline Coverage Path Planning	17
2.1	Problem formulation and environmental model	17
2.1.1	Problem formulation	17
2.1.2	Environmental model	17
2.2	Local search and initialization algorithms	18
2.2.1	Turn-away starting point algorithm	18
2.2.2	Backtracking spiral algorithm	19
2.3	Extension to hybrid genetic algorithm	20
2.3.1	Chromosome design	20
2.3.2	Initialization	21
2.3.3	Fitness function	21
2.3.4	Genetic operators	24
2.3.5	Termination	25
2.3.6	Point-to-point motion	26
2.3.7	Feasibility	26
2.4	Results and discussion	27
2.4.1	Fitness cost identification	28
2.4.2	Comparison of different approaches	30
2.5	Summary	35
3	Hybrid Systems Control	39
3.1	Introduction	39
3.2	Kinematic model	39
3.3	Hybrid systems approach	41
3.3.1	Input/output-linearization	42
3.3.2	Controller design	43
3.3.3	Stability analysis	45

3.4	Results and discussion	46
3.4.1	Simulation results	46
3.4.2	Experimental results	46
3.5	Summary	48
4	Probabilistic Safe Path Planning	51
4.1	Problem formulation	51
4.2	Hybrid A* algorithm and state and environment uncertainties	52
4.2.1	Kinematic model uncertainty propagation	52
4.2.2	Algorithm overview	54
4.2.3	Static and dynamic obstacles	55
4.3	Path planning under Gaussian uncertainty	56
4.3.1	Probabilistic collision check	57
4.3.2	Cost function extension	59
4.4	Arbitrary-shaped mobile robots	59
4.4.1	Results and discussion	62
4.5	Probabilistic robust path planning using the χ^2 -distribution	73
4.5.1	Generation of confidence ellipses	74
4.5.2	Collision detection	77
4.5.3	Extension for arbitrary-shaped robots	78
4.5.4	Results and discussion	79
4.6	Measurement feedback and adaptive velocity	82
4.6.1	Measurement feedback	83
4.6.2	Adaptive velocity	85
4.6.3	Results and discussion	86
4.7	Summary	89
5	Conclusions and Future Works	91
5.1	Conclusions	91
5.2	Future works	92
A	Linear transformation of variance and covariance	95
	List of Abbreviations	97
	Table of Symbols	99
	List of Figures	105
	List of Tables	107
	Bibliography	109

Notation

\mathbb{R}^n and $\mathbb{R}^{n \times m}$ represent an n -dimensional Euclidean space and $(n \times m)$ -dimensional matrix, respectively; \mathbf{A}^\top and I_n represent the transpose of the matrix \mathbf{A} and n -dimensional identity matrix, respectively. Bold uppercase, bold lowercase, and lowercase letters indicate matrices, vectors, and scalars, respectively. Uppercase Σ indicates a covariance matrix; σ_x^2 indicates the variance of variable x ; $\sigma_{x,y}$ indicates the covariance of variables x, y . The mean of a variable x is given by μ_x . $\boldsymbol{\varepsilon}_x \sim \mathcal{N}(\boldsymbol{\mu}_x, \Sigma_x)$ denotes the random Gaussian process disturbance of state vector \boldsymbol{x} .

1 Introduction

1.1 Motivation

In recent years much attention is given to autonomous vehicles and mobile robots, industrial areas start to deploy robots for several tasks. Most of all due to the defense advanced research projects agency (DARPA) Grand Challenge [115] and DARPA Urban Challenge [86, 116] many novel developments emerged in the area of path planning, perception, mapping, localization and control. Some recent mobile robots working in industrial fields are shown in Figure 1.1. The robot in Figure 1.1a is an agricultural autonomous robot that is used for transporting harvested products, spraying pesticides and fertilizers, weeding, and patrolling the fields, the mobile robot in Figure 1.1b is used in train station for cleaning, pedestrian support and security duties. Figure 1.1c shows a vacuum cleaning robot used in households, after vacuum cleaning the robot returns back to the charging station. The Mars rover Perseverance is shown in Figure 1.1d, it is currently deployed on Mars surface for exploration. Many challenges arise in the context of path planning in real-world environments for autonomous vehicles and mobile robots. Especially, dynamic environments and energy efficiency are key scenarios.

The above described robots are subject to two different path planning tasks, CPP [25, 38, 18] and PTP [63, 67, 66]. The former generates paths which objective is to move everywhere in a preassigned area and the latter generates paths from a starting position to a goal position while avoiding obstacles. Hereafter, the term path planning will be used interchangeably with PTP. Challenges in these areas differ, CPP works in general in known bounded environments, while PTP is often used in unknown, uncertain, dynamic environments.

Mobile robots have limit power supply, due to the fact that they have to carry their own battery. Thus, CPP is a power demanding problem for mobile robots. Existing CPP methods focus on complete coverage [71] and the number of repetitive visits of areas [37, 36]. Thus, one of the objectives of this thesis is an algorithm that generates optimal CPP paths in known bounded environments, where the objective of the cost function is either travelling time, repetitive visits or energy consumption. Flexible objectives allow a problem dependent coverage path generation, whereas e.g. energy consumption is selected for large environments or energy expensive mobile robots.

In general, PTP path planner do not consider real-world scenarios with uncertain motion patterns of dynamic obstacles, sensing noise, uncertain placement of static obstacles and noise in the robot motion. Therefore, the second objective of this research is the proposal of a PTP path planner that generates probabilistic safe paths in real-time in real-world environments. The focus is on paths in parking scenarios for autonomous cars and return paths in unstructured environments to the charging station for mobile robots, where precise motion and safety are



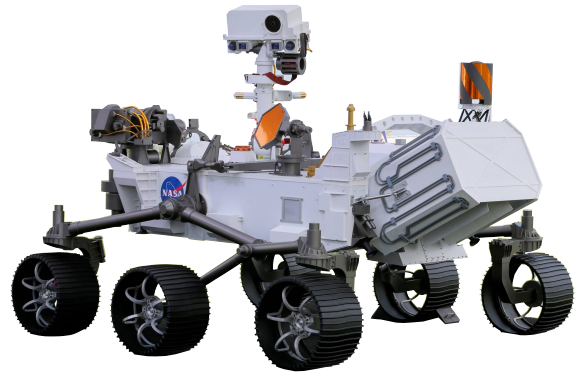
(a) Autonomous multifunctional agricultural robot for transportation, fertilizing, weeding, and patrolling.¹



(b) Industrial cleaning robot for cleaning train stations and performing security duties².



(c) Vacuum cleaning robot from iRobot³.



(d) Replica of the Mars rover Perseverance which is on an exploration mission on Mars surface⁴.

Figure 1.1: Examples of different mobile robots deployed in industry, households, and space exploration.

the main objectives of the generated path.

1.2 Fundamentals

This section briefly introduces some fundamental algorithms and methods used throughout this research.

¹https://www.maff.go.jp/e/policies/tech_res/smaagri/robot.html

²<https://asia.nikkei.com/Business/Transportation/Robots-roam-Tokyo-s-newest-train-station-to-patrol-and-sweep>

³<https://www.pngwing.com/en/free-png-ppwbt>

⁴<https://www.japantimes.co.jp/news/2020/07/30/world/science-health-world/nasa-rover-mars/>

Genetic algorithm and memetic algorithm

The genetic algorithm (GA) [39, 83, 20] is an evolution-based algorithm that uses different nature-inspired operators to change and improve current solutions of the optimization problem. Generally, a GA has a set of candidate solutions called *population* where one solution is a *chromosome*. A chromosome is encoded into a N_{GA} -element array, where each element, called *gene*, has a particular value. In discrete solution spaces chromosomes are in general binary. An optimal solution is found by taking the current population called *parent* to generate a new population (*offspring*) using the nature-inspired operators crossover, mutation and selection. These operators take the parent *generation* and modifies each chromosome to form the offspring. The crossover operator takes two parents and forms new offspring from both their genes, the mutation operator changes one or more genes of one parent to a different value and the selection operator selects the chromosomes, based on their cost (*fitness*), which will form the new offspring generation. Termination condition is set which is typically either a set number of generations or the algorithm terminates if no improvement is made in the last generation. The HGA [87, 88], also called memetic algorithm (MA), is an extension of the GA. The main difference is, that HGA perform a local search, after the modification with crossover and mutation, to improve the fitness of individual chromosomes. Figure 1.2 shows the cycle of a GA/HGA, first the initial population is set, then the chromosomes for the new generation are selected, which are modified with crossover and mutation operators. Thereafter, the fitness is evaluated and the termination condition will be checked. The dashed box applies the local search on the generated offspring before the fitness is evaluated.

Kalman filter

The Kalman filter (KF) [45] is a famous filtering method, which is used in robotics for localization [114] and sensor fusion [32]. The system model is assumed linear in the following form

$$\mathbf{x}_t = \mathbf{A}_t \mathbf{x}_{t-1} + \mathbf{B}_t \mathbf{u}_{t-1} + \boldsymbol{\varepsilon}_x, \quad (1.1)$$

where $\mathbf{x}_{t-1} \in \mathbb{R}^{n \times 1}$ is the state vector and $\mathbf{u}_{t-1} \in \mathbb{R}^{m \times 1}$ is the input vector at time instant $t-1$. $\mathbf{A}_t \in \mathbb{R}^{n \times n}$ is the state matrix and $\mathbf{B}_t \in \mathbb{R}^{n \times m}$ is the input matrix. The Gaussian random variable $\boldsymbol{\varepsilon}_x \sim \mathcal{N}(0, \Sigma_R)$ has zero mean and covariance matrix $\Sigma_R \in \mathbb{R}^{n \times n}$. The Gaussian random variable models the uncertainties of the system. Furthermore, the measurement equations are also assumed to be linear with additive Gaussian noise

$$\mathbf{y}_t = \mathbf{C}_t \mathbf{x}_{t-1} + \boldsymbol{\varepsilon}_y, \quad (1.2)$$

where $\mathbf{y}_t \in \mathbb{R}^{k \times 1}$ is the output vector, $\mathbf{C}_t \in \mathbb{R}^{k \times n}$ is the output matrix and $\boldsymbol{\varepsilon}_y \sim \mathcal{N}(0, \Sigma_Q)$ is a Gaussian random variable describing the measurement noise with zero mean and covariance matrix $\Sigma_Q \in \mathbb{R}^{k \times k}$. The belief of the state \mathbf{x}_t can be represented by the KF algorithm, provided the system ensures the above mentioned assumptions. Thus, the KF algorithm is denoted as

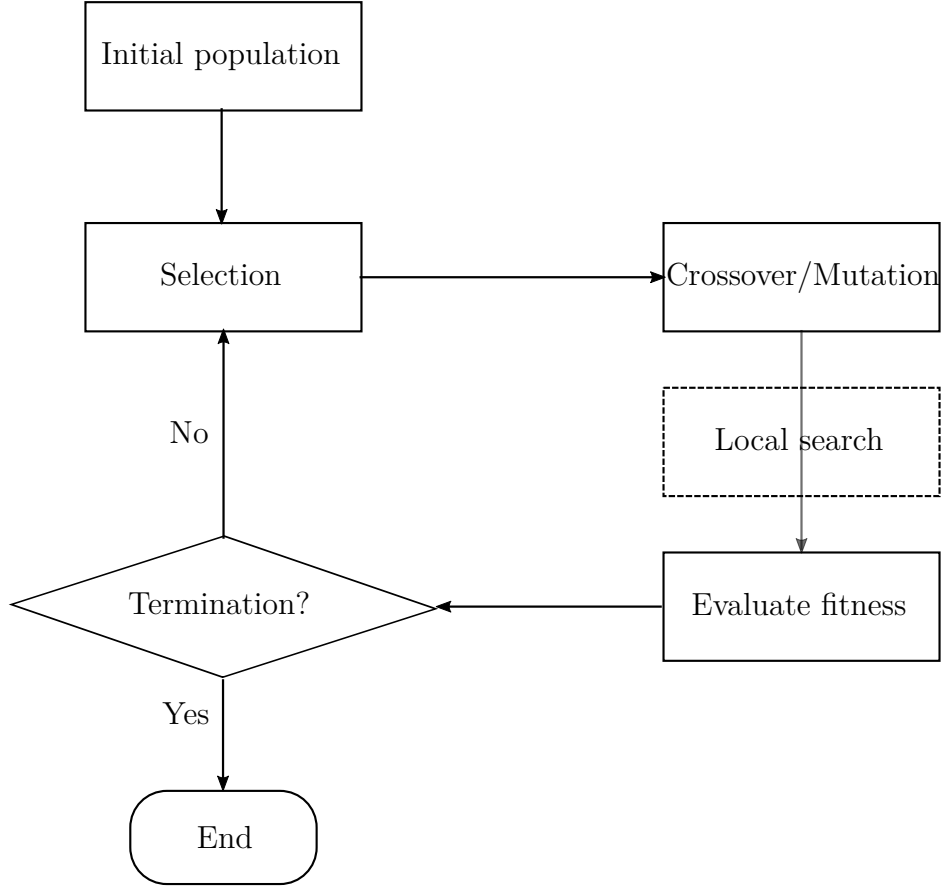


Figure 1.2: Cycle of GA and HGA

follows:

$$\bar{\boldsymbol{\mu}}_t = \mathbf{A}_t \boldsymbol{\mu}_{t-1} + \mathbf{B}_t \mathbf{u}_{t-1} \quad (1.3)$$

$$\bar{\boldsymbol{\Sigma}}_t = \mathbf{A}_t \boldsymbol{\Sigma}_{t-1} \mathbf{A}_t^\top + \boldsymbol{\Sigma}_R \quad (1.4)$$

$$\mathbf{K}_t = \bar{\boldsymbol{\Sigma}}_t \mathbf{C}_t^\top (\mathbf{C}_t \bar{\boldsymbol{\Sigma}}_t \mathbf{C}_t + \boldsymbol{\Sigma}_Q)^{-1} \quad (1.5)$$

$$\boldsymbol{\mu}_t = \bar{\boldsymbol{\mu}}_t + \mathbf{K}_t (\mathbf{y}_t - \mathbf{C}_t \bar{\boldsymbol{\mu}}_t) \quad (1.6)$$

$$\boldsymbol{\Sigma}_t = (\mathbf{I}_n - \mathbf{K}_t \mathbf{C}_t) \bar{\boldsymbol{\Sigma}}_t. \quad (1.7)$$

(1.3) to (1.4) represent the prediction step of the KF algorithm, where $\bar{\boldsymbol{\mu}}_t \in \mathbb{R}^{n \times 1}$, $\bar{\boldsymbol{\Sigma}}_t \in \mathbb{R}^{n \times n}$ denote the predicted mean state and predicted covariance matrix of the system at time instant t , respectively. $\boldsymbol{\Sigma}_{t-1} \in \mathbb{R}^{n \times n}$ represents the covariance matrix. (1.5) to (1.7) represent the correction step with the Kalman gain $\mathbf{K}_t \in \mathbb{R}^{n \times k}$ and the mean state $\boldsymbol{\mu}_t \in \mathbb{R}^{n \times 1}$ of the system. Dynamic systems are in general not linear, thus the KF will be extended for nonlinear systems

of the form

$$\mathbf{x}_t = \mathbf{f}(\mathbf{x}_{t-1}, \mathbf{u}_t) + \boldsymbol{\varepsilon}_x \quad (1.8)$$

$$\mathbf{y}_t = \mathbf{g}(\mathbf{x}_t) + \boldsymbol{\varepsilon}_y \quad (1.9)$$

to the extended Kalman filter (EKF). The EKF is similar to the KF with the difference that the nonlinear system will be linearized [34] around the mean state at each time instant

$$\mathbf{A}_t = \left. \frac{\partial \mathbf{f}}{\partial \mathbf{x}_{t-1}} \right|_{(\boldsymbol{\mu}_{t-1}, \mathbf{u}_t)} \quad (1.10)$$

$$\mathbf{B}_t = \left. \frac{\partial \mathbf{f}}{\partial \mathbf{u}} \right|_{(\boldsymbol{\mu}_{t-1}, \mathbf{u}_t)}. \quad (1.11)$$

Hence, only equations 1.3 and 1.6 change in the KF algorithm to

$$\bar{\boldsymbol{\mu}}_t = \mathbf{f}(\boldsymbol{\mu}_{t-1}, \mathbf{u}_t) \quad (1.12)$$

$$\boldsymbol{\mu}_t = \bar{\boldsymbol{\mu}}_t + \mathbf{K}_t (\mathbf{y}_t - \mathbf{g}(\bar{\boldsymbol{\mu}}_t)) \quad (1.13)$$

the other equations will be the same for the EKF.

Heuristics

In path planning heuristics [105] are used to make an “optimistic guess” of the cost-to-go from the current state \mathbf{x}_t to the goal state \mathbf{x}_G . A heuristic needs to be admissible to be used in path planning, which means that it should never overestimates the cost to reach the goal. In path planning one admissible heuristic is the euclidean distance between the current state and the goal state

$$h(\mathbf{x}_t) = \sqrt{(x_t - x_G)^2 + (y_t - y_G)^2}, \quad (1.14)$$

where (x_t, y_t) , (x_G, y_G) are the positions of the current and goal state, respectively and $h(\mathbf{x}_t)$ is the heuristic function. In general, heuristics reduce the number of expanded nodes and speed up the search process.

A* path planner

The A* [111] path planner is an extension of the Dijkstra algorithm [111]. It is a graph-based planner that uses heuristic costs to find the goal node in the search space. The cost function of each node of the planner

$$f(n) = g(n) + h(n) \quad (1.15)$$

takes the cost $g(n)$ to reach the current node and the heuristic cost $h(n)$ to calculate the cost of the node. Generally, A* uses an occupancy grid in mobile robot path planning, where the current node expands to one of the neighboring cells in the grid. A pseudo code is shown in Algorithm 1. The openList is a priority queue, the closedList is a set of nodes that are already expanded and the collision function checks if the successor node is colliding with any obstacle.

Algorithm 1 A* algorithm

```

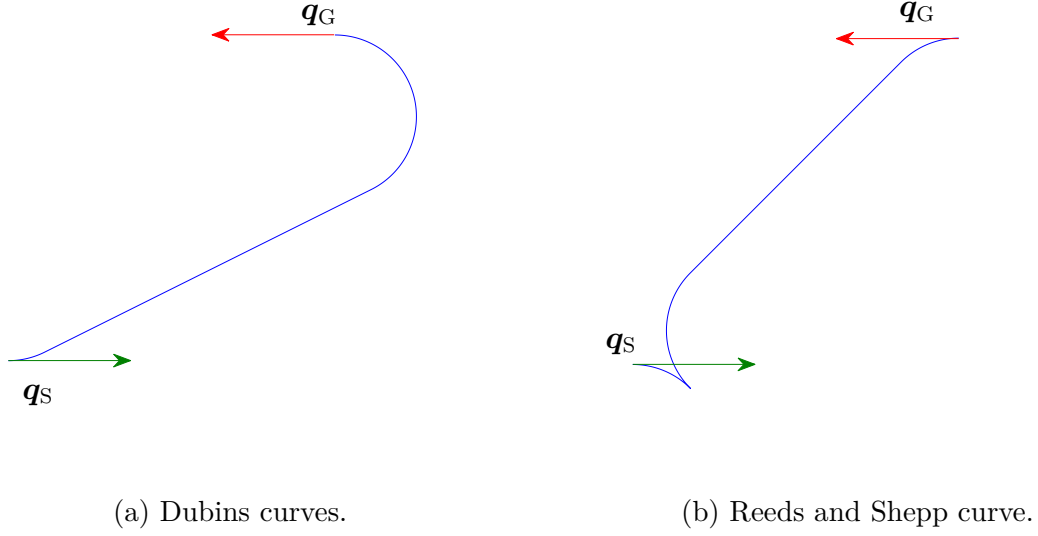
1: procedure A* SEARCH(start, goal)
2:   openList.push(start)
3:   while not(openList.isEmpty()) do
4:     node  $\leftarrow$  openList.pop()
5:     if node == goal then
6:       return true
7:     end if
8:     closedList.push(node)
9:     for all successor do
10:      if closedList.contains(successor) or collision(successor) then
11:        continue
12:      end if
13:      g_cost = node.g + cost(node, successor)
14:      if openList.contains(successor) or g_cost  $\geq$  successor.g then
15:        continue
16:      end if
17:      successor.predecessor  $\leftarrow$  node
18:      successor.g  $\leftarrow$  g_cost
19:      successor.f  $\leftarrow$  g_cost + h(successor)
20:      openList.push(successor)
21:    end for
22:  end while
23:  return false
24: end procedure

```

Dubins and Reeds & Shepp curves

Dubins curves [31, 63] return optimal curves from a start position q_S to a goal position q_G under the assumption that the car moves with a constant linear velocity u_v and maximum steering angle δ_{\max} , which results in a minimum turning angle R_{\min} . With $u_v = 1$ and $u_\omega \in \{-\tan \delta_{\max}, 0, \tan \delta_{\max}\}$ the optimal path of

$$\begin{aligned}
 \dot{x} &= \cos \theta \\
 \dot{y} &= \sin \theta \\
 \dot{\theta} &= u_\omega,
 \end{aligned} \tag{1.16}$$

Figure 1.3: Dubins and Reeds and Shepp curves from \mathbf{q}_S to \mathbf{q}_G

where (x, y, θ) are the pose of the robot, can be expressed as a combination of three different motion primitives. The motion primitives are *turn right* (R), *move straight* (S) and *turn left* (L). Hence, the optimal path is generated with one of the six possible combinations

$$\{LRL, RLR, LSL, RSR, LSR, RSL\}, \quad (1.17)$$

where each primitive is applied over an interval of time.

Reeds and Shepp curves [100, 63] are similar to Dubins curves, however they allow a reverse motion as well for the linear velocities. Thus $u_v \in -1, 1$ and the robot model is as follows:

$$\begin{aligned} \dot{x} &= u_v \cos \theta \\ \dot{y} &= u_v \sin \theta \\ \dot{\theta} &= u_v u_\omega. \end{aligned} \quad (1.18)$$

Hence, the shortest path using Reeds and Shepp curves with one of 48 possible combinations. Figure 1.3 shows a resulting path from \mathbf{q}_S to \mathbf{q}_G using Dubins curves (Figure 1.3a) and Reeds and Shepp curves (Figure 1.3b).

Canonical nonholonomic mobile robot kinematic model

The canonical nonholonomic mobile robot model generalizes the kinematic model of the unicycle model, the differential drive mobile robot, and the car-like mobile robot using Ackerman steering

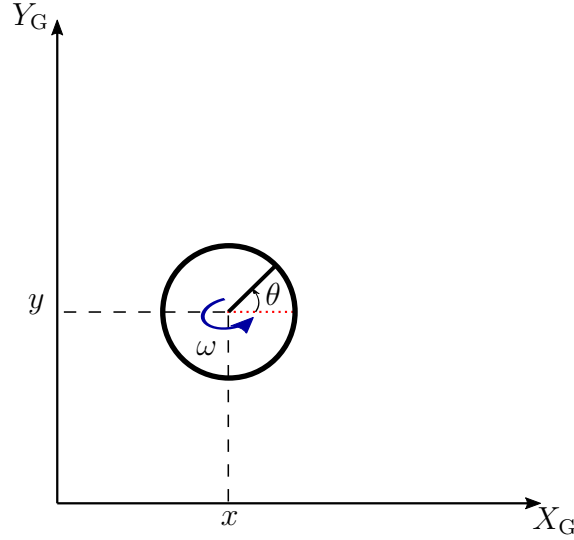


Figure 1.4: Canonical nonholonomic mobile robot kinematic model.

[79]. The model description is as follows:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos \theta & 0 \\ \sin \theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}, \quad (1.19)$$

where v and ω are the linear and angular velocities of the robot. Figure 1.4 depicts the canonical model in the global reference frame $\{X_G, Y_G\}$. Designing planners, controllers, etc. using the canonical model may lead to feasible solutions for the respective robot models as well, assuming that the respective constraints are considered.

Experimental systems

Two different industrial cleaning robots are used throughout this book. The robot shown in Figure 1.5 is a prototype version of the robot in Figure 1.6. It is a differential drive mobile robot with a mounted inertial measurement unit (IMU) and encoders for each driving wheel. The motors for the driving wheels can measure both current and voltage, hence the robot can be used to calculate the energy of the driven path. The industrial cleaning robot shown in Figure 1.6 does have an IMU mounted on the rear axis center, encoders for the drive wheels and a 2D Lidar sensor to read obstacles in the distance and generate a map of the environment. Around the robot are 14 ultrasonic sensors mounted used for detecting smaller obstacles such as curbs.

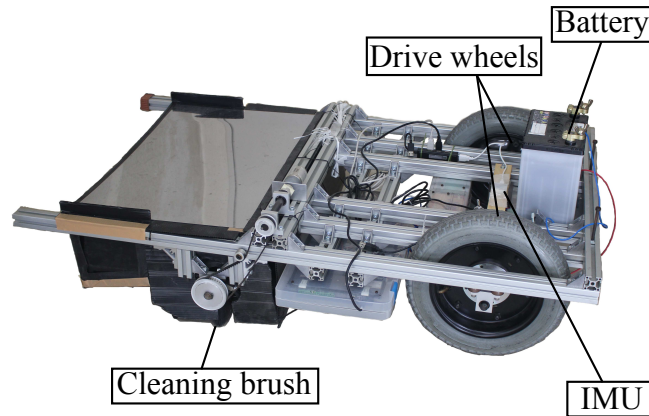


Figure 1.5: Prototype industrial cleaning robot.

1.3 Literature review

Coverage path planner

Due to the fact that CPP [38, 54] is a non-deterministic polynomial time hard (NP-hard) problem [6], finding optimal solutions or a solution at all is difficult. In recent years, many contributions have been made in this domain. Choset et al. proposed the boustrophedon cellular decomposition [26], which divides the known area into smaller cells. Thereupon, the robot moves from one cell to another, thereby covering each cell separately in zigzag motions. This approach was generalized using the critical points of Morse functions, called Morse-based cellular decomposition [3]. Furthermore, an online version of it uses a generalized Voronoi diagram with the boustrophedon cellular decomposition for the CPP [2]. In other approaches, the environment is decomposed into uniform grid cells, and the robot visits each unvisited cell. In [121], the wavefront algorithm was used to find a path that traverses each cell of the environment. Another approach generated a spiral path in the grid, on the basis of the spiral spanning tree coverage (Spiral-STC) algorithm, which can be used both online and offline [37, 36]. An extension of the Spiral-STC algorithm is the BSA, which uses a wall-following procedure to cover the area online [40, 41]. Yang and Luo proposed a neural network approach to online CPP, in which the neurons of the network are associated with each grid cell and every neuron is connected to its eight nearest neighbors [120, 78]. The aforementioned approach can be easily extended for multi-robot CPP. Kapanoglu et al. used a pattern-based GA for sensor-based CPP, where the area is divided into disks with a radius equal to the range of the sensing devices [47, 46]. The GA finds optimal paths by changing the segments of the existing candidate paths using operators such as selection and crossover. Miao et al. [82] proposed a scalable CPP algorithm that divides the map into rectangular sub-maps to reduce the execution time of the CPP process. An online algorithm called ϵ^* was proposed in [113]; it uses an exploratory turing machine as a supervisor for the mobile robot to guide it using adaptive navigation commands. Mitschke et al. [84] proposed the TASP algorithm as an online CPP approach that moves straight as long

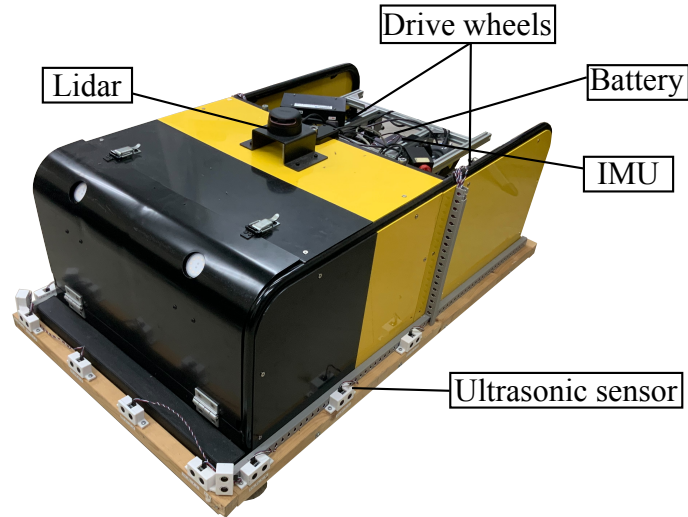


Figure 1.6: Industrial cleaning robot.

as possible to reduce both the number of turns and consumed energy of the path. Sasaki et al. [108] separated the cleaning area by considering the dust distribution, where areas with large quantities of dust had a priority while ignoring other areas. The optimality of the generated coverage paths in terms of path lengths or traveling time is addressed in several approaches. Jimenez et al. proposed an approach that used GA to achieve the optimal coverage [43]. The free space was divided into smaller subregions, and GA was used to find the optimal path to cover all the regions. Another approach used boustrophedon cellular decomposition to achieve complete coverage while minimizing the path of the mobile robot [80]. Lee et al. proposed an online approach that finds time- and energy-efficient complete coverage solutions by smoothing the coverage path to reduce accelerations and increase the average velocity [71]. Bochkarev et al. aimed to minimize the number of turns using convex decomposition; subsequently parallel path segments were placed on each region, and a minimum-cost tour through all regions was computed [15]. In recent studies, GA has been used in many different approaches to achieve complete coverage, and its effectiveness has been verified. Ryerson et al. proposed a GA-based CPP approach that finds the optimal path for farming operations [106]. Another approach used GA to find optimal mini-paths in terms of the traveled distance for complete coverage [119]. Sadek et al. used GA together with dynamic programming for online CPP with multiple objectives using only on-board sensors [107]. Dynamic programming was used to divide the complete problem into sub-problems and find coverage sub-paths, and GA was used to find the best multi-objective sub-paths. Energy-optimal paths were obtained using the GA approach by Schäfle et al. [109], where an energy-based fitness function was used.

Parking controller

Fuzzy systems receive much attention in the field of automotive parking control. Li et al. use a fuzzy sliding-mode controller for tracking control to the fifth-order polynomial trajectories for

garage parking and parallel parking [72]. An image-based fuzzy controller is used in [7], with inputs of the desired image features. Khoukhi uses a genetic neuro-fuzzy system for online multi objective motion planning, which is trained by simulation data from previous offline planning [55]. Demirli et al. also propose fifth-order polynomial references for a neuro-fuzzy sensor based controller for parallel parking with unknown parking space size [28]. Lee et al. separates the parking problem in two steps and uses a nonlinear controller based on the Lyapunov stability theory [70]. Ornik et al. proposed a hybrid control method called reach control, which uses a desired sequence of polytopes, that the trajectory needs to pass through, for parallel parking [94]. A navigation function to avoid obstacles or other parking cars, while parallel parking, is implemented in [97]. A geometric approach is proposed in [117], which can park parallel from any initial position and orientation of the robot. Zips et al. proposed an optimization based path planning algorithm for garage and parallel parking in narrow areas [123].

Point to point path planner

A* family of path planner

Many variations of the A* algorithm were introduced in the last decades. Were most researches focused on replanning [58, 56] and anytime planners [73]. Probably the most famous replanning algorithm is the D*-Lite from Koenig et al. [57]. The conventional A* algorithm suffers from the limitation, that it can only execute with a fixed set of directions, thus researchers developed algorithms which are able to move in any angle [92, 27, 91]. Recently the A* algorithm was adapted to consider the kinematics of the robot. The robot does not move anymore in discrete motions in the proposed HA* algorithm [30, 29]. The HA* algorithm was adapted to move from waypoint to waypoint in [96].

Sampling based path planner

Sampling-based path planners are generally separated into two categories, the probabilistic roadmap (PRM) planner [52] and the rapidly-exploring random tree (RRT) [64, 65, 68]. Karaman et al. improved both base algorithms to guarantee asymptotic optimality [49, 50, 48]. The optimal versions are named PRM* and RRT*. Kuffner et al. introduced the RRT-connect which expands two RRTs (one rooted at the start state and one rooted at the goal state) to find feasible solutions faster [60]. Anytime and replanning versions of the RRT algorithms were proposed in [51] and [33]. A real time version of RRT* for dynamic environments was proposed by Naderi et al. [90].

Local planner

Local planners are in general reactive planner which react based on the current state and measurement information of the robot. One famous method is the Bug algorithm [111] which moves along an obstacle until it can move again in a straight path to the goal state. Extensions of the algorithm improved the motion behaviour while moving along an obstacle [93, 24]. Potential field methods are using the concept of magnetism, where the robot is attracted to

goal state and repulsed by obstacles [111]. A variant of the potential fields method is the vector field histogram algorithm [17]. Fox et al. introduced the dynamic window approach which uses the motion dynamics of the robot to consider constraints in the states and actions of the robot [35]. A recent work proposes the timed elastic bands (TEB) method which locally optimizes the robots trajectory with respect to the execution time of the trajectory, separation of obstacles and kinodynamic constraints [103, 101, 102]. A model predictive control (MPC) approach considering non-rotational euclidean groups was proposed in [104].

Planning under uncertainty

Bry et al. introduces the rapidly-exploring random belief tree (RRBT) [19] which uses local linear quadratic Gaussian (LQG) control solutions to predict distributions over trajectories. A particle based rapidly exploring random trees path planner was developed by Melchior et al. [81], where each extension of the search tree is simulated multiple times under different conditions. Nodes are created by clustering simulation results. Model predictive control was combined with the particle approach to achieve optimal robust solutions [12, 13]. Particle based approaches are introduced to guarantee probabilistic safe paths, which considered non-Gaussian uncertainty [75]. Van den Berg et al. proposed a variant of LQG to consider motion uncertainty and imperfect state information [9]. Blackmore et al. presented a probabilistic approach which uses a maximum probability that a robot collides with an obstacle [11, 14]. The probability of collision is expressed as a disjunction of deterministic linear constraints. Luders et al. combined Blackmore et al.'s chance constrained method with rapidly exploring random trees and extended it by including uncertainty for obstacles [74]. The proposed chance constraint RRT (CCRRT) planner was then combined with a method to predict the future obstacle behavior in the planner [4]. Furthermore, it was extended for the RRT* [50, 48], which they called chance constraint RRT* (CCRRT*) [77]. Chen et al extended their space exploration guided heuristic path planner [21, 23] to consider perception and control uncertainties for self driving vehicles [22]. A Monte Carlo method named Monte Carlo Motion Planning was proposed, which fulfills probabilistic collision avoidance constraints [42]. Bopardikar et al. proposed a planner that searches for near optimal solutions of a multiobjective optimization problem which trade-off path length and safety in form of state estimation error covariance [16]. Miura et al. proposed a method to predict and model the motion behaviour of dynamic obstacles under path ambiguity, velocity uncertainty and observation uncertainty [85]. Thereafter, the best robot motion was selected. The heuristic arrival time field-biased random tree (HeAT-RT) using the arrival field method was proposed by Ardiyanto et al. to generate time optimal paths, where the robot moves slower near obstacles [5]. Kahn et al. introduced a reinforcement learning approach for collision avoidance [44]. The robot collided with obstacles at training time in order to learn to avoid obstacles. A convolutional neural network was introduced in [59], which predicted the trajectory of humans and estimated their risk of collision based on the awareness of the human. An analytical method for probability estimates for safe motion planning under Gaussian motion and sensing uncertainty was proposed in [95]. A framework and classification of motion planning uncertainties into categories can be found in [69].

1.4 Outline and summary of contributions

1.4.1 Outline

The thesis is separated into two parts, namely “Coverage Path Planning” and “Parking Control and Probabilistic Path Planning”. The first part is devoted to CPP algorithms, where optimal offline CPP algorithm is proposed. A brief summary of the chapter is denoted in the following

- Chapter 2 introduces an offline CPP algorithm. The algorithm uses the GA together with online CPP algorithms to search the solution space for optimal solutions under different objectives. A novel energy cost function is proposed, where the costs for the discrete motions per cell in the grid are experimentally acquired. The experimentally obtained costs for the discrete motion primitives are confirmed in an experimental setup in which the robot moved in a preassigned trajectory.

The second part of the thesis is devoted to parking control and probabilistic path planning. Chapter 3 proposes a parking control algorithm, which uses input/output-linearization (I/O-linearization) for successful parking. Chapter 4 introduces a novel family of probabilistic safe path planners based on the HA* algorithm. A brief summary of the chapters is given below

- Chapter 3 introduces a novel parking controller which uses I/O-linearization for a successful parking strategy. Inspired by human driving behaviour is the control strategy separated into three phases. The robot first orientates itself horizontal to the parking area, then it moves into a pose, where the mobile only has to drive in reverse for successful backwards parking.
- Chapter 4 introduces novel probabilistic safe path planners. Two of the planners are using chance constraints to guarantee a safe path to the goal pose. The two planners differ in their way of calculating the chance constraint and the result is a tradeoff between conservatism and computation between the two planners. The third planner uses a soft constraint instead of chance constraints, which results in an algorithm that trade-off the travelled distance and the safety of the path based on a tuning parameter.

Figure 1.7 shows the structure of the thesis.

1.4.2 Contribution

This thesis presents several contributions to solve problems relevant to path planning and coverage path planning of nonholonomic mobile robots. The contributions are as follows:

- The focus of optimal coverage path planning strategies was in literature the reduction of repetitive visits of the environment. This research proposes an algorithm which finds optimal solutions in terms of repetitive visits, travelling time or energy consumption. This was accomplished by designing a cost function which takes the sum of costs of all cells, where each possible motion primitive for a cell has a specific cost. Costs for

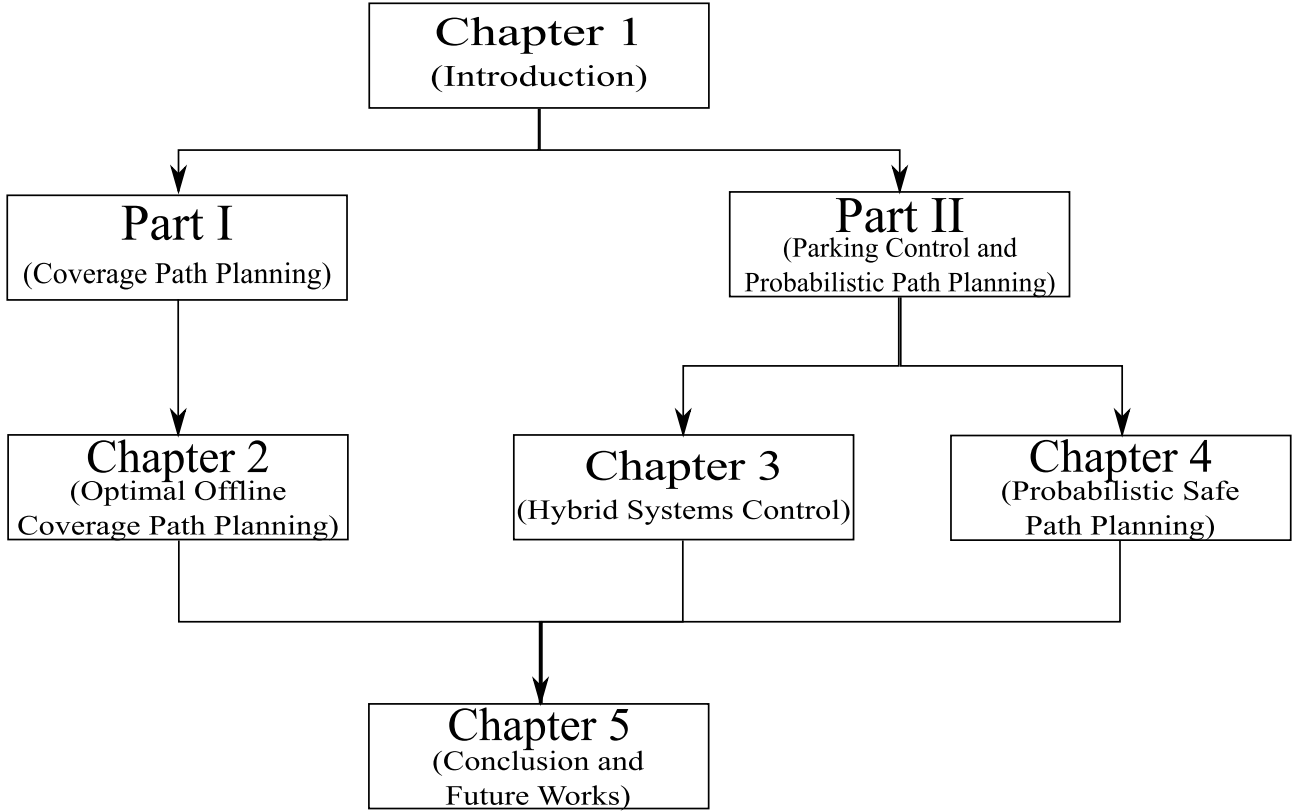


Figure 1.7: Outline of the thesis.

the energy consumption were experimentally obtained and experiments have shown that the costs predicted by the cost function match the energy consumption of the robot. The algorithm improves costs of existing coverage path planning methods. Furthermore, results show that optimizing repetitive visits is neither time optimal nor energy optimal.

- The parking controller uses a hybrid systems approach to garage parking of a differential drive mobile robot. The proposed system consists of three system states, in which each system state converges to its desired control value and then switches to the next state. Two system states use I/O-linearization for MIMO systems to conduct exact linearization of the plant, which can then be controlled by a linear controller. Simulation and experimental results show that the robot converges to its desired states and safely parks at the final position in a reasonable time.
- Most work in the literature on probabilistic path planning is dedicated to sampling based path planners. This thesis proposes novel HA* algorithms which consider uncertainty in the motion of the mobile robot, position uncertainty of static obstacles and position and velocity uncertainty of dynamic obstacles. A variant of the HA* algorithm is proposed which uses a soft constraint in the cost function instead of using chance constraints for the probability guarantees, which offers a trade-off between travelling distance and safety

of the path without pruning any additional nodes. Furthermore, this paper introduces a method to consider the shape of the mobile robot for probabilistic safe path planning. Performance is evaluated using the Monte Carlo simulation (MCS). Results show that safety can be improved without much increase of the travelling distance. Dynamic obstacles are safely avoided, while the conventional HA* algorithm has a high probability of collision. Furthermore, considering the shape of the robot for the probabilistic approach leads to safer paths overall. In addition, a probabilistic robust planner that uses a deterministic collision detection method for probabilistic robust planning is proposed. The resulting planner generates a confidence ellipse of the current state and covers the resulting ellipse with two circles of equal radius. The radius of the circles is used to inflate the obstacles and collision with obstacles can be detected with deterministic approaches. Results have shown that the planner is able to find probabilistic robust paths with a smaller computation time than existing planners. Lastly, a planner is proposed that receives measurement feedback from the environment and selects the linear velocity at each node according to the current probability of collision. The planner generates realistic probabilistic robust paths with adaptive velocities and the paths successfully avoid static and dynamic obstacles. The resulting paths have a shorter travelling time than existing planners.

2 Optimal Offline Coverage Path Planning

2.1 Problem formulation and environmental model

2.1.1 Problem formulation

Let \mathcal{C}^{CPP} be the grid environment for the CPP problem, where $\mathcal{C}^{\text{CPP}} \in \mathbb{R}^2$. Furthermore, \mathcal{C}^{CPP} is discretized into occupied and unoccupied cells, denoted by $\mathcal{C}_{\text{obs}}^{\text{CPP}}$ and $\mathcal{C}_{\text{free}}^{\text{CPP}}$, respectively. It is assumed that each cell in $\mathcal{C}_{\text{free}}^{\text{CPP}}$ is connected to at least another cell in $\mathcal{C}_{\text{free}}^{\text{CPP}}$. The cell of the robot is denoted by c , where $c \in \mathcal{C}^{\text{CPP}}$. The set $\mathcal{C}_{\text{free}}^{\text{CPP}}$ is further divided into unvisited cells in the set $\mathcal{C}_{\text{unvisited}}^{\text{CPP}}$ and visited cells in the set $\mathcal{C}_{\text{visited}}^{\text{CPP}}$. A cell is in $\mathcal{C}_{\text{visited}}^{\text{CPP}}$ if the robot visited the corresponding cell; otherwise, the cell is in $\mathcal{C}_{\text{unvisited}}^{\text{CPP}}$. The coverage of the environment is completed, when $\mathcal{C}_{\text{unvisited}}^{\text{CPP}} = \emptyset$, where \emptyset denotes an empty set, and the robot returns to the starting cell c_{start} . The permutation of the visited cells c_i is denoted by $C = (c_1 = c_{\text{start}}, c_2, \dots, c_i, \dots, c_{n_Q} = c_{\text{start}})$, where n_Q denotes the number of visited cells. The motion in each c_i is defined as m_i , and its permutation $M = (m_1, m_2, \dots, m_i, \dots, m_{n_Q})$. The optimal path is obtained by minimizing the fitness objective J of the resulting motions M . Therefore, the optimization problem is described as follows

$$\begin{aligned} & \text{minimize} && J(M) \\ & \text{subject to} && c_i \text{ in } C \in \mathcal{C}_{\text{free}}^{\text{CPP}}, \\ & && \mathcal{C}_{\text{unvisited}}^{\text{CPP}} = \emptyset. \end{aligned}$$

2.1.2 Environmental model

Each cell of the grid can take three states: obstacles (in $\mathcal{C}_{\text{obs}}^{\text{CPP}}$), previously visited (in $\mathcal{C}_{\text{visited}}^{\text{CPP}}$), or unvisited (in $\mathcal{C}_{\text{unvisited}}^{\text{CPP}}$). The size of a cell is equal to the length of the working tool of the mobile robot. Figure 2.1 shows a simple environment that is decomposed as a grid environment. The dark grey cells represent the occupied cells, the cell marked with an “S” represents the starting cell q_{start} , and cells with a line represent the visited cells. The dashed path sections denote repeated visits, solid lines indicate only one-time visits, and the dotted line represent the path from the last cell to the starting position after finishing the CPP. Therefore, the starting position is the same as the final position.

Each motion in M is recorded using numbers [47] from 0 to 3, where 0 represents the upwards motion in the grid, 1 is the left-ward motion, 2 denotes the downward motion, and 3 is the rightward motion, thereby enabling a simple representation of turn and U-turn motions. A change from the current motion number to an increasing/decreasing motion number implies a

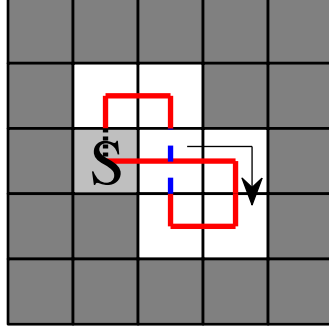


Figure 2.1: Path representation of the mobile robot (solid: one time visit; dashed: repeated visit; dotted: return path to the starting position).

turn motion. Moreover, an increment or decrement of two motion numbers represents a U-turn.

2.2 Local search and initialization algorithms

2.2.1 Turn-away starting point algorithm

TASP is a new approach to the online CPP proposed in [84]. It assumes that the robot can measure only the region in its vicinity using laser range finders, and records backtracking points (BPs) that are free and unvisited cells in the vicinity of the robot while moving in the environment. Its moving pattern was based on an experimental validation of the energy consumption, where it was shown that straight motions consume significantly less energy than turn motions. Therefore, this algorithm attempts to move straight as long as possible while covering the area. The straight motion is intuitively achieved by moving straight and away from the starting point continuously. When the robot reaches an obstacle, it has the following three choices depending on the number of free cells next to it:

- (I) If there are no free cells next to the robot, it moves to the closest BP, which is calculated by the number of cells along the X and Y directions between the current position and BP, with a PTP motion. Here, A^* [111] is used for PTP planning.
- (II) If there is only one free cell, the robot turns to that free cell and continues moving straight.
- (III) If two cells are free, the robot turns to the cell that points away from the starting position. However, if both the cells point away, it selects the one that provides a longer straight motion.

The distance for the selection is calculated by counting the cells from the position of the robot to the nearest obstacle. This distance includes previously visited cells if the number of those consecutive visited cells in the straight motion is less than or equal to a predefined threshold, which indicates the number of previously visited cells that can be crossed by the robot in straight motion. Algorithm 2 shows the pseudo code of TASP, and an example of the motion

Algorithm 2 Turn-away starting point algorithm

```

1: procedure TASP
2:   while backtracking points exist ( $\mathcal{C}_{\text{unvisited}}^{\text{CPP}} \neq \emptyset$ ), do
3:     get free neighboring cells from  $\mathcal{C}_{\text{unvisited}}^{\text{CPP}}$ 
4:     if two free cells, then
5:       if one shows away from starting point, then
6:         turn and move to the cell
7:       else
8:         estimate distance
9:         turn and move to the cell with longer distance
10:      end if
11:    else if one free cell then
12:      turn and move to the cell
13:    else
14:      point-to-point motion
15:    end if
16:    move forward until reaching obstacle
17:  end while
18: end procedure

```

behavior of the robot is shown in Figure 2.2.

2.2.2 Backtracking spiral algorithm

This section explains the online CPP algorithm called BSA [41]. The BSA uses a grid-based model to cover the environment. It has the following two main concepts: it covers simple regions using a spiral-like path and uses a backtracking mechanism to link the regions. A model of the environment is generated while the robot moves. Initially, all the cells in the environment are marked as *unknown*. A covered cell is denoted as *visited*, and cells that contain an obstacle, even if they are partially covered, are marked as *obstacle*.

One side of the robot must be next to an obstacle to start the BSA. The aforementioned side is named the reference lateral side (RLS), which indicates the relative direction where obstacles are to be referenced during the spiral procedure. The opposite side of the RLS is called the opposite lateral side (OLS). The BPs are updated whenever the robot moves to a new cell. The unvisited neighboring cells are added to the list, and the currently visited cell is deleted from the list. When the robot reaches a central ending point of a spiral path, it moves to the closest unvisited cell in a PTP motion and then continues with a new spiral procedure. The BPs are the potential next cells for the PTP motion. Every BP is recorded along with the distance by counting the number of cells along the x and y directions from the current position of the robot. Algorithm 3 shows a pseudo code of the BSA, and an example path of the motion behavior of the robot is shown in Figure 2.3.

Algorithm 3 Backtracking spiral algorithm

```

1: procedure BSA
2:   while backtracking points exist ( $\mathcal{C}_{\text{unvisited}}^{\text{CPP}} \neq \emptyset$ ), do
3:     if no free cell around robot, then
4:       search new starting point
5:       point-to-point motion
6:     else if no obstacle in RLS, then
7:       turn to RLS
8:     else if obstacle in front, then
9:       turn to OLS
10:    else
11:      move forward
12:    end if
13:  end while
14: end procedure

```

Table 2.1: Representation of the chromosome for the motion depicted in Figure 2.1.

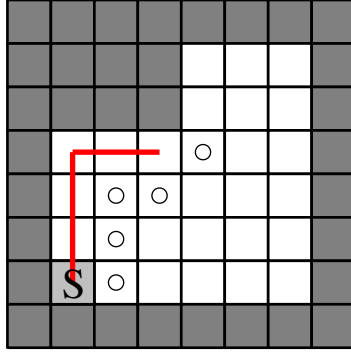
Parent	Genes
Path	3 3 2 1 0 0 1 2 2
Chromosome	3 3 2 1 0 1 2

2.3.2 Initialization

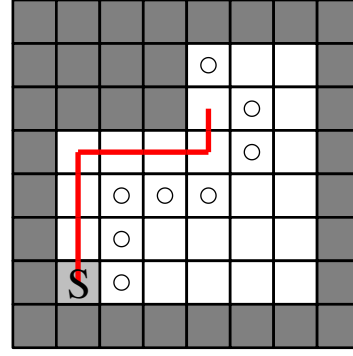
The initial paths correspond to the first parents of the HGA. Hence, it is essential to have a wide variety of initial paths. Additionally, the initial paths should already have good fitness values. Therefore, this approach uses five different procedures to generate the initial chromosomes. First, the online algorithms, i.e. BSA and TASP, are implemented as initial solutions, where both the algorithms have complete knowledge of the environment. Furthermore, two different approaches of TASP are used: one with a threshold value of 1, which allows the algorithm to cross only one previously visited cell, and the other with a threshold value of 2 which allows it to cross two previously visited cells. In addition, two methods with simple but effective motion behaviors, i.e. an exact spiral path and a zigzag path, are implemented. The example paths for all initialization methods are shown in Figures 2.4 and 2.5.

2.3.3 Fitness function

The robot moves from cell to cell with predefined motion behaviors, namely “straight”, “left turn”, “right turn”, and “U-turn”. Therefore, an energy-based fitness function can be designed by decomposing the four above mentioned motions into trajectory phases. Each of the turn motions has an acceleration phase and a deceleration phase. Furthermore, it is assumed that the turn phases of the left and right turns consume the same energy and require the same time.

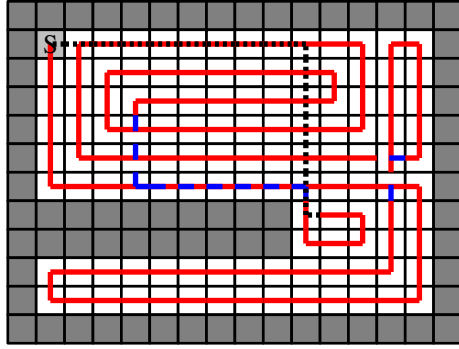


(a) Robot moves along the obstacle.

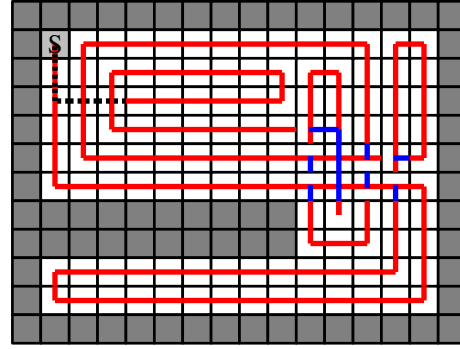


(b) Robot turns to RLS and continues motion along the obstacles.

Figure 2.3: Motion behavior of the robot and recorded backtracking points (circles) of the BSA algorithm.



(a) TASP (cross one visited cell).



(b) TASP (cross two visited cells).

Figure 2.4: Initial TASP paths for the proposed algorithm.

Hence, the trajectory phases are acceleration, deceleration, constant-velocity straight motion, constant-velocity turn motion, and constant-velocity U-turn motion. Therefore, the fitness for each motion can be calculated as follows:

$$\begin{aligned}
 \text{straight} &: 2J_S \\
 \text{turn} &: J_{\text{dec}} + J_T + J_{\text{acc}} \\
 \text{U-turn} &: J_{\text{dec}} + J_{\text{UT}} + J_{\text{acc}} \\
 \text{first cell} &: J_{\text{acc}} + J_S \\
 \text{last cell} &: J_{\text{dec}} + J_S,
 \end{aligned} \tag{2.1}$$

where the “turn” is either a left turn or right turn, “first cell” is the starting cell of the coverage path, and “last cell” is the last cell of the path. in addition, J_{acc} , J_{dec} , J_S , J_U and J_{UT} denote the fitness for the above mentioned trajectory phases, respectively. Therefore, the fitness function

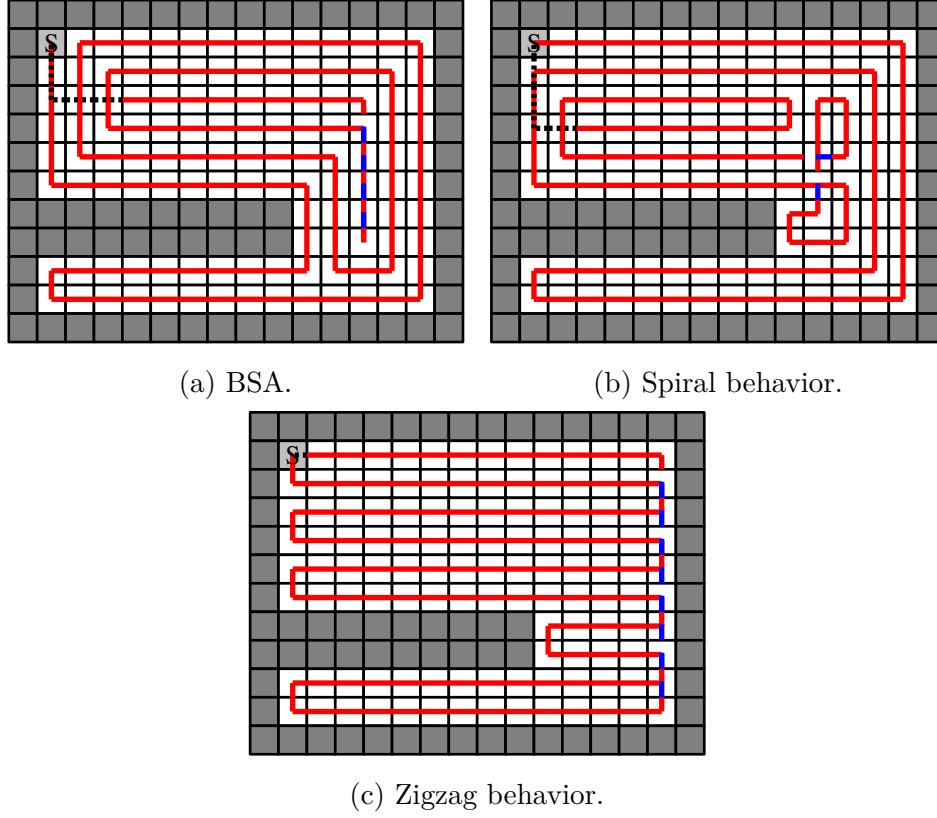


Figure 2.5: Initial paths for the proposed algorithm.

is given as follows:

$$J = \sum_{i=1}^{n_{\text{acc}}} J_{\text{acc}} + \sum_{i=1}^{n_{\text{dec}}} J_{\text{dec}} + \sum_{i=1}^{n_{\text{S}}} J_{\text{S}} + \sum_{i=1}^{n_{\text{T}}} J_{\text{T}} + \sum_{i=1}^{n_{\text{UT}}} J_{\text{UT}}. \quad (2.2)$$

The number of trajectory phases taken is saved in $n_{\{\text{acc,dec,S,T,UT}\}}$.

The energy cost for each phase is obtained experimentally using predefined velocity trajectories. The used current and voltage of the motors are saved while executing one of the trajectories. The obtained power is given as follows:

$$P_i = U_i I_i, \quad (2.3)$$

where P_i, U_i , and I_i denote the power, voltage, and current for the i -th sampling instant, respectively. The obtained power is integrated using the trapezoidal rule

$$E = \delta t \sum_{i=1}^{n_p-1} \frac{P_i + P_{i+1}}{2} \quad (2.4)$$

to obtain the consumed energy. Here E , n_p , and δt are the consumed energy, number of measurement points, and sampling time, respectively. The respective energy for each trajectory phase fitness J_{acc} , J_{dec} , J_S , J_T , and J_{UT} from (2.2) is obtained by calculating the power P of each measurement using (2.3) and then the energy E using (2.4). As the time of completion, T , of each trajectory phase is different from one another, each phase has a different number of measurement points, n .

Conventional approaches use the number of visited cells (e.g., [78]) to evaluate the performance of the coverage path, whereas this fitness function uses the consumed energy of the robot directly as the cost. Therefore, it can find optimal solutions, which may enable the robot to intentionally visit a cell more than once.

Notably, the fitness values must be measured whenever the surface of either the environment or robot changes. Depending on the surface roughness, the friction between the robot and surface changes, thereby changing the consumed energy. Furthermore, this function assumes a horizontal planar environment. Uneven areas require different energy consumption by the robot, which also depends on its moving direction. However, as the fitness values are experimentally obtained, they can be customized for respective situations. This is one of the advantages of the proposed fitness function over others.

In addition, the traveling time of the entire path of the robot, and the number of visited cells can be used as fitness functions to evaluate the performance of the HGAs.

2.3.4 Genetic operators

In nature, survival of the fittest, reproduction, and mutation are the operators to generate a wide variety of offspring for evolution. In GA, selection, crossover, and mutation are the analogous copies of these natural operators, respectively. The following operators are employed in the proposed GA approach.

2.3.4.1 Selection

Selection is the process of selecting parents for creating the next generation via crossover and mutation. The choice of the parents decides the convergence rate of the entire algorithm as selecting only the best chromosomes in every iteration might result in a local optimum. Hence, it is necessary to choose parents in such a way that the entire solution space is covered.

To provide a considerable diversity in the selection, this approach uses the tournament selection (TS), which randomly selects, among all the chromosomes in the population, k candidates. The candidate with the best fitness is chosen as a parent for the next generation. The process of selecting k candidates and choosing the best one is repeated until all the parents are selected. This algorithm provides a certain opportunity for weaker chromosomes to be chosen, as the set of the k candidates always changes at random.

An elitism selection is used in addition to the TS. It selects chromosomes with good fitness values of the current parent set. Therefore, the new population comprises chromosomes selected via both elitism and TS.

Table 2.2: Example of mutation operator.

Chromosome type	Genes
Parent	0 0 0 3 3 2 2 2
Offspring	0 0 0 3 <u>2</u> 2 2 2

Table 2.3: Example of crossover operator.

Parent one	0 0 0 3 2 2 2 2
Parent two	<u>2</u> <u>2</u> <u>2</u> <u>1</u> 0 0 0 <u>3</u>
Two point	0 0 0 <u>1</u> <u>0</u> 2 2 2

2.3.4.2 Mutation

Mutation is a small random change in the chromosome to obtain a new solution. The mutation operator is used to maintain diversity in the population, and is important for exploring the search space. A probability factor p_m decides whether a mutation can be applied in the current chromosome. This approach randomly selects one gene of the chromosome and changes it to a new randomly chosen gene. All motions, except the U-turn motion, are considered for mutation, due to the fact that U-turn motions will result in a previously visited cell. Notably, U-turns are used only in PTP motions. Table 2.2 illustrates an example, where the underlined gene is the mutated one.

2.3.4.3 Crossover

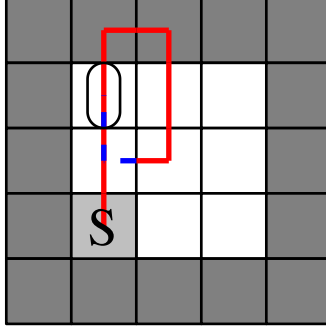
The crossover operator is analogous to reproduction in nature, and it is also known as recombination [1]. A pair of parents is selected, which provide, along with their genes, a new offspring for the next generation. In this study, the two-point crossover is used, as listed in Table 2.3, where the offspring inherits genes from one parent until a certain locus point, following which it inherits the genes from the second parent starting at that locus point. It switches again at the second locus point. The two locus points are randomly chosen for each new offspring. A probability factor p_c decides whether a crossover can be applied.

2.3.5 Termination

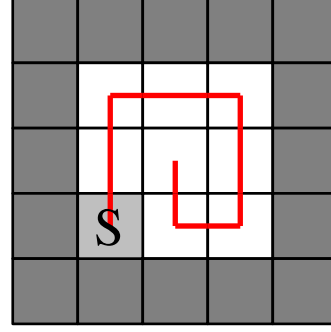
The termination condition of the GA or HGA is based on a predefined fixed number of generations. Upon reaching the last generation, the fitness function value is compared with those in the previous iterations. If no improvement occurs, the algorithm terminates. Otherwise, the number of generations are extended to a chosen number, followed by the continuation of the process.

Table 2.4: Adjustment of a gene from an infeasible offspring.

Infeasible offspring	0	0	0	3	2	2	1	0	0
Adjusted offspring	0	0	3	3	2	2	1	0	0



(a) Infeasible offspring (circled section runs into a wall).



(b) Adjusted feasible offspring (only one motion is adjusted).

Figure 2.6: Adjustment procedure in the GA.

2.3.6 Point-to-point motion

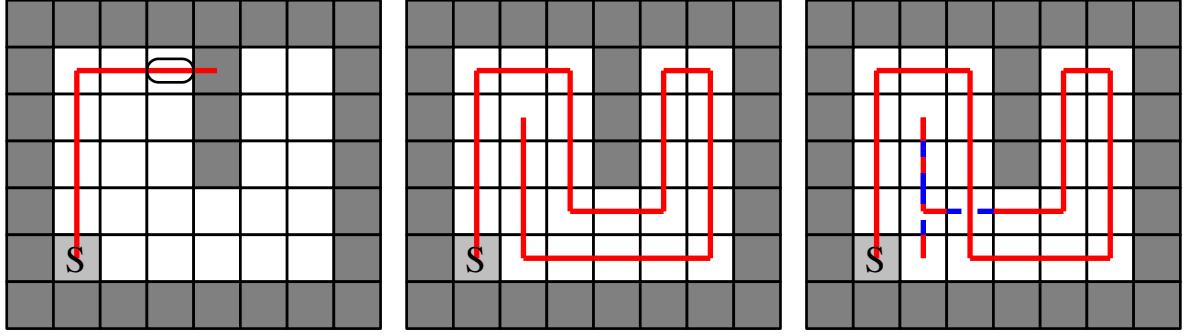
If the robot runs into a cell that is surrounded by only visited cells, it moves to the nearest unvisited cell with a PTP motion. This approach determines the nearest cells by counting the intermediate cells, both horizontally and vertically, to reach the unvisited cell. This method provides an approximate distance to the unvisited cells and ignores obstacles. Therefore, the unvisited cell with the smallest distance is calculated using the A* algorithm [111]. The path with the smallest distance is employed for the PTP motion of the robot.

2.3.7 Feasibility

There is no guarantee that the offspring would be feasible after applying mutation and crossover. Here, “feasible” means that the path of the offspring is not in $\mathcal{C}_{\text{obs}}^{\text{CPP}}$ and only in $\mathcal{C}_{\text{visited}}^{\text{CPP}}$ whenever no neighboring cell of the robots’ cell, c_i , is in $\mathcal{C}_{\text{unvisited}}^{\text{CPP}}$. Therefore, the feasibility of each offspring must be checked.

For the GA, this is accomplished by iteratively checking the feasibility of each gene of the offspring. If a gene is not feasible, it is changed to a feasible motion. First, the straight motion is tested; if it is also not feasible, the right or left turn motion is randomly selected. Subsequently, the remaining turn motion from the aforementioned random selection is evaluated. The U-turn motion is not tested, as it always runs into a previously visited cell and is only used in the PTP motion. If all the motions are not feasible, the PTP motion is applied, and, the gene is changed accordingly. Table 2.4, which corresponds to Figure 2.6, illustrates this gene adjustment, and the bold font emphasizes that the gene is adjusted.

The HGA approach differs from the GA approach, as the former uses TASP and BSA as local



(a) Infeasible offspring (circled section runs into a wall). (b) Locally improved feasible offspring using BSA. (c) Locally improved feasible offspring using TASP.

Figure 2.7: Local improvement procedures in the HGA approach.

improvement procedures instead of only adjusting the offspring. In addition, the genes of each offspring are evaluated for feasibility. The first gene, which is not feasible, is used as the starting position for TASP and BSA. The subsequent genes of the original offspring are deleted. Both TASP and BSA finish the coverage path for the offspring. These local procedures result in candidate solutions, which cannot be obtained using only the GA. The main reason is that both BSA and TASP can continue to generate feasible paths even upon reaching an obstacle or a previously visited cell. The example paths for both BSA and TASP are shown in Figure 2.7.

Three different variants of the CPP algorithm are proposed:

1. HGA/BSA
2. HGA/TASP
3. HGA/Both

Figure 2.8 shows the complete GA and HGA approaches, and it is evident that the difference between them lies in the update of the offspring.

2.4 Results and discussion

The differential-drive mobile robot shown in Figure 1.5 is used for obtaining the experimental values of the fitness function. The robot is used for cleaning industrial areas or other floor environments. Its differential wheels are located in the rear side of the robot, the cleaning tool is in the front side, and a caster wheel is in the center. The robot uses an inertial measurement unit sensor for detecting its orientation and location. Table 2.5 lists the chosen parameters after some tuning using a trial-and-error method for the HGAs.

The following subsections present the experimentally obtained energy results and validity of the energy-based fitness function, followed by a comparison of the HGAs with other CPP algorithms.

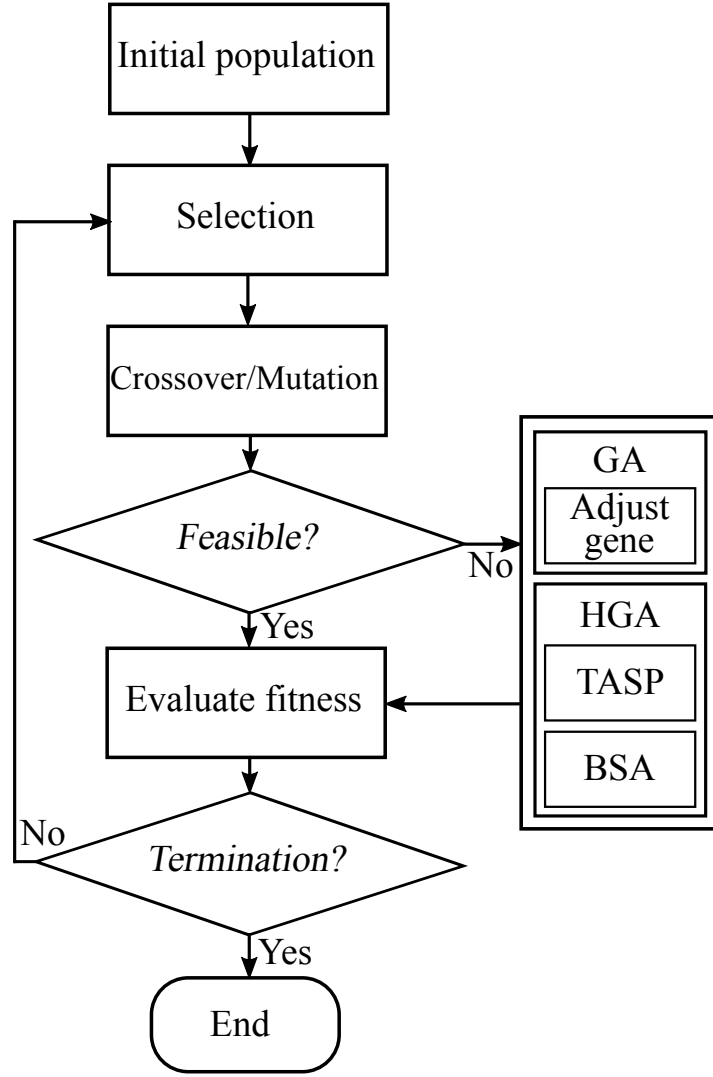


Figure 2.8: Flow chart of the GA and HGA approaches.

2.4.1 Fitness cost identification

Three experimental setups with different trajectories are considered to obtain the energy values for all the trajectory phases, where the maximum velocity and acceleration and deceleration times are predefined based on the robot specifications. Direct current motors are used to drive the left and right wheels, and subsequently, current and voltage are measured.

The experiments are conducted on a carpet-like surface to obtain the energy values for each trajectory phase of the fitness function. The energy for acceleration, deceleration and straight motion are received by accelerating, moving straight and then decelerating. The upper plot in Figure 2.9 shows the desired angular velocity for the left and right drive wheels, whereas the lower plot depicts the required current and voltage to perform the motion. $\dot{\phi}_{d_r}, \dot{\phi}_{d_l}$ denote the desired angular velocities for the right and left drive wheel, and U_r, I_r, U_l and I_l denote

Table 2.5: Parameters for HGAs.

Parameter	Values
Mutation probability p_m	0.1
Crossover probability p_c	0.6
Crossover method	Two-point
Selection method	TS + elitism
Number of generations	10
Population size	200

Table 2.6: Obtained time and energy costs for motions on carpet-like and smooth surfaces, respectively.

Motion	Energy [J]		Completion time $T[s]$
	Carpet-like surface	Smooth surface	
J_S	7.83	8.53	1.02
J_{acc}	17.09	14.95	2.10
J_{dec}	5.78	4.91	2.10
J_T	38.57	12.31	8.28
J_{UT}	51.22	19.16	10.56

the applied voltage and current for the right and left drive wheel, respectively. The vertical lines distinguish the acceleration section, straight motion section, and deceleration section for the calculation. Notably, the straight motion section comprises of multiple straight motions to neglect the effect of residual energies from the acceleration and deceleration. The total energy of the straight motion section is calculated and then divided by the number of straight motions to obtain the average energy value. Figures 2.10 and 2.11 show the desired angular velocity and used current and voltage for the turn and U-turn motions, respectively. Each experiment condition is performed five times, and the average of the obtained motions is calculated, such that the energy results are less affected by noise. Table 2.6 lists the energy and time values for the carpet-like surface and smooth surface, respectively, for each motion. The time value T denotes the required time for each motion. The energy for each motion is obtained using (2.3) and (2.4).

The fitness function must be validated, as it is still not known whether it can be used generally for each path. Therefore, a spiral motion is conducted, as shown in Figure 2.12. The path is traversed five times, and the resulting consumed energy is compared with the calculated energy using the fitness function, as depicted in the graph on the right side in Figure 2.12. The calculated energy of the spiral path is in agreement with the five experimentally obtained energies. Table 2.7 lists the resulting average, minimum and maximum deviations of the spiral path. The average deviation is 1.77 %, which is sufficient for energy estimation, thereby validating

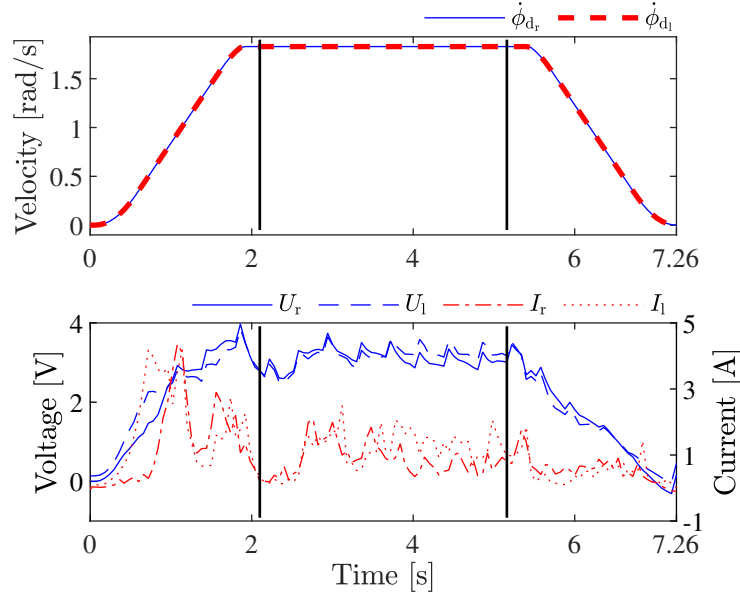


Figure 2.9: Desired angular velocity and the resulting current and voltage for acceleration, deceleration and straight motion (the black vertical lines divide acceleration, the straight motion and deceleration in the trajectory).

the fitness function.

2.4.2 Comparison of different approaches

In this section, comparison between the proposed HGAs with GA, TASP, BSA and distance transform (DT) wavefront algorithm [121] is presented. MATLAB[®] 2018a with an i7-2670QM CPU 2.20 GHz computer is used for the computation. The comparison is performed using three different-sized environments, with 122, 489, and 1348 unoccupied cells, as depicted in Figures 2.13–2.15, respectively. Furthermore, the comparison is performed for the energy-based fitness functions for smooth and carpet-like surface, traveling time, and number of visited cells. The results for a starting position are presented in Table 2.8, where the bold values indicate the best values for the respective fitness function objective, and “x” means that the algorithm is unable to improve the solution from the initial candidate solutions. In addition, HGA/TASP finds the best solutions in all three environments using the energy-based fitness function on the smooth

Table 2.7: Comparison of energy values in experiments and fitness function.

Spiral path	
Average error [%]	1.77
Min error [%]	0.03
Max error [%]	3.17

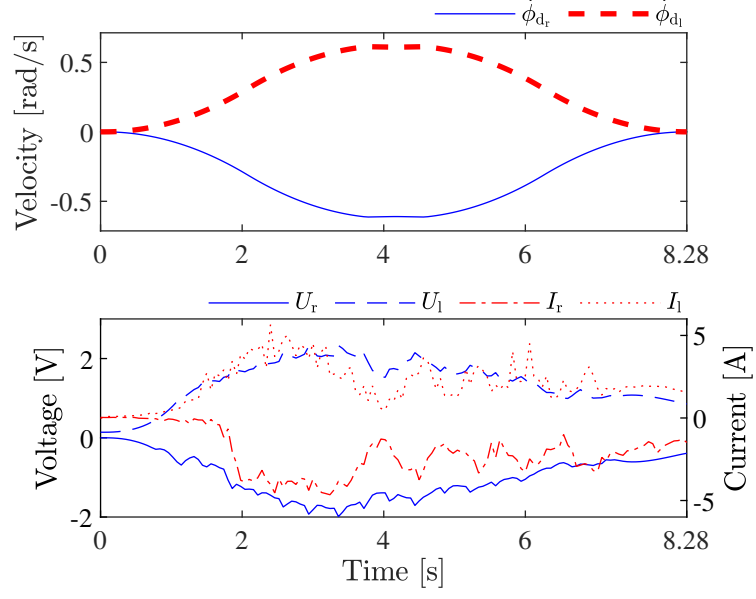


Figure 2.10: Desired angular velocity and the resulting current and voltage for turn motion.

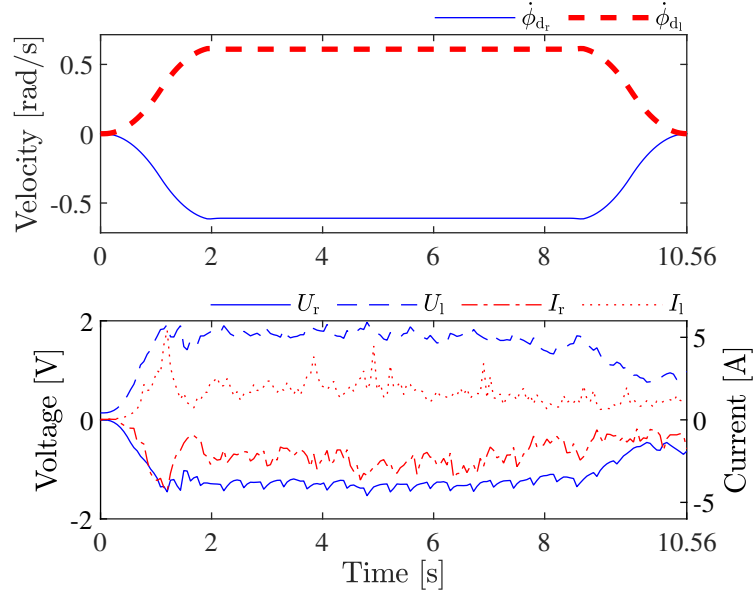
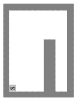




Figure 2.11: Desired angular velocity and the resulting current and voltage for U-turn motion.

Table 2.8: Comparison of HGAs with other approaches (grids are omitted in the maps for ensuring visibility).

Environment	Fitness function	HGA/BSA	HGA/TASP	HGA/Both	GA	BSA	TASP	DT
	Energy (smooth) [J]	2482.4	2452.6	2493.6	2456.5	2516.5	2743.6	2599.8
	Energy (carpet) [J]	2864.0	2864.0	x	x	2895.4	3295.3	3444.1
	Traveling time [s]	455.52	455.52	455.52	455.52	459.6	538.56	597.6
	Cell visits [-]	123	127	127	125	131	139	123
	Energy (smooth) [J]	11061	11001	11021	11251	11526	11626	12287
	Energy (carpet) [J]	x	12552	12552	x	17167	13638	17524
	Traveling time [s]	x	1945.0	1969.9	x	2322.7	2118.8	3157.6
	Cell visits [-]	543	571	571	x	549	583	545
	Energy (smooth) [J]	27843	27408	27869	x	28666	28326	29476
	Energy (carpet) [J]	31133	31205	31388	31405	33792	31819	38677
	Traveling time [s]	4849.0	4821.4	4849.0	4849.0	5494.2	4917.5	6666.4
	Cell visits [-]	1415	x	x	x	1443	1527	1403

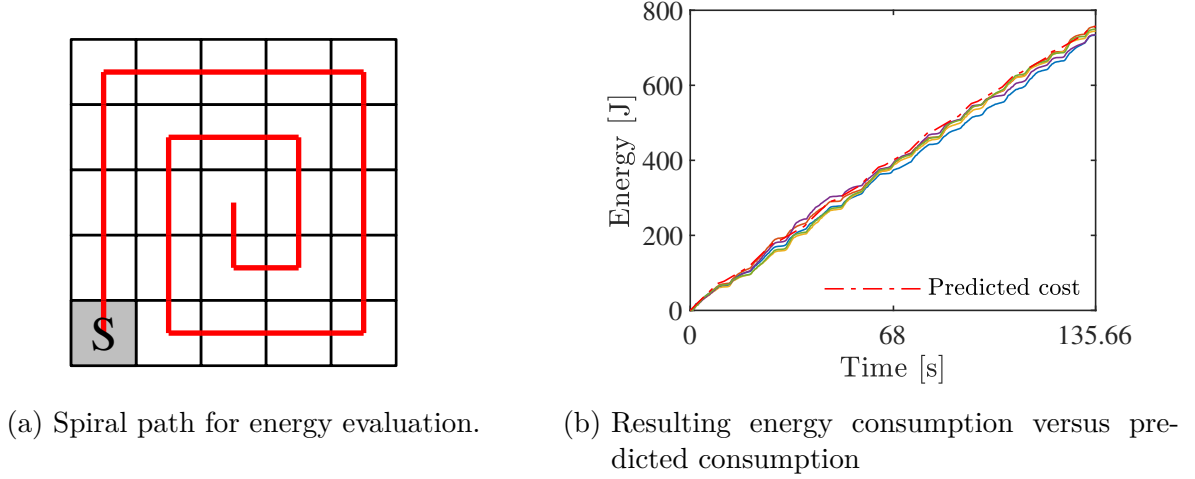


Figure 2.12: Experiments of spiral path for fitness function validation.

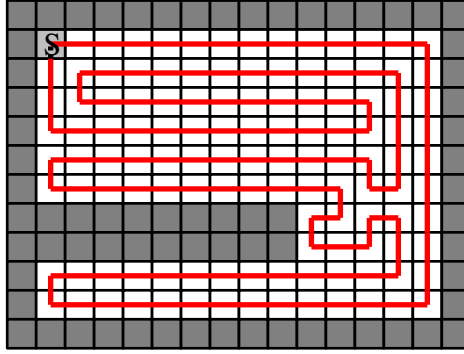


Figure 2.13: Resulting path using HGA/BSA with the number of cell visits as fitness function.

surface and traveling time. Its solution in the medium-sized environment with traveling time as fitness is $\sim 38\%$ faster than the DT solution. It also has the best solution for the small-sized and medium-sized environments on the carpet-like surface, where it consumes $\sim 13\%$ and $\sim 27\%$ less energy than the TASP and BSA solutions in the small-sized and medium-sized environment, respectively. Furthermore, HGA/TASP obtained the best solutions for travelling time and energy consumption on the smooth surface in the large environment. The travelling time is $\sim 28\%$ faster and the energy consumption is $\sim 7\%$ less compared to DT, respectively. The best solution in the large-sized environment with the energy-based fitness function on the carpet-like surface is obtained using the HGA/BSA, which consumes $\sim 20\%$ less energy than the DT solution. In addition, HGA/BSA obtains the best results on the number of cell visits in both the small-sized (global optimum) and medium-sized environments. Although the DT algorithm provides the best solution for the number of visits in the small-sized and large-sized environments, its solutions for the other fitness objectives are worse compared to the evolutionary algorithms. This shows that in general reducing the number of cell visits is not an appropriate objective for CPP algorithms, because a small number of visits may result

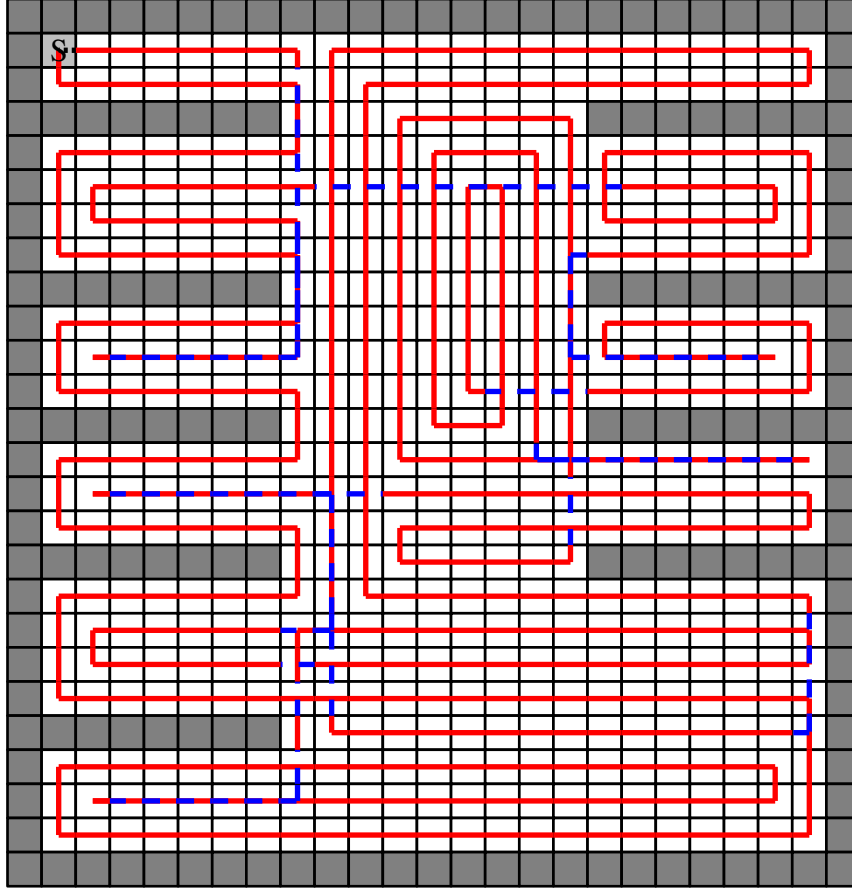


Figure 2.14: Resulting path using HGA/Both and HGA/TASP with the energy-based fitness function.

in higher traveling time or energy consumption. The resulting path of the HGA/BSA in the small-sized environment with the number of cell visits as fitness is shown in Figure 2.13. The path obtained using the HGA/TASP and HGA/Both in the medium-sized environment, and path obtained using the HGA/BSA in the large-sized environment with the energy-based fitness function on the carpet-like surface are shown in Figures 2.14 and 2.15, respectively.

The computation time for the evolutionary algorithms is presented in Table 2.9, where the HGA/TASP has the best computation time.

In addition, the algorithms are compared for the large-sized environment using 20 randomly chosen starting positions. Figures 2.16a, 2.16b, 2.16c, and 2.16d show the minimum, maximum, and average values of fitness based on the required energy-based fitness on a carpet-like surface, smooth surface, traveling time, and number of visited cells, respectively. It can be seen that the HGA/TASP and HGA/Both have the best solutions for the carpet-like fitness function (Figure 2.16a) and traveling time (Figure 2.16c), respectively. Furthermore, the HGA/BSA has the best average fitness for the smooth energy-based fitness function (Figure 2.16b) and best solution for the number of cell visits next to DT (Figure 2.16d). In addition, compared with DT, the

Table 2.9: Computation times of evolutionary algorithms for three environments.

	Small sized	Medium sized	Large sized
HGA/BSA	42.23 s	349.55 s	563.58 s
HGA/TASP	23.53 s	280.79 s	532.16 s
HGA/Both	104.33 s	622.86 s	1149.28 s
GA	32.56 s	893.62 s	1461.43 s

fitness for HGA/BSA is consistent and ranges from 1395 to 1423 visited cells, whereas the fitness for DT ranges from 1361 to 1489 visited cells. Moreover, the results for the carpet-like fitness function and traveling time (Figures 2.16a and 2.16c) are considerably consistent with those of the HGAs. However, the BSA fails for most of the randomly chosen starting points because it can start only next to a visited cell or an obstacle during path generation.

The above mentioned results show that the solutions of the HGA depend on the local search algorithms. The TASP algorithm attempts to move straight as long as possible; therefore, the HGA/TASP finds good solutions for both the energy-based fitness function and traveling time. However, the BSA moves in a spiral next to obstacles and visits the same cell again only if no other motion is possible. Therefore, the HGA/BSA finds the best solution for the number of cell visits. Although the HGA/Both uses both the TASP and BSA algorithms to perform local searches, it cannot produce better results than the HGA/TASP and HGA/BSA.

2.5 Summary

This chapter introduced a HGA algorithm to solve CPP problems for differential drive mobile robots. The algorithm uses different existing online CPP algorithms such as TASP and BSA as local search methods. Simulation results showed that the HGA algorithm obtained better results than existing works for the travelling time, energy consumption and cell visits as fitness. Furthermore, the HGA algorithm needed less computation time to solve the CPP problem compared to a GA algorithm that does not use local search methods. In addition, results have shown that the fitness of the HGA algorithm does not deviate much for different starting positions in the environment.

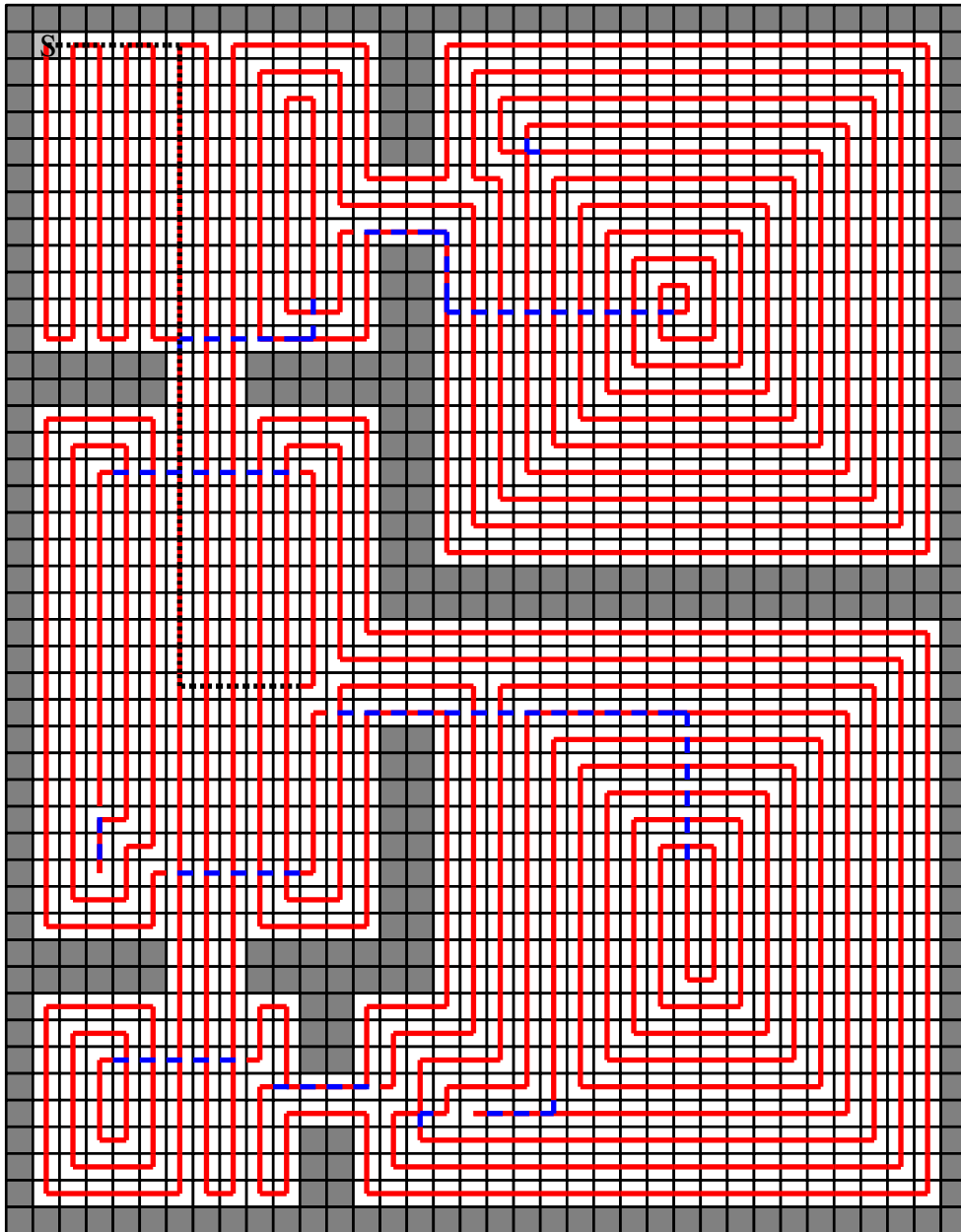


Figure 2.15: Resulting path using HGA/BSA with the energy-based fitness function on a carpet-like surface.

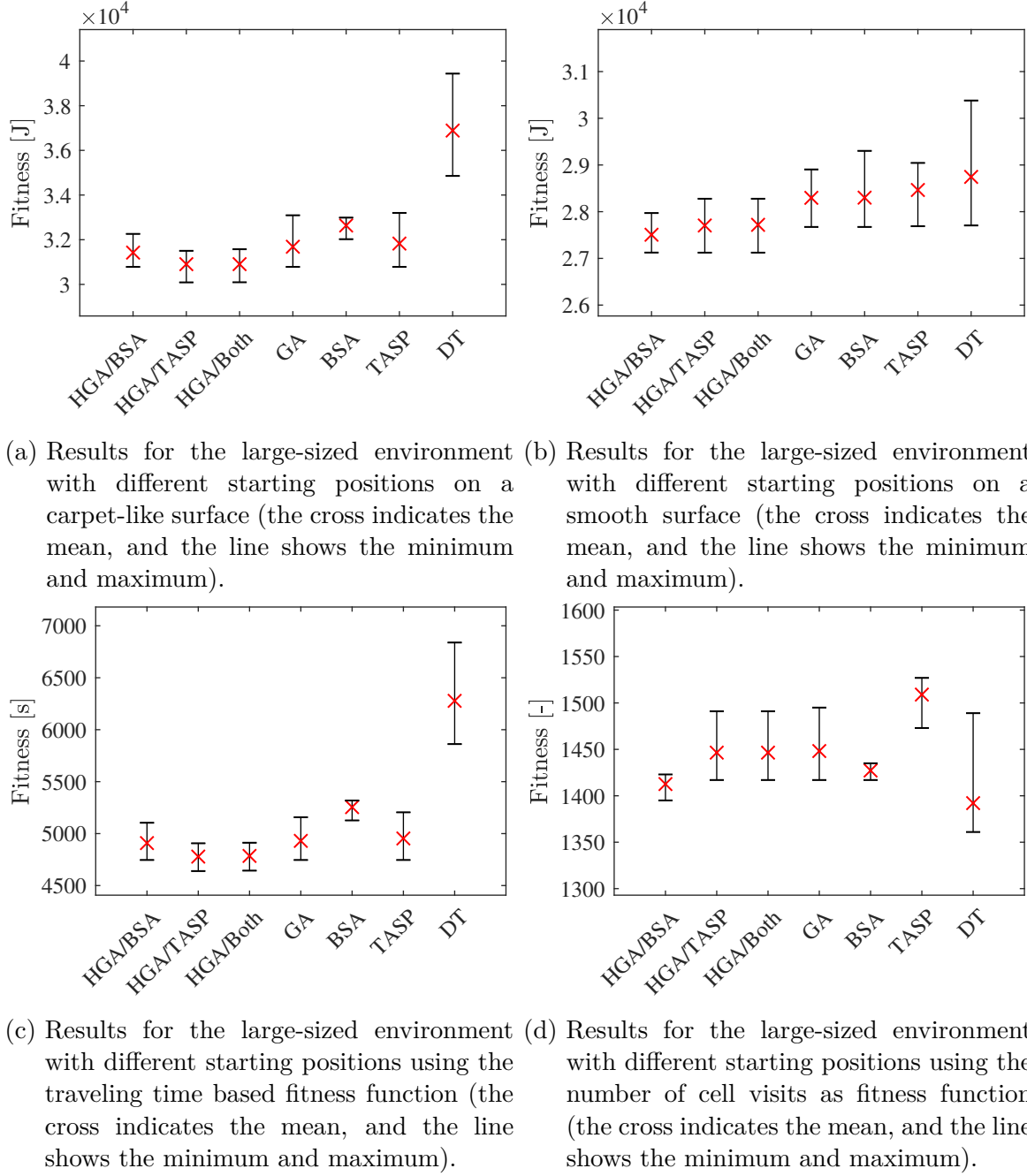


Figure 2.16: Performance results for different starting positions and objectives

3 Hybrid Systems Control

3.1 Introduction

This section proposes a hybrid systems approach consisting of three steps for garage parking of a differential-drive mobile robot. In the first system state, the orientation of the robot is controlled to a desired orientation required for the second system. In the second state, the robot position in the global x-axis and the orientation are controlled. An I/O-linearization for MIMO systems is used to design the linear controllers. Thereafter, the robot switches to the third state, in which the orientation is maintained and the position is controlled in the y-axis. This state also uses I/O-linearization for MIMO systems.

The next section presents a kinematic model of the robot, followed by the hybrid systems approach and the I/O-linearization for the parking procedure in section 3.3. Simulation and experimental results are shown in section 3.4. A conclusion and future works are given in chapter 5.

3.2 Kinematic model

In this section, a kinematic model for the nonholonomic differential-drive mobile robot is derived [111]. Figure 3.1 shows the robot with local and global reference frames, where the origin of the local reference frame is located in the center of the drive wheels axle. x and y represent the coordinates of the mobile robot in the global reference frame $\{X_G, Y_G\}$, and $\{X_R, Y_R\}$ depicts the local robot reference frame. $\dot{\phi}_l$ and $\dot{\phi}_r$ are the angular velocities of the left and right drive wheels, respectively. The radius of the drive wheels is denoted by r and the width of the robot is $2l$. The angular difference between local and global reference frame is given by θ . $\dot{\theta}$ is the angular velocity of the robot, which coincides with that of the local reference frame. The velocity v in X_R direction in Figure 3.1 can be derived as follows:

$$v = \frac{v_l + v_r}{2} \quad (3.1)$$

with

$$\begin{aligned} v_l &= r\dot{\phi}_l \\ v_r &= r\dot{\phi}_r. \end{aligned} \quad (3.2)$$

The drive wheels can not contribute to sideways motion in the local reference frame, hence the velocity in Y_R is always zero. From previous equations, the dynamics of the robot in the global

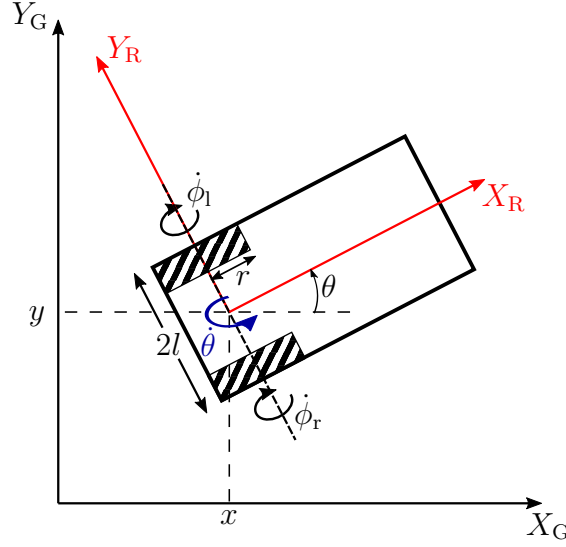


Figure 3.1: Differential-drive mobile robot and global and local reference frames.

reference frame with respect to $\dot{\phi}_l$ and $\dot{\phi}_r$ are as follows:

$$\begin{aligned} \dot{x} &= \cos \theta \left(\frac{r\dot{\phi}_r + r\dot{\phi}_l}{2} \right) \\ \dot{y} &= \sin \theta \left(\frac{r\dot{\phi}_r + r\dot{\phi}_l}{2} \right). \end{aligned} \quad (3.3)$$

Furthermore, the angular velocity of the robot is

$$\dot{\theta} = \frac{r}{2l}(\dot{\phi}_r - \dot{\phi}_l). \quad (3.4)$$

Hence, the kinematics model of the robot is

$$\begin{aligned} \dot{\mathbf{x}} &= \underbrace{\frac{r}{2} \begin{bmatrix} \cos \theta & \cos \theta \\ \sin \theta & \sin \theta \\ \frac{1}{l} & -\frac{1}{l} \end{bmatrix}}_{\mathbf{G}(\mathbf{x})} \mathbf{u} \\ \mathbf{y} &= \mathbf{h}(\mathbf{x}) \end{aligned} \quad (3.5)$$

with

$$\begin{aligned} \dot{\mathbf{x}} &= [\dot{x} \quad \dot{y} \quad \dot{\theta}]^\top, \\ \mathbf{u} &= [\dot{\phi}_r \quad \dot{\phi}_l]^\top \end{aligned}$$

and \mathbf{y} is the output of the system.

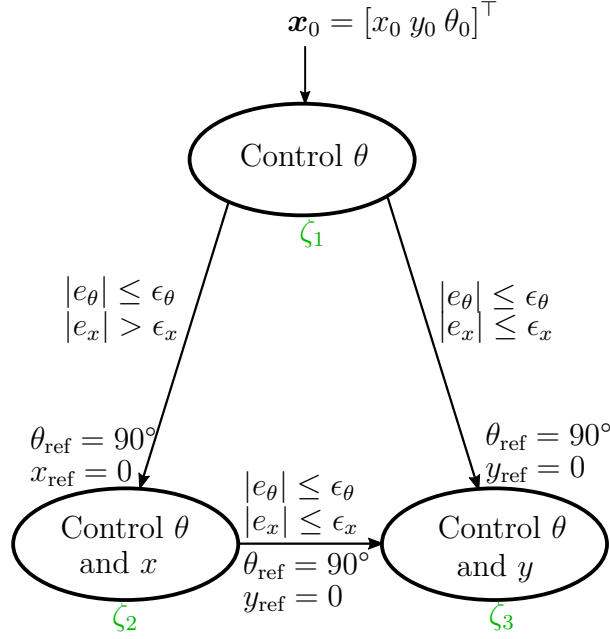


Figure 3.2: Hybrid system for parking control.

3.3 Hybrid systems approach

This section introduces the hybrid systems approach for the parking procedure, followed by the I/O-linearizations with the corresponding controller and stability analysis for each hybrid system.

Figure 3.2 shows the hybrid system for the parking procedure, where \mathbf{x}_0 denotes the initial state of the mobile robot. Each of the three subsystems $\zeta_i, i = 1, 2, 3$, is a state in the hybrid system. The first state ζ_1 controls θ to rotate the robot in the desired position for the next controller. The reference θ_{ref} depends therefore on the initial location x_0 of the robot in the x -axis. If it is in the left half plane and smaller than the threshold $-\epsilon_x$, then $\theta_{\text{ref}} = 0^\circ$, and if the robot is on the right half plane and greater than ϵ_x , then $\theta_{\text{ref}} = 180^\circ$, otherwise $\theta_{\text{ref}} = 90^\circ$. Thus, the reference for the first system is described as follows

$$\theta_{\text{ref}} = \begin{cases} 0^\circ, & \text{if } x_0 < -\epsilon_x \\ 180^\circ, & \text{if } x_0 > \epsilon_x \\ 90^\circ, & \text{otherwise.} \end{cases}$$

The hybrid system will switch to state ζ_2 if the absolute error e_θ is in between a certain tolerance ϵ_θ , and if the absolute error e_x is smaller than the tolerance ϵ_x , then the system switches to ζ_3 . State ζ_2 controls x and θ with the new reference $\theta_{\text{ref}} = 90^\circ$. If the absolute errors e_x and e_θ of the robot are in the tolerance range ϵ_x and ϵ_θ , respectively, then the hybrid system switches to ζ_3 . The third state controls y and θ , and the parking procedure is finished after the robot reaches its final position $y_{\text{ref}} = 0$, while maintaining θ at 90° and thus $x = 0$. The complete

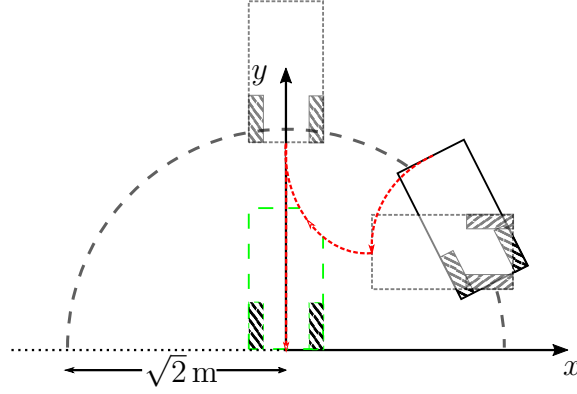


Figure 3.3: Parking control strategy.

procedure is shown in Figure 3.3, where the gray dashed arc shows the range of interest for the parking procedure, the red dotted lines are the trajectory of the robot, the gray dashed robots present the end posture after states ζ_1 and ζ_2 and the green dashed robot is the final posture of the robot in state ζ_3 . The system can only operate in the vicinity of the parking region due to the reason that the errors in the states ζ_2 and ζ_3 get higher the farther away the robot is from the parking position.

3.3.1 Input/output-linearization

The I/O-linearization [53, 112] is obtained by differentiating \mathbf{y} until all inputs \mathbf{u} appear in the equation, such that the Lie-derivative $L_{Gh}(\mathbf{x})$ of the robot model has to be nonzero. It can be seen that $L_{Gh}(\mathbf{x}) \neq 0$ if the states of the system are chosen as outputs \mathbf{y} . Hence, the I/O-linearization is realized after the first differentiation. Here, the I/O-linearization for state ζ_2 has outputs

$$\mathbf{y} = h(\mathbf{x}) = \begin{bmatrix} x \\ \theta \end{bmatrix}.$$

Consequently, the Lie-derivative

$$\begin{aligned} L_{Gh}(\mathbf{x}) &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \frac{r}{2} \begin{bmatrix} \cos \theta & \cos \theta \\ \sin \theta & \sin \theta \\ \frac{1}{l} & -\frac{1}{l} \end{bmatrix} \begin{bmatrix} \dot{\phi}_r \\ \dot{\phi}_l \end{bmatrix} \\ &= \frac{r}{2} \begin{bmatrix} \cos \theta & \cos \theta \\ \frac{1}{l} & -\frac{1}{l} \end{bmatrix} \begin{bmatrix} \dot{\phi}_r \\ \dot{\phi}_l \end{bmatrix} \end{aligned} \quad (3.6)$$

is obtained. It can be seen that the system is well defined in D_1 with

$$\left\{ D_1 \mid D_1 \subset \mathbb{R}, D_1 \neq \frac{(n+1)\pi}{2} \right\}, \quad n \in \mathbb{Z}.$$

Furthermore, the region of interest $\theta \in \Omega_1 \subset D_1$ is defined as

$$\left\{ \Omega_1 \mid 0 \leq \Omega_1 \leq \pi, \Omega_1 \neq \frac{\pi}{2} \right\}.$$

The input \mathbf{u} for the robot is obtained from (3.6) as follows:

$$\begin{bmatrix} \dot{\phi}_r \\ \dot{\phi}_l \end{bmatrix} = \begin{bmatrix} \frac{1}{r \cos \theta} & \frac{l}{r} \\ \frac{1}{r \cos \theta} & -\frac{l}{r} \end{bmatrix} \underbrace{\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}}_{\mathbf{w}}, \quad (3.7)$$

with w_1 and w_2 being the new pseudo inputs of the system.

The I/O-linearization for state ζ_3 uses

$$\mathbf{y} = \mathbf{h}(\mathbf{x}) = \begin{bmatrix} y \\ \theta \end{bmatrix}$$

as outputs. Thus, its Lie-derivative is as follows:

$$\begin{aligned} L_{\mathbf{G}}\mathbf{h}(\mathbf{x}) &= \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \frac{r}{2} \begin{bmatrix} \cos \theta & \cos \theta \\ \sin \theta & \sin \theta \\ \frac{1}{l} & -\frac{1}{l} \end{bmatrix} \\ &= \frac{r}{2} \begin{bmatrix} \sin \theta & \sin \theta \\ \frac{1}{l} & -\frac{1}{l} \end{bmatrix} \end{aligned} \quad (3.8)$$

and the I/O-Linearization is well defined for the region D_2

$$\{D_2 \mid D_2 \subset \mathbb{R}, D_2 \neq n\pi\}, \quad n \in \mathbb{Z}.$$

The region of interest $\theta \in \Omega_2 \subset D_2$ is in the neighborhood of $\pi/2$ and is therefore well defined.

The input \mathbf{u} is obtained from (3.8) as follows:

$$\begin{bmatrix} \dot{\phi}_r \\ \dot{\phi}_l \end{bmatrix} = \begin{bmatrix} \frac{1}{r \sin \theta} & \frac{l}{r} \\ \frac{1}{r \sin \theta} & -\frac{l}{r} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}. \quad (3.9)$$

3.3.2 Controller design

After the I/O-linearization, a linear controller with a single gain is applicable to states ζ_2 and ζ_3 . The controller for ζ_2

$$\begin{aligned} w_1 &= k_x e_x \\ w_2 &= k_{\theta_{\zeta_2}} e_\theta \end{aligned} \quad (3.10)$$

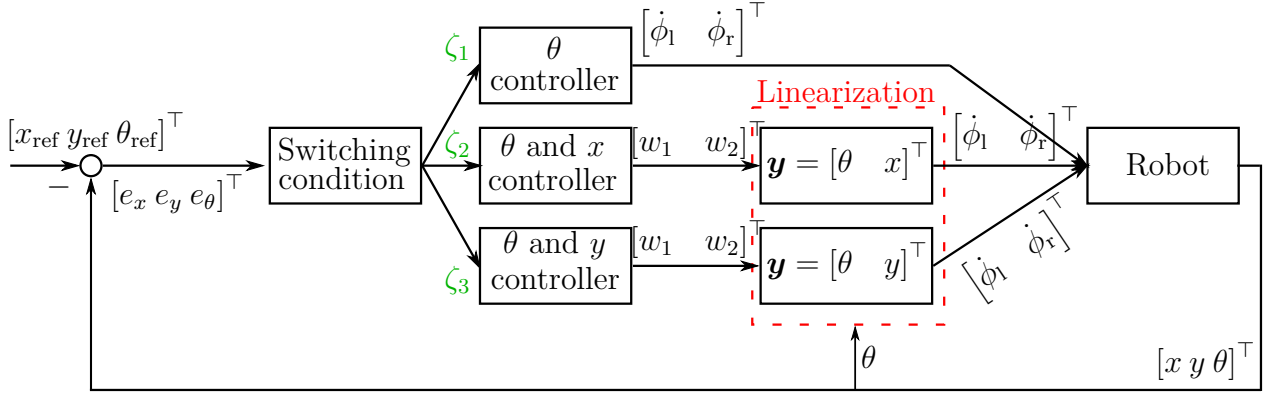


Figure 3.4: Blockdiagram of the system.

and for ζ_3

$$\begin{aligned} w_1 &= k_y e_y \\ w_2 &= k_{\theta_{\zeta_3}} e_\theta \end{aligned} \quad (3.11)$$

with

$$\begin{aligned} e_x &= x_{\text{ref}} - x \\ e_y &= y_{\text{ref}} - y \\ e_\theta &= \theta_{\text{ref}} - \theta \end{aligned} \quad (3.12)$$

are designed, with $k_x, k_{\theta_{\zeta_2}}, k_y$ and $k_{\theta_{\zeta_3}}$ being the control gains. e_x, e_y and e_θ are the errors with the desired references $x_{\text{ref}}, y_{\text{ref}}$ and θ_{ref} . Furthermore, the state ζ_1 controls θ with the linear dynamics of (3.4). Hence, I/O-linearization is not applied and a controller similar to (3.10) and (3.11) can be used as follows:

$$\dot{\phi}_r = \frac{l}{r} k_{\theta_{\zeta_1}} e_\theta \quad (3.13)$$

$$\dot{\phi}_l = -\frac{l}{r} k_{\theta_{\zeta_1}} e_\theta, \quad (3.14)$$

where $k_{\theta_{\zeta_1}}$ is the control gain. The fraction l/r is used to simplify the control system dynamics and thus gain tuning. The negative sign in (3.14) guarantees stability of the system, which is shown in the next subsection.

A full block diagram of the hybrid system is shown in Figure 3.4 with a block for the switching conditions, the robot plant, the controllers and the corresponding feedback linearizations.

3.3.3 Stability analysis

The error equation is differentiated with respect to time as follows:

$$\dot{e}_x = \dot{x}_{\text{ref}} - \dot{x} = -\dot{x} \quad (3.15)$$

$$\dot{e}_y = \dot{y}_{\text{ref}} - \dot{y} = -\dot{y} \quad (3.16)$$

$$\dot{e}_\theta = \dot{\theta}_{\text{ref}} - \dot{\theta} = -\dot{\theta}, \quad (3.17)$$

where the references are constant and thus zero. Substituting (3.4) into (3.17) leads to

$$\dot{e}_\theta = -\frac{r}{2l}(\dot{\phi}_r - \dot{\phi}_l). \quad (3.18)$$

Substituting (3.13) and (3.14) into (3.18), we have

$$\dot{e}_\theta = -\frac{r}{2l}\left(\frac{l}{r}k_{\theta_{\zeta_1}}e_\theta + \frac{l}{r}k_{\theta_{\zeta_1}}e_\theta\right) = -k_{\theta_{\zeta_1}}e_\theta, \quad (3.19)$$

which is stable for all $k_{\theta_{\zeta_1}} > 0$. Therefore, state ζ_1 is stable.

For the stability proof of ζ_2 , \dot{e}_x and \dot{e}_θ are required, and hence the following equation is obtained:

$$\begin{bmatrix} \dot{e}_x \\ \dot{e}_\theta \end{bmatrix} = \begin{bmatrix} -\dot{x} \\ -\dot{\theta} \end{bmatrix} = \frac{r}{2} \begin{bmatrix} -\cos \theta & -\cos \theta \\ -\frac{1}{l} & \frac{1}{l} \end{bmatrix} \begin{bmatrix} \dot{\phi}_r \\ \dot{\phi}_l \end{bmatrix}. \quad (3.20)$$

Substituting (3.7) and (3.10) into (3.20) leads to

$$\begin{bmatrix} \dot{e}_x \\ \dot{e}_\theta \end{bmatrix} = -\underbrace{\begin{bmatrix} k_x & 0 \\ 0 & k_{\theta_{\zeta_2}} \end{bmatrix}}_{\mathbf{K}_{\zeta_2}} \begin{bmatrix} e_x \\ e_\theta \end{bmatrix}, \quad (3.21)$$

which is stable if \mathbf{K}_{ζ_2} is *positive definite*.

The same procedure is used for the system ζ_3 with error dynamics \dot{e}_y , \dot{e}_θ , and (3.9) and (3.11):

$$\begin{aligned} \begin{bmatrix} \dot{e}_y \\ \dot{e}_\theta \end{bmatrix} &= \begin{bmatrix} -\dot{y} \\ -\dot{\theta} \end{bmatrix} = \frac{r}{2} \begin{bmatrix} -\sin \theta & -\sin \theta \\ -\frac{1}{l} & \frac{1}{l} \end{bmatrix} \begin{bmatrix} \dot{\phi}_r \\ \dot{\phi}_l \end{bmatrix} \\ &= -\underbrace{\begin{bmatrix} k_y & 0 \\ 0 & k_{\theta_{\zeta_3}} \end{bmatrix}}_{\mathbf{K}_{\zeta_3}} \begin{bmatrix} e_y \\ e_\theta \end{bmatrix}. \end{aligned} \quad (3.22)$$

Thus, system ζ_3 is stable if \mathbf{K}_{ζ_3} is *positive definite*. Since there is no cycle in the hybrid system in Figure 3.2 (no state can switch to a previous state) and each state converges to its desired values, the entire system is stable.

Table 3.1: Initial position and orientation \mathbf{x}_0 , control gains k_x, k_y and $k_{\theta_{\zeta_i}}$ for the controllers, and threshold values $\epsilon_x, \epsilon_\theta$, for simulation and experiment.

Parameter	Value			
	Simulation	Exp. #1	Exp. #2	Exp. #3
x_0	$-\sqrt{2}$ m	-1 m	1 m	0.5 m
y_0	0 m	0 m	0 m	0.5 m
θ_0	210°	90°	45°	30°
$k_{\theta_{\zeta_1}}$	1.6 1/s	0.84 1/s	0.84 1/s	0.84 1/s
k_x	0.9 rad/s	0.4 rad/s	0.4 rad/s	0.4 rad/s
$k_{\theta_{\zeta_2}}$	0.6 1/s	0.3 1/s	0.3 1/s	0.3 1/s
k_y	0.7 rad/s	0.3 rad/s	0.3 rad/s	0.3 rad/s
$k_{\theta_{\zeta_3}}$	0.6 1/s	0.3 1/s	0.3 1/s	0.3 1/s
ϵ_θ	0.5°	1.5°	1.5°	1.5°
ϵ_x	0.01 m	0.05 m	0.05 m	0.05 m

3.4 Results and discussion

This section presents the simulation and experimental results for the proposed hybrid systems approach.

3.4.1 Simulation results

Simulations for the proposed controller are done with MATLAB® 2018a. Table 3.1 shows the gains for the controllers, the initial position and orientation of the mobile robot, and the chosen threshold values for the switching conditions. The maximum possible angular velocity $\dot{\phi}_{\max}$ of the robot is 10.163 rad/s. The trajectories for x, y, θ and states $\zeta_i, i = 1, 2, 3$ are shown in Figure 3.5. The reference $\theta_{\text{ref}1}$ correspond to θ_{ref} in ζ_1 , and $\theta_{\text{ref}2}$ corresponds to the angular references in ζ_2 and ζ_3 . The robot converged to the desired orientation in state ζ_1 in less than 5 s, and then it switched to state ζ_2 in which the robot converged to the desired orientation and position in less than 15 s. The parking procedure completed in less than 20 s. The angular velocity commands for the left and right drive wheel are shown in Figure 3.6. Both, the angular velocity for the right wheel and the left wheel do not reach the maximum possible angular velocity, hence the gains are chosen well enough for the controllers. Furthermore, since the robots initial position is at the boundary of the range of interest for the parking controller, the angular velocities will not saturate to their limits for any initial position for the chosen gains.

3.4.2 Experimental results

The differential drive mobile robot for the experiment is shown in Figure 1.6, which uses an IMU to measure heading θ and encoders for measuring angular velocities of each wheel. Both

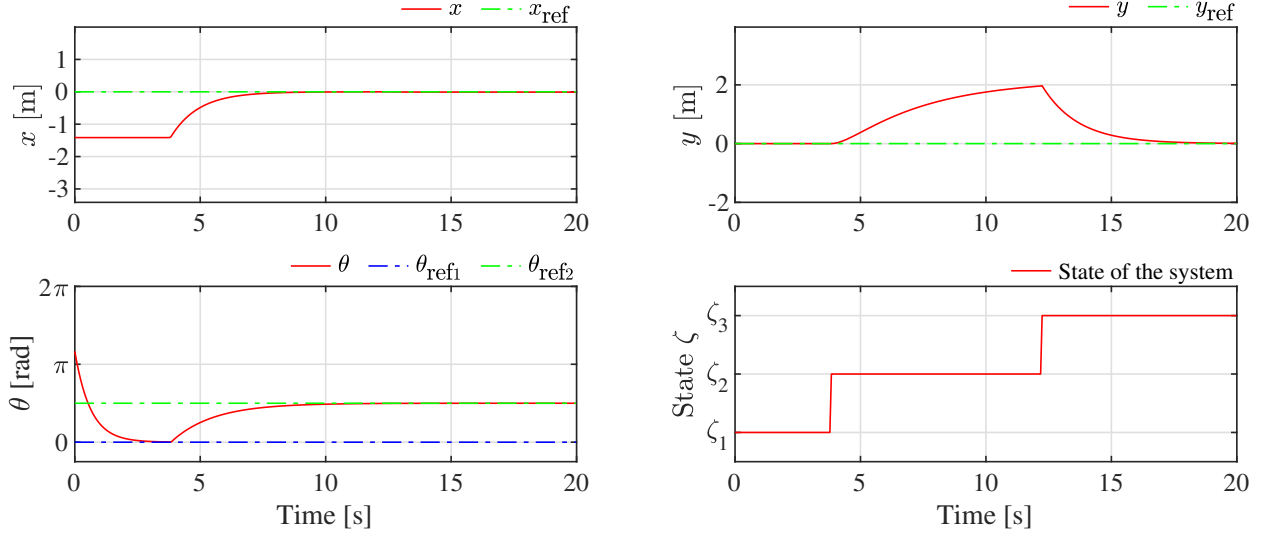


Figure 3.5: Simulation results of robot trajectories and state transitions.

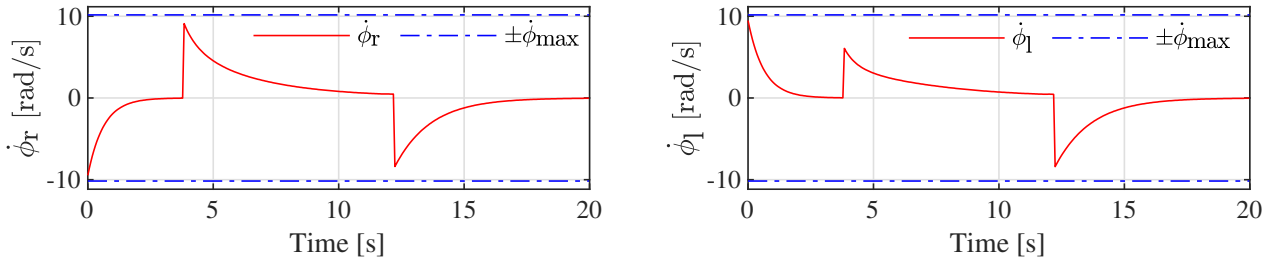


Figure 3.6: Simulation results of wheel angular velocities.

measurements are used to calculate the robots position in the global reference frame. The angular velocities of each wheel are the reference commands for an internal controller of the robot. The experimental parameter values are shown in Table 3.1. Due to safety reasons, gains are set smaller than those in the simulation. Furthermore, the thresholds are set to larger values in the experiment. Figure 3.7 shows the resulting robot trajectories and system states. The profiles are similar to the simulation results. In all experiments, the robot converged to the desired orientation in state ζ_1 in less than 6 s, and then it switched to state ζ_2 in which the robot converged to the desired orientation and position in less than 20 s. The parking for all experiments completed in less than 30 s. The measured angular velocities of each wheel are illustrated in Figure 3.8, where the profiles are similar to the simulations with smaller angular velocities due to the chosen gains. The trajectory of the robot in the global reference frame is shown in Figure 3.9.

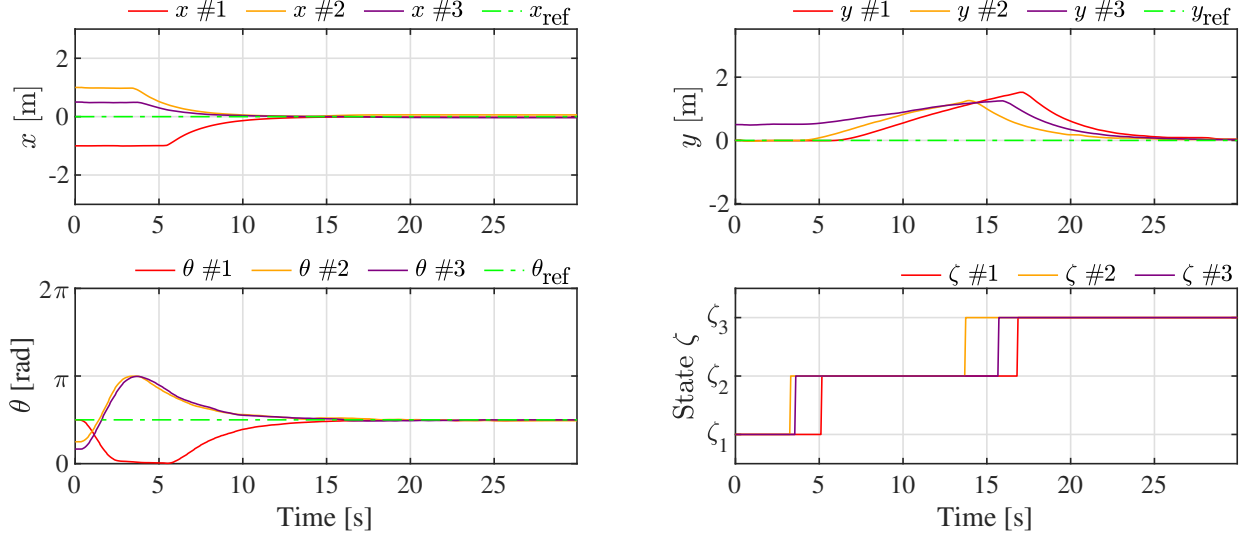


Figure 3.7: Experimental results of robot trajectories and state transitions.

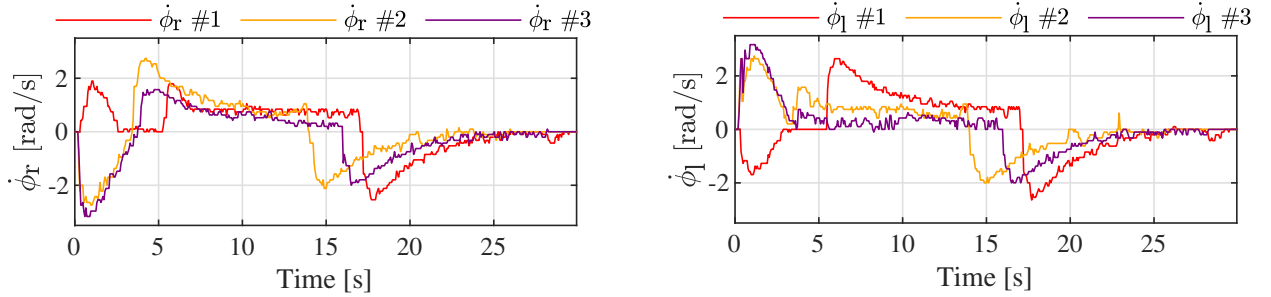


Figure 3.8: Experimental results of wheel angular velocities.

3.5 Summary

This chapter proposed a parking control approach for differential drive mobile robots. An hybrid systems approach consisting of three states was introduced. The first state controls the orientation of the robot and from there I/O-linearization with a linear controller was used to control the position of the robot such that the robot parks at the desired pose. Simulation and experimental results have shown that the robot is able to reach its desired states in each control step and parks at the desired pose.

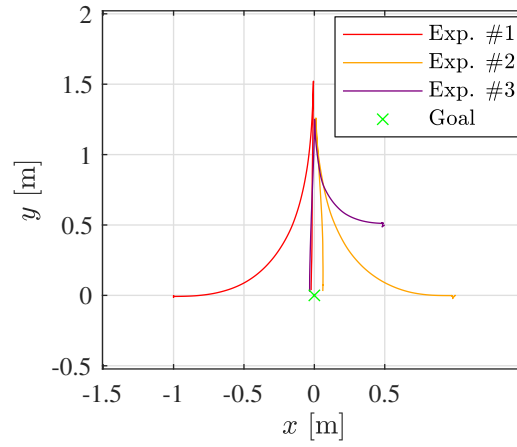


Figure 3.9: Experimental results of robot planar trajectories.

4 Probabilistic Safe Path Planning

4.1 Problem formulation

This study considers a planar motion of arbitrary-shaped mobile robots whose configuration space $\mathcal{C}^{\text{PTP}} \in \text{SE}(2) = \mathbb{R}^2 \times \text{SO}(2)$ is the special Euclidean group which consists of two-dimensional position and orientation. Furthermore, the configuration space is divided into open space $\mathcal{C}_{\text{free}}^{\text{PTP}} \subseteq \mathcal{C}^{\text{PTP}}$ and obstacles $\mathcal{C}_{\text{obs}}^{\text{PTP}} \subset \mathcal{C}^{\text{PTP}}$, where $\mathcal{C}_{\text{free}}^{\text{PTP}} + \mathcal{C}_{\text{obs}}^{\text{PTP}} = \mathcal{C}^{\text{PTP}}$. In a deterministic environment, the objective is to find a path from the start node $\mathbf{x}_S \in \mathcal{C}_{\text{free}}^{\text{PTP}}$ to the goal node $\mathbf{x}_G \in \mathcal{C}_{\text{free}}^{\text{PTP}}$. The complete path $\mathcal{P} = (\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_t, \mathbf{x}_{t+1}, \dots, \mathbf{x}_{t_{\text{goal}}})$ is described by the set of nodes whose ends are the start and goal nodes, $\mathbf{x}_S = \mathbf{x}_0$ and $\mathbf{x}_G = \mathbf{x}_{t_{\text{goal}}}$, respectively. Moreover, \mathbf{x}_t is the parent node of \mathbf{x}_{t+1} , $t \in \mathbb{N}$ is the respective integer time instant, and t_{goal} is the final time instant. The optimization problem is described as follows:

$$\begin{aligned} & \text{minimize} && \sum_{t=1}^{t_{\text{goal}}} J(\mathbf{x}_t) \\ & \text{subject to} && \mathbf{x}_t \in \mathcal{C}_{\text{free}}^{\text{PTP}} \quad \forall t \in \{0, \dots, t_{\text{goal}}\}, \end{aligned}$$

where $J(\mathbf{x}_t)$ is the cost function. In belief space planning, the collision condition is described by chance constraints and the cost function is the expected cost of $J(\mathbf{x}_t)$. Accordingly, the stochastic optimization problem is described as

$$\begin{aligned} & \text{minimize} && \sum_{t=1}^{t_{\text{goal}}} E[J(\mathbf{x}_t)] \\ & \text{subject to} && p(\mathbf{x}_t \in \mathcal{C}_{\text{obs}}^{\text{PTP}}) \leq \Gamma \quad \forall t \in \{0, \dots, t_{\text{goal}}\}, \end{aligned}$$

where $E[\cdot]$, $p(\cdot)$ and Γ denote the expected value, probability of the outcome, and threshold value for collision, respectively.

Static and dynamic obstacles in the configuration space are represented as convex polygons, as illustrated in Figure 4.1, which allows a fast collision analysis and makes it possible to incorporate a probabilistic collision check [11]. It is assumed that each vertex of an obstacle is known, and the obstacles are obtained from an offline map. In the following section, the robot is assumed to be a point robot, and therefore collision will occur when the robot point is inside the obstacle. Using the known obstacle vertices and robot position $\mathbf{q}_t = [x_t, y_t]^\top$, the collision

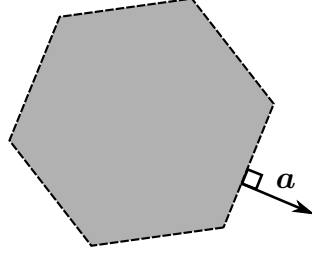


Figure 4.1: Convex obstacle representation in the configuration space.

conditions are expressed as follows:

$$\bigwedge_{i=1}^{n_e} \mathbf{a}_i^\top \mathbf{q}_t < b_i, \quad (4.1)$$

where n_e is the number of edges and \mathbf{a}_i and b_i denote the parameters of the straight-line equation $\mathbf{a}_i^\top \mathbf{q}_t = b_i$ of edge i , respectively.

4.2 Hybrid A* algorithm and state and environment uncertainties

This section describes the uncertainty propagation for a kinematic model of a nonholonomic mobile robot [114]. After briefly explaining the conventional HA* algorithm [29], the linear velocity model of dynamic obstacles and their uncertainty propagation is introduced.

4.2.1 Kinematic model uncertainty propagation

In this work, the velocity model from [114] is used to model the kinematics of the nonholonomic mobile robot. The deterministic kinematic model of the form

$$\mathbf{x}_{t+1} = \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t) \quad (4.2)$$

with nonzero angular velocity is as follows [114]:

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{bmatrix} = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} + \begin{bmatrix} -\frac{v_t}{\omega_t} \sin \theta_t + \frac{v_t}{\omega_t} \sin(\theta_t + \omega_t \delta t) \\ \frac{v_t}{\omega_t} \cos \theta_t - \frac{v_t}{\omega_t} \cos(\theta_t + \omega_t \delta t) \\ \omega_t \delta t \end{bmatrix}, \quad (4.3)$$

where $\mathbf{x}_t = [x_t, y_t, \theta_t]^\top \in \text{SE}(2)$ is the pose of the robot at time instant t with x_t, y_t being the position in the global reference frame and θ_t being the heading of the robot. Moreover, $\mathbf{u}_t = [v_t, \omega_t]^\top$, where v_t and ω_t are the linear and angular velocities, respectively, and δt is the sampling time. The model for straight motions can be obtained using L'Hôpital's rule for

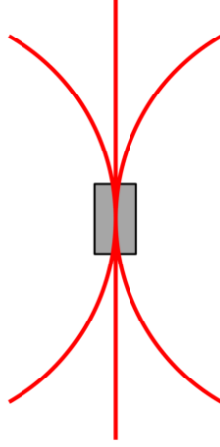


Figure 4.2: Example motions for HA* using the velocity model.

$\lim_{\omega_t \rightarrow 0} f(\mathbf{x}_t, \mathbf{u}_t)$ as follows:

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{bmatrix} = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} + \begin{bmatrix} v_t \delta t \cos \theta_t \\ v_t \delta t \sin \theta_t \\ 0 \end{bmatrix}. \quad (4.4)$$

Example motions using the velocity model for HA* are shown in Figure 4.2. The additive Gaussian process noise is integrated into the model by replacing $\mathbf{u}_t = [v_t, \omega_t]^\top$ with $\hat{\mathbf{u}}_t = [\hat{v}_t, \hat{\omega}_t]^\top$, where

$$\begin{bmatrix} \hat{v}_t \\ \hat{\omega}_t \end{bmatrix} = \begin{bmatrix} v_t \\ \omega_t \end{bmatrix} + \boldsymbol{\varepsilon}_M \quad (4.5)$$

with

$$\boldsymbol{\varepsilon}_M \sim \mathcal{N}(0, \Sigma_M). \quad (4.6)$$

Here, Σ_M is the covariance matrix of the random variable $\boldsymbol{\varepsilon}_M$. The Gaussian probability distribution Σ_{x_t} is obtained using the prediction step of the EKF [45]. Hence, the nonlinear kinematic model of the robot has to be linearized at each time instant around the input \mathbf{u}_t and state mean $\boldsymbol{\mu}_{x_t}$

$$\begin{aligned} \boldsymbol{\mu}_{x_{t+1}} &\approx \tilde{\mathbf{A}}_t \boldsymbol{\mu}_{x_t} + \tilde{\mathbf{B}}_t \mathbf{u}_t \\ \tilde{\mathbf{A}} &= \frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\boldsymbol{\mu}_{x_t}, \mathbf{u}_t) \\ \tilde{\mathbf{B}} &= \frac{\partial \mathbf{f}}{\partial \mathbf{u}}(\boldsymbol{\mu}_{x_t}, \mathbf{u}_t), \end{aligned} \quad (4.7)$$

where

$$\tilde{\mathbf{A}}_t = \begin{bmatrix} 1 & 0 & -\frac{v_t}{\omega_t} \cos \mu_{\theta_t} + \frac{v_t}{\omega_t} \cos(\mu_{\theta_t} + \omega_t \delta t) \\ 0 & 1 & -\frac{v_t}{\omega_t} \sin \mu_{\theta_t} + \frac{v_t}{\omega_t} \sin(\mu_{\theta_t} + \omega_t \delta t) \\ 0 & 0 & 1 \end{bmatrix} \quad (4.8)$$

$$\begin{aligned} \tilde{\mathbf{B}}_t &= [\tilde{\mathbf{b}}_1, \tilde{\mathbf{b}}_2] \\ \tilde{\mathbf{b}}_1 &= \begin{bmatrix} -\frac{\sin \mu_{\theta_t} + \sin(\mu_{\theta_t} + \omega_t \delta t)}{\omega_t} \\ \frac{\cos \mu_{\theta_t} - \cos(\mu_{\theta_t} + \omega_t \delta t)}{\omega_t} \\ 0 \end{bmatrix} \\ \tilde{\mathbf{b}}_2 &= \begin{bmatrix} \frac{v_t(\sin \mu_{\theta_t} - \sin(\mu_{\theta_t} + \omega_t \delta t))}{\omega_t^2} + \frac{v_t \cos(\mu_{\theta_t} + \omega_t \delta t) \delta t}{\omega_t} \\ -\frac{v_t(\cos \mu_{\theta_t} - \cos(\mu_{\theta_t} + \omega_t \delta t))}{\omega_t^2} + \frac{v_t \sin(\mu_{\theta_t} + \omega_t \delta t) \delta t}{\omega_t} \\ \delta t \end{bmatrix} \end{aligned} \quad (4.9)$$

for $\omega_t \neq 0$. For straight motions, the linearization is given as follows:

$$\tilde{\mathbf{A}}_t = \begin{bmatrix} 1 & 0 & -v_t \delta t \sin \mu_{\theta_t} \\ 0 & 1 & v_t \delta t \cos \mu_{\theta_t} \\ 0 & 0 & 1 \end{bmatrix} \quad (4.10)$$

$$\tilde{\mathbf{B}}_t = \begin{bmatrix} \delta t \cos \mu_{\theta_t} & 0 \\ \delta t \sin \mu_{\theta_t} & 0 \\ 0 & 0 \end{bmatrix}. \quad (4.11)$$

With the linearized model, Gaussian uncertainty distribution $\Sigma_{\mathbf{x}_{t+1}}$ can be updated as follows:

$$\begin{aligned} \mu_{\mathbf{x}_{t+1}} &= \mathbf{f}(\mu_{\mathbf{x}_t}, \mathbf{u}_t) \\ \Sigma_{\mathbf{x}_{t+1}} &= \tilde{\mathbf{A}}_t \Sigma_{\mathbf{x}_t} \tilde{\mathbf{A}}_t^\top + \tilde{\mathbf{B}}_t \Sigma_M \tilde{\mathbf{B}}_t^\top, \end{aligned} \quad (4.12)$$

where $\tilde{\mathbf{B}}_t \Sigma_M \tilde{\mathbf{B}}_t^\top$ is the mapping of the motion noise from the control space to state space [114].

4.2.2 Algorithm overview

The conventional HA* [29] is a variation of the A* algorithm [105]. Unlike the A* algorithm, the HA* algorithm assigns a continuous mobile robot coordinate to each discrete cell in the working space. Each node of the planner is expanded by applying three different actions: no-turn, left turn, and right turn for forward and reverse motions (Figure 4.2). Reverse motions are only applied if a forward motion violates a collision constraint. A kinematic model of the mobile robot is used to generate new states for the planner ((4.3) and (4.4)), and the same cost-to-goal heuristics as in [29] are applied: The *non-holonomic-without-obstacles* heuristic, which uses Reeds and Shepp paths [100] to compute the shortest path to the goal (assuming no obstacles) from every point in the working space, and the *holonomic-with-obstacles* heuristic, which employs the occupancy grid of the configuration space to compute the shortest path to

the goal via dynamic programming. The maximum heuristic cost of both heuristics is used for the path planner.

For every N^{th} node, an expansion of the current node to the goal with Reeds and Shepp paths is generated, where N depends on the cost-to-goal heuristic. If the expansion does not collide with any obstacles, the HA* algorithm is completed.

The HA* algorithm uses a cost function to penalize the reverse motion and switching of the motion direction of the path traveled:

$$J(\mathbf{x}_t) = l_t(1 + r_t C_{\text{rev}}) + |r_t - r_{t-1}| C_{\text{sw}}, \quad (4.13)$$

where l_t is the distance traveled from time instant $t - 1$ to time instant t , and C_{rev} and C_{sw} are the penalty gains for driving in reverse and switching the direction of motion, respectively. r_t denotes the boolean state of driving forward or reverse (reverse is set to 1).

This work modifies the HA* algorithm by considering Gaussian uncertainties in the robot's motion, as well as the static obstacle position. Furthermore, dynamic obstacles with uncertain state information are considered. Hence, the robot's uncertain state is updated at each node expansion using (4.12). If the condition

$$p(\mathbf{x}_t \in \mathcal{C}_{\text{obs}}) \leq \Gamma \quad (4.14)$$

is satisfied the current node is probabilistically safe and the expected cost

$$E[J(\mathbf{x}_t)] = \mu_{l_t}(1 + r_t C_{\text{rev}}) + |r_t - r_{t-1}| C_{\text{sw}} \quad (4.15)$$

is calculated, where μ_{l_t} is the mean distance traveled from time instant $t - 1$ to time instant t . If the condition is not satisfied the node will be pruned.

4.2.3 Static and dynamic obstacles

4.2.3.1 Static obstacles

The static obstacles are expressed in a manner that is similar to [74]. The probability distributions of the static obstacles are time invariant, i.e.,

$$\underbrace{\begin{bmatrix} x_{\mathcal{S}^j} \\ y_{\mathcal{S}^j} \end{bmatrix}}_{\mathbf{x}_{\mathcal{S}^j}} = \underbrace{\begin{bmatrix} x_{\mathcal{S}_0^j} \\ y_{\mathcal{S}_0^j} \end{bmatrix}}_{\mathbf{x}_{\mathcal{S}_0^j}} + \boldsymbol{\varepsilon}_{\mathcal{S}^j} \quad (4.16)$$

with

$$\boldsymbol{\varepsilon}_{\mathcal{S}^j} \sim \mathcal{N}(0, \Sigma_{\mathcal{S}^j}) \quad j \in \{1, \dots, n_{\mathcal{S}}\}, \quad (4.17)$$

where $[x_{\mathcal{S}^j}, y_{\mathcal{S}^j}]^{\top}$ is the position of obstacle j , $[x_{\mathcal{S}_0^j}, y_{\mathcal{S}_0^j}]^{\top}$ is the nominal position of obstacle j , and $n_{\mathcal{S}}$ is the number of static obstacles in the configuration space. Hereafter, all static

obstacles have the same covariance, i.e., $\Sigma_{\mathcal{S}_j} = \Sigma_{\mathcal{S}}$.

4.2.3.2 Dynamic obstacles

The dynamic obstacles are assumed to be convex with an estimated mean heading and velocity. This work uses a linear velocity model for the dynamic obstacle

$$\begin{bmatrix} x_{\mathcal{D}_{t+1}} \\ y_{\mathcal{D}_{t+1}} \\ \dot{x}_{\mathcal{D}_{t+1}} \\ \dot{y}_{\mathcal{D}_{t+1}} \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & \delta t & 0 \\ 0 & 1 & 0 & \delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\mathbf{A}_{\mathcal{D}}} \underbrace{\begin{bmatrix} x_{\mathcal{D}_t} \\ y_{\mathcal{D}_t} \\ \dot{x}_{\mathcal{D}_t} \\ \dot{y}_{\mathcal{D}_t} \end{bmatrix}}_{\mathbf{x}_{\mathcal{D}_t}} + \boldsymbol{\varepsilon}_{\mathcal{D}}, \quad (4.18)$$

where $x_{\mathcal{D}_t}, y_{\mathcal{D}_t}$ denote the position, and $\dot{x}_{\mathcal{D}_t}, \dot{y}_{\mathcal{D}_t}$ denote the velocity in x and y direction in the global reference frame, respectively. $\boldsymbol{\varepsilon}_{\mathcal{D}} \sim \mathcal{N}(0, \Sigma_{\mathcal{D}})$ is a random variable with Gaussian noise distribution with covariance matrix $\Sigma_{\mathcal{D}}$. In the following, it is assumed that the robot is able to detect and track a dynamic obstacle at each time instant for example with the methods in [99] and [8]. It is often sufficient to assume a linear motion of a dynamic obstacle [110]. A linear model allows a one time computation of the mean state and the covariance of the dynamic obstacle using the KF [45]

$$\boldsymbol{\mu}_{\mathbf{x}_{\mathcal{D}_{t+1}}} = \mathbf{A}_{\mathcal{D}}^t \boldsymbol{\mu}_{\mathbf{x}_{\mathcal{D}_0}} \quad (4.19)$$

$$\begin{aligned} \Sigma_{\mathbf{x}_{\mathcal{D}_{t+1}}} &= \mathbf{A}_{\mathcal{D}}^{t+1} \Sigma_{\mathbf{x}_{\mathcal{D}_0}} (\mathbf{A}_{\mathcal{D}}^{\top})^{t+1} \\ &+ \sum_{k=0}^t \mathbf{A}_{\mathcal{D}}^{t-k-1} \Sigma_{\mathcal{D}} (\mathbf{A}_{\mathcal{D}}^{\top})^{t-k-1}, \end{aligned} \quad (4.20)$$

where $\boldsymbol{\mu}_{\mathbf{x}_{\mathcal{D}_t}}$ and $\Sigma_{\mathbf{x}_{\mathcal{D}_t}}$ are the mean and the covariance matrix of the dynamic obstacle at time instant t , respectively. Process noise is used to increase the uncertainty while propagating the mean state and covariance in time, which increases robustness without specific knowledge of the dynamic obstacles' behavior. In this case, the process noise of the dynamic obstacle can be tuned based on confidence of the unknown future behavior of the dynamic obstacle. If it is certain that the dynamic obstacle will continue moving in the measured direction at the measured velocity, small values for the process noise matrix can be selected as parameters inside the algorithm. If the dynamic obstacle behaved in a nonlinear manner in previous time instants, larger values for the process noise should be selected.

4.3 Path planning under Gaussian uncertainty

This section introduces the collision condition for probabilistic safe motion of robot points and arbitrary-shaped mobile robots. Results and a discussion are given in the end of this section.

4.3.1 Probabilistic collision check

Two methods for calculating the probability of collision are presented in the following.

4.3.1.1 Chance constraints

The objective is to guarantee that the probability of a collision with any obstacle is less than the assigned value Γ . In a deterministic world, a collision will occur if (4.1) is satisfied. Hence, in belief space context the inequality, i.e.,

$$p(\mathbf{x}_t \in \mathcal{C}_{\text{obs}}) = p\left(\bigwedge_{i=1}^{n_e} \mathbf{a}_i^\top \mathbf{q}_t < b_{it}\right) \leq \Gamma \quad (4.21)$$

has to be evaluated. Here, b_{it} is only time dependent for a dynamic obstacle and a is not time dependent since we assume no rotation of obstacles. Since normality is preserved by linear transformations [10], a new random variable, $\varepsilon_d \sim \mathcal{N}(\mu_d, \sigma_d^2)$ can be introduced with

$$\mu_d = \mathbf{a}^\top \boldsymbol{\mu}_{q_t} - b \quad (4.22)$$

$$\sigma_d^2 = \mathbf{a}^\top \Sigma_{q_t} \mathbf{a}, \quad (4.23)$$

where $\boldsymbol{\mu}_{q_t}$ is the mean value of the position of the mobile robot with respective covariance matrix Σ_{q_t} at time instant t . Mean and covariance are taken from the mean vector and covariance matrix of robot state \mathbf{x}_t , respectively. Therefore, the new inequality condition for the probability of collision is as follows:

$$p(\varepsilon_d < 0) \leq \Gamma. \quad (4.24)$$

This probabilistic constraint is shown to be equivalent to the following deterministic constraint using the mean and variance of d , together with the inverse Gaussian error function [11, 14]:

$$\mu_d \geq \sqrt{2\sigma_d^2} \text{erf}^{-1}(1 - 2\Gamma). \quad (4.25)$$

The left-hand side of the above equation is the mean distance between the center of the robot and the line segment of the obstacle, and the right-hand side represents the minimum allowed distance between the center of the robot and the line segment. Hence, the chance constraint is satisfied if

$$\bigvee_{i=1}^{n_e} \mathbf{a}_i^\top \boldsymbol{\mu}_{q_t} - b_{it} \geq \sqrt{2\mathbf{a}_i^\top \Sigma_{q_t} \mathbf{a}_i} \text{erf}^{-1}(1 - 2\Gamma) \quad (4.26)$$

is true. This approach can be extended for obstacles with an uncertain position and translation [74]. The current work assumes static and dynamic obstacles. The mean distance μ_d is unchanged for static obstacles and can be calculated for dynamic obstacles with updated parameters b_{it} based on the translation of the dynamic obstacle at time instant t . The variance

for d for both static and dynamic obstacles is as follows:

$$\sigma_d^2 = \mathbf{a}^\top (\Sigma_{q_t} + \Sigma_{\mathcal{O}}) \mathbf{a}, \quad (4.27)$$

where $\Sigma_{\mathcal{O}}$ is the covariance matrix of the respective obstacle. Then, the final inequality equations for a single obstacle are shown as below:

$$\bigvee_{i=1}^{n_e} \mathbf{a}_i^\top \boldsymbol{\mu}_{q_t} - b_{it} \geq \sqrt{2\mathbf{a}_i^\top (\Sigma_{q_t} + \Sigma_{\mathcal{O}}) \mathbf{a}_i} \operatorname{erf}^{-1}(1 - 2\Gamma). \quad (4.28)$$

This inequality condition has to be extended for multiple obstacles, which can be obtained using an upper bound for the probability of collision with at least one obstacle using Boole's inequality [11], i.e.,

$$p(\mathbf{x}_t \in \mathcal{C}_{\text{obs}}) \leq \sum_{j=1}^{n_{\mathcal{O}}} p\left(\bigwedge_{i=1}^{n_{\mathcal{O}_j}} \mathbf{a}_{ji}^\top \boldsymbol{\mu}_{q_t} < b_{jit}\right) \leq \sum_{j=1}^{n_{\mathcal{O}}} \gamma_j = \Gamma, \quad (4.29)$$

where $n_{\mathcal{O}_j}$ represents the number of line-segments for obstacle j and $n_{\mathcal{O}}$ is the total number of obstacles. It is shown that this inequality can only be guaranteed if

$$\gamma_j = \gamma = \frac{\Gamma}{n_{\mathcal{O}}}. \quad (4.30)$$

Hence, the conjunction of inequalities at each time instant t , i.e.,

$$\bigwedge_{j=1}^{n_{\mathcal{O}}} \left(\bigvee_{i=1}^{n_{\mathcal{O}_j}} \mu_{d_{ji}} \geq \sqrt{2\sigma_{d_{ji}}^2} \operatorname{erf}^{-1}(1 - 2\gamma) \right) \quad (4.31)$$

has to be satisfied to guarantee safe execution. Hereafter, the HA* algorithm that employs this method for the probabilistic collision check will be referred to as the chance constraint hybrid A* (CCHA*).

4.3.1.2 Exact collision probability

The inverse Gaussian error function will be over-conservative for an increasing number of obstacles in the configuration space (see (4.30)). Hence, many nodes will be pruned and a solution may not be found in a many-obstacle case. This drawback can be circumvented using the Gaussian error function directly to obtain the exact probability of collision for the respective obstacle [74]:

$$\delta_{jit} = \frac{1}{2} \left[1 - \operatorname{erf} \left(\frac{\mathbf{a}_{ji}^\top \boldsymbol{\mu}_{q_t} - b_{jit}}{\sqrt{2\mathbf{a}_{ji}^\top (\Sigma_{q_t} + \Sigma_{\mathcal{O}}) \mathbf{a}_{ji}}} \right) \right], \quad (4.32)$$

where δ_{jit} is the probability of collision with line segment i of obstacle j at time instant t . The inequality equation for all obstacles at one time instant using the Boole's inequality is as follows

$$p(\mathbf{x}_t \in \mathcal{C}_{\text{obs}}) \leq \sum_{j=1}^{n_{\mathcal{O}}} \min_{i=1, \dots, n_{\mathcal{O}_j}} \delta_{jit} = \Delta_t(\mathbf{x}_t), \quad (4.33)$$

where $\Delta_t(\mathbf{x}_t)$ is the probability of collision for \mathbf{x}_t at time instant t . The CCHA* algorithm that uses this method are referred to as CCxHA*, where “x” indicates that the algorithm uses the Gaussian error function and as such calculates the exact probability of collision per obstacle.

4.3.2 Cost function extension

This section introduces an extension of the cost function for the HA* algorithm

$$\begin{aligned} E[J(\mathbf{x}_t)] = & \mu_{l_t}(1 + r_t C_{\text{rev}}) \\ & + |r_t - r_{t-1}| C_{\text{sw}} \\ & + k \ln(1 - \Delta_t(\mathbf{x}_t)). \end{aligned} \quad (4.34)$$

A new soft constraint is added for considering the probability of collision for the mobile robot, where $k \leq 0$ is a tuning gain employed as a trade-off between traveled distance and path safety. Instead of pruning nodes that do not satisfy the probability of collision Γ , the cost function penalizes the cost of nodes with high probability of collision. This approach has the advantage that no nodes will be pruned, and accordingly the algorithm may find a path where other approaches will fail. This path may, however have a high probability of collision. Hereafter, all HA* algorithms using the adapted cost function are labeled SCHA*, where SC refers to “soft constraint”.

4.4 Arbitrary-shaped mobile robots

In general, path planning algorithms work with point robots [29]. This can be achieved by inflating the obstacles based on the robot's shape and heading. For circular mobile robots, the obstacles simply have to be inflated by the radius of the robot. However, for arbitrary-shaped mobile robots, the obstacles must be inflated using the Minkowski difference [63], where each possible heading also has to be considered. This can computationally and memory-wise be very demanding. Uncertainty in the robot's heading complicates the inflation. One simplification for the configuration space is the segmentation of the robot into smaller circular parts to achieve a similar result as for circular mobile robots [122], where the obstacles will be inflated with the radius of the respective segment of the robot. In this work, the robot is separated into several smaller circles as shown in Figure 4.3.

The collision check for the center of motion is the same as in (4.31) and (4.33) using the parameters for the inflated obstacle. For the other segments, the uncertainty of the heading

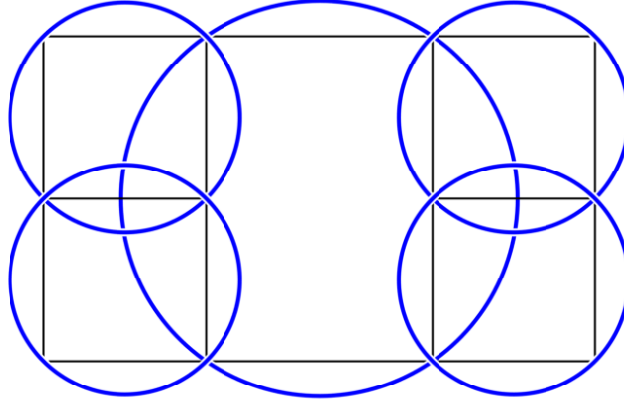


Figure 4.3: Segmentation of a rectangular robot into smaller segments.

has to be included. The general equation for deriving the center of each segment is as follows:

$$\begin{bmatrix} x_t^j \\ y_t^j \end{bmatrix} = \underbrace{\begin{bmatrix} x_t \\ y_t \end{bmatrix} + \begin{bmatrix} \cos \theta_t & -\sin \theta_t \\ \sin \theta_t & \cos \theta_t \end{bmatrix} \begin{bmatrix} l_{x_t^j} \\ l_{y_t^j} \end{bmatrix}}_{g(\mathbf{x}_t)}, \quad (4.35)$$

where $[x_t^j, y_t^j]^\top$ is the position of the center of a selected square segment j at time step t , and indicates the selected square segment, and $[l_{x_t^j}, l_{y_t^j}]^\top$ is the corresponding translation value. This transformation is nonlinear, which means that resulting variables are no longer Gaussian [10]. Therefore, the transformation will be approximated by the Taylor series expansion as follows:

$$\begin{aligned} \begin{bmatrix} x_t^j \\ y_t^j \end{bmatrix} &\approx g(\boldsymbol{\mu}_{x_t}) + \nabla g(\mathbf{x}_t)|_{\boldsymbol{\mu}_{x_t}} (\mathbf{x} - \boldsymbol{\mu}_{x_t}) \\ &= \begin{bmatrix} x_t \\ y_t \end{bmatrix} + \begin{bmatrix} -s_{\mu_{\theta_t}} & -c_{\mu_{\theta_t}} \\ c_{\mu_{\theta_t}} & -s_{\mu_{\theta_t}} \end{bmatrix} \begin{bmatrix} l_{x_t^j} \\ l_{y_t^j} \end{bmatrix} \theta_t \\ &\quad + \begin{bmatrix} c_{\mu_{\theta_t}} + s_{\mu_{\theta_t}} \mu_{\theta_t} & -s_{\mu_{\theta_t}} + c_{\mu_{\theta_t}} \mu_{\theta_t} \\ s_{\mu_{\theta_t}} - c_{\mu_{\theta_t}} \mu_{\theta_t} & c_{\mu_{\theta_t}} + s_{\mu_{\theta_t}} \mu_{\theta_t} \end{bmatrix} \begin{bmatrix} l_{x_t^j} \\ l_{y_t^j} \end{bmatrix}, \end{aligned} \quad (4.36)$$

where $s_{\mu_{\theta_t}}$ and $c_{\mu_{\theta_t}}$ are abbreviations for $\sin \mu_{\theta_t}$ and $\cos \mu_{\theta_t}$, respectively; x_t^j and y_t^j on the right-hand side of this equation are of the form $x_t^j = a_1 x_t + b_1 \theta_t + c_1$ and $y_t^j = a_2 y_t + b_2 \theta_t + c_2$ with the constants $a_1, a_2, b_1, b_2, c_1, c_2$. Hence, it is sufficient to calculate the expectation and variance for the general case, i.e., $Z = aX + bY + c$, where a, b , and c are constants and X, Y , and Z are random variables. Subsequently, the expectation and variance are calculated in terms of moment expression. Since it is a linear transformation, the expectation for Z is as follows:

$$E[Z] = E[aX + bY + c] = aE[X] + bE[Y] + c. \quad (4.37)$$

The complete calculation of the variance is shown in Appendix A, and the resulting equation is as follows:

$$\text{var}(Z) = a^2 \text{var}(X) + b^2 \text{var}(Y) + 2ab \text{cov}(X, Y). \quad (4.38)$$

Therefore, the mean and variance for the transformed coordinates are

$$\begin{bmatrix} \mu_{x_t^j} \\ \mu_{y_t^j} \end{bmatrix} = \begin{bmatrix} \mu_{x_t} \\ \mu_{y_t} \end{bmatrix} + \begin{bmatrix} c_{\mu_{\theta_t}} & -s_{\mu_{\theta_t}} \\ s_{\mu_{\theta_t}} & c_{\mu_{\theta_t}} \end{bmatrix} \begin{bmatrix} l_{x_t^j} \\ l_{y_t^j} \end{bmatrix} \quad (4.39)$$

$$\begin{aligned} \begin{bmatrix} \sigma_{x_t^j}^2 \\ \sigma_{y_t^j}^2 \end{bmatrix} &= \begin{bmatrix} \sigma_{x_t}^2 \\ \sigma_{y_t}^2 \end{bmatrix} + \begin{bmatrix} s_{\mu_{\theta_t}}^2 l_{x_t^j}^2 + s_{\mu_{\theta_t}} c_{\mu_{\theta_t}} l_{x_t^j} l_{y_t^j} + c_{\mu_{\theta_t}}^2 l_{y_t^j}^2 \\ c_{\mu_{\theta_t}}^2 l_{x_t^j}^2 - s_{\mu_{\theta_t}} c_{\mu_{\theta_t}} l_{x_t^j} l_{y_t^j} + s_{\mu_{\theta_t}}^2 l_{y_t^j}^2 \end{bmatrix} \sigma_{\theta_t}^2 \\ &+ 2 \begin{bmatrix} \sigma_{x_t, \theta_t} & 0 \\ 0 & \sigma_{y_t, \theta_t} \end{bmatrix} \begin{bmatrix} -s_{\mu_{\theta_t}} & -c_{\mu_{\theta_t}} \\ c_{\mu_{\theta_t}} & -s_{\mu_{\theta_t}} \end{bmatrix} \begin{bmatrix} l_{x_t^j} \\ l_{y_t^j} \end{bmatrix} \end{aligned} \quad (4.40)$$

Then, $\text{cov}(x_t^j, y_t^j)$ can be calculated in a similar manner (the full equation is shown in Appendix A):

$$\begin{aligned} &\text{cov}(a_1 x_t + b_1 \theta_t + c_1, a_2 y_t + b_2 \theta_t + c_2) \\ &= a_1 a_2 \text{cov}(x_t, y_t) + a_1 b_2 \text{cov}(x_t, \theta_t) \\ &+ a_2 b_1 \text{cov}(y_t, \theta_t) + b_1 b_2 \text{var}(\theta_t). \end{aligned} \quad (4.41)$$

The covariance can be expressed in vector form as follows:

$$\begin{aligned} \sigma_{x_t^j, y_t^j} &= \sigma_{x_t, y_t} + \begin{bmatrix} \sigma_{x_t, \theta_t} \\ \sigma_{y_t, \theta_t} \end{bmatrix}^\top \begin{bmatrix} -s_{\mu_{\theta_t}} & -c_{\mu_{\theta_t}} \\ c_{\mu_{\theta_t}} & -s_{\mu_{\theta_t}} \end{bmatrix} \begin{bmatrix} l_{x_t^j} \\ l_{y_t^j} \end{bmatrix} \\ &+ \begin{bmatrix} s_{\mu_{\theta_t}} c_{\mu_{\theta_t}} (l_{y_t^j}^2 - l_{x_t^j}^2) - c_{2\mu_{\theta_t}} l_{x_t^j} l_{y_t^j} \end{bmatrix} \sigma_{\theta_t}^2. \end{aligned} \quad (4.42)$$

Hence, the covariance matrix for the square center positions is

$$\Sigma_{q_t^j} = \begin{bmatrix} \sigma_{x_t^j}^2 & \sigma_{x_t^j, y_t^j} \\ \sigma_{x_t^j, y_t^j} & \sigma_{y_t^j}^2 \end{bmatrix}. \quad (4.43)$$

The above equations can be used to calculate the probability of collision for each of the square centers with either (4.31) or (4.33) using the inflated obstacle parameters, where the maximum probability of collision of all segment centers will be used per obstacle. An example figure of the resulting confidence ellipses for the square segments is shown in Figure 4.4, where the robot's heading is 45° and the selected covariance matrix of the robot is

$$\Sigma_x = \begin{bmatrix} 0.003 & -0.0009 & -0.00005 \\ -0.0009 & 0.003 & -0.00005 \\ -0.00005 & -0.00005 & 0.05 \end{bmatrix}.$$

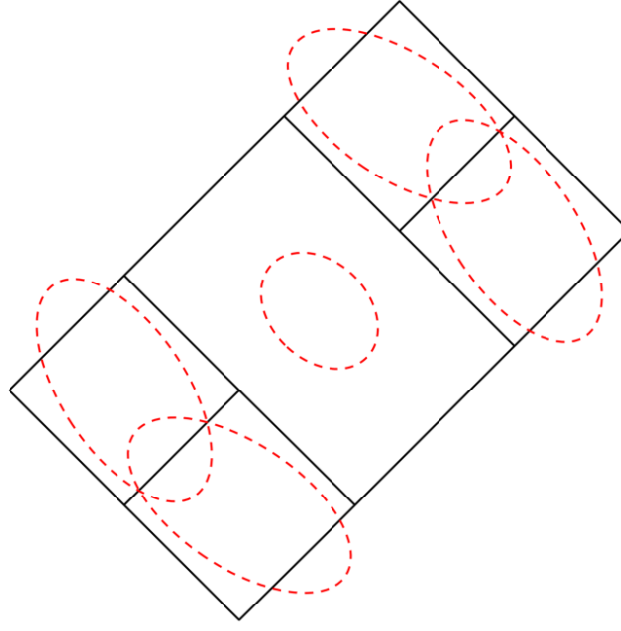


Figure 4.4: Rectangular robot with 95 % confidence ellipses.

It is shown that the confidence ellipses for four square centers are angled and have larger radii than the ellipse of the robot's center. This follows from the fact that variance σ_θ^2 of the heading and covariances $\sigma_{x,\theta}, \sigma_{y,\theta}$ are considered. Hereafter, algorithms (i.e., HA*, CCHA*, CCxHA*, and SCHA*) that incorporate the shape of the robot are labeled with the prefix arbitrary-shaped mobile robot (ASR).

4.4.1 Results and discussion

This subsection demonstrates the validity of the approaches applied in this section. The variants of the proposed algorithms (i.e., CCHA*, CCxHA*, SCHA*, ASR CCHA*, ASR CCxHA* and ASR SCHA*) are examined to evaluate their strengths and weaknesses. The above proposed algorithms are compared with existing approaches in a clustered, static environment, and an environment with a dynamic obstacle. For the comparison, the conventional A* [105], the HA* algorithm [29, 86], ASR HA* considering the shape of the robot using the approach of [122], the RRT algorithm [64], the closed-loop rapidly-exploring random tree (CLRRT) algorithm [61, 76], the HeAT-RT [5] as well as the probabilistic approaches CCRRT and online CCRRT (CCxRRT) [74, 75], which are based on CLRRT, are all implemented. All kinematic planners use the same motion model, and all probabilistic planners use the same uncertainties and uncertainty propagation method. The RRT-based planners for comparison are random in their search, which generate different solution paths in every calculation, and therefore each RRT algorithm is executed 1000 times for both environments. The success rate, shortest distance, mean computation time, standard deviation σ_{time} , mean distance, and standard deviation of the distance σ_{dist} are calculated for evaluation. A plan is considered successful if it finds a

solution within 20 s. Furthermore, RRT-based planners are generally unable to exactly reach the goal pose, but the vicinity of $\boldsymbol{\mu}_{\text{goal}} \in \mathcal{X}_{\text{goal}} \subset \text{SE}(2)$. Here, the vicinity to the goal is defined to be in a range of 0.5 m with a heading difference of less than 20° .

An evaluation of the proposed cost function (see (4.34)) is presented. Finally, the performance of the algorithms is evaluated using MCS [105].

The robot used for simulation has a length of 1.27 m and a width of 0.75 m. The robot will move with a fixed linear velocity of $v = 0.5$ m/s (except for HeAT-RT) and, for the HA*-based planners, an angular velocity of $\omega = 10\pi/180$ rad/s. Furthermore, the environments are discretized with a cell size of 0.5 m and an angle increment of $5\pi/180$ rad. The penalties for driving in reverse and switching the driving direction are set to the same values, i.e., $C_{\text{rev}} = 1.0$ and $C_{\text{sw}} = 1.0$, respectively. The algorithms are tested in ROS Kinetic [98] on a GNU/Linux Ubuntu 16.04 laptop with an Intel® Core™ i7-8550U CPU @ 1.80 GHz \times 8 and 8 GB memory.

4.4.1.1 Results on static environment

In this section, the algorithms are compared in a static environment with nine obstacles (Figure 4.5). The process noise and initial covariance matrix for the robot are as follows:

$$\Sigma_M = \begin{bmatrix} 0.001 & 0 \\ 0 & 0.0005 \end{bmatrix}$$

$$\Sigma_{x_0} = 0.0001 I_3.$$

The covariance matrix for the static obstacles is

$$\Sigma_S = 0.1 I_2.$$

The gain for the soft constraint is $k = -1.5$ and the chance constraint parameter is $\Gamma = 0.25$. Figure 4.5 shows the resulting path attaining the shortest distance (solid red) and expanded nodes (solid orange) for the ASR CCxHA* algorithm. It is shown that the algorithm pruned all nodes that were too close to the obstacles, and the final path attempted to keep as much distance as possible from all obstacles. The resulting path attaining the shortest distance (solid red) using the CCRRT algorithm and a path with “zig-zag” motions (dash-dotted blue) using the CCxRRT algorithm are shown in Figure 4.6. The path segments with “zig-zag” motions are emphasized with black circles. The paths for all algorithms are shown in Figure 4.7. Both the A* and RRT algorithms move close to obstacles to generate short distance paths. The conventional HA* algorithm, which did not consider the robot shape, moved close to the obstacles while CCHA* moved very conservatively, although it did not consider the shape of the robot. The ASR CCHA* was not able to find a solution (not shown in the figure) because it pruned all nodes that were considered “too dangerous”. This result was derived from the number of obstacles, where the allowed probability of collision per obstacle (see (4.30)), i.e., $\gamma \approx 0.028$ was very conservative in terms of satisfying Boole’s inequality. The CCxHA* and ASR CCxHA* algorithms were able to find a solution that guaranteed $\Delta_t(\mathbf{x}_t) \leq \Gamma$ for $\forall t$. Furthermore, the SCHA* and ASR SCHA* algorithms found robust solutions without pruning

any nodes. The HA* algorithms move the robot in reverse only if the forward motion violates the collision constraints, whereas the CLRRT-based algorithms take a backward motion based on the position of the randomly expanded node. Therefore, the best solution of the CLRRT-based algorithms took a different route than the other algorithms. Table 4.1 shows the performance of the algorithms. The probabilistic approaches required more time to check for collisions than the conventional HA* and CLRRT algorithms. CCHA* and CCRRT use the same method for collision check and have, therefore, the same collision check computation time per node. Accordingly, the collision check time per node is the same for CCxHA* and CCxRRT. The CCHA* algorithm, which used the inverse Gaussian error function, required approximately 10 times more time per node than the HA* algorithm, while the algorithms that employed a Gaussian error function (CCxHA*, ASR CCxHA*, SCHA* and ASR SCHA*) needed slightly more; this was because they calculated the Gaussian error function for each line segment. Additional time per node was needed if the shape of the robot was considered. It is shown that a trade-off occurred between computation time and robustness. The RRT-based planners were able to find paths that are shorter in distance than the HA*-based algorithms, but the mean distance of all trials was larger than the distance of any HA*-based planner. Furthermore, only 9.5% of the paths of CCRRT were successful in finding a path within the given time frame. The mean computation time of CCRRT and CCxRRT is much larger than that of CCHA* and CCxHA*. In addition, the standard deviation of the distance and computation time is large for the CCRRT and CCxRRT path planners. The HeAT-RT algorithm expanded fewer nodes than most other planners but generated a path near an obstacle and had a larger mean computation time compared to the proposed probabilistic algorithm CCxHA*. In addition, the HeAT-RT had the largest standard deviation of the distance.

The best performance of the proposed algorithms in the static environment in terms of travelling distance and computation time are provided by the CCxHA* and the ASR SCHA*.

4.4.1.2 Results in a dynamic environment

In this section, the algorithms are evaluated in a dynamic environment. The dynamic obstacle had a square size of 0.5 m with linear velocity $v = 0.3536$ m/s and a heading direction of 135° . The initial covariance matrix and process noise of the dynamic obstacle are given as follows:

$$\Sigma_{x_{\mathcal{D}_0}} = 0.001I_4$$

$$\Sigma_{\mathcal{D}} = \begin{bmatrix} 0.001 & 0 & 0 & 0 \\ 0 & 0.001 & 0 & 0 \\ 0 & 0 & 0.0001 & 0 \\ 0 & 0 & 0 & 0.0001 \end{bmatrix}.$$

The covariance matrix of the static obstacles was the same as in the static environment, and elements of the covariance matrix and the process noise of the mobile robot were increased for

Table 4.1: Traveled distance and computation time per node in static environment.

Algorithm	Shortest distance [m]	Collision check time per node [μ s]	Expanded nodes (best solution)	Computation time [s]	Distance [m]	Successful nodes [%]
CCHA*	23.70	~ 149	2620	~ 1.21	—	—
CCxHA*	20.51	~ 240	408	~ 0.22	—	—
SCHA*	20.98	~ 222	1333	~ 0.592	—	—
ASR CCHA*	x	x	x	x	—	—
ASR CCxHA*	22.83	~ 898	5086	~ 13.84	—	—
ASR SCHA*	22.15	~ 885	2022	~ 3.5	—	—
A*	22.63	~ 15	509	~ 0.019	—	—
HA*	20.19	~ 15	334	~ 0.02	—	—
ASR HA*	21.40	~ 67	580	~ 0.11	—	—
RRT	17.70	~ 15	191	$\sim 0.022 \pm 0.02$	24.04 ± 3.49	100
CLRRT	18.64	~ 15	1720	$\sim 0.57 \pm 0.74$	28.25 ± 4.82	100
CCRRT	20.47	~ 149	10628	$\sim 8.40 \pm 5.64$	30.78 ± 4.32	9.5
CCxRRT	19.25	~ 240	10242	$\sim 2.65 \pm 3.02$	28.39 ± 4.71	97.5
HeAT-RT	18.90	~ 15	288	$\sim 0.32 \pm 0.51$	26.51 ± 5.39	100

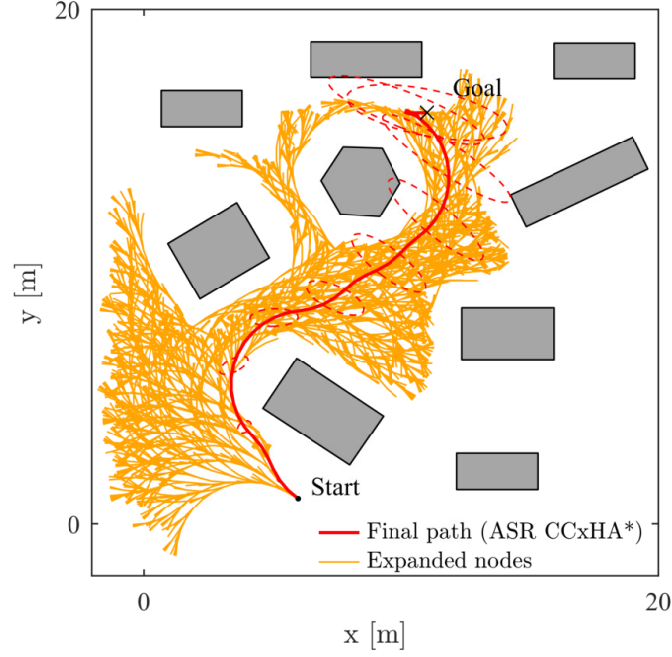


Figure 4.5: The resulting path (solid red) and all expanded nodes (solid orange) using the ASR CCxHA* algorithm (the dashed ellipses are plotted every 5th node with a confidence of 95 %).

this environment, where $\Sigma_{x_0} = 0.01I_3$ and the process noise is given as follows:

$$\Sigma_M = \begin{bmatrix} 0.01 & 0 \\ 0 & 0.005 \end{bmatrix}.$$

The parameters for the chance constraints and the soft constraint are $\Gamma = 0.4$ and $k = -1.5$, respectively.

Results are shown in Figures 4.8 and 4.9, where the former shows the result of the ASR CCHA* algorithm at a selected node. The robot attempted to cross the path of the dynamic obstacle behind it while considering its own uncertainty and that of the dynamic obstacle. Figure 4.9 shows the results for all the algorithms. Most HA* algorithm variants crossed the path of the dynamic obstacle behind it, except for the HA* algorithm, the CCxHA* algorithm, which did not consider the robot shape and the ASR SCHA* algorithm, which conservatively crossed the path of the dynamic obstacle ahead of it. Table 4.2 shows the results for each of the planner. It is shown that the SCHA* variations expanded many nodes and therefore had a larger computation time. Due to the dynamic obstacle, the soft constraint part increased, which will lead to the expansion of many nodes. Furthermore, most of the algorithms had a similar traveled distance as the HA* and ASR HA* algorithms. The CCxRRT algorithm had a similar computation time as the CCxHA*, but a larger mean distance with a large standard deviation. The CCRRT algorithm had the second-largest mean computation time, which is more than two times larger as for the ASR CCHA*. Both the A* and RRT moved, as in the

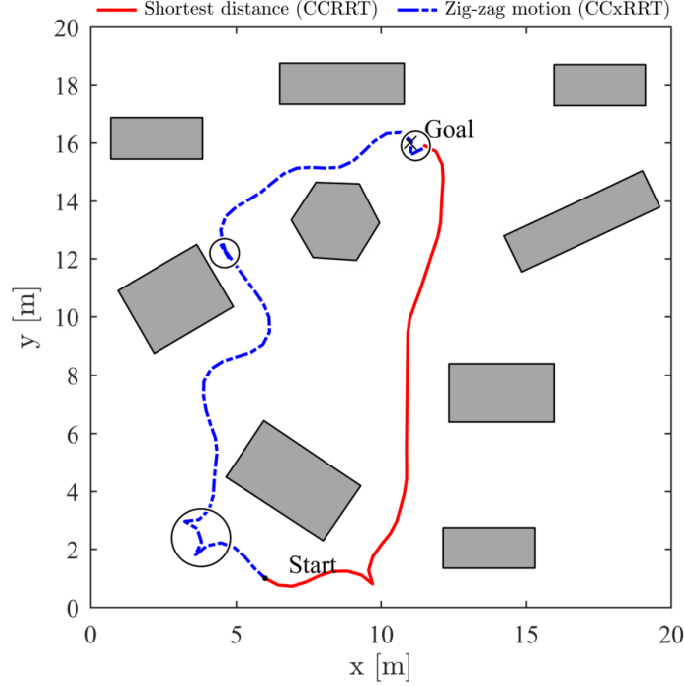


Figure 4.6: The resulting path attaining the shortest distance (solid red) using the CCRRT algorithm and a “zig-zag” motion (dash-dotted blue) using the CCxRRT algorithm (the black circles emphasize the path segments with a “zig-zag” motion).

static environment, very close to the obstacle. Furthermore, the HeAT-RT expanded the fewest nodes but had the largest standard deviation of the distance and a larger mean computation time compared to the proposed CCHA*, CCxHA* and SCHA* algorithms.

The CCxHA* and ASR CCxHA* performed best among the proposed algorithms in terms of computation time and travelling distance.

4.4.1.3 Results for different soft constraint gains

This section compares the result of the HA* algorithm with those of the SCHA* algorithm using different k gains for the soft constraint. Figure 4.10 shows the resulting paths, where the conservatism of the SCHA* algorithm increases with a decreasing gain value for k . Furthermore, Table 3 shows that the number of expanded nodes, the traveled distance and the computation time also increases. For $k = -0.1$, the traveled distance is less than for the HA* algorithm (see Table 4.1); however the SCHA* algorithm with $k = -0.1$ moves close to the obstacles. The traveled distance of the SCHA* algorithm with $k = -0.5$ and $k = -1.5$ is similar to the traveled distance of HA*. Based on these results, the gain value of the soft constraint should be set between -0.5 and -1.5 to achieve a good traveling distance, computation time, and safety.

Table 4.2: The traveled distance and computation time per node in a dynamic environment.

Algorithm	Shortest distance [m]	Collision check time per node [μ s]	Expanded nodes (best solution)	Computation time [s]	Distance [m]	Successful nodes [%]
CCHA*	18.75	~ 179	745	~ 0.39	—	—
CCxHA*	18.25	~ 224	550	~ 0.3	—	—
SCHA*	18.75	~ 212	2064	~ 0.9	—	—
ASR CCHA*	20.35	~ 410	1063	~ 1.32	—	—
ASR CCxHA*	18.36	~ 637	671	~ 1.08	—	—
ASR SCHA*	18.84	~ 589	4138	~ 4.9	—	—
A*	18.38	~ 14	447	~ 0.011	—	—
HA*	17.97	~ 14	309	~ 0.02	—	—
ASR HA*	18.25	~ 56	584	~ 0.11	—	—
RRT	17.36	~ 14	207	$\sim 0.016 \pm 0.005$	20.28 ± 2.33	100
CLRRT	17.46	~ 14	699	$\sim 0.27 \pm 0.57$	20.31 ± 2.68	100
CCRRT	18.85	~ 179	4653	$\sim 2.81 \pm 3.65$	26.19 ± 4.44	93.9
CCxRRT	17.60	~ 224	788	$\sim 0.55 \pm 0.71$	$21.79.52$	100
HeAT-RT	19.05	~ 14	72	$\sim 0.96 \pm 0.51$	25.53 ± 4.73	99.9

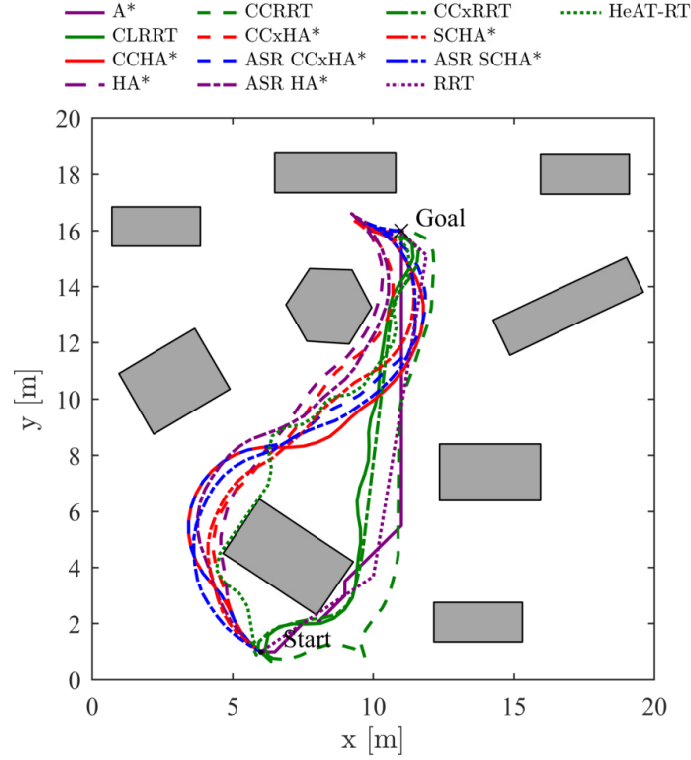


Figure 4.7: Resulting paths for all algorithms in a clustered, static environment.

4.4.1.4 Performance Comparison

This work adapted the MCS algorithm in [62], which was modified for dynamic obstacles and robots with nonlinear dynamics, where the robot state was updated at each expanded node for the motion command of the final path. Algorithm 4 shows a pseudo-code of the MCS algorithm. Variable $\overline{p_{\text{coll}}}(\mathcal{R}, \cdot)$ denotes the probabilities for robot \mathcal{R} to not collide with the tested obstacle. $\mathcal{S}^i, \mathcal{D}^j$ denote static obstacle i and dynamic obstacle j , respectively. $\mathcal{A}_{\mathcal{R}}, \mathcal{A}_{\mathcal{S}^i}$ and $\mathcal{A}_{\mathcal{D}^j}$ denote the area of the robot \mathcal{R} , static obstacle i and dynamic obstacle j , respectively. \mathcal{O} is the set of all obstacles and $n_{\mathcal{S}}, n_{\mathcal{D}}$ is the number of static and dynamic obstacles, respectively. Function $\text{randc}()$ calculates the Gaussian random variable. This work used $P = 1000$ particles for the MCS, and the probability of collision was calculated based on the shape of the robot. The results of the static and dynamic environments are shown in Figure 4.17. The red lines indicate the MCS results for the proposed algorithms assuming a point robot, the blue lines indicate the algorithms that consider the shape of the robot, and the green and purple lines show the results of the planners used for comparison, except A* and RRT, which cannot be evaluated using the MCS since they do not use a robot model. The probability of collision was significantly reduced for the static environment. Only CCxRRT and the proposed CCxHA* had high values, likely because they did not consider the shape of the robot and allowed a probability of collision of $\Delta_t(\mathbf{x}_t) \leq 0.25$, but CCxHA* was still $\sim 31\%$ better than CCxRRT. The result of the dynamic environment was similar. In both environments, the CCHA* algorithm variation indicated

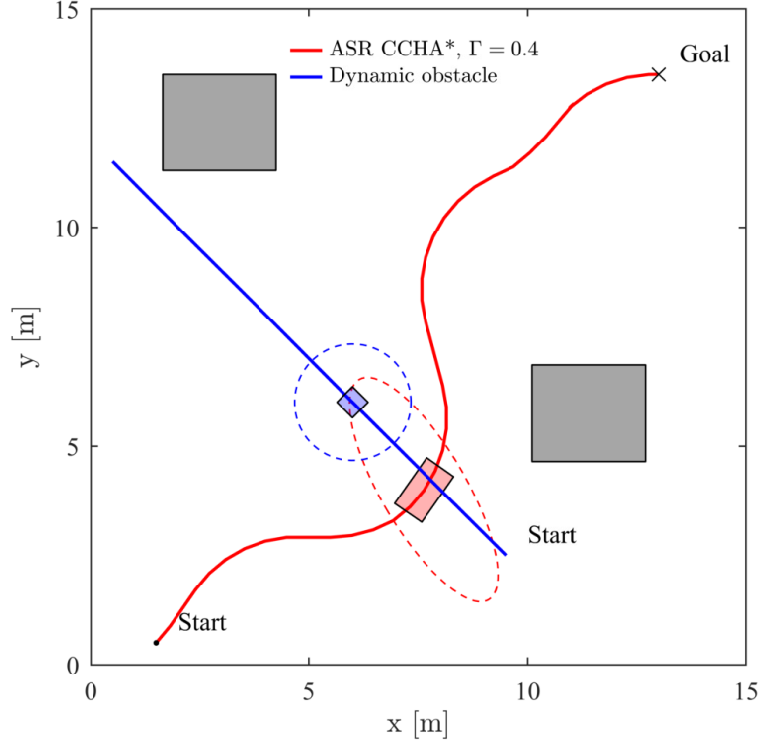


Figure 4.8: The ASR CCHA* algorithm result at a selected node in the dynamic environment.

almost no probability of collision while the CCxHA* algorithm had a small value based on the setting of Γ . ASR CCxHA* improved the probability of collision by $\sim 54\%$ compared to the deterministic planner ASR HA*. It can be seen that the algorithms that do not consider uncertainties had the largest probability of collision. Furthermore, most ASR variations (blue lines) of the proposed algorithms have a very low probability (close to 0%) of collision in both the static and dynamic environments.

The performance of the algorithms was evaluated using a clustered, static environment, and an environment with a dynamic obstacle. In addition, the true probability of a collision was calculated using the MCS.

The results showed that the proposed algorithms were able to find probabilistically safe paths within the set threshold, i.e., Γ . Furthermore, the CCHA* algorithm was very conservative, thus exhibited a very low probability of collision (Figure 4.17). However, with an increase in the number of obstacles, the CCHA* algorithm may fail to find a solution due to the pruning of all expanded nodes. The CCxHA* algorithm showed good results with a small increase in the computation time per node compared with the CCHA* algorithm (Tables 4.1 and 4.2). All proposed algorithms were able to avoid the dynamic obstacle with a decreased probability of collision (Table 4.2). The method considering arbitrary-shaped mobile robots further increased probabilistic safety at the cost of additional computation time per node. Therefore, the ASR variations of CCHA*, CCxHA*, and SCHA* can be used as global approaches for predicting

Algorithm 4 Monte Carlo Simulation

```

1: procedure PROBABILISTICCOLLISIONCHECK
2:   for all Nodes do
3:      $reset(p_{coll})$ 
4:     for  $j \leftarrow n_S$  do
5:       for  $i \leftarrow P$  do
6:          $\hat{\mathbf{u}} \leftarrow \mathbf{u} + randc(0, \Sigma_M)$ 
7:          $\mathbf{x}_{t+1} = f(\mathbf{x}_t, \hat{\mathbf{u}})$ 
8:          $\mathbf{x}_{S^j} = \mathbf{x}_{S_0^j} + randc(0, \Sigma_S)$ 
9:          $\mathcal{S}^j \leftarrow \mathbf{x}_{S^j}$ 
10:        if  $\mathcal{A}_{\mathcal{R}} \cap \mathcal{A}_{S^j} = \emptyset$  then
11:           $\overline{p_{coll}(\mathcal{R}, \mathcal{S}^j)} \leftarrow \overline{p_{coll}(\mathcal{R}, \mathcal{S}^j)} + 1$ 
12:        end if
13:      end for
14:    end for
15:    for  $j \leftarrow n_D$  do
16:      for  $i \leftarrow P$  do
17:         $\hat{\mathbf{u}} \leftarrow \mathbf{u} + randc(0, \Sigma_M)$ 
18:         $\mathbf{x}_{t+1} = f(\mathbf{x}_t, \hat{\mathbf{u}})$ 
19:         $\mathbf{x}_{D_{t+1}^j} = \boldsymbol{\mu}_{\mathbf{x}_{D_{t+1}^j}} + randc(0, \Sigma_{D^j})$ 
20:         $\mathcal{D}^j \leftarrow \mathbf{x}_{D_{t+1}^j}$ 
21:        if  $\mathcal{A}_{\mathcal{R}} \cap \mathcal{A}_{D^j} = \emptyset$  then
22:           $\overline{p_{coll}(\mathcal{R}, \mathcal{D}^j)} \leftarrow \overline{p_{coll}(\mathcal{R}, \mathcal{D}^j)} + 1$ 
23:        end if
24:      end for
25:    end for
26:     $\overline{p_{coll}(\mathcal{R}, \mathcal{O})} \leftarrow 1$ 
27:    for  $k \leftarrow 1$  to  $n_S$  do
28:       $\overline{p_{coll}(\mathcal{R}, \mathcal{O})} \leftarrow \overline{p_{coll}(\mathcal{R}, \mathcal{O})} \cdot \overline{p_{coll}(\mathcal{R}, \mathcal{S}^k)}$ 
29:    end for
30:    for  $k \leftarrow 1$  to  $n_D$  do
31:       $\overline{p_{coll}(\mathcal{R}, \mathcal{O})} \leftarrow \overline{p_{coll}(\mathcal{R}, \mathcal{O})} \cdot \overline{p_{coll}(\mathcal{R}, \mathcal{D}^k)}$ 
32:    end for
33:     $p_{coll}(\mathcal{R}, \mathcal{O}) \leftarrow 1 - \frac{\overline{p_{coll}(\mathcal{R}, \mathcal{O})}}{P^{n_{\mathcal{O}}}}$ 
34:     $store(p_{coll}(\mathcal{R}, \mathcal{O}))$ 
35:  end for
36: end procedure

```

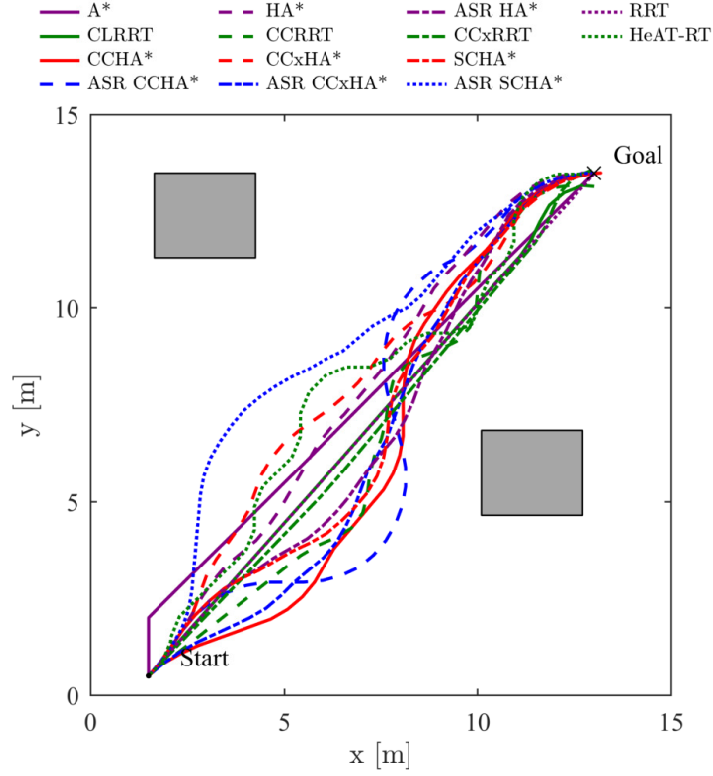


Figure 4.9: Resulting paths in the dynamic environment.

an initial probabilistically safe path.

The proposed soft constraint approach successfully found probabilistic robust paths for both environments. However, computation time and number of expanded nodes was large for the dynamic environment, which indicates that the heuristics approach should be extended to consider dynamic obstacles. The ASR SCHA* outperformed both ASR CCHA* and ASR CCxHA* in the static environment. The ASR CCHA* failed to find a solution and the ASR CCxHA* had a large computation time (Table 4.1). Moreover, results indicated that the gain values for the SCHA* algorithm should be in the range of $k = -0.5$ to -1.5 to achieve good results for safety and traveled distance (Table 4.3).

CCHA* and CCxHA* outperformed CCRRT and CCxRRT in terms of mean computation time and mean traveled distance (Tables 4.1 and 4.2). In addition, the results of the HA* algorithms are deterministic, whereas the results of the RRT-based planners are random. The shorter computation time of the proposed HA*-based planners is the result of two factors. First, the RRT-based planners compute a new motion command for each node, whereas the HA*-based planners use a fixed motion command that can be stored beforehand in a lookup table. Second, HA*-based planners store each expanded node in a list, and the node with the lowest total cost (heuristic cost plus cost-so-far) is selected for expansion. Such a procedure requires a heap algorithm; here, binary heap [118] is used, which has an average time complexity for inserting a value of $\mathcal{O}(\log n)$ and extracting the minimum value of $\mathcal{O}(\log n)$, where

Table 4.3: Traveled distance and computation time per node in a static environment for the SCHA* algorithm.

Soft constraint k	Distance [m]	Expanded nodes	Comp. time [s]
-0.1	19.82	400	~ 0.174
-0.5	20.62	813	~ 0.406
-1.5	20.81	1333	~ 0.592
-3.0	21.34	1846	~ 0.936
-5.0	22.01	1713	~ 0.794
-10.0	21.88	1917	~ 0.994
-100.0	23.10	5499	~ 3.176

n is the number of nodes in the list. Meanwhile, the RRT-based approaches use a list for all expanded nodes, where the cost of each node has to be updated based on the newly expanded node, which requires $\mathcal{O}(n)$ time. In CLRRT, CCRRT and CCxRRT the updated list has to be sorted, which takes $\mathcal{O}(n \log n)$ time [89].

The larger mean distance of the RRT-based planners is a result of detours and “zig-zag” motions (Figure 4.6) caused by random selection of new nodes in the configuration space [75]. In contrast the proposed algorithms are guided by heuristics, therefore expand nodes close to the goal, and change motion directions only if a collision constraint is violated, hence avoiding “zig-zag” motions (Figure 4.5).

The heuristics approach used for the proposed algorithms do not consider dynamic obstacles, which result in a large number of node expansions. Therefore, future studies should extend the algorithms with heuristics considering dynamic obstacles to reduce the number of expanded nodes and computation time.

4.5 Probabilistic robust path planning using the χ^2 -distribution

In the previous section the probability of collision is calculated using the Gaussian error function or the inverse Gaussian error function. Calculating the probability of collision using the former can be computationally expensive (depends on the number of obstacles) on the other hand, planners using the inverse Gaussian error function (i.e., CCHA*) are very conservative and may not find a solution. This section introduces an approach that uses the χ^2 -distribution to detect a collision with an obstacle under state and environment uncertainties. Confidence ellipses generated with the χ^2 -distribution are used to get an area of possible robot states for each time instant t . The area of the confidence ellipse is enclosed with two circles with equal radius. Collision occurs if one of the circles overlaps with an obstacle. The following explains the approach in detail with comparisons with the proposed planners of the previous section.

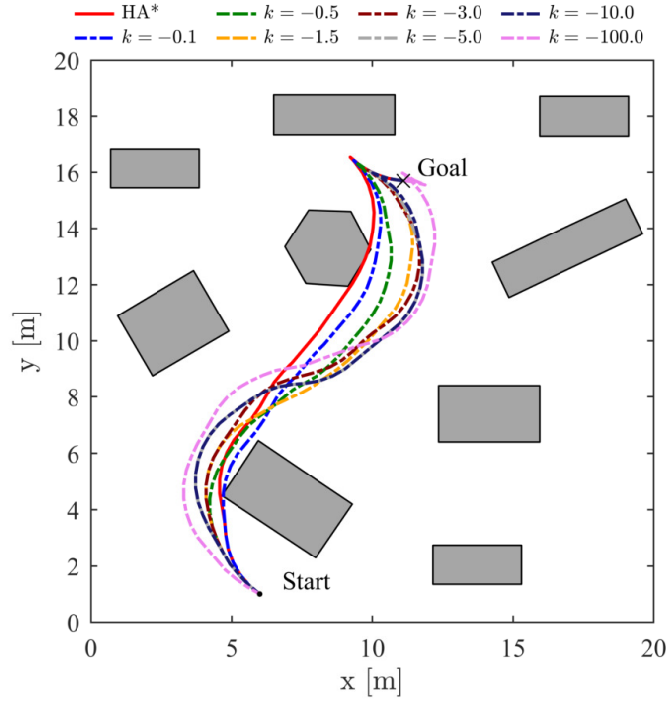


Figure 4.10: Results for different gains of the SCHA* algorithm.

4.5.1 Generation of confidence ellipses

A not rotated ellipse at the origin is described by

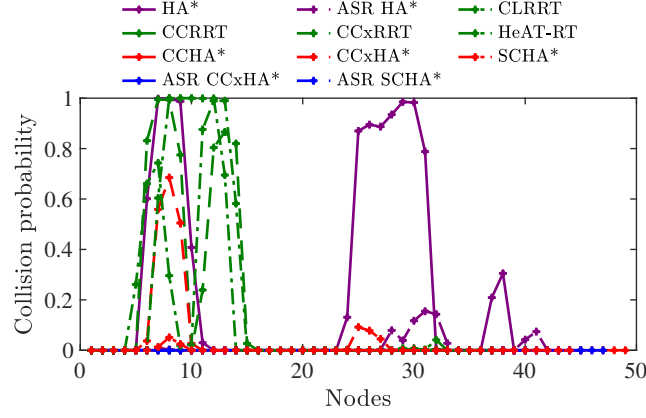
$$\left(\frac{x}{r_x}\right)^2 + \left(\frac{y}{r_y}\right)^2 = 1, \quad (4.44)$$

where r_x and r_y are the radii of the ellipse in x and y -direction, respectively. The confidence ellipse of an uncorrelated Gaussian distribution of the form

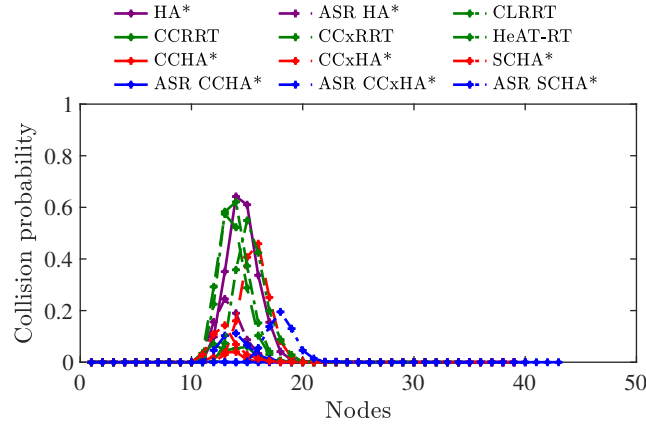
$$\Sigma = \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix} \quad (4.45)$$

is expressed with the following

$$\left(\frac{x}{\sigma_x}\right)^2 + \left(\frac{y}{\sigma_y}\right)^2 = s, \quad (4.46)$$



(a) Collision probability for the static environment.



(b) Collision probability for the dynamic environment.

Figure 4.11: Monte Carlo simulation for static and dynamic environments.

where s denotes the scale factor of the confidence ellipse. It can be seen that (4.46) follows a χ^2 -distribution. The χ^2 -distribution is defined as the sum of squares of Gaussian distributions

$$Q = \sum_{i=1}^{k_{\text{DOF}}} X_i^2, \quad (4.47)$$

where k_{DOF} is the number of degrees of freedom, X_i is a Gaussian distributed random variable and Q is a χ^2 distributed random variable. Therefore, $s \sim \chi_2^2$ in (4.46) is a χ^2 -distributed random variable with two degrees of freedom. The variable s is calculated using the selected confidence p_{conf} for the confidence ellipse. The following equation

$$p(s < k_{\text{th}}) = p_{\text{conf}} \quad (4.48)$$

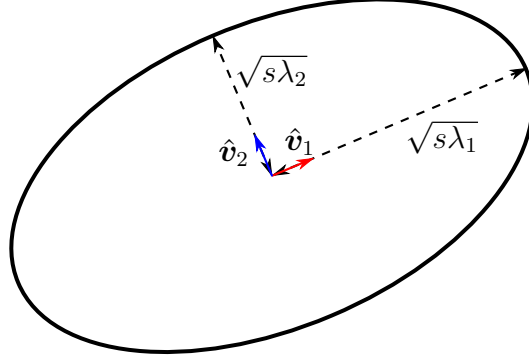


Figure 4.12: Confidence ellipse of a correlated covariance matrix.

with the threshold value k_{th} has to be satisfied. Since the χ^2 -distribution has two degrees of freedom s can be calculated with

$$s = -2 \ln(1 - p_{conf}). \quad (4.49)$$

The area of the resulting confidence ellipse contains all Gaussian distributed data points of Σ up to a confidence of p_{conf} .

Generally for robotic systems the Gaussian random variables are correlated, hence the covariance matrix is expressed as follows

$$\Sigma = \begin{bmatrix} \sigma_x^2 & \sigma_{x,y} \\ \sigma_{x,y} & \sigma_y^2 \end{bmatrix}. \quad (4.50)$$

(4.46) can still be used but has to be adjusted for $\sigma_{x,y}$. Now the eigenvalues λ_1, λ_2 are used for the radii, hence the adjusted equation is as follows

$$\left(\frac{x}{\lambda_1}\right)^2 + \left(\frac{y}{\lambda_2}\right)^2 = s. \quad (4.51)$$

Furthermore, the normalized eigenvectors \hat{v}_1, \hat{v}_2 represent the rotated coordinate system for the confidence ellipse. Since a covariance matrix is always positive definite the eigenvalues and eigenvectors can be calculated with

$$\lambda_{1,2} = \frac{1}{2} \left(\sigma_x + \sigma_y \pm \sqrt{(\sigma_x - \sigma_y)^2 + 4\sigma_{x,y}^2} \right) \quad (4.52)$$

$$\mathbf{v}_{1,2} = \begin{bmatrix} \frac{-\sigma_y + \lambda_{1,2}}{\sigma_{x,y}} & 1 \end{bmatrix}^\top \quad (4.53)$$

$$\hat{\mathbf{v}}_{1,2} = \frac{\mathbf{v}_{1,2}}{\|\mathbf{v}_{1,2}\|_2}. \quad (4.54)$$

Figure 4.12 shows a generated confidence ellipse for a correlated covariance matrix Σ .

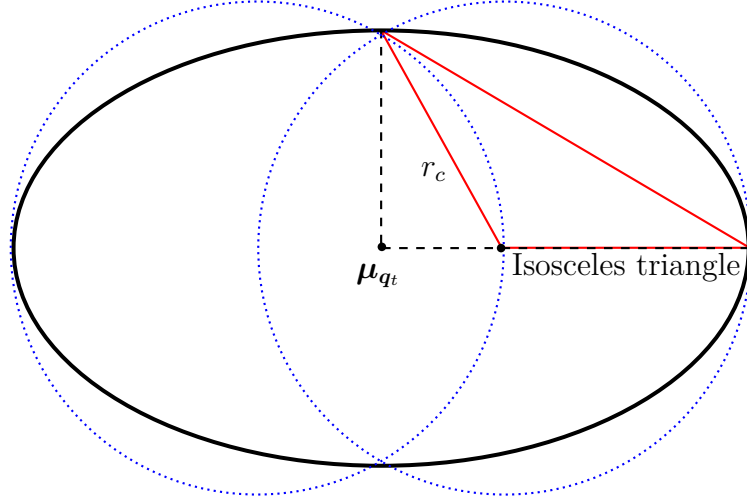


Figure 4.13: Confidence ellipse with the isosceles triangle and the two circles.

4.5.2 Collision detection

In this approach collision occurs if an obstacle enters the area of the confidence ellipse. Therefore, the same approach as shown in Figure 4.3 can be used to cover the confidence ellipse. Here, two circles of equal radius are selected to cover the confidence ellipse along the larger eigenvalue. It can be seen that the smallest possible radius of the circles equals the length of the equally sized sides of an isosceles triangle, where the hypotenuse is the connection of the intersection points of the confidence ellipse with the lines along the eigenvectors $\hat{\mathbf{v}}_1, \hat{\mathbf{v}}_2$. Figure 4.13 shows a confidence ellipse with the isosceles triangle and the resulting circles. Using trigonometric equations the radius of the circles is as follows

$$r_c = \frac{s(\lambda_1 + \lambda_2)}{2\sqrt{s\lambda_{\max}}}, \quad (4.55)$$

where r_c is the radius of the circles. The center of the circles are calculated using the eigenvector $\hat{\mathbf{v}}_{\max}$ of the larger eigenvalue λ_{\max} together with the corresponding radius of the confidence ellipse

$$\mathbf{c}_{1,2} = \boldsymbol{\mu}_{q_t} \pm \left(\sqrt{s\lambda_{\max}} - r_c \right) \hat{\mathbf{v}}_{\max}, \quad (4.56)$$

where $\mathbf{c}_{1,2}$ are the coordinates of the centers of both circles and $\boldsymbol{\mu}_{q_t}$ is the mean position of the mobile robot and therefore the center of the confidence ellipse. Figure 4.13 shows a confidence ellipse covered by two circles of equal radius. After obtaining the centers of the circles and the radius the collision check can be done using (4.1). The equations for each obstacle have to be adjusted for the circle radius. Hence, collision occurs if the following conjunction of inequalities

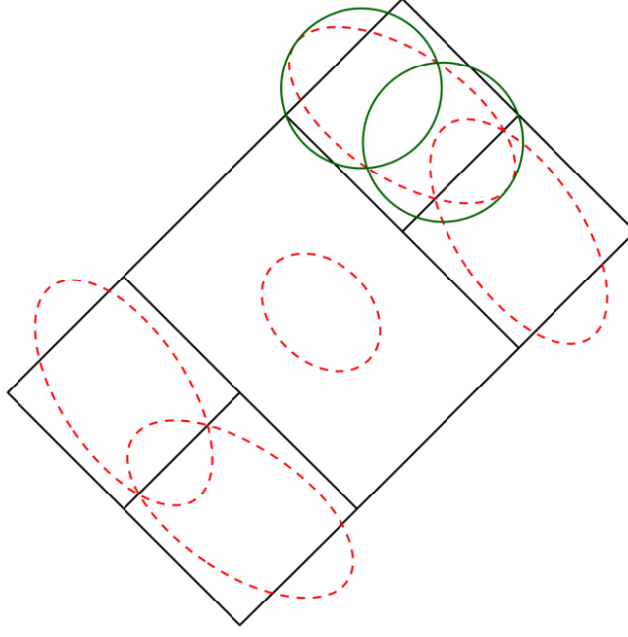


Figure 4.14: 95 % confidence ellipses with circles covering the confidence ellipse of the front left square segment.

for an inflated obstacle is satisfied

$$\bigwedge_{i=1}^{n_e} \mathbf{a}_i^\top (\boldsymbol{\mu}_{q_t} - \mathbf{x}_{p_i} + r_c \mathbf{a}_i) < 0, \quad (4.57)$$

where \mathbf{x}_{p_i} is a point on line segment i . The current state $\boldsymbol{\mu}_{q_t}$ of the mobile robot is collision free if the following conjunction for all obstacles is satisfied

$$\bigwedge_{j=1}^{n_o} \left(\bigvee_{i=1}^{n_{o_j}} \mathbf{a}_{ji}^\top (\boldsymbol{\mu}_{q_t} - \mathbf{x}_{p_{jit}} + r_c \mathbf{a}_{ji}) \geq 0 \right), \quad (4.58)$$

where the parameter $\mathbf{x}_{p_{jit}}$ is time dependent to accommodate for dynamic obstacles. The planners that use the confidence ellipses to generate collision free paths are named confidence ellipse hybrid A* (CEHA*).

4.5.3 Extension for arbitrary-shaped robots

The approach can be extended for arbitrary-shaped mobile robot by covering the confidence ellipse of each square and rectangle segment (Figure 4.4) of the robot with two circles. An example using the same covariance matrix as for Figure 4.4 is given in Figure 4.14. The path

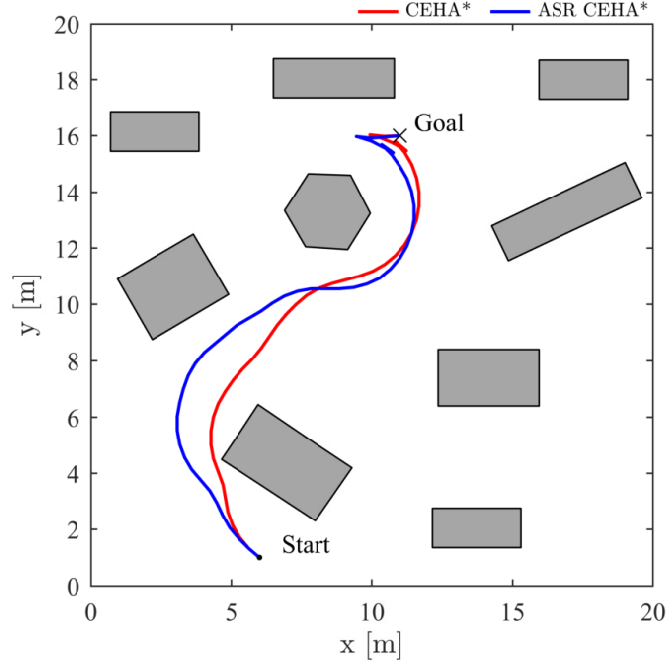


Figure 4.15: Resulting paths of the planners in the static environment.

of the ASR is collision free if the following conjunction of inequality constraints is satisfied

$$\bigwedge_{j=1}^{n_{\mathcal{O}}} \left[\bigwedge_{k=1}^{n_{\text{Seg}}} \left(\bigvee_{i=1}^{n_{\mathcal{O}_j}} \mathbf{a}_{ji}^{\top} (\boldsymbol{\mu}_{q_t} - \mathbf{x}_{p_{jit}} + (r_{c_k} + r_k) \mathbf{a}_{ji}) \geq 0 \right) \right], \quad (4.59)$$

where n_{Seg} is the number of segments of the robot, r_{c_k} is the radius of the circles covering the confidence ellipse of segment k , and r_k is the inflation radius of segment k . Here the inflation radii are r_{sq} and r_{rec} for the square segment and rectangle segment of the robot as seen in Figure 4.3. The planners that consider the shape of the robot and use the confidence ellipses are named arbitrary-shaped mobile robot confidence ellipse hybrid A* (ASR CEHA*).

4.5.4 Results and discussion

The proposed algorithms using confidence ellipse to find probabilistic robust paths are compared in the following in the same static and dynamic environments as in the previous section. The best performing algorithms in terms of computation time and probabilistic robustness from the previous section, namely CCxHA*, ASR CCxHA*, ASR SCHA*, ASR HA*, and HeAT-RT are selected for comparison. The same process noise and environment uncertainties as in the previous section or chosen for the path generation. The covariance matrix used to generated the confidence ellipses is the sum of the covariance matrix of the robots or square segment position $\Sigma_{\{q_t, q_t^j\}}$ and the covariance matrix of the position of the respective static or dynamic

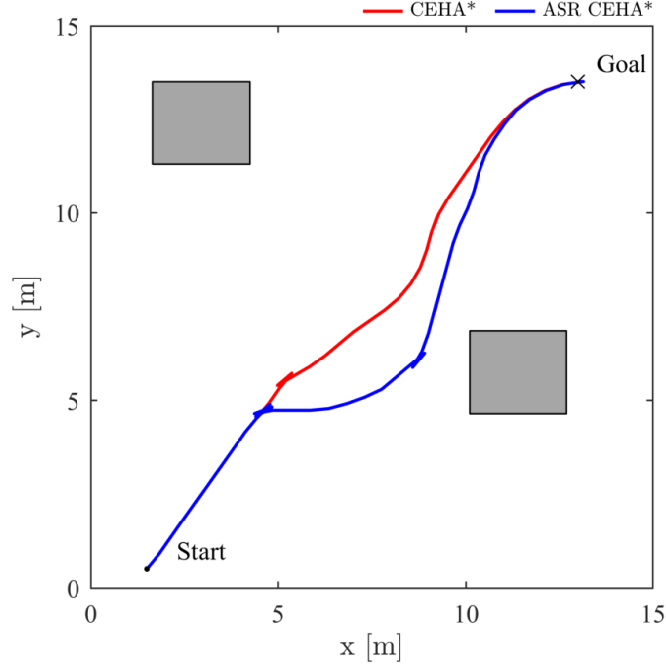


Figure 4.16: Resulting paths of the planners in the dynamic environment.

obstacle $\Sigma_{\mathcal{O}}$

$$\Sigma_{\text{total}} = \Sigma_{\{q_t, q_t^j\}} + \Sigma_{\mathcal{O}}, \quad (4.60)$$

where Σ_{total} is the covariance matrix of the sum. Furthermore, the confidence value was set to $p_{\text{conf}} = 0.1$ for CEHA* and $p_{\text{conf}} = 0.02$ for ASR CEHA*. Finally, the performance is evaluated using MCS.

The results for the static and dynamic environment are shown in Figures 4.15 and 4.16, respectively. It can be seen that the ASR CEHA* is more conservative than the CEHA* even though the confidence value is much smaller. Furthermore, both planners moved forward and backwards on the spot in the dynamic environment until the dynamic obstacle passed the mobile robot. Table 4.4 shows the comparison with other planners. Among all the planners is CEHA* the fastest with a computation time of ~ 0.069 s in the static environment and ~ 0.032 s in the dynamic environment. CEHA* is more than 3 times faster than the second fastest probabilistic robust planner CCxHA* in the static environment and more than 9 times faster in the dynamic environment. In the dynamic environment the confidence ellipse approach considering the shape of the robot has a smaller computation time than the CCxHA* planner which does not consider the shape of the robot. Furthermore, ASR CEHA* is almost as fast as the deterministic planner ASR HA*. The travelling distance of the proposed planners is larger compared to the other planners, especially ASR CEHA* has a larger travelling distance compared to other ASR approaches.

The MCS shows that both planners are able to reduce the probability of collision. But the

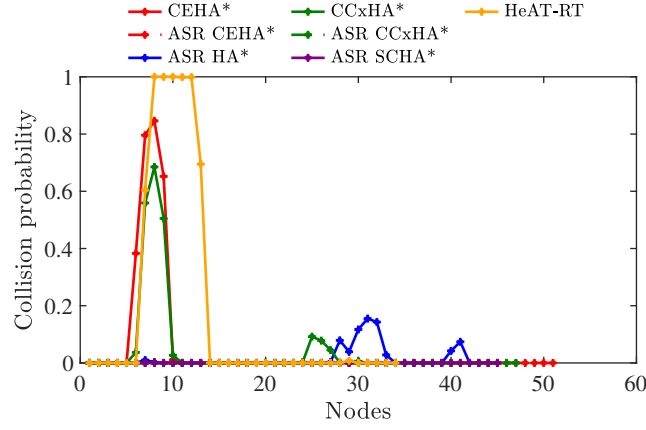
Table 4.4: The traveled distance and computation time per node in the static and dynamic environments compared with selected planners from the previous section (the last two columns are omitted for brevity).

Algorithm	Shortest distance [m]	Collision check time per node [μ s]	Expanded nodes (best solution)	Computation time [s]
Static environment				
CEHA*	21.99	~ 153	174	~ 0.069
ASR CEHA*	24.49	~ 448	424	~ 0.48
CCxHA*	20.51	~ 240	408	~ 0.22
ASR CCxHA*	22.83	~ 898	5086	~ 13.84
ASR SCHA*	22.15	~ 885	2022	~ 3.5
ASR HA*	21.40	~ 67	580	~ 0.11
HeAT-RT	18.90	~ 15	288	$\sim 0.32 \pm 0.51$
Dynamic environment				
CEHA*	19.25	~ 173	103	~ 0.032
ASR CEHA*	22.25	~ 360	140	~ 0.119
CCxHA*	18.25	~ 224	550	~ 0.3
ASR CCxHA*	18.36	~ 637	671	~ 1.08
ASR SCHA*	18.84	~ 589	4138	~ 4.9
ASR HA*	18.25	~ 56	584	~ 0.11
HeAT-RT	19.05	~ 14	72	$\sim 0.96 \pm 0.51$

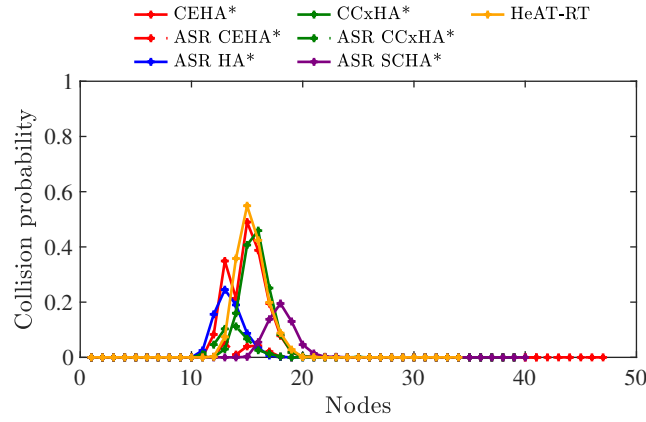
CEHA* planner has a larger probability of collision compared to the other probabilistic robust planners. On the other hand, the ASR CEHA* has almost no probability of collision (close to 0%) in both the static and dynamic environments. Furthermore, CEHA* and CCxHA* have a similar probability of collision in the dynamic environment but CEHA* is more than 9 times faster in generating the path. CEHA* improved the probability of collision by $\sim 15\%$ in the static environment compared to HeAT-RT with a faster computation time.

In this section a new planning approach for probabilistic robustness using confidence ellipses was proposed. The planners were compared with other probabilistic robust planners in a clustered, static environment and a dynamic environment. The true probability of collision was calculated using the MCS.

Results showed that the planners were able to find probabilistically safe paths in both the static and dynamic environments (Figures 4.15 and 4.16). Furthermore, the ASR variation generated very conservative paths, thus the paths had a very low probability of collision. The planners outperformed other probabilistic robust algorithms in terms of computation time with a similar probability of collision. However, setting the p_{conf} parameter is not intuitive compared to setting the parameter Γ of the previous section.



(a) Collision probability of the confidence ellipse planners for the static environment.



(b) Collision probability of the confidence ellipse planners for the dynamic environment.

Figure 4.17: Monte Carlo simulation for static and dynamic environments of the confidence ellipse planners and selected planners of the previous section.

Even though the computation time of the collision check is faster for the confidence ellipse approach compared to the planners of the previous section, the time complexity is the same. As with the chance constraint and exact chance constraint planners, the time complexity for inserting a value is $\mathcal{O}(\log n)$ and the time complexity of extracting a value of the list is $\mathcal{O}(\log n)$ as well.

4.6 Measurement feedback and adaptive velocity

This section introduces measurement feedback to the planner such that the state uncertainty of the mobile robot is updated leading to a more realistic uncertainty provided the real system is equipped with sensors for measurements. In addition, an approach is proposed that selects the

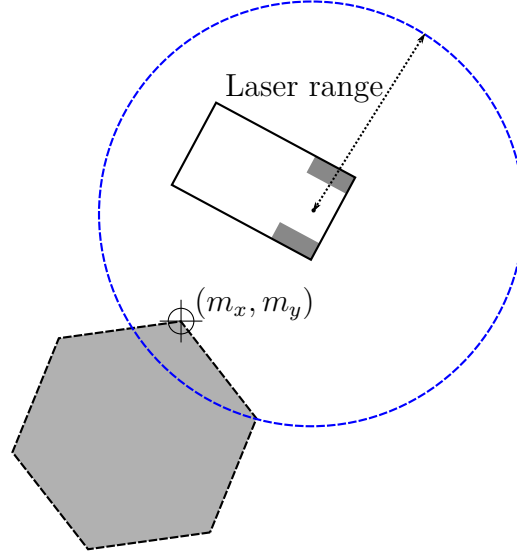


Figure 4.18: Feature detection of a static obstacle.

robots linear velocities with respect to the current probability of collision. The SCHA* planner of the previous section will be extended for measurement feedback and adaptive velocities and the performance will be compared with an existing planner. A new cost function considering travelling time instead of travelling distance will be introduced for the adaptive velocities.

4.6.1 Measurement feedback

In the following it is assumed that the measurement is received from laser range sensors (e.g., Lidar), where the localization of the robot is calculated by detecting known features such as corners from static obstacles. Figure 4.18 shows a mobile robot detecting with a Lidar sensor a feature of the static obstacle, where m_x and m_y denote the coordinates of the feature in the global reference frame. The blue dashed circle denotes the laser range of the Lidar sensor.

4.6.1.1 Measurement model

The same measurement model as in [114] is used for the feature detection, where the equation of the deterministic measurement model is in the following form

$$\mathbf{z}_t^i = \mathbf{h}(\mathbf{x}_t, j, m), \quad (4.61)$$

where \mathbf{z}_t^i is the measurement vector of the i th-feature which corresponds to the j th-landmark of the feature map m . Here, it is assumed that the correspondence between the detected feature and the landmark on the map is known. The nonlinear measurement model with additive

Gaussian noise is expressed as follows

$$\begin{bmatrix} r_t^i \\ \phi_t^i \\ s_t^i \end{bmatrix} = \begin{bmatrix} \sqrt{(m_{j,x} - x_t)^2 + (m_{j,y} - y_t)^2} \\ \text{atan2}(m_{j,y} - y_t, m_{j,x} - x_t) - \theta_t \\ m_{j,s} \end{bmatrix} + \boldsymbol{\varepsilon}_Q, \quad (4.62)$$

where r_t^i is the range to the landmark, ϕ_t^i is the corresponding bearing and s_t^i is the signature of the landmark. $[m_{j,x}, m_{j,y}]^\top$ are the coordinates of the landmark and $m_{j,s}$ is the landmarks signature. The signature is not used in the following but is kept for completeness. The additive Gaussian noise is expressed with $\boldsymbol{\varepsilon}_Q \sim \mathcal{N}(0, \Sigma_Q)$ with the covariance matrix Σ_Q .

4.6.1.2 Kalman filter update

The function $\mathbf{h}(\mathbf{x}_t, j, m)$ has to be linearized for the correction step of the EKF. Hence, the resulting measurement matrix $\tilde{\mathbf{H}}_t^i$ is

$$\tilde{\mathbf{H}}_t^i = \frac{\partial \mathbf{h}}{\partial \mathbf{x}_t}(\boldsymbol{\mu}_{x_t}, j, m) \quad (4.63)$$

$$= \begin{bmatrix} -\frac{m_{j,x} - \mu_{x_t}}{\sqrt{(m_{j,x} - \mu_{x_t})^2 + (m_{j,y} - \mu_{y_t})^2}} & -\frac{m_{j,y} - \mu_{y_t}}{\sqrt{(m_{j,x} - \mu_{x_t})^2 + (m_{j,y} - \mu_{y_t})^2}} & 0 \\ \frac{m_{j,y} - \mu_{y_t}}{(m_{j,x} - \mu_{x_t})^2 + (m_{j,y} - \mu_{y_t})^2} & -\frac{m_{j,x} - \mu_{x_t}}{(m_{j,x} - \mu_{x_t})^2 + (m_{j,y} - \mu_{y_t})^2} & -1 \\ 0 & 0 & 0 \end{bmatrix}. \quad (4.64)$$

The correction step of the EKF to obtain the updated state covariance matrix $\Sigma_{\mathbf{x}_{t+1}}$ is as follows

$$\mathbf{S}_t = \mathbf{H}_t \bar{\Sigma}_{\mathbf{x}_{t+1}} \mathbf{H}_t^\top + \mathbf{Q}_t \quad (4.65)$$

$$\mathbf{L}_t = \bar{\Sigma}_{\mathbf{x}_{t+1}} \mathbf{H}_t^\top \mathbf{S}_t^{-1} \quad (4.66)$$

$$\Sigma_{\mathbf{x}_{t+1}} = (\mathbf{I}_3 - \mathbf{L}_t \mathbf{H}_t) \bar{\Sigma}_{\mathbf{x}_{t+1}}, \quad (4.67)$$

where the above equations correspond to the correction step for all detected features n at time instant t with

$$\mathbf{H}_t = [\mathbf{H}_t^1, \dots, \mathbf{H}_t^n]^\top \in \mathbb{R}^{3n \times 3} \quad (4.68)$$

$$\mathbf{Q}_t = \text{diag}(\Sigma_Q, \dots, \Sigma_Q) \in \mathbb{R}^{3n \times 3n}. \quad (4.69)$$

Furthermore, $\bar{\Sigma}_{\mathbf{x}_{t+1}}$ is the state covariance matrix of the mobile robot after the prediction step of the EKF and $\mathbf{L}_t \in \mathbb{R}^{3 \times 3n}$ is the Kalman gain. It can be seen, that the computation of the inverse of the matrix $\mathbf{S}_t \in \mathbb{R}^{3n \times 3n}$ will be large if many features are detected. Generally, in an implementation the state covariance matrix is updated iteratively for each detected feature. But a better solution is the use of the extended information filter (EIF) in the correction step instead of using EKF [32]. In the EIF the inverse of \mathbf{S}_t does not have to be calculated, instead the sum of each measurement is taken and the inverse of a $\mathbb{R}^{3 \times 3}$ matrix is taken ones to obtain

$\Sigma_{x_{t+1}}$

$$\bar{\Omega}_{x_{t+1}} = \bar{\Sigma}_{x_{t+1}}^{-1} \quad (4.70)$$

$$\Sigma_{x_{t+1}} = \left[\bar{\Omega}_{x_{t+1}} + \sum_{i=1}^n [\mathbf{H}_t^i]^\top \Sigma_Q^{-1} \mathbf{H}_t^i \right]^{-1}, \quad (4.71)$$

where $\bar{\Omega}_{x_{t+1}}$ is the information matrix of the prediction step, which corresponds to the inverse of the covariance matrix $\bar{\Sigma}_{x_{t+1}}$.

4.6.1.3 Control feedback

Using measurement feedback in a path planner may lead to unreachable states for the real system, due to the fact that the state covariance matrix Σ_{x_t} may be much smaller than the state covariance matrix of the prediction step $\bar{\Sigma}_{x_t}$. There is no guarantee that the real system can reach the new state with a small covariance matrix from the previous state with a large covariance matrix. Hence, Bry et al. [19] proposed a planning approach that incorporates a stabilizing controller in the correction step of the EKF. The result is a larger covariance matrix for states that can guarantee reachability for the mobile robot. The following shows the update step of the EKF with a stabilizing control gain \mathbf{K}_t

$$\mu_{x_{t+1}} = (\tilde{\mathbf{A}}_t - \tilde{\mathbf{B}}_t \mathbf{K}_t) \mu_{x_t} \quad (4.72)$$

$$\Sigma_{\Lambda_{t+1}} = (\tilde{\mathbf{A}}_t - \tilde{\mathbf{B}}_t \mathbf{K}_t) \Sigma_{\Lambda_t} (\tilde{\mathbf{A}}_t - \tilde{\mathbf{B}}_t \mathbf{K}_t)^\top + \mathbf{L}_t \mathbf{H}_t \bar{\Sigma}_{t+1} \quad (4.73)$$

$$\Sigma_{t+1} = \Sigma_{x_{t+1}} + \Sigma_{\Lambda_{t+1}}, \quad (4.74)$$

where Σ_{Λ_t} is the additional covariance matrix calculated with \mathbf{K}_t . Σ_{t+1} is the state covariance matrix which is the summation of the state covariance matrix obtained from the EKF and the control covariance matrix. The above equations can be adjusted such that the EIF will be used instead of the EKF for the correction step. The main problem is the calculation of $\Sigma_{\Lambda_{t+1}}$ which is expressed with the Kalman gain \mathbf{L}_t . The term $\mathbf{L}_t \mathbf{H}_t \bar{\Sigma}_{t+1}$ can be calculated with (4.67) and (4.71)

$$\mathbf{L}_t \mathbf{H}_t \bar{\Sigma}_{t+1} = \bar{\Sigma}_{x_{t+1}} - \Sigma_{x_{t+1}} \quad (4.75)$$

$$\Sigma_{\Lambda_{t+1}} = (\tilde{\mathbf{A}}_t - \tilde{\mathbf{B}}_t \mathbf{K}_t) \Sigma_{\Lambda_t} (\tilde{\mathbf{A}}_t - \tilde{\mathbf{B}}_t \mathbf{K}_t)^\top + \bar{\Sigma}_{x_{t+1}} - \Sigma_{x_{t+1}}. \quad (4.76)$$

The obtained covariance matrix Σ_{t+1} can be used for the proposed probabilistic robust path planners in the previous sections.

4.6.2 Adaptive velocity

This subsection introduces the policy that sets the linear velocity of the mobile robot with respect to the current probability of collision. In addition, a new cost function is introduced that minimizes the travelling time instead of the travelling distance.

The absolute values of the linear velocity are assumed to range from $v_{\min} = 0.1 \text{ m/s}$ to $v_{\max} = 1.0 \text{ m/s}$ and are discretized with a step size of $\delta v = 0.1 \text{ m/s}$ for the planner. The planner should select large velocities for the next time instant if the probability of collision is small and vice versa. Hence, if the probability of collision $\Delta(\mathbf{x}_t) = 0.0$ the largest velocity and as an upper bound the smallest velocity should be selected if $\Delta(\mathbf{x}_t) \geq 0.45$. Therefore the linear velocities can be selected with the following expression

$$v_t = v_{\max} - \left(\delta v \left\lfloor \frac{\min(\Delta_t(\mathbf{x}_t), \Delta_{\max})}{\delta \Delta} \right\rfloor \right), \quad (4.77)$$

where $\delta \Delta$ is the discretization of the collision probabilities and $\lfloor \cdot \rfloor$ expresses the *floor* function. The HA* algorithms always move with the same turn radius, therefore the angular velocity of the mobile robot should be adjusted with respect to the current linear velocity

$$\omega_t = \frac{v_t}{R}, \quad (4.78)$$

where R is the turn radius of the mobile robot in the HA* planner.

4.6.2.1 Cost function

This approach will select linear velocities with respect to the probability of collision of the mobile robot, therefore the cost function should minimize the travelling time instead of the travelling distance. The following is the cost function for this path planning approach

$$E[J(\mathbf{x}_t)] = \delta t [1 + k \ln(1 - \Delta_t(\mathbf{x}_t)) + k_\lambda \lambda_{\max}], \quad (4.79)$$

where $k_\lambda \lambda_{\max}$ is a penalty for the covariance matrix with a gain k_λ and $\lambda_{\max} = \text{eig}(\Sigma_{t+1})$. The penalty guides the path to areas where a measurement feedback is received such that the mobile robot is more certain about its current state. The planners using measurement feedback and adaptive velocities are named adaptive soft constraint hybrid A* (ASCHA*).

4.6.3 Results and discussion

This section demonstrates the performance of the ASCHA* planner. The planner is compared with existing planners (i.e., HeAT-RT, CLRRT, CCxRRT) and planners of the previous sections in the clustered, static environment and the dynamic environment. The same parameters as in the previous sections are used for the planners, in addition the ASCHA* planners gains for the cost function are $k = -0.5$ and $k_\lambda = 0.6$. The range for the laser sensor is set to 2.5 m and the covariance matrix for the measurement model is set to

$$\Sigma_Q = \begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 0.01 & 0 \\ 0 & 0 & 0.01 \end{bmatrix}. \quad (4.80)$$

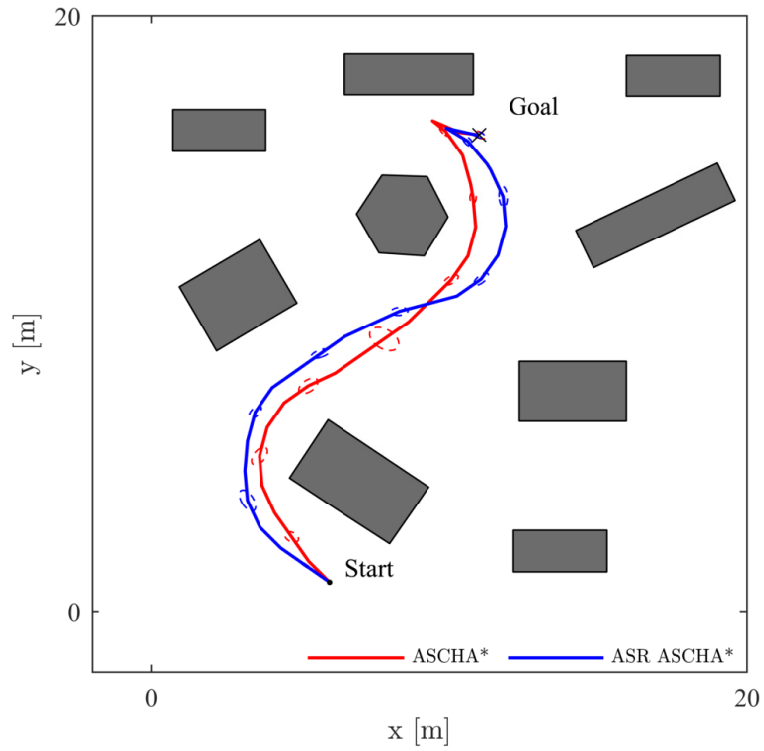


Figure 4.19: Resulting paths with adaptive velocities in the static environment.

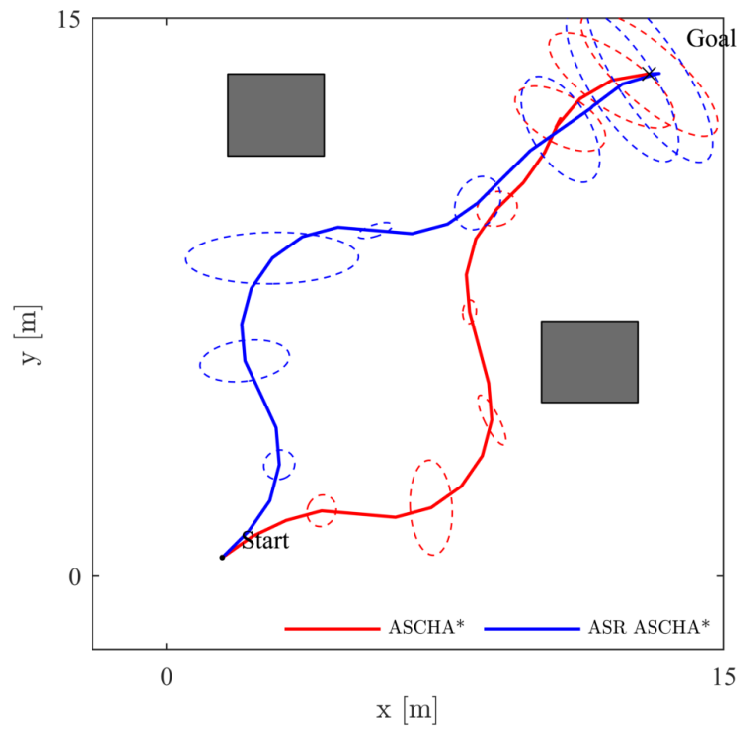


Figure 4.20: Resulting paths with adaptive velocities in the dynamic environment.

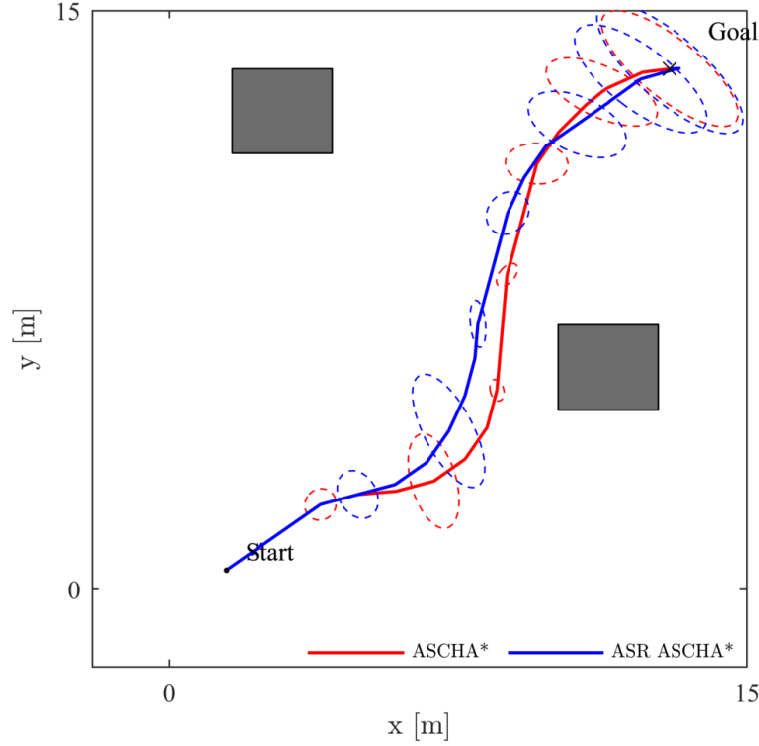


Figure 4.21: Performance of the eigenvalue penalty in a static open environment.

Furthermore, the controller \mathbf{K}_t is generated using pole placement for the linearized and discretized canonical nonholonomic mobile robot kinematic model.

$$\tilde{\mathbf{A}} = \begin{bmatrix} 1 & 0 & -\sin \theta \delta t v \\ 0 & 1 & \cos \theta \delta t v \\ 0 & 0 & 1 \end{bmatrix} \quad (4.81)$$

$$\tilde{\mathbf{B}} = \begin{bmatrix} \cos \theta \delta t & 0 \\ \sin \theta \delta t & 0 \\ 0 & \delta t \end{bmatrix} \quad (4.82)$$

$$\lambda_{1,2,3} = [-0.05, -0.05, -0.5], \quad (4.83)$$

where $\lambda_{1,2,3}$ are the selected poles and the control gain is generated for a discrete set of headings θ and a discrete set of velocities v , where it is assumed that the controller will be stable for headings and velocities between the discretization steps.

Figures 4.19 and 4.20 show the resulting paths of ASCHA* and ASR ASCHA* in the static and dynamic environments. The 95 % confidence ellipses are plotted every third node. Especially, in the dynamic environment it can be seen, that the state uncertainty first increases until the robot is close enough to a static obstacle. The mobile robot receives a measurement feedback near the static obstacles and hence the state uncertainty decreases. Table 4.5 shows the comparison with the existing planners, where the travelling time, collision check time per node,

expanded nodes, and the computation time are compared. In both the static and dynamic environments does the adaptive velocity approach has the fastest travelling time but also a larger computation time compared to most other planners. The larger computation time and collision check time per node results from the additional computation time for the measurement feedback and correction step with the EIF. Even though the computation time is larger, the adaptive velocity showed its effectiveness. The travelling time of ASCHA* is $\sim 36\%$ and $\sim 37\%$ faster than the second fastest planner HeAT-RT in the static and dynamic environment, respectively.

In the following the effectiveness of the λ_{\max} penalty is shown. Figure 4.21 shows the paths of ASCHA* and ASR ASCHA* in an open space without a dynamic obstacle. Both planners moved first near a static obstacle to receive measurement feedback and then continued moving to the goal location instead of moving straight and directly to the goal. Hence, the penalty in the cost function can guarantee that the mobile robot will not move to the goal location with a large uncertainty, provided static obstacles are in the vicinity of the goal.

The performance of the adaptive velocity planners was evaluated using a clustered, static environment and an environment with a dynamic obstacle.

The results showed that the planner generated in both environments probabilistic robust paths with the smallest travelling time. Furthermore, the penalty for the eigenvalues of the state uncertainty will guide the robot close to static obstacles such that measurements can be received (Figure 4.21). Due to the measurement feedback and the correction step calculation is the computation time of the proposed planners larger than the computation time of most other planners. Hence, these planners may be more suitable as global planners and local corrections can be generated using faster probabilistic robust planners such as CEHA*.

4.7 Summary

This chapter proposed probabilistic robust path planning algorithms for nonholonomic mobile robots. The algorithms consider Gaussian uncertainty of the robot, static obstacles, and dynamic obstacles. The proposed planners find probabilistic robust paths in both static and dynamic environments. The uncertainty of the mobile robot was extended for mobile robots of any shape were the proposed algorithms that consider the shape of the robot are able to further increase the safety. Furthermore, a variation of the planner was proposed that omits computationally heavy calculations in the planner (such as the Gaussian error function) and instead uses a geometric approach. The proposed planner finds probabilistic safe paths and reduces the computation time compared to other approaches. Moreover, a planner is proposed that uses a measurement model for a laser range sensor in the planning process to receive information of the robots localization. In addition, the planner adapts its current linear velocity according to the current probability of collision. The resulting paths have a shorter travelling time than other planners and the mobile robot attempts to move in the vicinity of static obstacles such that it can receive a measurement feedback.

Table 4.5: Comparison of the travelling time and computation time of the adaptive planners with existing approaches.

Algorithm	Travelling time [s]	Collision check time per node [μ s]	Expanded nodes (best solution)	Computation time [s]
Static environment				
ASCHA*	21.68	~ 379	1412	~ 1.025
ASR ASCHA*	23.22	~ 1101	1769	~ 3.58
CEHA*	43.97	~ 153	174	~ 0.069
ASR CEHA*	48.97	~ 448	424	~ 0.48
CCxHA*	41.03	~ 240	408	~ 0.22
ASR CCxHA*	45.65	~ 898	5086	~ 13.84
ASR SCHA*	44.29	~ 885	2022	~ 3.5
ASR HA*	42.80	~ 67	580	~ 0.11
HeAT-RT	34.0	~ 15	288	$\sim 0.32 \pm 0.51$
CLRRT	37.0	~ 15	1720	$\sim 0.57 \pm 0.74$
CCxRRT	38.0	~ 240	10242	$\sim 2.65 \pm 3.02$
Dynamic environment				
ASCHA*	21.28	~ 221	2408	~ 1.014
ASR ASCHA*	21.39	~ 477	2980	~ 2.78
CEHA*	38.5	~ 173	103	~ 0.032
ASR CEHA*	44.5	~ 360	140	~ 0.119
CCxHA*	36.5	~ 224	550	~ 0.3
ASR CCxHA*	36.71	~ 637	671	~ 1.08
ASR SCHA*	37.68	~ 589	4138	~ 4.9
ASR HA*	36.5	~ 56	584	~ 0.11
HeAT-RT	34.0	~ 14	72	$\sim 0.96 \pm 0.51$
CLRRT	34.0	~ 14	699	$\sim 0.27 \pm 0.57$
CCxRRT	35.0	~ 224	788	$\sim 0.55 \pm 0.71$

5 Conclusions and Future Works

5.1 Conclusions

This thesis proposed algorithms for CPP problems that address different objectives of the coverage path. Furthermore, a parking controller and probabilistic robust PTP path planners are introduced. A separate conclusion is given for each chapter of the thesis:

- Chapter 2 proposes a HGA algorithm to generate coverage paths that improve the traveling time, energy consumption or number of cell visits. The proposed fitness function for the energy consumption is proven to be valid for differential-drive mobile robots in planar environments. Results have shown that energy-efficient paths are different from those with the number of cell visits as a fitness function. Hence, energy cannot be optimized by reducing the number of repetitive visits. The HGA algorithm with TASP and BSA shows significant results: HGA/BSA finds good solutions for both the number of visited cells and smooth energy-based fitness function, whereas HGA/TASP provides good solutions for the both energy-based fitness functions and traveling time. HGA/TASP reduced the travelling time by $\sim 28\%$ and the energy consumption on the smooth surface by $\sim 7\%$ on the large environment compared to DT. Furthermore, HGA/TASP has the fastest computation time among the HGAs. In addition, the evolutionary algorithms can find solutions for different-sized environments, and their fitness was consistent for randomly selected starting positions.

The HGA can improve the solutions obtained from other CPP algorithms such as TASP and BSA. TASP is an appropriate algorithm for faster traveling, whereas BSA shows good results for the number of cell visits. Because the properties of TASP and BSA are inherited for the HGA, HGA/TASP has good results for the traveling time, and HGA/BSA provides good results for the number of cell visits.

- Chapter 3 proposes a hybrid systems approach for parking control. The proposed controller provides good results in simulations and experiments. The robot is able to reach its desired states in each control step and is safely parked at the desired pose.
- Chapter 4 proposes novel variations of the HA* algorithm, which consider the probability of collision with an obstacle. Furthermore, a method is proposed which considers the shape of the robot in the probabilistic collision calculations. The performance is evaluated with an environment with many static obstacles and an environment with a dynamic obstacle. In addition, the true probability of collision is calculated using the MCS algorithm. Results show that the variations are able to find probabilistic safe paths inside

the set threshold p_{coll} . Results show that the CCHA* algorithms are very conservative and therefore have a very low probability of collision. However, with increasing number of obstacles the CCHA* may fail in finding a solution due to pruning all the expanded nodes. The CCxHA* shows good results with small increase in the computation time per node compared to the CCHA* algorithms. All proposed algorithms are able to avoid the dynamic obstacle with a decreased probability of collision. Furthermore, results have shown that best gain values for the SCHA* algorithm should be in the range of $k = -0.5$ and $k = -1.5$ to achieve a good result for safety and travelled distance. The method to consider the shape of the robot further increased the probabilistic safety at the cost of additional computation time per node. ASR CCxHA* improved the probability of collision by $\sim 54\%$ compared to the deterministic planner ASR HA*. Most ASR of the proposed planners had close to 0% probability of collision. Since the computation time is larger for the ASR variations the algorithms can be used for a global plan as a first probabilistic safe path.

The planners that use the confidence ellipse for obstacle detection had small computation times, compared to other existing planners, with resulting probabilistic robust paths making these planners versatile for path generation in any environment or robot shape. The ASR variation of the CEHA* approach also had close to 0% probability of collision. The proposed planners that employ measurement feedback and adaptive velocities with respect to the current probability of collision successfully generated paths in both the static and dynamic environments. In addition, the resulting paths had the smallest travelling time among all compared path planners. The resulting path of the ASCHA* was $\sim 36\%$ and $\sim 37\%$ faster than the second fastest planner HeAT-RT in the static and dynamic environment, respectively.

5.2 Future works

In the future the HGA can be extended with more different local search algorithms. The combination of several local search algorithms should be analyzed more in order to obtain better results in terms of travelling time and energy efficiency. So far the algorithm assumes full knowledge of the environment, an extension could be an online approach, where the environment is separated into smaller regions based on the sensor limits of the mobile robot. A combination of an online algorithm e.g. TASP and HGA/TASP could lead to optimal online paths.

So far the parking controller is not aware of its environment, the approach could be extended by including constraints so that the algorithm is able to perform the parking motion without hitting any obstacles.

So far only Gaussian noise is considered, the planners can be adapted e.g. using particles to consider non-Gaussian noise as well. Furthermore, the current model for dynamic obstacles assumes constant velocities, using a better model with e.g. constant turning radius could result in a more precise estimation of the dynamic obstacle behaviour during the planning process. In addition, heuristics that consider dynamic obstacle may further improve the computation time of the path planners. The approach considering the arbitrary-shape of the mobile robot

in $SE(2)$ can be extended for $SE(3)$, which can then be used for e.g., robotic manipulators to provide probabilistic robust solutions.

A Linear transformation of variance and covariance

It is known that variance and covariance can be expressed using the expectation of random variables as follows:

$$\text{var}(X) = E[(X - E[X])^2], \quad (\text{A.1})$$

$$\text{cov}(X, Y) = E[(X - E[X])(Y - E[Y])]. \quad (\text{A.2})$$

Furthermore, the expectation of $aX + bY + c$ is

$$E[aX + bY + c] = aE[X] + bE[Y] + c. \quad (\text{A.3})$$

Using (A.1) and (A.3)

$$\begin{aligned} \text{var}(aX + bY + c) &= E[(aX + bY + c - E[aX + bY + c])^2] \\ &= E[(a(X - E[X]) + b(Y - E[Y]) + c - c)^2] \end{aligned} \quad (\text{A.4})$$

Using the principles of (A.3) again and the binomial formula,

$$\begin{aligned} &= E[a^2(X - E[X])^2] + E[b^2(Y - E[Y])^2] \\ &\quad + E[2ab(X - E[X])(Y - E[Y])] \end{aligned} \quad (\text{A.5})$$

Hence,

$$\begin{aligned} \text{var}(aX + bY + c) &= a^2\text{var}(X) + b^2\text{var}(Y) \\ &\quad + 2ab\text{cov}(X, Y). \end{aligned} \quad (\text{A.6})$$

The covariance for $Z_1 = a_1X_1 + b_1Y_1 + c_1$ and $Z_2 = a_2X_2 + b_2Y_2 + c_2$ can be calculated similarly, i.e.,

$$\begin{aligned} \text{cov}(Z_1, Z_2) &= E[(a_1X_1 + b_1Y_1 + c_1 - E[a_1X_1 + b_1Y_1 + c_1]) \\ &\quad (a_2X_2 + b_2Y_2 + c_2 - E[a_2X_2 + b_2Y_2 + c_2])], \end{aligned} \quad (\text{A.7})$$

which can be simplified using (A.3),

$$\begin{aligned}\text{cov}(Z_1, Z_2) = E[(a_1(X_1 - E[X_1]) + b_1(Y_1 - E[Y_1])) \\ (a_2(X_2 - E[X_2]) + b_2(Y_2 - E[Y_2]))].\end{aligned}\tag{A.8}$$

Rearranging above equation and using (A.2) led to the final result for the covariance as follows:

$$\begin{aligned}\text{cov}(Z_1, Z_2) = a_1a_2\text{cov}(X_1, X_2) + b_1b_2\text{cov}(Y_1, Y_2) \\ + a_1b_2\text{cov}(X_1, Y_2) + a_2b_1\text{cov}(X_2, Y_1).\end{aligned}\tag{A.9}$$

List of Abbreviations

ASCHA*	adaptive soft constraint hybrid A*
ASR	arbitrary-shaped mobile robot
ASR ASCHA*	arbitrary-shaped mobile robot adaptive soft constraint hybrid A*
ASR CCHA*	arbitrary-shaped mobile robot chance constraint hybrid A*
ASR CCxHA*	arbitrary-shaped mobile robot exact collision probability chance constraint hybrid A*
ASR CEHA*	arbitrary-shaped mobile robot confidence ellipse hybrid A*
ASR HA*	arbitrary-shaped mobile robot hybrid A*
ASR SCHA*	arbitrary-shaped mobile robot soft constraint hybrid A*
BP	backtracking point
BSA	backtracking spiral algorithm
CCHA*	chance constraint hybrid A*
CCRRT	chance constraint RRT
CCRRT*	chance constraint RRT*
CCxHA*	exact collision probability chance constraint hybrid A*
CCxRRT	exact collision probability chance constraint RRT
CEHA*	confidence ellipse hybrid A*
CLRRT	closed-loop rapidly-exploring random tree
CPP	coverage path planning
DT	distance transform
EIF	extended information filter
EKF	extended Kalman filter
GA	genetic algorithm
HA*	hybrid A*
HeAT-RT	heuristic arrival time field-biased random tree
HGA	hybrid genetic algorithm
HGA/Both	HGA with both BSA and TASP
HGA/BSA	HGA with BSA
HGA/TASP	HGA with TASP
I/O-linearization	input/output-linearization
IMU	inertial measurement unit
KF	Kalman filter
MA	memetic algorithm
MCS	Monte Carlo simulation
MIMO	multiple input multiple output

OLS	opposite lateral side
PRM	probabilistic roadmap
PTP	point-to-point
RLS	reference lateral side
RRBT	rapidly-exploring random belief tree
RRT	rapidly-exploring random tree
SCHA*	soft constraint hybrid A*
TASP	turn-away starting point
TS	tournament selection

Table of Symbols

Symbol	Unit	Description
$\mathbf{A}_{\mathcal{D}}$	—	Transition matrix of the dynamic obstacle
$\dot{\phi}_{\text{dl}}$	rad/s	Desired angular velocity of the left drive wheel
$\dot{\phi}_{\text{dr}}$	rad/s	Desired angular velocity of the right drive wheel
ϕ_t^i	rad	Bearing of the landmark i
r_t	—	Boolean state of driving forward or reverse (reverse is set to 1)
C	—	Permutation of visited cells
c	m	Coordinates of the current cell
\mathcal{C}^{CPP}	—	2-dimensional configuration space for the CPP algorithm
$\mathcal{C}_{\text{free}}^{\text{CPP}}$	—	obstacle free cells in the configuration space for the CPP algorithm
$\mathcal{C}_{\text{obs}}^{\text{CPP}}$	—	obstacle cells in the configuration space for the CPP algorithm
$\mathcal{C}_{\text{unvisited}}^{\text{CPP}}$	—	unvisited cells in the configuration space for the CPP algorithm
$\mathcal{C}_{\text{visited}}^{\text{CPP}}$	—	visited free cells in the configuration space for the CPP algorithm
$\mathbf{c}_{1,2}$	—	Coordinates of the centers of the two circles covering the confidence ellipse
r_c	m	Radius of the circles covering the confidence ellipse
\mathbf{u}	—	Input vector of the continuous mobile robot system
\mathbf{y}	—	Output vector of the continuous mobile robot system
\mathbf{x}	—	State vector of the continuous mobile robot system
Σ_{Λ_t}	—	Control covariance matrix at time instant t
$\Sigma_{\mathbf{x}_{\mathcal{D}_t}}$	—	Covariance matrix of the dynamic obstacle at time instant t
$\Sigma_{\mathcal{O}}$	—	Covariance matrix of current obstacle
$\Sigma_{\mathbf{q}_t}$	—	Covariance matrix of the mobile robot position at time instant t
$\bar{\Sigma}_{t+1}$	—	State covariance matrix in the prediction step
Σ_M	—	Covariance matrix of the random Gaussian process noise
$\Sigma_{\mathcal{D}}$	—	Covariance matrix of the process noise of the dynamic obstacle
Σ_Q	—	Covariance matrix of the measurement model
$\Sigma_{\mathbf{x}_t}$	—	Covariance matrix of the robot at time instant t
$\Sigma_{\mathbf{q}_t^j}$	—	Covariance matrix of square segment j at time instant t
Σ_S	—	Covariance matrix of the static obstacles
Σ_{total}	—	Sum of all covariance matrices
σ_{x_t, θ_t}	m rad	Covariance of the mobile robot of the x -axis position and the heading at time instant t

σ_{x_t, y_t}^j	m^2	Covariance of square segment j of x -axis and y -axis at time instant t
σ_{y_t, θ_t}	m rad	Covariance of the mobile robot of the y -axis position and the heading at time instant t
\mathcal{C}^{PTP}	—	Configuration space of the PTP path planner
$\mathcal{C}_{\text{free}}^{\text{PTP}}$	—	Free space in the configuration space of the PTP path planner
$\mathcal{C}_{\text{obs}}^{\text{PTP}}$	—	Obstacle space in the configuration space of the PTP path planner
C_{rev}	—	Penalty for driving in reverse
c_{start}	m	Coordinates of the starting cell
C_{sw}	m	Penalty for switching the direction of motion
I_i	A	Current of the motors at the i -th sampling instant
I_l	A	Applied current on the left drive wheel
I_r	A	Applied current on the right drive wheel
Δ_t	—	Probability of collision with all obstacles
δ_{jit}	—	Probability of collision of line segment i of obstacle j at time instant t
$\delta\Delta$	—	Discretization step of the collision probability
Δ_{max}	—	Maximum collision probability
δt	s	Sampling time
δv	m/s	Discretization step of the linear velocity
ε_d	m	Random Gaussian variable of the distance from mobile robot to obstacle line segment
$\varepsilon_{\mathcal{D}}$	—	Random Gaussian noise of the dynamic obstacle
$\mathbf{x}_{\mathcal{D}_t}$	—	State vector of the dynamic obstacle at time instant t
$x_{\mathcal{D}_t}$	m	Position of the dynamic obstacle in x -axis at time instant t
$y_{\mathcal{D}_t}$	m	Position of the dynamic obstacle in y -axis at time instant t
\mathcal{D}^j	—	Description of dynamic obstacle j
$\mathcal{A}_{\mathcal{D}^j}$	—	Area of dynamic obstacle j
$\lambda_{1,2}$	—	Eigenvalues of the covariance matrix for the positional uncertainty
$\mathbf{v}_{1,2}$	—	Eigenvectors of the covariance matrix for the positional uncertainty
$\hat{\mathbf{v}}_{\text{max}}$	—	Eigenvector of the maximum eigenvalue
E	J	Energy consumption of the motors
ϵ_{θ}	rad	Threshold in the heading
ϵ_x	m	Threshold in x -axis
e_{θ}	rad	Error in the heading
e_x	m	Error in x -axis
e_y	m	Error in y -axis
I_n	—	n -dimensional identity matrix
Γ	—	Threshold probability of collision
$\mathcal{X}_{\text{goal}}$	—	Goal space
\mathbf{H}_t^i	—	Measurement matrix of the linearized measurement model
ζ_i	—	State i of the hybrid system

$\bar{\Omega}_{t+1}$	—	Information matrix of the state in the prediction step
J	—	Cost or fitness objective function of the optimization problem
J_{acc}	J; s; —	Cost for the acceleration phase
J_{dec}	J; s; —	Cost for the deceleration phase
J_S	J; s; —	Cost for the straight motion phase
J_U	J; s; —	Cost for the turn motion phase
J_{UT}	J; s; —	Cost for the U-turn motion phase
k_λ	—	Gain for the covariance matrix eigenvalues penalty
k_x	1/s	Control gain in x -axis
k_y	1/s	Control gain in y -axis
$k_{\theta_{\zeta_i}}$	1/s	Control gain for the heading of the i -th hybrid system
\mathbf{L}_t	—	Kalman gain at time instant t
k_{DOF}	—	Number of degrees of freedom of the χ^2 -distribution
λ_{max}	—	Maximum eigenvalue of the covariance matrix
l	m	Distance from the rear axis center to the left or right drive wheel
$l_{x_t^j}$	—	Distance to the robot center of square segment j in x -axis at time instant t
$l_{y_t^j}$	—	Distance to the robot center of square segment j in y -axis at time instant t
$\tilde{\mathbf{A}}$	—	Transition matrix of the linearized system at time instant t
$\tilde{\mathbf{B}}$	—	Input matrix of the linearized system at time instant t
\mathbf{x}_{p_i}	—	Point on line segment i
M	—	Permutation of motions
m_i	—	Selected motion for cell c_i
m	—	Feature map
μ_d	m	Mean distance between mobile robot and obstacle line segment
$\boldsymbol{\mu}_{q_t}$	—	Mean position of the mobile robot at time instant t
$\boldsymbol{\mu}_{x_t}$	—	Mean state at time instant t
$\boldsymbol{\mu}_{x_{\mathcal{D}_t}}$	—	Mean state of the dynamic obstacle at time instant t
$\boldsymbol{\mu}_{\text{goal}}$	—	Mean state at goal area
μ_{θ_t}	rad	Mean heading of the mobile robot at time instant t
μ_{l_t}	m	Mean travelled distance at time instant t
μ_{x_t}	m	Mean position of the mobile robot in x -axis at time instant t
$\mu_{x_t^j}$	m	Mean position of square segment j in x -axis at time instant t
μ_{y_t}	m	Mean position of the mobile robot in y -axis at time instant t
$\mu_{y_t^j}$	m	Mean position of square segment j in y -axis at time instant t
m_x	m	Global coordinate in direction of the x -axis of the feature point
m_y	m	Global coordinate in direction of the x -axis of the feature point
n_S	—	Number of static obstacles
n_{acc}	—	Number of acceleration phases in the CPP path
n_{dec}	—	Number of deceleration phases in the CPP path
N_{GA}	—	Number of <i>genes</i> in the <i>chromosome</i>

$\mathbf{x}_{S_0^j}$	—	Nominal state of static obstacle j
$x_{S_0^j}$	m	Nominal position of static obstacle j in x -axis at time instant t
$y_{S_0^j}$	m	Nominal position of static obstacle j in y -axis at time instant t
$\hat{\mathbf{v}}_{1,2}$	—	Normalized eigenvectors of the covariance matrix for the positional uncertainty
n_p	—	Number of measurement points
n_Q	—	Number of visited cells
n_S	—	Number of straight motion phases in the CPP path
n_{Seg}	—	Number of line segments of the current obstacle
n_T	—	Number of turn motion phases in the CPP path
$n_{\mathcal{D}}$	—	Number of dynamic obstacles
n_e	—	Number of line segments of the obstacle
n_{O_j}	—	Number of line segments of obstacle j
n_O	—	Number of static and dynamic obstacles
n_S	—	Number of static obstacles
n_{UT}	—	Number of U-turn motion phases in the CPP path
\mathcal{O}	—	Set of static and dynamic obstacles
ω_t	rad/s	Angular velocity of the mobile robot at time instant t
$\hat{\omega}_t$	rad/s	Angular velocity of the mobile robot at time instant t subject to additive Gaussian noise
P	—	Number of particles
\mathcal{P}	—	Set of states that represent the generated path
p_c	—	Probability of applying crossover in the HGA approach
p_{coll}	—	Probability of collision
p_{conf}	—	Selected confidence for the confidence ellipse
x_t^j	—	Position of square segment j in x -axis at time instant t
y_t^j	—	Position of square segment j in y -axis at time instant t
P_i	W	Power consumption of the motors at the i -th sampling instant
γ	—	Threshold probability of collision for one obstacle
ε_M	—	Random Gaussian process noise
\mathbf{w}	—	Pseudo input vector of the system
ε_Q	—	Random Gaussian variable of the measurement model
\mathbf{q}_t	—	Position vector at time instant t
r	m	Wheel radius of the drive wheels
r_{rec}	m	Radius of the rectangular segment of the robot
r_{sq}	m	Radius of the square segment of the robot
r_t^i	m	Range of the landmark i to the current robot position
\mathcal{R}	—	Description of the mobile robot
$\mathcal{A}_{\mathcal{R}}$	—	Area of the mobile robot
s	—	Scale factor of the confidence ellipse
s_t^i	—	Signature of landmark i
k	m	Gain for the soft constraint of SCHA*

\mathbf{x}_0	—	Initial state of the mobile robot system
Σ_{t+1}	—	Sum of the state covariance matrix and the control covariance matrix
\mathcal{S}^j	—	Description of static obstacle j
$\mathcal{A}_{\mathcal{S}^j}$	—	Area of static obstacle j
$\varepsilon_{\mathcal{S}^j}$	—	Random Gaussian noise of static obstacle j
$\mathbf{x}_{\mathcal{S}^j}$	—	State of static obstacle j
$x_{\mathcal{S}^j}$	m	Position of static obstacle j in x -axis at time instant t
$y_{\mathcal{S}^j}$	m	Position of static obstacle j in y -axis at time instant t
t	s	Time
t_{goal}	s	Final time instant
θ	rad	Heading angle of the mobile robot
θ_{ref}	rad	Reference point for the heading
θ_t	rad	Heading of the mobile robot at time instant t
l_t	m	Travelled distance at time instant t
R	m	Turn radius of the mobile robot
\mathbf{u}_t	—	Input vector at time instant t
$\hat{\mathbf{u}}_t$	—	Input vector at time instant t subject to additive Gaussian noise
σ_d^2	m ²	Variance of the distance between mobile robot and obstacle line segment
$\sigma_{\theta_t}^2$	rad ²	Variance of the heading at time instant t
$\sigma_{x_t}^2$	m ²	Variance of the mobile robot in x -axis at time instant t
$\sigma_{x_t^j}^2$	m ²	Variance of square segment j in x -axis at time instant t
$\sigma_{y_t}^2$	m ²	Variance of the mobile robot in y -axis at time instant t
$\sigma_{y_t^j}^2$	m ²	Variance of square segment j in y -axis at time instant t
v	m/s	Linear velocity of the mobile robot
v_l	m/s	Linear velocity of the left wheel
v_{max}	m/s	Maximum velocity for the adaptive velocity planner
v_{min}	m/s	Minimum velocity for the adaptive velocity planner
U_i	V	Voltage of the motors at the i -th sampling instant
U_l	V	Applied voltage on the left drive wheel
U_r	V	Applied voltage on the right drive wheel
v_r	m/s	Linear velocity of the right wheel
v_t	m/s	Linear velocity of the mobile robot at time instant t
\hat{v}_t	m/s	Linear velocity of the mobile robot at time instant t subject to additive Gaussian noise
$\dot{\phi}_l$	rad/s	Angular velocity of the left drive wheel
$\dot{\phi}_r$	rad/s	Angular velocity of the right drive wheel
x	m	Position of the mobile robot in x -axis
x_{ref}	m	Reference point for the x -axis coordinate
x_t	m	Position of the mobile robot in x -axis at time instant t
\mathbf{x}_t	—	State vector at time instant t

y	m	Position of the mobile robot in y -axis
y_{ref}	m	Reference point for the y -axis coordinate
y_t	m	Position of the mobile robot in y -axis at time instant t
\mathbf{z}_t^i	—	Measurement vector of feature i at time instant t

List of Figures

1.1	Examples of different mobile robots deployed in industry, households, and space exploration.	2
1.2	Cycle of GA and HGA	4
1.3	Dubins and Reeds and Shepp curves from \mathbf{q}_S to \mathbf{q}_G	7
1.4	Canonical nonholonomic mobile robot kinematic model.	8
1.5	Prototype industrial cleaning robot.	9
1.6	Industrial cleaning robot.	10
1.7	Outline of the thesis.	14
2.1	Path representation of the mobile robot (solid: one time visit; dashed: repeated visit; dotted: return path to the starting position).	18
2.2	Motion behavior of the robot and recorded backtracking points (circles) of the TASP algorithm [84]	20
2.3	Motion behavior of the robot and recorded backtracking points (circles) of the BSA algorithm.	22
2.4	Initial TASP paths for the proposed algorithm.	22
2.5	Initial paths for the proposed algorithm.	23
2.6	Adjustment procedure in the GA.	26
2.7	Local improvement procedures in the HGA approach.	27
2.8	Flow chart of the GA and HGA approaches.	28
2.9	Desired angular velocity and the resulting current and voltage for acceleration, deceleration and straight motion (the black vertical lines divide acceleration, the straight motion and deceleration in the trajectory).	30
2.10	Desired angular velocity and the resulting current and voltage for turn motion. .	31
2.11	Desired angular velocity and the resulting current and voltage for U-turn motion.	31
2.12	Experiments of spiral path for fitness function validation.	33
2.13	Resulting path using HGA/BSA with the number of cell visits as fitness function.	33
2.14	Resulting path using HGA/Both and HGA/TASP with the energy-based fitness function.	34
2.15	Resulting path using HGA/BSA with the energy-based fitness function on a carpet-like surface.	36
2.16	Performance results for different starting positions and objectives	37
3.1	Differential-drive mobile robot and global and local reference frames.	40
3.2	Hybrid system for parking control.	41
3.3	Parking control strategy.	42

3.4	Blockdiagram of the system.	44
3.5	Simulation results of robot trajectories and state transitions.	47
3.6	Simulation results of wheel angular velocities.	47
3.7	Experimental results of robot trajectories and state transitions.	48
3.8	Experimental results of wheel angular velocities.	48
3.9	Experimental results of robot planar trajectories.	49
4.1	Convex obstacle representation in the configuration space.	52
4.2	Example motions for HA* using the velocity model.	53
4.3	Segmentation of a rectangular robot into smaller segments.	60
4.4	Rectangular robot with 95 % confidence ellipses.	62
4.5	The resulting path (solid red) and all expanded nodes (solid orange) using the ASR CCxHA* algorithm (the dashed ellipses are plotted every 5 th node with a confidence of 95 %).	66
4.6	The resulting path attaining the shortest distance (solid red) using the CCRRT algorithm and a “zig-zag” motion (dash-dotted blue) using the CCxRRT algorithm (the black circles emphasize the path segments with a “zig-zag” motion).	67
4.7	Resulting paths for all algorithms in a clustered, static environment.	69
4.8	The ASR CCHA* algorithm result at a selected node in the dynamic environment.	70
4.9	Resulting paths in the dynamic environment.	72
4.10	Results for different gains of the SCHA* algorithm.	74
4.11	Monte Carlo simulation for static and dynamic environments.	75
4.12	Confidence ellipse of a correlated covariance matrix.	76
4.13	Confidence ellipse with the isosceles triangle and the two circles.	77
4.14	95 % confidence ellipses with circles covering the confidence ellipse of the front left square segment.	78
4.15	Resulting paths of the planners in the static environment.	79
4.16	Resulting paths of the planners in the dynamic environment.	80
4.17	Monte Carlo simulation for static and dynamic environments of the confidence ellipse planners and selected planners of the previous section.	82
4.18	Feature detection of a static obstacle.	83
4.19	Resulting paths with adaptive velocities in the static environment.	87
4.20	Resulting paths with adaptive velocities in the dynamic environment.	87
4.21	Performance of the eigenvalue penalty in a static open environment.	88

List of Tables

2.1	Representation of the chromosome for the motion depicted in Figure 2.1.	21
2.2	Example of mutation operator.	25
2.3	Example of crossover operator.	25
2.4	Adjustment of a gene from an infeasible offspring.	26
2.5	Parameters for HGAs.	29
2.6	Obtained time and energy costs for motions on carpet-like and smooth surfaces, respectively.	29
2.7	Comparison of energy values in experiments and fitness function.	30
2.8	Comparison of HGAs with other approaches (grids are omitted in the maps for ensuring visibility).	32
2.9	Computation times of evolutionary algorithms for three environments.	35
3.1	Initial position and orientation \mathbf{x}_0 , control gains k_x, k_y and $k_{\theta_{\zeta_i}}$ for the controllers, and threshold values $\epsilon_x, \epsilon_\theta$, for simulation and experiment.	46
4.1	Traveled distance and computation time per node in static environment.	65
4.2	The traveled distance and computation time per node in a dynamic environment.	68
4.3	Traveled distance and computation time per node in a static environment for the SCHA* algorithm.	73
4.4	The traveled distance and computation time per node in the static and dynamic environments compared with selected planners from the previous section (the last two columns are omitted for brevity).	81
4.5	Comparison of the travelling time and computation time of the adaptive planners with existing approaches.	90

Bibliography

- [1] E. Aarts and J. K. Lenstra. *Local Search in Combinatorial Optimization*. Princeton University Press, 2003.
- [2] E. U. Acar, H. Choset, and J. Y. Lee. “Sensor-based coverage with extended range detectors”. In: *IEEE Transactions on Robotics* 22.1 (Feb. 2006), pp. 189–198.
- [3] E. U. Acar, H. Choset, A. A. Rizzi, P. N. Atkar, and D. Hull. “Morse Decompositions for Coverage Tasks”. In: *The International Journal of Robotics Research* 21.4 (Apr. 2002), pp. 331–344.
- [4] G. S. Aoude, B. D. Luders, J. M. Joseph, N. Roy, and J. P. How. “Probabilistically safe motion planning to avoid dynamic obstacles with uncertain motion patterns”. In: *Autonomous Robots* 35.1 (July 2013), pp. 51–76.
- [5] I. Ardiyanto and J. Miura. “Real-time navigation using randomized kinodynamic planning with arrival time field”. In: *Robotics and Autonomous Systems* 60.12 (2012), pp. 1579–1591.
- [6] E. M. Arkin, S. P. Fekete, and J. S. Mitchell. “Approximation algorithms for lawn mowing and milling”. In: *Computational Geometry* 17.1-2 (2000), pp. 25–50.
- [7] Y. Y. Aye, K. Watanabe, S. Maeyama, and I. Nagai. “Image-based fuzzy control of a car-like mobile robot for parking problems”. In: *2016 IEEE International Conference on Mechatronics and Automation*. 2016, pp. 502–507.
- [8] V. Badrinarayanan, A. Kendall, and R. Cipolla. “SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.12 (2017), pp. 2481–2495.
- [9] J. van den Berg, P. Abbeel, and K. Goldberg. “LQG-MP: Optimized path planning for robots with motion uncertainty and imperfect state information”. In: *The International Journal of Robotics Research* 30.7 (2011), pp. 895–913.
- [10] D. P. Bertsekas and J. N. Tsitsiklis. *Introduction to Probability*. Athena Scientific, 2002.
- [11] L. Blackmore, H. Li, and B. Williams. “A probabilistic approach to optimal robust path planning with obstacles”. In: *2006 American Control Conference*. 2006, 7 pp.-.
- [12] L. Blackmore. “A Probabilistic Particle Control Approach to Optimal, Robust Predictive Control”. In: *AIAA Guidance, Navigation, and Control Conference and Exhibit*.
- [13] L. Blackmore, M. Ono, A. Bektassov, and B. C. Williams. “A Probabilistic Particle-Control Approximation of Chance-Constrained Stochastic Predictive Control”. In: *IEEE Transactions on Robotics* 26.3 (June 2010), pp. 502–517.

- [14] L. Blackmore, M. Ono, and B. C. Williams. “Chance-Constrained Optimal Path Planning With Obstacles”. In: *IEEE Transactions on Robotics* 27.6 (Dec. 2011), pp. 1080–1094.
- [15] S. Bochkarev and S. L. Smith. “On minimizing turns in robot coverage path planning”. In: *2016 IEEE International Conference on Automation Science and Engineering (CASE)*. 2016, pp. 1237–1242.
- [16] S. D. Bopardikar, B. Englot, and A. Speranzon. “Multiobjective Path Planning: Localization Constraints and Collision Probability”. In: *IEEE Transactions on Robotics* 31.3 (June 2015), pp. 562–577.
- [17] J. Borenstein and Y. Koren. “The vector field histogram-fast obstacle avoidance for mobile robots”. In: *IEEE Transactions on Robotics and Automation* 7.3 (June 1991), pp. 278–288.
- [18] R. Bormann, F. Jordan, J. Hampp, and M. Hägele. “Indoor Coverage Path Planning: Survey, Implementation, Analysis”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 1718–1725.
- [19] A. Bry and N. Roy. “Rapidly-exploring Random Belief Trees for motion planning under uncertainty”. In: *2011 IEEE International Conference on Robotics and Automation*. 2011, pp. 723–730.
- [20] J. Carr. “An Introduction to Genetic Algorithms”. In: *Senior Project* 1.40 (2014).
- [21] C. Chen, M. Rickert, and A. Knoll. “Kinodynamic motion planning with Space-Time Exploration Guided Heuristic Search for car-like robots in dynamic environments”. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2015, pp. 2666–2671.
- [22] C. Chen, M. Rickert, and A. Knoll. “Motion planning under perception and control uncertainties with Space Exploration Guided Heuristic Search”. In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. June 2017, pp. 712–718.
- [23] C. Chen, M. Rickert, and A. Knoll. “Path planning with orientation-aware space exploration guided heuristic search for autonomous parking and maneuvering”. In: *2015 IEEE Intelligent Vehicles Symposium (IV)*. 2015, pp. 1148–1153.
- [24] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, S. Thrun, J. Latombe, and R. Arkin. *Principles of Robot Motion: Theory, Algorithms, and Implementation*. MIT press, 2005.
- [25] H. Choset. “Coverage for robotics - A survey of recent results”. In: *Annals of Mathematics and Artificial Intelligence* 31.1-4 (2001), pp. 113–126.
- [26] H. Choset and P. Pignon. “Coverage Path Planning: The Boustrophedon Cellular Decomposition”. In: *Field and Service Robotics*. London: Springer London, 1998, pp. 203–209.
- [27] K. Daniel, A. Nash, S. Koenig, and A. Felner. “Theta*: Any-Angle Path Planning on Grids”. In: *Journal of Artificial Intelligence Research* 39 (Sept. 2010), pp. 533–579.

-
- [28] K. Demirli and M. Khoshnejad. “Autonomous parallel parking of a car-like mobile robot by a neuro-fuzzy sensor-based controller”. In: *Fuzzy Sets and Systems* 160.19 (2009), pp. 2876–2891.
 - [29] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel. “Path Planning for Autonomous Vehicles in Unknown Semi-structured Environments”. In: *The International Journal of Robotics Research* 29.5 (Apr. 2010), pp. 485–501.
 - [30] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel. “Practical Search Techniques in Path Planning for Autonomous Driving”. In: *AAAI Workshop - Technical Report* (Jan. 2008).
 - [31] L. E. Dubins. “On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents”. In: *American Journal of Mathematics* 79.3 (1957), pp. 497–516.
 - [32] H. Durrant-Whyte. *Multi Sensor Data Fusion*. Australien Center for Field Robotics, The University of Sydney, 2001.
 - [33] D. Ferguson, N. Kalra, and A. Stentz. “Replanning with RRTs”. In: *2006 IEEE International Conference on Robotics and Automation (ICRA)*. 2006, pp. 1243–1248.
 - [34] O. Föllinger. *Regelungstechnik*, VDE-Verlag, 12. 2013.
 - [35] D. Fox, W. Burgard, and S. Thrun. “The dynamic window approach to collision avoidance”. In: *IEEE Robotics Automation Magazine* 4.1 (Mar. 1997), pp. 23–33.
 - [36] Y. Gabriely and E. Rimon. “Competitive on-line coverage of grid environments by a mobile robot”. In: *Computational Geometry* 24.3 (2003), pp. 197–224.
 - [37] Y. Gabriely and E. Rimon. “Spanning-tree based coverage of continuous areas by a mobile robot”. In: *Annals of Mathematics and Artificial Intelligence* 31.1 (2001), pp. 77–98.
 - [38] E. Galceran and M. Carreras. “A survey on coverage path planning for robotics”. In: *Robotics and Autonomous Systems* 61.12 (2013), pp. 1258–1276.
 - [39] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison Wesley, 1989.
 - [40] E. González, M. Alarcón, P. Aristizábal, and C. Parra. “BSA: a coverage algorithm”. In: *2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Vol. 2. 2003, pp. 1679–1684.
 - [41] E. González, O. Álvarez, Y. Díaz, C. Parra, and C. Bustacara. “BSA: A Complete Coverage Algorithm”. In: *2005 IEEE International Conference on Robotics and Automation (ICRA)*. 2005, pp. 2040–2044.
 - [42] L. Janson, E. Schmerling, and M. Pavone. “Monte Carlo Motion Planning for Robot Trajectory Optimization Under Uncertainty”. In: *Robotics Research: Volume 2*. Springer International Publishing, 2018, pp. 343–361.
-

- [43] P. A. Jimenez, B. Shirinzadeh, A. Nicholson, and G. Alici. “Optimal area covering using genetic algorithms”. In: *2007 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*. 2007, pp. 1–5.
- [44] G. Kahn, A. Villaflor, V. Pong, P. Abbeel, and S. Levine. *Uncertainty-Aware Reinforcement Learning for Collision Avoidance*. 2017.
- [45] R. E. Kalman. “A New Approach to Linear Filtering and Prediction Problems”. In: *Journal of Basic Engineering* 82.1 (Mar. 1960), pp. 35–45.
- [46] M. Kapanoglu, M. Alikalfa, M. Ozkan, A. Yazıcı, and O. Parlaktuna. “A pattern-based genetic algorithm for multi-robot coverage path planning minimizing completion time”. In: *Journal of Intelligent Manufacturing* 23.4 (2012), pp. 1035–1045.
- [47] M. Kapanoglu, M. Ozkan, A. Yazıcı, and O. Parlaktuna. “Pattern-Based Genetic Algorithm Approach to Coverage Path Planning for Mobile Robots”. In: *Computational Science – ICCS 2009*. Springer Berlin Heidelberg, 2009, pp. 33–42.
- [48] S. Karaman and E. Frazzoli. “Incremental Sampling-based Algorithms for Optimal Motion Planning”. In: *Robotics: Science and Systems*. Vol. 6. MIT Press Journals, 2011, pp. 267–274.
- [49] S. Karaman and E. Frazzoli. “Optimal kinodynamic motion planning using incremental sampling-based methods”. In: *49th IEEE Conference on Decision and Control (CDC)*. 2010, pp. 7681–7687.
- [50] S. Karaman and E. Frazzoli. “Sampling-based algorithms for optimal motion planning”. In: *The International Journal of Robotics Research* 30.7 (2011), pp. 846–894.
- [51] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller. “Anytime Motion Planning using the RRT*”. In: *2011 IEEE International Conference on Robotics and Automation*. 2011, pp. 1478–1483.
- [52] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars. “Probabilistic roadmaps for path planning in high-dimensional configuration spaces”. In: *IEEE Transactions on Robotics and Automation* 12.4 (1996), pp. 566–580.
- [53] H. K. Khalil. *Nonlinear Systems*. Prentice-Hall, 2002.
- [54] A. Khan, I. Noreen, and Z. Habib. “On Complete Coverage Path Planning Algorithms for Non-holonomic Mobile Robots: Survey and Challenges.” In: *Journal of Information Science & Engineering* 33.1 (Jan. 2017), pp. 101–121.
- [55] A. Khoukhi. “Data-driven multi-stage multi-objective motion planning of mobile robots, application to near minimum power fuzzy parking”. In: *Computers Electrical Engineering* 43 (2015), pp. 218–237.
- [56] S. Koenig and M. Likhachev. “Fast replanning for navigation in unknown terrain”. In: *IEEE Transactions on Robotics* 21.3 (June 2005), pp. 354–363.
- [57] S. Koenig and M. Likhachev. “D* Lite”. In: *National Conference on Artificial Intelligence*. 2002, pp. 476–483.

-
- [58] S. Koenig, M. Likhachev, and D. Furcy. “Lifelong Planning A”. In: *Artificial Intelligence* 155.1 (2004), pp. 93–146.
 - [59] K. Koide and J. Miura. “Collision Risk Assessment via Awareness Estimation Toward Robotic Attendant”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020, pp. 11011–11016.
 - [60] J. Kuffner and S. LaValle. “RRT-connect: An efficient approach to single-query path planning”. In: *IEEE International Conference on Robotics and Automation*. Vol. 2. 2000, pp. 995–1001.
 - [61] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, and J. P. How. “Real-Time Motion Planning With Applications to Autonomous Urban Driving”. In: *IEEE Transactions on Control Systems Technology* 17.5 (Sept. 2009), pp. 1105–1118.
 - [62] A. Lambert, D. Gruyer, and G. Saint Pierre. “A fast Monte Carlo algorithm for collision probability estimation”. In: *2008 10th International Conference on Control, Automation, Robotics and Vision*. 2008, pp. 406–411.
 - [63] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.
 - [64] S. M. LaValle. “Rapidly-Exploring Random Trees: A New Tool for Path Planning”. In: *Computer Science Department* 98.11 (Oct. 1998).
 - [65] S. M. LaValle and J. J. Kuffner. “Rapidly-Exploring Random Trees: Progress and Prospects”. In: *4th Workshop on Algorithmic and Computational Robotics: New Directions* (2000), pp. 293–308.
 - [66] S. LaValle. “Motion Planning Part II: Wild Frontiers”. In: *IEEE Robotics Automation Magazine* 18.2 (June 2011), pp. 108–118.
 - [67] S. M. LaValle. “Motion Planning Part I: The Essentials”. In: *IEEE Robotics Automation Magazine* 18.1 (Mar. 2011), pp. 79–89.
 - [68] S. M. LaValle and J. J. Kuffner. “Randomized Kinodynamic Planning”. In: *The International Journal of Robotics Research* 20.5 (2001), pp. 378–400.
 - [69] S. M. LaValle and R. Sharma. “On Motion Planning in Changing, Partially Predictable Environments”. In: *The International Journal of Robotics Research* 16.6 (1997), pp. 775–805.
 - [70] S. Lee, M. Kim, Y. Youm, and W. Chung. “Control of a car-like mobile robot for parking problem”. In: *1999 IEEE International Conference on Robotics and Automation*. Vol. 1. 1999, pp. 1–6.
 - [71] T.-K. Lee, S.-H. Baek, Y.-H. Choi, and S.-Y. Oh. “Smooth coverage path planning and control of mobile robots based on high-resolution grid map representation”. In: *Robotics and Autonomous Systems* 59.10 (2011), pp. 801–812.
 - [72] T.-H. Li and S.-J. Chang. “Autonomous fuzzy parking control of a car-like mobile robot”. In: *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 33.4 (July 2003), pp. 451–465.
-

- [73] M. Likhachev, G. Gordon, and S. Thrun. “ARA*: Anytime A* with Provable Bounds on Sub-Optimality”. In: *Neural Information Processing Systems (NeurIPS)*. Dec. 2003, pp. 767–774.
- [74] B. Luders, M. Kothari, and J. How. “Chance Constrained RRT for Probabilistic Robustness to Environmental Uncertainty”. In: *AIAA Guidance, Navigation, and Control (GNC) Conference*. Aug. 2010.
- [75] B. D. Luders and J. How. “Probabilistic Feasibility for Nonlinear Systems with Non-Gaussian Uncertainty using RRT”. In: *AIAA Infotech at Aerospace 2011*. Mar. 2011.
- [76] B. D. Luders, S. Karaman, E. Frazzoli, and J. P. How. “Bounds on tracking error using closed-loop rapidly-exploring random trees”. In: *2010 American Control Conference*. 2010, pp. 5406–5412.
- [77] B. D. Luders, S. Karaman, and J. P. How. “Robust Sampling-based Motion Planning with Asymptotic Optimality Guarantees”. In: *AIAA Guidance, Navigation, and Control (GNC) Conference*. Aug. 2013.
- [78] C. Luo and S. X. Yang. “A Bioinspired Neural Network for Real-Time Concurrent Map Building and Complete Coverage Robot Navigation in Unknown Environments”. In: *IEEE Transactions on Neural Networks* 19.7 (July 2008), pp. 1279–1298.
- [79] K. M. Lynch and F. C. Park. *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press, 2017.
- [80] R. Mannadiar and I. Rekleitis. “Optimal coverage of a known arbitrary environment”. In: *2010 IEEE International Conference on Robotics and Automation*. 2010, pp. 5525–5530.
- [81] N. A. Melchior and R. Simmons. “Particle RRT for Path Planning with Uncertainty”. In: *2007 IEEE International Conference on Robotics and Automation*. 2007, pp. 1617–1624.
- [82] X. Miao, J. Lee, and B.-Y. Kang. “Scalable Coverage Path Planning for Cleaning Robots Using Rectangular Map Decomposition on Large Environments”. In: *IEEE Access* 6 (2018), pp. 38200–38215.
- [83] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT press, 1998.
- [84] M. Mitschke, N. Uchiyama, and O. Sawodny. “Online Coverage Path Planning for a Mobile Robot Considering Energy Consumption”. In: *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*. 2018, pp. 1473–1478.
- [85] J. Miura and Y. Shirai. “Probabilistic Uncertainty Modeling of Obstacle Motion for Robot Motion Planning”. In: *Journal of Robotics and Mechatronics* 14.4 (2002), pp. 349–356.
- [86] M. Montemerlo et al. “Junior: The Stanford entry in the Urban Challenge”. In: *Journal of Field Robotics* 25.9 (2008), pp. 569–597.

-
- [87] P. Moscato and C. Cotta. “A Gentle Introduction to Memetic Algorithms”. In: *Handbook of Metaheuristics*. Springer US, 2003, pp. 105–144.
 - [88] P. Moscato, C. Cotta, and A. Mendes. “Memetic Algorithms”. In: *New Optimization Techniques in Engineering*. Springer Berlin Heidelberg, 2004, pp. 53–85.
 - [89] D. R. Musser. “Introspective Sorting and Selection Algorithms”. In: *Software: Practice and Experience* 27.8 (1997), pp. 983–993.
 - [90] K. Naderi, J. Rajamäki, and P. Hämmäläinen. “RT-RRT*: A Real-Time Path Planning Algorithm Based on RRT*”. In: *8th ACM SIGGRAPH Conference on Motion in Games*. MIG ’15. Association for Computing Machinery, 2015, pp. 113–118.
 - [91] A. Nash and S. Koenig. “Any-Angle Path Planning”. In: *AI Magazine* 34.4 (Sept. 2013), pp. 85–107.
 - [92] A. Nash, S. Koenig, and M. Likhachev. “Incremental Phi*: Incremental Any-Angle Path Planning on Grids”. In: *International Joint Conference on Artificial Intelligence (IJ-CAI)*. International Joint Conferences on Artificial Intelligence, 2009, pp. 1824–1830.
 - [93] J. Ng and T. Bräunl. “Performance Comparison of Bug Navigation Algorithms”. In: *Journal of Intelligent and Robotic Systems* 50.1 (2007), pp. 73–84.
 - [94] M. Ornik, M. Moarref, and M. E. Broucke. “An Automated Parallel Parking Strategy Using Reach Control Theory”. In: *IFAC-PapersOnLine* 50.1 (July 2017), pp. 9089–9094.
 - [95] S. Patil, J. van den Berg, and R. Alterovitz. “Estimating probability of collision for safe motion planning under Gaussian motion and sensing uncertainty”. In: *2012 IEEE International Conference on Robotics and Automation*. 2012, pp. 3238–3244.
 - [96] J. Petereit, T. Emter, C. W. Frey, T. Kopfstedt, and A. Beutel. “Application of Hybrid A* to an Autonomous Mobile Robot for Path Planning in Unstructured Outdoor Environments”. In: *ROBOTIK 2012; 7th German Conference on Robotics*. 2012, pp. 1–6.
 - [97] M. Przybyła, W. Kowalczyk, and K. Kozłowski. “Navigation Function Used for Parallel Parking in Restricted Area by a Differentially-Driven Mobile Robot”. In: *IFAC-PapersOnLine* 50.1 (2017). 20th IFAC World Congress, pp. 4312–4317.
 - [98] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. “ROS: an open-source Robot Operating System”. In: *ICRA Open-Source Software Workshop*. 2009.
 - [99] J. Redmon and A. Farhadi. “YOLOv3: An Incremental Improvement”. In: *arXiv preprint arXiv:1804.02767* (2018).
 - [100] J. A. Reeds and L. A. Shepp. “Optimal paths for a car that goes both forwards and backwards”. In: *Pacific Journal of Mathematics* 145.2 (1990), pp. 367–393.
 - [101] C. Rösmann, F. Hoffmann, and T. Bertram. “Integrated online trajectory planning and optimization in distinctive topologies”. In: *Robotics and Autonomous Systems* 88 (Feb. 2017), pp. 142–153.
-

- [102] C. Rösmann, F. Hoffmann, and T. Bertram. “Kinodynamic trajectory optimization and control for car-like robots”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017, pp. 5681–5686.
- [103] C. Rösmann, F. Hoffmann, and T. Bertram. “Planning of multiple robot trajectories in distinctive topologies”. In: *2015 European Conference on Mobile Robots (ECMR)*. 2015, pp. 1–6.
- [104] C. Rösmann, A. Makarow, and T. Bertram. “Online Motion Planning based on Nonlinear Model Predictive Control with Non-Euclidean Rotation Groups”. In: *arXiv preprint arXiv:2006.03534* (2020).
- [105] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2002.
- [106] A. E. F. Ryerson and Q. Zhang. “Vehicle Path Planning for Complete Field Coverage Using Genetic Algorithms”. In: *Agricultural Engineering International: the CIGR Ejournal IX* (July 2007).
- [107] M. G. Sadek, A. E. Mohamed, and A. M. El-Garhy. “Augmenting Multi-Objective Genetic Algorithm and Dynamic Programming for Online Coverage Path Planning”. In: *2018 13th International Conference on Computer Engineering and Systems (ICCES)*. 2018, pp. 475–480.
- [108] T. Sasaki, G. Enriquez, T. Miwa, and S. Hashimoto. “Adaptive Path Planning for Cleaning Robots Considering Dust Distribution”. In: *Journal of Robotics and Mechatronics* 30.1 (2018), pp. 5–14.
- [109] T. R. Schäfle, S. Mohamed, N. Uchiyama, and O. Sawodny. “Coverage path planning for mobile robots using genetic algorithm with energy optimization”. In: *2016 International Electronics Symposium (IES)*. 2016, pp. 99–104.
- [110] C. Schöller, V. Aravantinos, F. Lay, and A. Knoll. “What the Constant Velocity Model Can Teach Us About Pedestrian Motion Prediction”. In: *IEEE Robotics and Automation Letters* 5.2 (Apr. 2020), pp. 1696–1703.
- [111] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza. *Introduction to Autonomous Mobile Robots*. The MIT Press, 2011.
- [112] J.-J. E. Slotine and W. Li. *Applied Nonlinear Control*. Prentice Hall, 1991.
- [113] J. Song and S. Gupta. “ ϵ^* : An Online Coverage Path Planning Algorithm”. In: *IEEE Transactions on Robotics* 34.2 (Apr. 2018), pp. 526–533.
- [114] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, 2005.
- [115] S. Thrun et al. “Stanley: The robot that won the DARPA Grand Challenge”. In: *Journal of Field Robotics* 23.9 (2006), pp. 661–692.
- [116] C. Urmson et al. “Autonomous driving in urban environments: Boss and the Urban Challenge”. In: *Journal of Field Robotics* 25.8 (2008), pp. 425–466.

- [117] H. Vorobieva, S. Glaser, N. Minoiu-Enache, and S. Mammar. “Geometric Path Planning for Automatic Parallel Parking in Tiny Spots”. In: *IFAC Proceedings Volumes* 45.24 (Sept. 2012). 13th IFAC Symposium on Control in Transportation Systems, pp. 36–42.
- [118] J. W. J. Williams. “Algorithm 232: heapsort”. In: *Communications of the ACM* 7 (1964), pp. 347–348.
- [119] M. A. Yakoubi and M. T. Laskri. “The path planning of cleaner robot for coverage region using Genetic Algorithms”. In: *Journal of Innovation in Digital Ecosystems* 3.1 (June 2016), pp. 37–43.
- [120] S. Yang and C. Luo. “A neural network approach to complete coverage path planning”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 34.1 (Feb. 2004), pp. 718–724.
- [121] A. Zelinsky, R. Jarvis, J. C. Byrne, and S. Yuta. “Planning Paths of Complete Coverage of an Unstructured Environment by a Mobile Robot”. In: *International Conference on Advanced Robotics*. 1993, pp. 533–538.
- [122] Y. Zhang, H. Chen, S. L. Waslander, J. Gong, G. Xiong, T. Yang, and K. Liu. “Hybrid Trajectory Planning for Autonomous Driving in Highly Constrained Environments”. In: *IEEE Access* 6 (2018), pp. 32800–32819.
- [123] P. Zips, M. Böck, and A. Kugi. “Optimisation based path planning for car parking in narrow environments”. In: *Robotics and Autonomous Systems* 79 (2016), pp. 1–11.

Publications

Journals (peer-reviewed)

- **T. R. Schäfle**, N. Uchiyama. “Probabilistic Robust Path Planning for Nonholonomic Arbitrary-Shaped Mobile Robots Using a Hybrid A* Algorithm”. In: *IEEE Access* 9 (2021), pp. 93466-93479.
- **T. R. Schäfle**, M. Mitschke , and N. Uchiyama. “Generation of Optimal Coverage Paths for Mobile Robots Using Hybrid Genetic Algorithm”. In: *Journal of Robotics and Mechatronics* 33.1 (2021), pp. 11-23.

International Conferences (peer-reviewed)

- **T. R. Schäfle**, A. Tokui, and N. Uchiyama. “A Hybrid Systems Approach with Input-Output Linearization for Automotive Parking Control of a Nonholonomic Mobile Robot”. In: *2018 3rd International Conference on Advanced Robotics and Mechatronics (ICARM)*, 2018, pp. 1-6.
- **T. R. Schäfle**, S. Mohamed, N. Uchiyama, and O. Sawodny. “Coverage path planning for mobile robots using genetic algorithm with energy optimization”. In: *2016 International Electronics Symposium (IES)*, 2016, pp. 99-104.