Czech Technical University in Prague
Faculty of electrical engineering
Department of cybernetics

A guided approach to sampling-based motion planning

Doctoral thesis

Ph.D. Programme: Electrical Engineering and Information Technology (P2612)
Branch of study: Artificial Intelligence and Biocybernetics (3902V035)

Supervisor: Ing. Libor Přeučil, CSc.

August, 2015 Vojtěch Vonásek

Contents

Abstract

The thesis deals with the motion planning problem. In this problem, the task is to find a path or trajectory between two places in a known environment. Motion planning is mostly studied in robotics, but its applications are far beyond robotics in areas like computational biology or surgery. A wide range of motion planning problems can be solved using the concept of configuration space. Due to high number of dimensions of the configuration space, that is equal to the number of degrees of freedom of the robot, it is not possible to discretize the space and search it using standard state-space searching methods.

Sampling-based motion planner like Probabilistic Roadmaps of Rapidly Exploring Random Tree solves the planning problems by randomized sampling of the configuration space. A well know bottleneck of the methods is the narrow passage problem. In order to speed up motion planner and to increase reliability of the planners, we propose to utilize the knowledge of the workspace to help sample the configuration space. The knowledge is represented using a path, the guides the sampling in the configuration space from the start configuration to the goal configuration.

The guided sampling is studied in three challenging scenarios. The basic principle of the guided sampling is introduced on the example of motion planning for mobile robots, which requires to sample the three-dimensional configuration space. The low dimensionality of the configuration space allows us to compute the guiding path as a geometric path in the workspace using standard path planning methods.

A different approach to compute the guiding path is proposed solve the path planning problem for 3D objects, that requires to search the six-dimensional configuration space. The proposed method first solves a relaxed version of the problem by scaling down the geometry of the robot. The found solution is then iterative improved until the solution of the original problem is found.

Finally, a novel motion planner is proposed for motion planning for modular robots. Modular robots are formed by connecting basic robotic modules. These robots can be reconfigured to various shapes and they represent systems with more than 6 degrees of freedom. Motion planning for modular robots is challenging also due to necessity to control many actuators in order to achieve a motion of the whole robot. We propose a novel motion planning system that utilize several locomotion generators in order to realize basic motions of the robots — motion primitives. The proposed planner then constructs the plan using these primitives.

Acknowledgments

I would like to thank to my supervisor Dr. Libor Přeučil for his help and guidance during my PhD study, and for giving me opportunity to work on EU project Symbrion. I would like to thank to all my former colleagues from Intelligent and Mobile robotics group who were always ready to help with HW or SW problems. Special thanks go to Jan Faigl and Tomáš Krajník who directed my research to motion planning and who always had time time to encourage me and give me valuable suggestions. Thanks also go to my colleagues Lutz Winkler and Jens Liedke from Karlsruhe Institute of Technology who helped me to conduct experiments with modular robots.

Chapter 1

Introduction

The ability to move is a crucial skill of robots. Simple reactive motions can be generated based on actual sensory data, which is suitable for tasks like floor cleaning or lawn mowing. To achieve a motion in complex scenarios, e.g. for robots with many degrees of freedom, motion planning techniques are required.

The motion planning problem has been studied since the beginning of robotics. It should be noted that motion planning is sometimes called path planning (and vice versa) in the literature. Although these problems are similar, we can identify several differences. In the **path planning** problem, the task is to find a collision-free path in the environment for a holonomic robot. **The result of the path planning is a geometric path** in the environment, which can be represented e.g. as a sequence of waypoints. A geometric path however does not specify how fast a robot should traverse the path, so time information and information about control inputs are not considered in path planning. The task of **motion planning** (sometimes called **trajectory planning**) is to find a trajectory for a robot considering its geometry, motion model and time. **The result of motion planning is a trajectory** describing desired positions in time, sometimes together with control inputs that can be used to drive the robot.

1.1 Problem formulation

A robot operates in a workspace $\mathcal{W}$, which is $\mathcal{W} = \mathbb{R}^2$ in the case of 2D environments or $\mathcal{W} = \mathbb{R}^3$ in the case of 3D environments. Let $\mathcal{O} \subseteq \mathcal{W}$ denote the obstacle region in the workspace. In this thesis, 2D workspaces are described using the polygonal representation, i.e., each obstacle is described by a polygon. To represent 3D workspaces and robots, 3D triangle meshes are used.

A position of a robot is described by its configuration $q \in \mathcal{C}$, where $\mathcal{C}$ is the configuration space of the robot, i.e., the set of all possible configurations. The dimension of the configuration space equals to the number of degrees of freedom of the robot. Rigid body of the robot is denoted $B(q) \subset \mathcal{W}$, where positions of its vertices are determined by the current configuration $q \in \mathcal{C}$. The obstacle region of the configuration space $\mathcal{C}_{obstacle} \subseteq \mathcal{C}$ is defined as $\mathcal{C}_{obstacle} = \{q \in \mathcal{C} | B(q) \cap \mathcal{O} \neq \emptyset\}$. The free space $\mathcal{C}_{free}$ is then $\mathcal{C}_{free} = \mathcal{C} \backslash \mathcal{C}_{obstacle}$. In the rest of the text, we denote configurations located in $\mathcal{C}_{free}$ as free or collision-free, while configurations located in $\mathcal{C}_{obstacle}$ are referred to as non-free.

One of the fundamental problems studied in robotics is the path planning problem (also known as Piano mover's problem [151] or Generalized Mover's problem [206]):

Formulation 1.1.1 *The task of path planning is to find a continuous collision-free path* $\tau : [0, 1] \to \mathcal{C}_{free}$ *from the start configuration* $q_{start} \in \mathcal{C}_{free}$ *to a desired goal configuration* $q_{goal} \in \mathcal{C}_{free}$ *such that* $\tau(0) = q_{start}$, $\tau(1) = q_{goal}$ *and* $\tau(t) \in \mathcal{C}_{free}$ *for all* $0 \leq t \leq 1$.

The formulation of the path planning problem assumes a holonomic robot, that can move arbitrarily in the configuration space. For practical purposes, the path is often represented as a polygonal path (a sequence of waypoints) instead as a continuous path. This waypoint representation is considered in this thesis. The complexity of the problem for a robot with limited number of polyhedral bodies moving amongst polyhedral obstacles was shown to be PSPACE-hard [206]. Therefore, exact solutions can be found only in limited cases [147]. Optimal planning is NP-hard even for a point in 3D polyhedral environment without differential constraints [31]. The high complexity of the problem led to development of many heuristics that provide a solution in a reasonable computation time.

Motions of real robots are usually constrained due to kinematics and dynamics. The constraints restrict allowable velocities in the configuration space, which can be described using a forward motion model

$$\dot{q} = f(q, u), \tag{1.1}$$

where $u \in \mathcal{U}$ is a control input and $\mathcal{U}$ is a set of all possible control inputs. The motion model describes how a configuration q changes after a control input u is applied. The motion planning problem under differential constraints [151] can be formulated as follows.

Formulation 1.1.2 *The task of motion planning is to find an action trajectory $\tilde{u} : [0, T] \to \mathcal{U}$ defining how to apply actions to the system starting from the configuration $q(0) = q_{start}$ so that after integration:*

$$q(t) = q(0) + \int_0^t f(q(\tau), \tilde{u}(\tau))\, d\tau, \tag{1.2}$$

the resulting trajectory $q : [0, T] \to \mathcal{C}$ satisfies: $q(0) = q_{start}$, $q(T) = q_{goal}$, and $q(t) \in \mathcal{C}_{free}$ for all $t \in [0, T]$.

The above introduced first-order constraints are usually caused by kinematics of the robot. The dynamic constraints, e.g. velocity limits, can be described using second-order constraints $\ddot{q} = h(q, \dot{q}, u)$. By introducing new configuration $x = (q, \dot{q})$, we can shift the problem to new space $\mathcal{X}$, that is called phase space. The phase space has $2n$ dimensions, where first n components are configurations and next n components are their time derivatives. In literature, the phase space can be formulated to contain also high-order differential equations [152].

The planning problem under differential constraints of order at least two is called kinodynamic motion planning [151] and its formulation is similar to Formulation 1.1.2 except the configuration space is replaced by the phase space $\mathcal{X}$ and the states obtained using the integration have to hold all high-order constraints. Motion planning under differential constraints with obstacles is extremely difficult, and exact solutions are known only for the double integrator system for $\mathcal{C} = \mathbb{R}$ [189] or $\mathcal{C} = \mathbb{R}^2$ [30].

In the above described formulations of the path planning and motion planning problems, the only task is to find a feasible path or trajectory between two given configurations. The only criterion to be satisfied is the feasibility of the solution regardless its efficiency. Both problems can be formulated considering an optimality criterion such as time to reach the goal, length of the trajectory or consumed energy.

In this thesis, we study the path/motion planning problems without considering any optimality criteria as finding a feasible solution is already challenging for many motion planning problems. Non-optimal motion planners are often used to provide an initial solution to optimization-based approaches like evolutionary algorithms [287, 57] or model predictive control [118, 53]. It is necessary to compute the non-optimal plans as fast as possible, which motivates us to design a fast motion planner. Besides, the non-optimal plans are satisfactory in several applications like assembly/disassembly studies [237, 5, 36] or controller testing [127, 19]. Again, fast yet non-optimal planners are required in these applications.

1.2 Applications of motion planning

Motion planning has most applications in robotics, e.g. for path/trajectory computation of robotic manipulators, mobile robots, autonomous cars [64, 53, 144], Unmanned Aerial Vehicles [69, 275, 72, 273, 118, 200], humanoids [133, 88], planetary robots [139] and modular robots [33, 261, 255].

Applications of motion planning can also be found far beyond robotics. For example, motion planning is necessary for needle manipulation during surgical operations [274, 6, 180, 169, 182] and for tumor treatment [241]. Paths for 3D objects are required in assembly/disassembly studies in CAD systems [248, 36, 185, 227, 237, 5]. In these applications, providing a non-optimal yet feasible solution is satisfactory, as it is used to decide if the parts are maintainable or not. Motion planning can be used to design routes for evacuation planning [210] and to verify feasibility of escape routes for wheelchairs [87, 198]. Computer games can utilize motion planners to control bots [130, 134, 201, 110, 18, 115, 148].

In computational biology, motion planning has been used to study protein docking and protein folding problems. A protein structure can be considered as a robot with many degrees of freedom [119, 230] and the protein docking and protein folding problems can be formulated as path planning tasks [230, 7, 170, 55, 199, 228].

1.3 Motivation & thesis goals

Various applications of motion and path planning bring their own challenges. Some problems can be approached geometrically considering disc robots, while other problems require to consider complex geometries of the robots including their motion capabilities. Many problems that are different in terms of geometry or kinematics can be approached using the configuration space concept [147]. The configuration space allows us to convert complicated geometric models and motion constraints to a general problem of finding a path in the space.

The configuration space can be searched by sampling-based motion planners. The main idea of sampling-based methods is to randomly sample the configuration space in order to create its approximation. This approximation is called roadmap and it is represented as a graph. A path in the graph is then related to a trajectory in the workspace. Due to utilization of the configuration space concept, the sampling-based methods can solve various planning problems for robots with many degrees of freedom (DOF) as well as for systems with motion constraints.

Despite the versatility of the sampling-based planners, there are several issues that can decrease performance of the methods in certain scenarios. The well known bottleneck of these methods is the narrow passage problem. A narrow passage is a small free region of the configuration space, which position is usually unknown. A prohibitively large number of random samples must be generated over the whole configuration space in order to hit the narrow passage and to connect it to the roadmap.

The goal of the thesis is to introduce a novel and practical motion planner to cope with the narrow passage problem. The proposed planner is based on Rapidly Exploring Random Tree (RRT) [150] that builds the roadmap as a configuration tree rooted at the initial configuration q_{start}. The RRT method has been selected due to its ability to cope with differential constraints, which is necessary for practical application in mobile and modular robotics. The main idea of the proposed planner is to utilize a guiding path computed from the start configuration to the goal configuration in order to sample the configuration space. We propose several methods to compute the guiding path and we will show how to apply the guiding principle for motion planning of mobile and modular robots as well as for path planning of 3D objects.

The effect of guided sampling is twofold. First, the guiding path helps to sample narrow passages, which increases probability of finding a solution in a given amount of time. This is depicted in Fig. 1.1, where the roadmap (tree) has to pass the narrow passage in order to connect the start and goal configurations. The tree is quickly built in the first room, but the narrow passage is not entered due to its low volume, and the goal configuration is not reached. If the guided sampling is used, the tree can enter and grow through the narrow passages and consequently it can reach the goal configuration.

Second, guided sampling decreases the size of the tree, as the random samples are generated mainly along the guiding path. The second effect is important especially in motion planning under differential constraints, where satisfying the constraints may be time consuming and exploration of unnecessary parts of the configuration space should be suppressed. An example is motion planning for modular robots that requires a time-consuming physical simulation to properly model motion of the robots in an environment (Fig. 1.2). Decreasing the number of nodes in the tree significantly decreases runtime of the planning process.

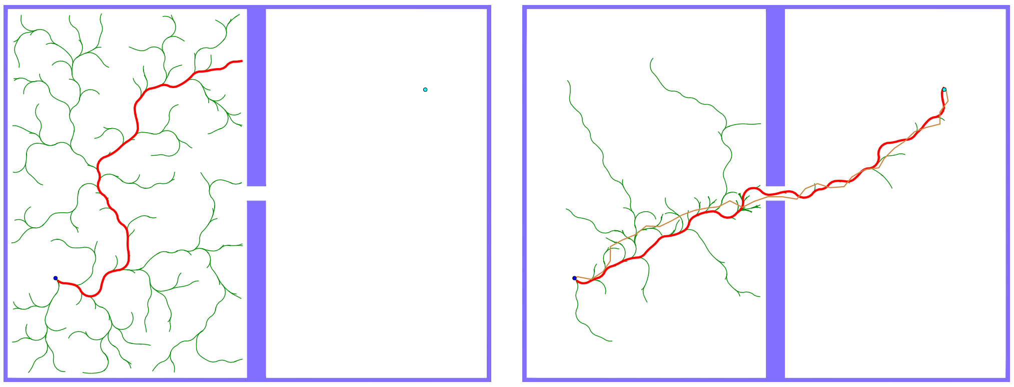

(a) uniform sampling (b) guided sampling

Figure 1.1: Difference between trees constructed by RRT using uniform samples (a) and samples generated along a guiding path (b). The configuration trees are depicted in green with a highlighted red trajectory. The trees are constructed using 2000 iterations of the the RRT method. The left tree has $\sim$ 1500 nodes and the right tree has only $\sim$ 800 nodes. The configuration tree built using the uniform samples explores more the configuration space before the narrow passage is entered. Contrary, the guiding path (brown) helps to quickly traverse the narrow passage and consequently, the goal configuration is reached in less number of iterations. Moreover, the built configuration tree is smaller.

Due to high complexity of the motion planning problem it is believed, that there is no universal algorithm that can efficiently solve instances of all types [75, 77]. To fulfill the aim of

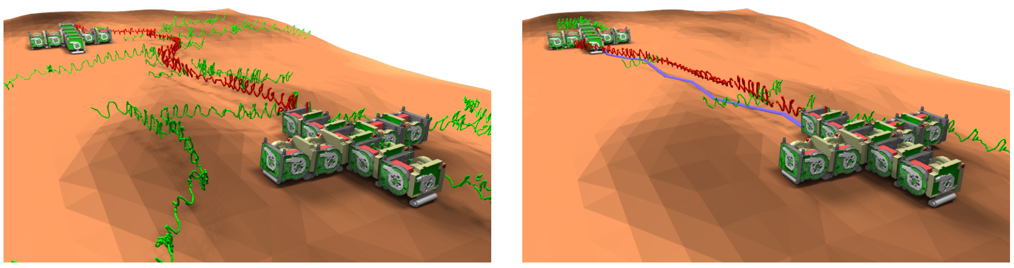

(a) uniform sampling (b) guided sampling

Figure 1.2: The effect of the guided sampling in motion planning for modular robots. The configuration tree is depicted in green, the resulting trajectory in red. The result of the uniform sampling is a tree covering large area in the space (a), while the guided sampling steers the configuration tree directly towards the goal configuration (b). The motion planning using the guided sampling therefore needs less number of iterations to reach the goal configuration. The guided path is depicted in blue.

the thesis, i.e., to provide a practical motion planner, we will study the herein proposed guiding principle in three typical motion planning tasks. The tasks are selected to represent planning problems in three-, six-, and high-dimensional configuration spaces. The increasing dimension causes, that it is more difficult to estimate narrow passages from the description of the workspace. The motion planning is studied in the following scenarios.

- **Motion planning for mobile robots in 2D environments**

  In this task, collision-free trajectories need to be found for the mobile robots with three degrees of freedom. The trajectories have to be computed considering motion models of the robots. Despite low dimension of the configuration space, the problem is challenging due to the presence of the narrow passages. To cope with the narrow passage problem, we propose a novel sampling-based planner. This novel strategy, referred to as RRT–Path (Rapidly Exploring Random Tree with guiding Path) in the rest of the text, utilizes a geometric description of the workspace to speed up the search in the configuration space. RRT–Path samples the configuration spaces primarily along the guiding path, which speeds up growth of the configuration tree towards the goal configuration.

- **Path planning of 3D rigid bodies in 3D workspaces**

  In the second domain, a collision-free path has to be found for a 3D rigid object moving in a 3D workspace. This requires to work in a 6D configuration space, as both 3D position and 3D orientation of the object have to be considered. The problem is challenging not only due to more DOFs, but especially due to occurrence of complicated narrow passages, that cannot be easily predicted based on geometric description of the objects and obstacles. We propose a novel algorithm for finding paths for the 3D rigid objects. The proposed method first solves a relaxed version of the problem and then it iteratively improves this initial result to solve the original problem. The improvement of the relaxed solution is realized using the guiding principle. The proposed method, called RRT–IS (RRT with Iterative Scaling) is an extension of RRT–Path to the six-dimensional configuration space.

- **Motion planning for modular robots**

  The task in the third scenario is to find a feasible trajectory for a modular robot while avoiding collisions with obstacles. Modular robots consist of many basic mechatronic modules and they move by controlling many actuators. Motion planning for modular robots leads to a search in a high-dimensional configuration space (with more than 6 dimensions). This task is challenging also due to necessity to derive control signals for many actuators considering a black-box motion model. We propose a novel extension of RRT for the purpose of modular robots. The method utilizes a set of locomotion generators to generate basic motion primitives of the robots and it is referred to as RRT–MP (RRT with Motion Primitives). The utilization of motion primitives significantly reduces complexity of the planning task, as it does not depend on the number of the actuators, but only on the number of used primitives. To further increase speed of the planning process, the guiding principle is used.

1.4 Thesis outline

The basic path planning and motion planning techniques are described in Chapter 2 with stress to sampling-based motion planners. A novel motion planner, called RRT–Path, that combines path planning and the sampling-based principle is proposed in Chapter 3. The RRT–Path planner is experimentally verified and compared to several state-of-the-art methods in Chapter 4. An extension of RRT–Path for the application of path planning of 3D rigid objects is described in Chapter 5. This novel algorithm, called RRT–IS, is compared to state-of-the-art methods.

The experiments are described in Chapter 6. Chapter 7 is dedicated to motion planning for modular robots using a novel extension of RRT called RRT–MP. Experimental verification of the RRT–MP planner in simulation as well as on physical robots is described in Chapter 8. Finally, conclusion and future work are presented in Chapter 9.

1.5 Overview of thesis contributions

The development of motion planning methods presented in the thesis has been inspired by several applications, mainly in mobile and modular robotics. Particular contributions of the thesis together with references to related publications are as follows.

- Motion planning for 2D mobile robots (Chapter 3)
  - Novel principle for guided sampling of configuration space called RRT–Path. Initial version of the method was published in [252], extended version in [253].
  - Utilization of guided sampling of configuration space for modular robots [259].
- Path planning for 3D objects (Chapter 5)
  - The method, called RRT–IS (RRT with Iterative Scaling), was published in [253].
  - A novel Hedgehog in the cage benchmark, was accepted to the set of benchmarks for 3D motion planning [141].
- Motion planning for modular robots (Chapter 7)
  - A naive implementation of RRT for modular robots utilizing random control signals [255].
  - Comparison of several RRT-based planners in the task of motion planning for joint-controlled modular robots [255].
  - A novel motion planner for modular robots with motion primitives called RRT–MP (RRT with Motion Primitives). Preliminary idea was presented at ICRA 2013 [260], the final version of the planner was published in Robotics and Autonomous Systems journal [263].
  - The RRT–MP planner was used in Symbrion/Replicator EU projects [156].
  - Fast motion planning for modular robots moving on a plane using simplified motion model. The method was presented at ICRA 2014 [264].
  - A fast method for optimization of locomotion gaits using Particle Swarm Optimization [258].
  - Guided sampling for motion planning for modular robots [259].
  - Robot3D simulator for modular robots [271]. The simulator was used in the Symbrion/Replicator projects to study artificial evolution of modular robots. Another simulator called Sim [254] was developed for fast simulation on computational grids. The Sim simulator provides general physical simulation, and it allows users to create their own robots. Besides utilization of the Sim simulation in Symbrion/Replicator projects, it was used in the task of formation driving [218, 219, 220]. Both simulators were co-developed by the author of this thesis.

Chapter 2

Related work

Path planning and motion planning problems have been studied for decades in robotics. The early path planners solved the problem by a deterministic search of a discrete state-space, e.g. by finding path in a grid representing the workspace. These methods are limited for robots with few degrees of freedom, for which the discretized state-space can be prepared. In late 1980s, randomized planners were introduced to cope with robots with more degrees of freedom. Later, the randomization techniques were proposed to sample the configuration space, which further enabled to consider differential constraints. This chapter briefly reviews the path planning and motion planning approaches with the emphasis to the sampling-based motion planners.

2.1 Basic path planning methods

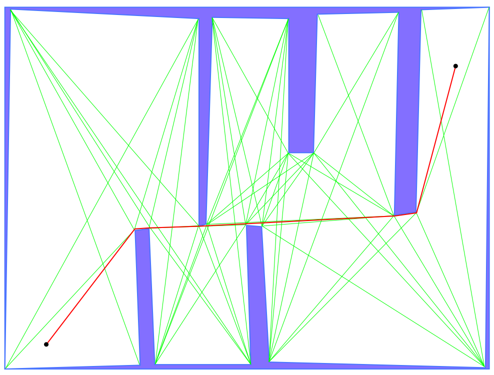
Basic path planning techniques rely on a graph-based description of the workspace, in which a path can be found using standard graph-search methods like A* or Dijkstra. In the Visibility graph approach (VG) [147], the graph is constructed from vertices of the polygons describing the obstacles. If two vertices in a polygonal map can be connected by a collision-free line, the corresponding vertices in the graph are connected by an edge. In the naive approach of VG construction, all possible pairs of nodes are considered and tested whether a line segment connecting them intersects any obstacle. This naive method has complexity $O(n^3)$, where n is the number of vertices of the polygonal representation. The intersection test can be speeded up using a line-sweep method, which improves the overall complexity to $O(n^2 \log n)$ [50]. An example of a VG is depicted in Fig. 2.1a.

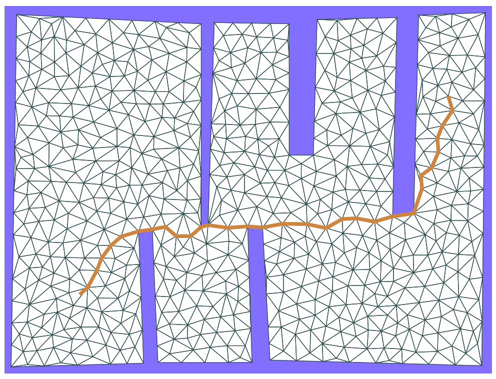
Decomposition-based techniques split the free parts of the workspace into a set of non-overlapping regions. The relations between the regions are then described in a graph, where the nodes represent the regions and the edges connect neighboring regions. To find a path between two places, first the regions containing start and goal points are determined and the path is found in the graph between corresponding nodes. Several methods for workspace decomposition have been proposed such as the vertical cell decomposition [37] or triangulation (Fig. 2.1b).

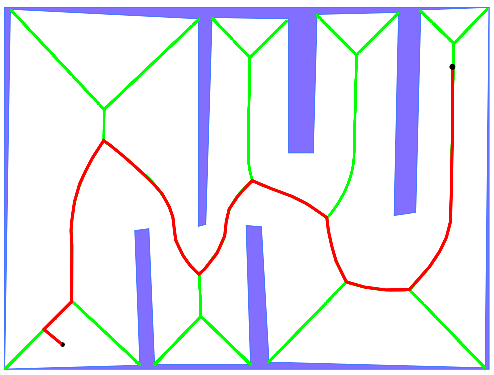
The VG approach and the decomposition-based methods can provide a path that is too close to obstacles. A path with the maximal clearance can be computed using Voronoi diagram [14]. The point Voronoi diagram of a set of n points can be computed in $O(n \log n)$ time using Fortune's algorithm [67]. Generalized Voronoi Diagram (GVD) [29] is more suitable for polygonal domains, as it can compute the diagram considering line segments. The complexity of GVD is $O(m \log m)$, where m is the number of disconnected line segments. The example of GVD is depicted in Fig. 2.1c. GVD is suitable for environments with narrow corridors, where paths with maximal clearance are preferred. However, the paths provided by GVD are usually longer in large free areas. A combination of Voronoi diagram and Visibility graph was proposed in [268].

The above mentioned path planning methods consider only a point robot. To consider a non-point robot, the polygonal map needs to be dilated by computing Minkowski sum between the obstacles and a disk representing the robot.

Although the geometric path planning approaches seem to be straightforward and easy to implement, it is not trivial to achieve a robust implementation [86], because the methods may suffer due to degenerate input data describing the workspace obstacles. An example of the degenerate input are three points lying on a same line, which can cause a problem in the computation of visibility in the VG approach. Further problems are caused by limited arithmetic precision of computer variables. To increase robustness of the computations, the input data should be preprocessed. For example, collinear or almost-collinear points can be removed using Ramer-Douglas-Peucker algorithm [89] or using Discrete Curve Evolution [146].


(a) Visibility graph


(b) Triangulation


(c) Generalized Voronoi diagram

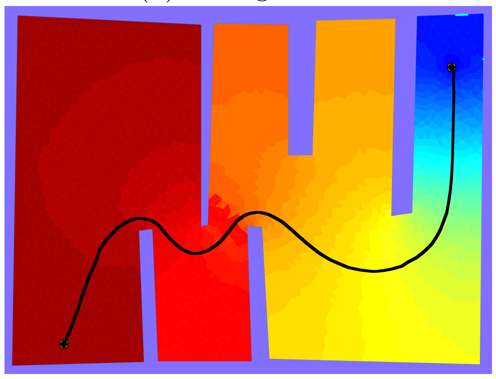

(d) Potential field

Figure 2.1: Examples of path planning in polygonal domain. The potential field was computed using [164].

In the potential field approach, a robot is considered as a positively charged particle that moves under the influence of the potential field towards a negatively charged goal configuration [125]. The obstacles have assigned a positive charge to repeal the robot. Example of a potential field is depicted in Fig. 2.1d. A path in the potential field is found using a gradient descent method. The gradient descend is guaranteed to converge to a local minimum, but it is not guaranteed that this minimum is the global minimum representing the goal configuration. If a local minimum is reached, the gradient descend algorithm is not able to escape it to reach the goal. The problem with local minima can be avoided if a navigation function is used [45]. A navigation function is designed to have only one (global) minimum.

The path planning problem has been studied also using soft-computing methods like genetic algorithms [229], evolutionary-based approaches [287], ant colony optimization [73], etc. The soft-computing techniques often represent the path of a robot using splines or polynoms, whose parameters have to be optimized to avoid collisions with the obstacles [39]. An important advantage of the soft-computing techniques is that they can consider an optimality criteria such as traveled distance or time required to traverse the path. The performance of the soft-computing optimization techniques may be decreased if finding of the initial feasible solution is already a challenging problem. This problem arises e.g. if the cost function describing quality of the solution cannot distinguish between unfeasible solutions, and consequently, it does not provide any selection pressure to the evolutionary process. It is therefore suitable to provide an initial feasible solution to the optimization-based methods. A comprehensive survey of existing soft-computing techniques for path and motion planning can be found in [166].

An advantage of the basic planning methods is the fast computation. However, the methods are tightly coupled with the geometric description of the environment and to consider robot's geometry, the environment needs to be dilated using Minkowski sum. The Minkowski sum needs to be computed also for a set of considered rotations of the robot. This practically limits usage of the basic path planners to simple mobile robots, that can be approximated by a disc.

2.2 Randomized motion planning

Motion planning under differential constraints, where also motion models of robots, their geometries, and additional constraints need to be considered, can be solved using the configuration space rather than using the workspace. The configuration space allows us to describe different motion planning problems and the problem of finding a path or a trajectory in the workspace is converted to the problem of searching a path in the configuration space. The main difference between planning in the workspace and planning in the configuration space is caused by the description of the obstacles. The workspace obstacles $\mathcal{O}$ are described explicitly e.g. using the polygonal representation, which allows us to directly construct Visibility graphs or Voronoi diagrams. In contrary, obstacles in the configuration space are described implicitly, i.e., as a set of configurations $q \in \mathcal{C}$ satisfying $R(q) \cap \mathcal{O} \neq \emptyset$. Due to the implicit representation of the obstacles in the configuration space, the path planning methods designed for workspaces cannot be used to search a path in the configuration space.

A naive approach to find a path in the configuration space would be to discretize it, e.g. into a grid, and search the grid using A* or other state-space search methods. An obvious disadvantage of this approach is the exponentially-growing number of discretized cells with the dimension of the configuration space. Moreover, state-spaces of most real systems are infinite, therefore discretization is intractable [151].

The configuration space can be searched using a randomized method without need to discretize the whole space. One of the first randomized planners was the Randomized Path Planner (RPP) [11]. RPP uses several potential fields computed on a grid that is defined over the configuration space. Each potential field corresponds to a control point on a robot. A potential of each cell is defined as a combination of the potential fields in the workspace. The planner starts at an initial state and descends the gradient of the potential field until a goal or a local minimum is reached. If the goal state is reached, the algorithm terminates. To escape from the local minimum, several random walks are performed. If the planner escapes it, the gradient descend is performed until the goal state or another local minimum is reached, or a predefined planning time is elapsed. The planning time must be limited, because RPP is probabilistically complete [143] and it cannot recognize that no solution exists. To speed up the escape from a local minimum, the configurations connected in previous iterations can be used to build a roadmap graph [34]. If a new local minimum is reached, the planer attempts to reach the nearest configuration in the graph. The local minimum is then escaped using a path in the graph. The utilization of several potential fields allows RPP to compute trajectories for many-DOF systems [110].

Two planners (global and local) are used in the Z^3 planner [81]. The task of the global planner is to place random subgoals in the configuration space, while the local planners tries to find a connection between the subgoals. The local planner is realized using a potential field. If both start and goal configurations are connected to a subgoal, the planner terminates, because the path can be found in the graph of the subgoals. Otherwise it attempts to connect unconnected subgoals to other subgoals.

In the Ariadne's clew algorithm [168], two main procedures are used: Explore and Search. The Explore phase adds new configurations to a tree rooted at the initial configuration. The task of the Search procedure is to connect known configurations to the goal. Both procedures are formulated as optimization tasks and they are solved using a genetic algorithm. For example, the task of the optimization in the Explore procedure is to place a new configuration to maximize its distance to all known configurations.

In early 1990s, several papers proposed to sample the configuration space using randomized methods to create an approximation of the space, in which a solution can be found [162]. The main idea of these methods, called *sampling-based motion planners*, is to create an approximation of $\mathcal{C}_{free}$ by random sampling of the whole configuration space and using collision detection to determine the free samples. This approximation is called *roadmap* and it is represented as a graph. A path in the graph then corresponds to a trajectory or path in the workspace. One of the first attempts to build a roadmap of the configuration space using the random samples was proposed in Horch's planner [92]. The vertices in the roadmap are random configurations in the configuration space, and the edges connect them with straight lines if possible. If a subgraph is disconnected, a random ray is "shot" from a vertex of the graph. If the ray hits an obstacle, the contact configuration on the boundary of the obstacle is computed, added to the roadmap and connected with its nearest neighbors. If this contact configuration cannot be added to the roadmap, a new random ray is shot from this configuration.

The early randomized planners like Z^3, Ariadnes's clew and Horch's planner were able to solve problems with more degrees of freedom than other planners of that time. These planners introduced a new paradigm in the motion planning: creating a plan by randomized sampling of the configuration space. This idea was further investigated, which resulted in the nowadays most used sampling-based planners: Probabilistic Roadmaps (PRM) [121] and Rapidly Exploring Random Trees (RRT) [150].

2.3 Probabilistic roadmaps (PRM)

PRM [121] builds the roadmap in two steps. First, random samples drawn from uniform distribution are generated in $\mathcal{C}$ and tested for collisions using collision detection. The collision-free samples are stored in the roadmap. After the random samples are generated, they are connected to their neighbors using a local planner. For simplicity, the straight-line planner is used as the local planner. The straight-line planner connects two configurations by a straight line segment. The line segment is tested for collisions at discrete points computed with resolution ε. To find a trajectory or path in the workspace, both initial and goal configurations are connected to the roadmap and the path between them is found using a graph-search method like Dijkstra's algorithm. The search for the path between the start and goal configurations is called *query phase*. The example of a roadmap construction is depicted in Fig. 2.2.

Depending on the local planner, PRM can be used for both path and motion planning. The path planning problem is solved using the straight-line local planner, that simply connects two configurations by a line segment. A more advanced local planners are required for motion planning, especially if differential constraints need to be considered. The local connection under differential constraints can be made e.g. using RRT method [202, 46, 224].

PRM is suitable in situations, where multiple planning queries have to be answered in the same (static) environment, e.g. for motion planning of a robotic manipulator in the pick & place task. The relatively high time required to prepare the roadmap is not an issue, as the roadmap is built only once. A modification of PRM for motion planning in dynamic environments was proposed in [277, 129, 159].

One of the time consuming routines is collision detection that is used to determine feasibility

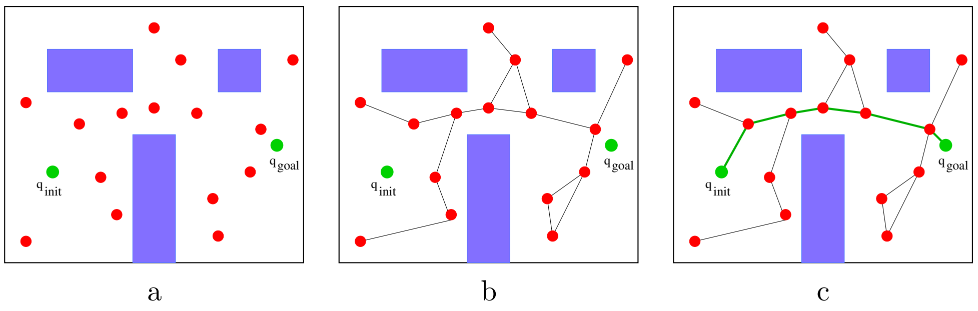


Figure 2.2: The configurations sampled in the first phase of the PRM (red). Initial and goal configurations are not necessary sampled (a). Roadmap constructed from the samples using the straight-line planner (b). In the query phase, both start and goal configurations are connected to the roadmap and a path is found (c).

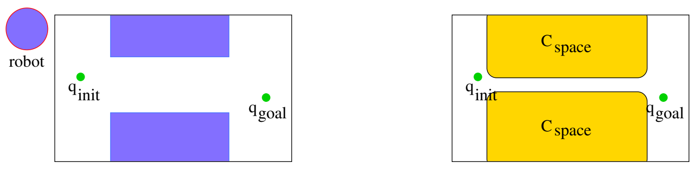
of the samples and edges. During the roadmap construction process, many samples are tested for collisions even though they are not used in the final path. In Lazy-PRM [17], collision detection at the nodes and the edges is postponed to the query phase. If the path found in the roadmap is completely collision-free, it is returned. Otherwise, colliding nodes and edges are iteratively removed from the roadmap and the path is searched again in the updated roadmap until a valid path is found. This speeds up the planning process in applications where the robot and workspace geometry consist of hundred thousands of elements [217]. The idea of the postponed collision detection approach is also part of other PRM-based planners [217, 231, 16, 185, 95].

In Fuzzy-PRM [185], collision detection is performed only for the nodes during the construction of the roadmap and the edges are not checked. Instead, the probability of being valid is estimated according to the length of the edge. In the query phase, the path with the largest probability of being valid is selected and its edges are validated using the collision detection mechanism and updated if necessary. The process is repeated until a valid path is found. Contrary to Lazy-PRM, which postpones collision detection of both the nodes and the edges, Fuzzy-PRM postpones only collision detection of the edges.

2.3.1 Narrow passage problem

A well known bottleneck of sampling-based methods is the narrow passage problem [120, 97]. Narrow passages are small regions in the configuration space whose removal changes connectivity of the space. The volume of a narrow passage is small in comparison to the volume of the whole configuration space, and it is therefore difficult to obtain sufficient number of samples there if the samples are drawn from uniform distribution.

To cope with the narrow passages, the distribution of the random samples should be modified in order to sample the passages densely. The probability distribution however cannot be changed so easily, as the locations of narrow passages are unknown. Although their shape and volume depend on the shape of the robot and obstacles, it is not always possible to estimate their



(a) Workspace (b) Configuration space

Figure 2.3: Example of a workspace without narrow passage (a). Configuration space of a disc robot contains a narrow passage (b).

positions only from the description of the workspace. For example, a workspace depicted in Fig. 2.3a does not contain any narrow region, but the related configuration space of the circular robot (Fig. 2.3b) contains a narrow passage.

Due to difficulty to estimate the positions of the narrow passages from the description of the workspace, several methods attempt to evaluate the random samples before these are added to the roadmap. In the Gaussian sampling strategy [191], the samples are generated more frequently around obstacles. A random configuration $q_1 \in \mathcal{C}$ is generated uniformly and another random configuration $q_2 \in \mathcal{C}$ is drawn around using Normal distribution $N(q_1, \sigma^2)$. If both q_1 and q_2 are free or both are non-free, they are not added to the roadmap. If only one of these configurations is free, then the free one is added to the roadmap. The disadvantage of this strategy is the necessity to choose suitable value of the parameter σ^2, which depends on the used map and the shape of the robot. The Bridge-Test sampling [93] employs the Gaussian strategy to generate two samples in order to compute their midpoint. If the midpoint is collision-free and both end configurations are non-free, the midpoint lies in a narrow passage, and it is added to the roadmap. Both Gaussian and Bridge-Test strategies have to be combined with the uniform sampling in order to ensure sampling of free-regions [236]. Despite the simplicity of these two modifications, they were proven to significantly improve performance of PRM [96, 78, 80, 267].

In the Visibility roadmap approach [149], uniformly distributed samples are used to build the roadmap. A new node can be added to the roadmap only if it is not visible from other nodes in the roadmap or if it is visible from at least two nodes in a different component of the roadmap. This minimizes the number of the nodes in the roadmap, however the visibility check increases runtime of the roadmap construction process.

2.3.2 Workspace-based sampling

The sampling distribution can be modified based on obstacles in the workspace as the narrow passages in the workspace are believed to be related with narrow passages in the configuration space [138]. To localize the narrow passages in the configuration space, authors of [138] suggest to construct tetrahedralization of the workspace and to compute importance of each face. The faces located at the boundary of the free space and the obstacles have more importance. The importance of each tetrahedron is computed as an average importance of its faces. To generate a random sample, first a tetrahedron is selected randomly according to the probability distribution given by the importance of the tetrahedrons. A random sample is then generated uniformly in the selected tetrahedron. The disadvantage of the method is the high computational burden of the tetrahedralization. Similar approach is used in [247], where the workspace is decomposed into cells that are labeled by the watershed algorithm to find borders between open and close regions. The border cells are then sampled more dense.

Methods [270, 68, 84, 90, 276] generate random samples around the medial axis, which helps to sample narrow passages more densely. This requires to precompute the medial axis, which can be time consuming. Another approach is to shift the random samples towards the medial axis [8] or into a direction of estimated medial axis [91].

2.3.3 Adaptive sampling

The workspace-based sampling is useful for robots with few degrees of freedom, because the narrow passages in the configuration space can be estimated from the narrow passages of the workspace. The importance of the workspace knowledge however decreases as the dimension of the configuration space increases [138]. For example, two narrow passages can be detected in the workspace of the Alpha puzzle problem [141] (Fig. 2.4). Configuration of the moving part can be described by its 3D position and 3D rotation. The only knowledge we can extract from the workspace is the 3D position of the narrow passage in the workspace (Fig. 2.4b), but the

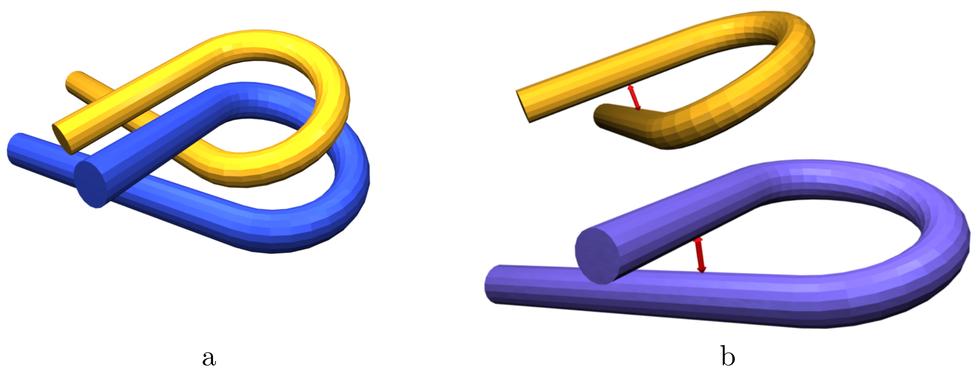


Figure 2.4: Alpha puzzle problem [141] consists of two solid objects that need to be removed from each other (a). The red arrows denote narrow passages of the geometric models (b).

exact position and shape of the corresponding narrow passage in the configuration space remain unknown.

A solution is to estimate the difficult parts of the configuration space on-line and to change the distribution of random samples accordingly. This requires to maintain a separate model of the configuration space to discover the possibly difficult parts. Such a model can be realized using a mixture of Gaussians [23, 25], a locally weighted regression [24] or nearest-neighbor model [26].

The performance of the sampling-based methods varies according to type of environment and used robot. The planners are usually tuned for some type of environment and their performance can degrade in different environments [75, 77]. To create a more universal planning system, multiple motion planners can be combined. The environment can be split to several regions in which specialized planners are used [173, 174, 171]. To divide the configuration space into several possibly overlapping regions, features like the number of obstacles in the region, the radius of the largest free ball together with suitable performance indicators (e.g. the number of nodes in the region roadmap or the connectivity ratio) are measured. A machine learning technique is then used to classify the regions to several classes (free, cluttered, narrow-passage, non-homogeneous). A suitable motion planner is then selected for each region based on the classification. If the region cannot be classified, it is further split into smaller regions and the classification process is repeated. The disadvantage of this approach is the necessity to provide labeled data for the employed supervised machine learning. Later, authors improved the system by employing an unsupervised strategy for region classification [240]. A bank of several planners with known behavior is utilized also in [246]. In this approach, the configurations added to the roadmap are classified as free-space samples or narrow passage samples. A suitable local planner is then selected in order to connect two nodes according their classification.

2.4 Rapidly Exploring Random Trees

Rapidly Exploring Random Tree method was introduced by LaValle [150] in 1998. The RRT method builds a roadmap as a tree rooted at q_{start}. The basic RRT algorithm works as follows. In each iteration, a random sample q_{rand} is chosen from $\mathcal{C}$ and the nearest node q_{near} in the tree to q_{rand} is found. The node q_{near} is expanded using a local planner to obtain a set of new configurations reachable from q_{near}. The nearest configuration towards q_{rand} is selected from this set and added to the tree. The edge from q_{near} to the newly added configuration contains control inputs used by the local planner to reach the new configuration. The algorithm terminates if the distance between a node in the tree and q_{goal} is less than d_{goal} or after I_{max} of planning iterations. The main loop of RRT is listed in Alg. 1 and the expansion procedure is described in

Alg. 2. The resulting trajectory is represented as a sequence of nodes (configurations) together with control information of each edge (e.g. a control input and time). RRT can therefore provide both paths and trajectories.

The RRT algorithm differs from PRM in two main aspects. First, the configuration tree built by RRT cannot have cycles, while the PRM roadmap can contain cycles. Second, the configuration tree is extended incrementally. Contrary to PRM, which first samples the configuration space and then connects the stored free samples into the roadmap, RRT samples the configuration space simultaneously with the construction of the configuration tree. To create a dense tree in a given region, the simple increase of probability of sampling in the region is not satisfactory as it has to be ensured first, that the tree can grow towards the region. Many PRM-based methods to cope with the narrow passage problem are not applicable to RRT due to this reason.

Algorithm 1: Main loop of RRT

```
Input: Configurations $q_{start}$ and $q_{goal}$, maximum number of iterations, $I_{max}$, maximum distance
       to goal $d_{goal}$
Output: Trajectory $P$ or failure
1  $\mathcal{T}.add(q_{start})$; // create new tree and add initial configuration $q_{start}$ in it
2  for $iteration=1:I_{max}$ do
3      $q_{rand}$ = random configuration in $\mathcal{C}$ using uniform distribution;
4      $q_{near}$ = nearest node in tree $\mathcal{T}$ to $q_{rand}$;
5      expandTree($q_{rand}, q_{near}$);
6      $d$ = distance from tree to $q_{goal}$;
7      if $d < d_{goal}$ then
8          $P$ = extract trajectory from $q_{start}$ to $q_{goal}$;
9          return $P$;
10     end
11 end
12 return failure; // no solution was found within $K$ iterations
```

Algorithm 2: expandTree(q_{rand}, q_{near}): Expansion procedure of RRT

```
Input: Random configuration $q_{rand} \in \mathcal{C}$, configuration tree $\mathcal{T}$, its nearest node in the tree
       $q_{near} \in \mathcal{T}$
Output: Extended tree $\mathcal{T}$
1  $R = \emptyset$; // set of configurations reachable from $q_{near}$ together with control inputs
2  foreach $u \in \mathcal{U}$ do
3      $q = q_{near} + \int_0^{\Delta t} f(q_{near}, u)\, \mathrm{d}t$;
4      if $q$ is feasible then
5          $R = R \cup \{(q, u)\}$;
6      end
7  end
8  if $R \neq \emptyset$ then
9      $(q_{new}, u)$ = select a configuration from $R$ closest to $q_{rand}$;
10     $\mathcal{T}$.addNode($q_{new}$);
11     $\mathcal{T}$.addEdge($q_{near}, q_{new}, \Delta t, u$);
12 end
```

The implementation of the expansion step in RRT differs in path planning and motion planning. In motion planning, where a forward motion model $\dot{q} = f(q, u)$ is considered, the expansion of a node q is realized by applying several control inputs $u \in \mathcal{U}$ to the model in order to obtain new configurations reachable from the node. The control inputs are applied over time Δt. New configurations are obtained by integration of the motion model, which can be solved analytically in the case of simple systems like Car-like robots [221], or using numerical integration like Euler integration or Runge-Kutta methods for complex systems. As many robotic systems can be driven by continuous values (e.g. speeds of wheels of mobile robots), the set $\mathcal{U}$ has to be discretized in order to allow RRT to expand the node q_{near} to a reasonable number of candidate configurations. $\mathcal{U}$ can be created e.g. by enumerating combinations of discretized control inputs of the actuators. An example of tree expansion using three control inputs $\mathcal{U} = \{u_1, u_2, u_3\}$ is depicted in Fig. 2.5 and examples of constructed trees are depicted in Fig. 2.6.

The advantage of the tree expansion using a forward motion model is that kinematics, dynamics and motion constraints can be considered, which allows RRT to find feasible trajectories even under differential constraints. This is one of the reasons for wide usage of RRT in robotics for motion planning of systems like mobile robots [71, 154, 107], autonomous cars [140], humanoids [133] or Unmanned Aerial Vehicles [273, 118, 200].

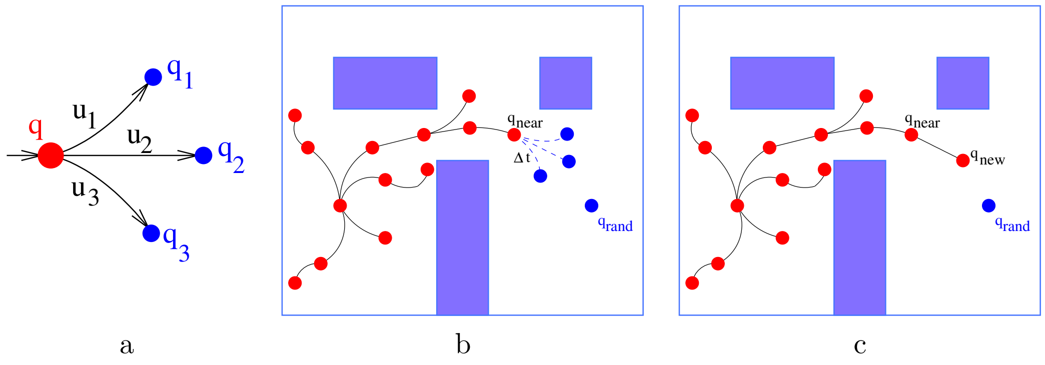


Figure 2.5: Example of expansion of configuration q using control inputs u_1, u_2 and u_3 applied over time Δt. The arrows denote orientation of the robot (a). The usage of this expansion in the RRT algorithm (b) and the extended tree (c).

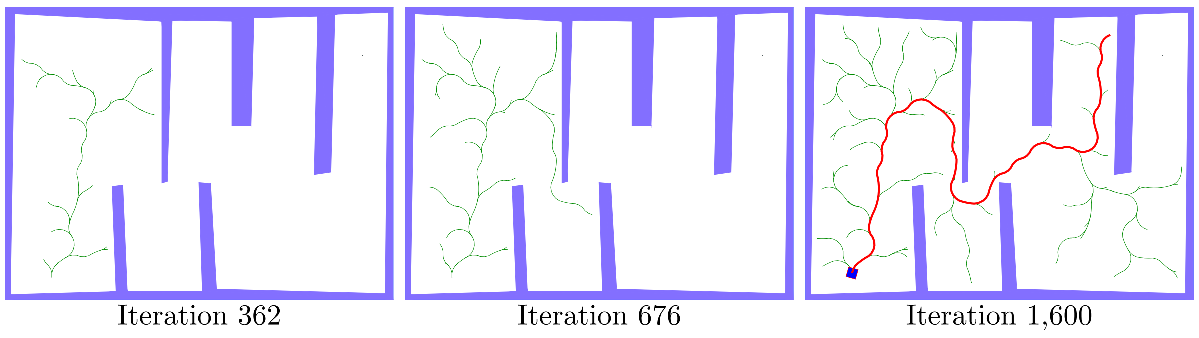


Figure 2.6: Progress of RRT configuration tree for a mobile robot.

The RRT planner can also be used for path planning [266, 211, 51] simply by using the straight-line planner in the expansion step. A line between q_{near} and q_{rand} is constructed and the tree is extended by a configuration q_{new} lying on the line in the distance ε from q_{near}, where ε is the resolution of the straight-line planner. This is visualized in Fig. 2.7a and examples of large trees constructed for 2D objects are depicted in Fig. 2.7b,c. RRT can be used for path planning of complex 3D models, e.g. in assembly/disassembly studies [248, 36, 227, 5].

The ability of RRT to spread the tree in the configuration space is caused by the nearest-neighbor selection of nodes for the expansion. As the random samples are generated uniformly in the configuration space, the probability that a node will be selected for the expansion is given by the volume of its Voronoi cell. Due to this *Voronoi bias*, the nodes at a boundary of the tree are selected more frequently for the expansion than the inner nodes [161, 279], which boosts the tree to grow towards unexplored areas of the configuration space. The Voronoi bias is performed implicitly by the sampling and nearest-neighbor searching. The study [160] has shown that exploration of the configuration space can be speeded up by selecting nodes with the largest

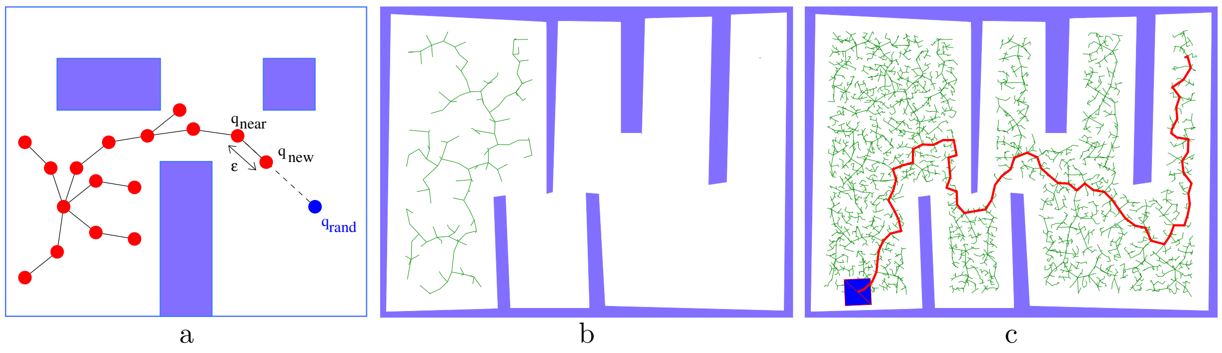


Figure 2.7: The tree expansion for path planning using the straight-line local planner with resolution ε (a). Examples of configuration trees with 318 nodes (b) and 2,900 nodes (c).

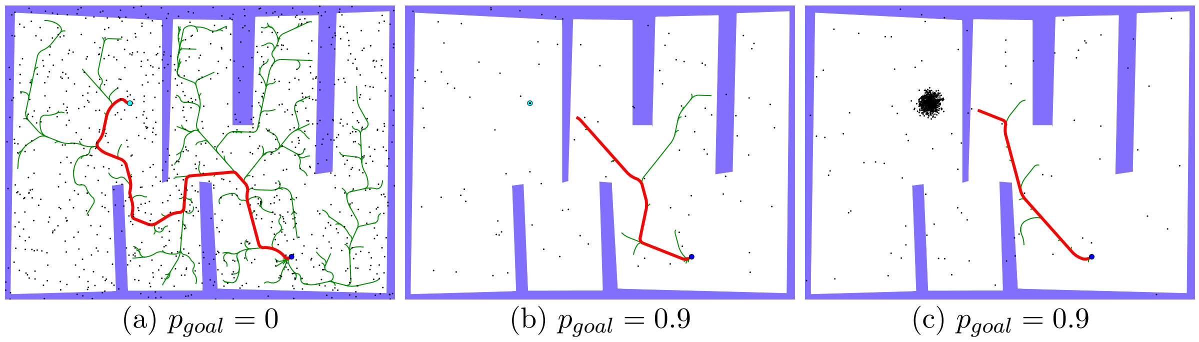


Figure 2.8: Examples of trees constructed with various goal biases during $1,000$ iterations. The black dots represent the random samples. The uniform sampling enables the tree to explore the whole configuration space (a). The goal-bias that generates 90 % of samples in the goal configuration causes that the tree is attracted mostly to the same place, therefore it cannot grow in the environment. The growth of the tree is suppressed even if the random samples are generated around the goal configuration (c).

exactly computed Voronoi cell, but this exact approach is time consuming due to necessity to maintain the Voronoi diagram of the nodes of the tree.

The growth of the tree towards the goal configuration can be speeded up using the *goal-bias* principle [153]: the random sample q_{rand} is replaced by q_{goal} with a probability p_{goal}. The goal-bias is very effective in environments without obstacles. However, it can slow down the growth of the tree if the goal is hidden by an obstacle, because the same boundary nodes are frequently selected for the expansion, which is precluded due to the obstacle. This situation is depicted in Fig. 2.8. The random samples drawn from the whole configuration space allow the tree to grow into the whole space (Fig. 2.8a). By replacing q_{rand} for q_{goal} with $p_{goal} = 0.9$, the tree is attracted to the goal, but it is stuck due to the obstacle (Fig. 2.8b). The picture Fig. 2.8c shows a modified goal-bias, where q_{rand} is generated around q_{goal} using Normal distribution.

The goal-bias can be generalized to multiple waypoints towards which the tree has to grow. Authors of [238] proposed to place the waypoints between difficult and easy regions of the configuration space, but they did not proposed any strategy to find the waypoints. In Cell-RRT [85], the waypoints are found using A* in the grid-based representation of the workspace and trees are constructed separately withing each cell on the path.

Another approach to bias the tree growth towards the goal configuration is to sample around trees constructed in previous iterations, which is suitable in situations, where RRT is repeatedly run in the same environment, e.g. in the robotic soccer. In these scenarios, a plan is computed for a short time horizon and it has to be delivered as fast as possible. Despite possible changes in the environment, the trees constructed in previous iterations can be used to speed up growth of the new one [65, 20]. In [20], the tree is built from scratch but its growth is biased along the trajectory found in the previous step by replacing q_{rand} by a random point from the trajectory with a certain probability (authors suggest 60 % of iterations). An approach to fully preserve the previously built tree was proposed in [65]. Instead of removing the tree, the algorithm only removes parts of the tree that are invalidated due to new situation in the environment.

2.4.1 Multiple trees

Other approaches to speed up growth of the tree and consequently to connect the start and goal configurations in shorter time are based on multiple trees. RRT–Bidirect [153] utilizes two trees, one rooted at q_{start} and the second one at q_{goal}. The trees can be grown simultaneously, i.e., both are expanded towards q_{rand} in each iteration, the expansion procedures can alternate [135] or only a tree containing less nodes is expanded. The RRT–Bidirect terminates if the trees are close enough to each other and can be connected by an edge.

To connect two trees $\mathcal{T}_1$ and $\mathcal{T}_2$ using their nodes $q_1 \in \mathcal{T}_1$ and $q_2 \in \mathcal{T}_2$, an appropriate control input leading the robot from q_1 to q_2 needs to be found. This operation can be solved easily for holonomic robots or robots with simple differential constraints. Generally, this problem is the two-point boundary problem, whose analytical solution is known only for limited class of systems [40]. The problem is usually solved numerically by an optimization technique, e.g. by perturbing the control inputs [142, 40]. The connection of two trees, especially under differential constraints, is therefore a time demanding operation. The bidirectional search is more suitable in the path planning task, where the configuration trees can be simply connected by the straight line.

The bidirectional search can be further extended by employing multiple trees [157, 46, 234, 266, 202, 224], which requires to define how to create the multiple trees, how to extend them and how to connect them to be able to find a trajectory. For example, authors of [234] suggest to create a new tree if a random sample q_{rand} cannot be connected to any of the existing trees. The new tree is rooted at the configuration q_{rand}. To prevent creation of too many trees, a new tree is created only with a given probability. A bounding box is maintained for each tree. The connection procedure is called only if the newly added node changes the bounding box.

Another way to maintain the RRT trees is to employ PRM as a high-level planner [202, 46, 224, 172]. The PRM planner samples the configuration space and RRT is used to grow a local tree around each node of the PRM's roadmap. The nodes of the PRM's roadmap are connected using the local trees. The combination of PRM and RRT planners is similar to the early Z^3 planner [81]. The roadmap of the trees can be easily parallelized, because each tree-based planner can run on its own processor. Similarly, the method [224] uses PRM to cover the large free areas of the configuration space. Each sample is tested for being located in a narrow passage. If a sample lies in a narrow passage, RRT is started in this configuration. A sample is said to be in a narrow passages if it cannot be connected to its neighbors or only to a few of them. Contrary to [234], the method [224] starts a new tree only in narrow passages, which is motivated by the fact that RRT grows well in narrow passages despite the fact that it is hard to reach them.

2.4.2 Modifications of the expansion step

RRT–Connect [134, 135] repeatedly expands the tree from the last added node until an obstacle is hit or the goal is reached. This expansion can be implemented e.g. by applying same control input to the last added node. The RRT–Connect method can create large chains of nodes, which quickly traverse free regions. The repeated application of the same control inputs is however not suitable in tortuous narrow passages, where control inputs need to be frequently changed in order to steer a robot through the passage.

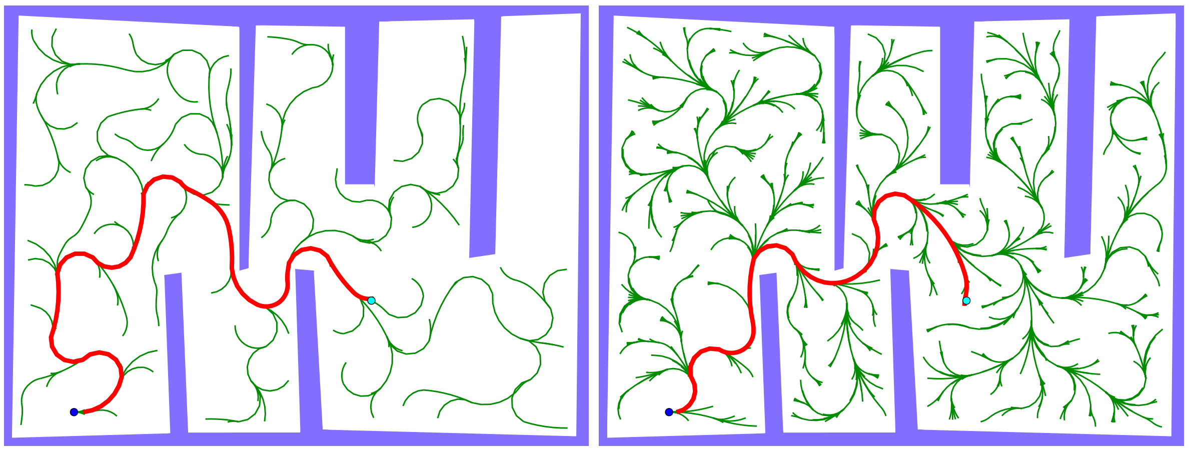


Figure 2.9: Examples of RRT trees for a mobile robot constructed using RRT and RRT–Blossom in 2, 000 iterations. The tree constructed by RRT–Blossom is more dense, as upto five new nodes are added to the tree in each expansion, while RRT can add at most one node.

While the original RRT algorithm extends the tree using only one new node in each iteration, RRT–Blossom extends the tree using multiple nodes [111]. In the expansion procedure, all control inputs are applied to steer the system from q_{near} in order to obtain a set of reachable configurations. A configuration from the set can be added to the tree only if the distance to its parent is smaller than the distance to its nearest neighbor in the tree. This pruning boosts the growth of the tree towards unexplored regions of the configuration space. Another pruning method based on filtering of non-viable states, i.e., the states leading to a collision, was proposed in [112]. In comparison to RRT, RRT–Blossom generates more dense trees. Examples of trees constructed by RRT and RRT–Blossom are depicted in Fig. 2.9.

An analysis of the performance of RRT in the narrow passages was presented in [279]. The nodes in the tree can be divided into two groups: frontier nodes, whose Voronoi cells grow together with growth of the environment and boundary nodes, which are close to the obstacles. The frontier nodes lie at the boundary of the tree and they should be selected for the tree expansion. In a narrow passage, the nodes are both boundary and frontier. These nodes are frequently selected for the expansion, because they are the frontier nodes, however the tree cannot be expanded from them, because they are also the boundary nodes. A Dynamic-Domain strategy for RRT (RRT–DD) is proposed in [279]: each node holds an action radius defining how far can be a random sample q_{rand} that activates the node for the expansion. The RRT–DD algorithm generates random samples q_{rand} and finds its nearest neighbor in the tree q_{near} until $\rho(q_{rand}, q_{near}) < q_{near}.radius$. The extension of q_{near} is performed as in the original RRT. If the connection is successful, the radius of q_{new} is set to ∞, otherwise the radius of q_{near} is set to a predefined constant R_{dd} (Fig. 2.10). This ensures that the boundary nodes will not be selected frequently for the tree expansion. Although the algorithm is efficient in the narrow passages, it is strongly influenced by the parameter R_{dd}. Too small radius decreases performance of the planner as it takes more time to generate samples that can induce a tree's node for expansion. Too large radius supports only exploration of the space and the refinement does not have any effect. To set the initial radius R_{dd}, authors suggest to use 100ε, where ε is the resolution step of the local planner.

To decrease the sensitivity of the method to the parameter R_{dd}, it can be automatically adjusted, which was proposed in RRT–ADD (RRT with Adaptive Dynamic Domain) [108]. The idea of RRT–ADD is to decrease the activation radius of a node that cannot be expanded. The activation radius is increased after the node is successfully expanded. The radius is increased by scaling it by factor $(1 + \alpha_{dd})$ and it is decreased by scaling it by factor $(1 - \alpha_{dd})$, where α_{dd}

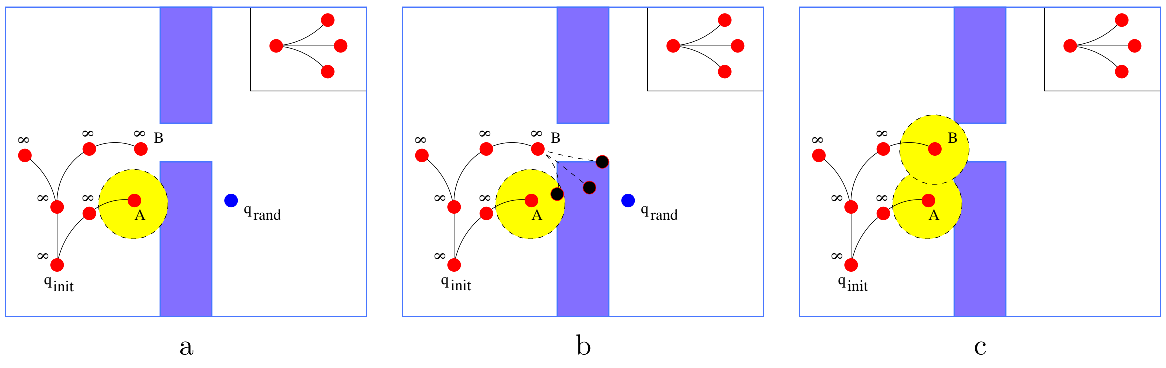


Figure 2.10: An example of an unsuccessful expansion in RRT–DD due to low activation radius. The upper right corner shows possible expansions. The nearest neighbor to q_{rand} is node A, however it cannot be selected as its activation radius (yellow circle) is smaller than the distance to q_{rand}, so node B is selected for expansion, because its radius is ∞ (a). The node B cannot be extended, as all states lead to a collision (black nodes) (b). Therefore, the radius of the node B is decreased from ∞ to R_{dd} (c).

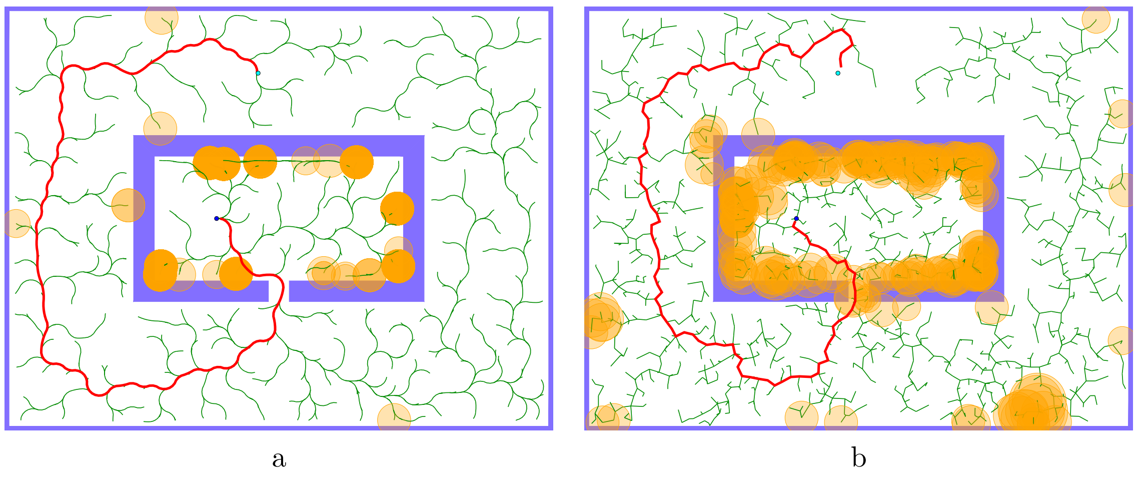


Figure 2.11: Example of a tree constructed using RRT–ADD with dynamic domain radius for the Car-like robot (a) and the 2D holonomic robot (b). The orange circles denote activation radius smaller than ∞. The initial radius is set to $R_{dd} = 70$ mu, $\alpha_{dd} = 0.1$. Size of the maps is $1,500 \times 1,000$ mu. The activation radius is limited at nodes close to the obstacles.

is parameter of RRT–ADD. The effect of this automatic adjustment is that nodes close to the obstacles have smaller activation radius, which prevents them to be frequently chosen for the expansion. Examples of configuration trees built by RRT–ADD are depicted in Fig. 2.11.

RRT–Retraction [286] improves sampling in the narrow passages for motion planning of 3D objects. After q_{rand} is generated and its nearest neighbor $q_{near} \in \mathcal{T}$ is found, the segment from q_{rand} to q_{near} is checked for collision. The tree is extended normally if the segment is free, otherwise the retraction step is performed. The task of the retraction step is to find a contact configuration that minimizes the distance to q_{rand}. The contact configuration q is found on the line and its neighborhood is searched to find another contact configuration q' such that $\varrho(q', q_{rand}) < \varrho(q, q_{rand})$. The retraction is terminated after I_{ret} steps or if no new contact configuration can be found. The configuration trees built by RRT–Retraction can therefore contain more nodes than is the number of planning iterations, as the tree can be expanded by upto I_{ret} contact configurations in each iteration. An example of the retraction is depicted in Fig. 2.12. The RRT–Retraction can quickly penetrate to narrow passages, but its computational burden is significantly increased due to necessity to find the contact configurations. The method

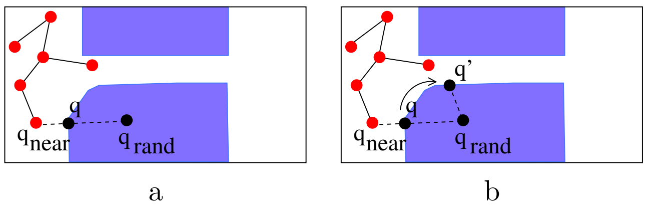


Figure 2.12: The retraction procedure of RRT–Retraction. The random configuration q_{rand} is non-free and a contact configuration q is found on segment q_{rand}, q_{near} (a). The neighborhood of q is searched to find a new contact configuration q' that minimizes distance to q_{rand} (b). The tree is then extended towards q'.

was designed mainly for 3D path planning, but it can be combined with other planners for motion planning of many-DOF robots [197].

The paper [286] lacks technical details about construction of the contact configurations. For the purpose of our experiments, waypoints are computed on the line segment (q_{near}, q_{rand}) with resolution ε (resolution of the straight-line planner). Collision detection is run in each point. The last collision-free configuration on the line starting from q_{near} is then considered as the contact configuration. In order to find the next contact configuration around a previous configuration q, N_{ret} random samples are generated uniformly in the radius r_{ret} around q. From these configurations, a collision-free configuration that is nearest towards q_{rand} becomes the next contact configuration. The procedure is repeated at most I_{ret} times. The expansion procedure of RRT–Retraction used in our experiments is listed in Appendix A.

2.4.3 Optimality of RRT-based planners

The RRT planner belongs to the family of non-optimal planners. The probability that RRT returns an optimal path is almost zero [116]. Two basic paradigms are used to compute optimal or near-optimal solutions using RRT: to construct the trees considering an optimality criteria [104, 117, 200, 154, 245, 107, 106, 60] and to improve the plans by a post-optimization [76, 186, 28, 187, 13, 79, 38, 10, 217].

In MA-RRT [154], a cost map is used to define difficulty of traversing areas in the environment. To select a node for the expansion, all nodes in the tree are examined to estimate the cost to the goal configuration. The node with the lowest resulting cost is selected and added to the tree. MA-RRT generates trajectories with lower costs, but the method is computationally intensive, because all nodes need to be examined during each expansion step. The necessity of examining all nodes is eliminated in [245], where only several neighbors of q_{near} are tested for the expansion. The costmap is utilized also in the approaches [107, 106], where Monte Carlo optimization and simulated annealing are used to accept or reject new states based on the gradient of the cost function of the local motion. Another approach for computing good paths in rough terrains was proposed in [60]. In this approach, the cost-map defines "obstacleness" as a cost of reaching a given place. The expansion of the tree is allowed from the nodes with a low obstacleness. If no solution can be found, the allowed level of obstacleness is increased.

Recently, PRM* and RRT* [117] extensions have been proposed, which evaluate quality of each node as a cost of the path from the initial configuration to the node. Contrary to RRT, where the tree is extended only from the node q_{near}, RRT* considers also its neighbors. The tree is extended from such a node that minimizes cost of the newly added node. Moreover, after a new node is added, the parent node can be reconnected to different nodes, if the reconnection is feasible and if it decreases the cost.

A different approach to achieve optimal or near-optimal plans is based on post-optimization. A path or trajectory is optimized by moving or even deleting its configurations. A simple approach for path smoothing is to test if three consecutive points can be replaced by a straight line made of the end points [13]. This is generalized in the Shortcut pruning [79], which tries to replace a whole sequence of points by a single line. The Shortcut strategy selects the sequence randomly. Other ways of selecting the sequence are discussed in [38, 10, 217]. Methods to replace parts of path by other geometric elements (e.g. circular segments) were proposed in [115, 13].

The path manipulation is not easy if kinematic and dynamic constraints need to be considered. Therefore, postprocessing is used mainly for path planning of mobile robots, where the usual criterion is to improve smoothness of the paths [76, 53]. Smooth paths are preferred, as they can be followed by mobile robots easier than jerky paths or paths with sharp turns [53].

During path pruning also clearance can be considered [79]. The path smoothing methods usually change the paths only locally, i.e., the improved path is located in the same homotopy class as the original one. The approach [83] allows more drastic changes in the path homotopy, so that the path can skip obstacles during the post-optimization process.

2.5 Implementation details

The motion planning problem is usually studied on the algorithmic level and less number of papers deal with implementation details. The implementation details become crucial in challenging tasks like autonomous car driving [140], or navigation of planetary rowers [140], where the plans have to be delivered very fast or where computational resources are limited. Several implementation details can be found in [49], and current issues and challenges in the motion planning are summarized in [4].

An important part of the sampling-based planners is the nearest-neighbor search. In PRM, the nearest-neighbor search is used to connect samples in the roadmap. Fast search can be realized by dedicated data structures like KD-trees or metric-trees. In RRT, the nearest-neighbor search is realized on an incrementally growing dataset, which requires fast insertion to an existing KD-tree data structure. KD-trees suitable for RRT planners are supported by MPNN library [9]. The exact nearest-neighbor search using KD-trees becomes impractical in high-dimensional configuration spaces, as almost all the stored nodes have to be visited [203]. In such a case, an approximate nearest-neighbor method can be used. The performance of KD-trees can be negatively influenced by frequent cache misses. A cache efficient KD-tree was proposed in [3], but it supports only one-time construction of the tree, which is not suitable for RRT-based planners. A cache aware RRT* that is able to construct KD-tree on-line was presented in [99]. It subdivides the configuration space into several regions so the resulting dataset is small enough to fit in the cache.

The collision detection is another important procedure of sampling-based planning. It is used to decide whether a configuration $q \in \mathcal{C}$ is free or non-free. Collisions between 3D objects can be computed fast using hierarchical data structures like OBB or AABB trees [56], which have been implemented in several libraries like Rapid or Solid [207]. Parallelization of collision detection can be achieved on multi-core architectures [52] or even using dedicated hardware [15, 196, 126].

Parallelization of computations is a widely used technique to obtain plans in a shorter time, and it has been already used in several early planners [10, 35, 163]. A simple parallel version of RRT [32] executes the same instance of RRT on several processors and the first processor that finds a solution terminates the others. The disadvantage of this approach is that all the processors explore the configuration space separately and therefore many computations (e.g. collision detections) are repeated, rather than shared amongst the processors. To share common data structures a global communication between processors is required. A parallelization framework for single-query planners was presented in [190]. In this approach, the processors construct configuration trees separately until a new best solution is found (e.g. the shortest path to the goal or the closest path to the goal). If a better plan is found, it is exchanged between the trees and irrelevant nodes are pruned. A possible drawback of parallelized planning is the global communication between the processors, which can have a considerable overhead. The communication overhead can be decreased by splitting the configuration space to several regions, that are searched independently by each processor [209, 105].

The ability of RRT to explore the configuration space is also influenced by the employed metric ϱ [41]. The Euclidean metric, that is supported by many KD-tree libraries, is often used in RRT-based planners. The Euclidean metric is suitable only for non-holonomic systems, which can freely move between two configurations on a straight line. In the presence of kinematic constraints (joint limits, obstacles or other non-holonomic constraints) or dynamic constraints (e.g. torque limits), the Euclidean metric does not describe the real distance between the states correctly. Consequently, the nearest-neighbor search repeatedly selects nodes without growing closer to the sampled region [226, 27], which slows down the tree growth process.

An ideal metric would be an optimal cost-to-go function computed e.g. as a time or energy required to move the system between given configurations [69]. Computation of such a metric is intractable for most of the high-order systems as it would require to solve the original motion planning problem [151]. The cost-to-go metric can be approximated using simple models [127, 194]. The drawback of using cost-to-go functions is that fast data structures like KD-trees cannot be used. Linear search is used instead, which can decrease the speed of the planning process [194].

2.6 Conclusion

The presented overview has shown different approaches to solve the path and motion planning problems. The basic path planning methods rely on a geometric description of the workspace, from which a graph-based representation can be directly built. Examples are Voronoi diagram or decomposition-based methods. An important advantage of these methods is a fast computation of collision-free paths, especially if robots can be approximated by a disc. The methods are therefore suitable for path planning of the mobile robots.

Sampling-based methods like PRM and RRT attempt to solve the planning problems by randomized sampling of the configuration space. The methods utilize black-box collision detection to determine free and non-free samples which allows them to cope with robots of arbitrary shapes. Another important advantage of the sampling-based methods is the ability to cope with differential constraints. As the methods work primarily in the configuration space, they can be applied to robots with many degrees of freedom.

One of the challenging issues of sampling-based planning is the narrow passage problem. Due to low volume of narrow passages, it is difficult to sample them and therefore to construct a path through them. The presence of the narrow passages decreases performance of the sampling-based planners and consequently it increases time necessary to find a solution.

PRM-based planners can cope with the narrow passage problem by focusing the sampling to regions that are believed to contain a narrow passage [191, 138, 247, 22, 173, 174, 172, 171, 24, 27, 98, 95]. The biased sampling in a given region can improve the performance of RRT as well, especially in environments without obstacles [150, 238], but it may become counterproductive in environments with obstacles. The obstacles can block the growth of the tree, which slows down the planning process.

In the task of motion planning for robots with few degrees of freedom, the knowledge about the obstacles in the workspace can help to estimate the narrow passages in the configuration space. In this thesis, we propose a novel sampling schema for the RRT planner. The core of the method is the utilization of a workspace knowledge represented by a geometric path found in the workspace. The path is used to focus the sampling of the configuration space to the regions that are believed to contain a solution. The usage of such a path speeds up the tree growing process and it also helps in the case of the narrow passages. To compute the path as fast as possible, the path planning methods can be utilized.

Chapter 3

Guided sampling of configuration space

The ability of RRT to quickly explore the configuration space is caused by the combination of uniform sampling and the nearest-neighbor rule used to select nodes for the expansion. The growth of the configuration tree into some parts of the configuration space can however be slowed down due to the narrow passages. As has been proposed in many PRM-based planners, the knowledge about the obstacles in the workspace can help to estimate positions of narrow passages in the configuration space. In this chapter, we propose a novel sampling schema for RRT to cope with the narrow passage problem. To maximize benefit of the workspace knowledge, we propose to compute a path in the workspace, and we use it to guide the tree through the configuration space towards the goal configuration.

3.1 Problem analysis

The growth of the configuration tree in RRT can be attracted to a region $\mathcal{R} \subseteq \mathcal{C}$ by replacing random configuration $q_{rand} \in \mathcal{C}$ by $q'_{rand} \in \mathcal{R}$ with a non-zero probability. An example of the biased sampling is the goal-bias [150], which replaces q_{rand} by q_{goal} with probability p_{goal} to attract the tree towards the goal configuration. Sampling around predefined configurations was proposed in [238]. Authors suggest to place the configurations close to narrow passages, but they do not provide any method to find such configurations. The biased sampling can also be used to rebuild a tree if the environment slightly changes, e.g. during robotic soccer. In [20, 243], the configuration space is sampled more along a previously built tree.

Increasing probability of sampling in a given region however cannot ensure that a tree will grow faster towards the region. The reason is that a sample can be added to the tree only if the tree can reach the sample. The effect of goal-bias is investigated in the following two scenarios in environments with and without obstacles.

3.1.1 Goal-bias without obstacles

Let consider the scenario with a long corridor depicted in Fig. 3.1a with an initial configuration q_{start} and two goal configurations q_{down} and q_{top}. Both goal configurations are equally distant from q_{start}. Due to the nearest-neighbor rule used to select the nodes for the expansion and the implicit Voronoi bias, the uniform sampling supports fast growth of the tree towards unexplored areas of the configuration space [161]. In the long corridor, the nodes on the top of the tree have larger Voronoi cells than the nodes on the bottom (Fig. 3.1b), therefore the tree will prefer to grow upwards if uniform sampling is used. Consequently, the tree will probably reach q_{top} faster than q_{down} although both goals are equally distant from q_{start}. Examples of typical configuration trees are depicted in Fig. 3.1c,d.

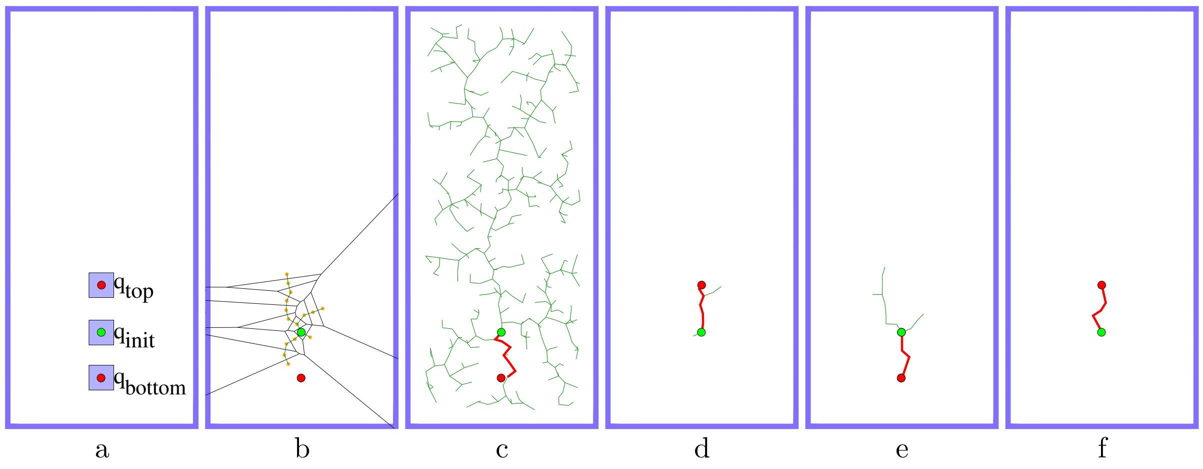


Figure 3.1: The scenario with long corridor of size 400×900 map units (mu). The holonomic robot is represented by a box of size 50×50 mu (a). Voronoi diagram of a tree with 16 nodes (b). Example of configuration tree with highlighted path for q_{down} (c) and q_{top} variants (d) computed without goal-bias. A small goal-bias $p_{goal} = 0.2$ forces the trees to grow directly to q_{down} (e) as well as to q_{top} (f).

Table 3.1 shows difference between planning without ($p_{goal} = 0$) and with ($p_{goal} = 0.2$) goal-bias. The plans have been computed using basic RRT with $I_{max} = 500$ iterations and distance threshold $d_{goal} = 15$ mu for a holonomic 2D robot represented by a box of size 50×50 mu, where mu denotes the "map units". The table shows average values over 50 trials. The success rate is the percentage of trials where the tree approached a given goal configuration to a distance $d_{goal} = 15$ mu or less.

Without the goal-bias, RRT builds the trees with ~ 412 nodes in average to reach q_{down}, while only ~ 215 nodes are required to reach q_{top}. The goal q_{down} was not always reached by the tree, which is indicated by the distance to goal that is larger than the threshold $d_{goal} = 15$ mu. The ability to reach both goals significantly improves if a small goal-bias ($p_{goal} = 0.2$) is introduced. The average size of the built trees is reduced from ~ 412 nodes to ~ 17 nodes in the case of q_{down} and from ~ 215 nodes to ~ 10 nodes in the case of q_{top} respectively. The examples of the configuration trees constructed with goal-bias are depicted in Fig. 3.1e,f.

The introduction of the goal-bias also increases the success rate. Under uniform sampling ($p_{goal} = 0$), q_{top} is approached in 78 % of cases, while q_{down} only in 48 % of cases. When the goal-bias is introduced, both configurations are visited in 100 % of cases. The higher success rate is also indicated by the decreased distance to goal.

This simple experiment shows the positive effect of the goal-bias to the growth of the configuration tree. It also shows, that the relative position of the start/goal configurations in the environment significantly influences performance (especially success rate) of the RRT planner. Although the both goals are equally distant from the start configuration, the success rate significantly differs if no goal-bias is used.

3.1.2 Goal-bias in scenarios with obstacles

An important parameter of the goal-bias is the probability p_{goal} of drawing the samples around q_{goal}. A right choice of p_{goal} depends on the environment as well as on the maneuverability of a robot, which is demonstrated in the following experiment.

The experiment has been performed in four 2D environments (Fig. 3.2) with two Car-like robots (Car-like$_{\uparrow}$ and Car-like$_{\updownarrow}$), two Differential drive robots (Diff$_{\uparrow}$ and Diff$_{\updownarrow}$) and two 2D holonomic robots. The robots Diff$_{\uparrow}$ and Car-like$_{\uparrow}$ cannot move backward, while the robots Diff$_{\updownarrow}$ and Car-like$_{\updownarrow}$ can move in both directions. For each robot and each map, 200 start/goal feasible

Table 3.1: Effect of goal-bias sampling in the scenario with long corridor.

Goal-bias		q_{down} Mean	q_{down} Dev	q_{top} Mean	q_{top} Dev
$p_{goal} = 0$	**Tree size**	412.40	130.90	215.74	226.82
	Runtime [s]	0.03	0.01	0.02	0.01
	Distance to goal [mu]	18.7	21.5	12.5	14.4
	Success rate	48 %		78 %	
$p_{goal} = 0.2$	**Tree size**	17.40	7.70	10.42	5.10
	Runtime [s]	0.02	0.01	0.02	0.01
	Distance to goal [mu]	1.00	4.47	3.57	4.92
	Success rate	100 %		100 %	

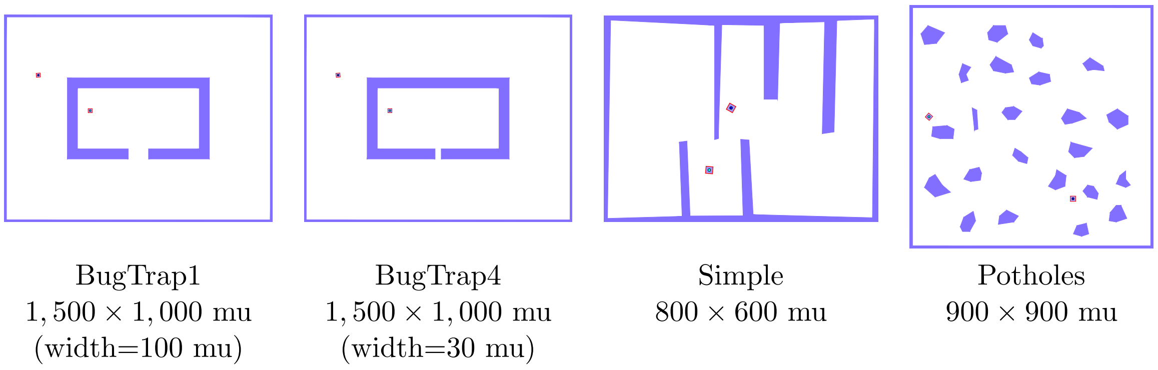


Figure 3.2: 2D maps with a random start/goal configuration pair. All robots are represented as box of size 20 map units. Numbers in parentheses denote width of narrow passage in BugTrap* maps.

configurations are randomly generated. Basic RRT is run 40 times for each start/goal pair and each tested value of $p_{goal} = \{0, 0.05, 0.1, \ldots, 0.95\}$. Maximum number of allowed iterations $I_{max} = 1,000$, and the goal region is defined by radius $d_{goal} = 30$ mu.

The performance of RRT is measured as percentage of start/goal pairs that are connected by the planner at least once during the 40 trials. A start/goal pair is connected if the tree approaches the goal configuration to distance less than the threshold $d_{goal} = 30$ mu. The results are visualized in Fig. 3.4 and Fig. 3.5. A small goal bias ($p_{goal} = 0.1$) significantly increases the number of visited start/goal pairs compared to motion planning without goal-bias ($p_{goal} = 0$). The goal-bias has the most positive effect (i.e., increase of percentage of solved instances) in the Potholes environment. The Potholes map contains 23 obstacles and the tree can easily grow around them without being stuck. Even with a high goal-bias ($p_{goal} = 0.7$), almost all instances can be solved (with a minor exception of the Car-like robot, which is discussed later).

Contrary, the increased goal-bias brings a negative effect in the Simple environment. The Simple map contains long walls, which requires the robots to perform zig-zag motions in order to maneuver through the environment. Such motions are however suppressed by the goal-bias. The goal-bias can help mainly in cases where the start and goal configurations are not separated by any obstacles (similarly as in the experiment described in Sec. 3.1.1). In this experiment, the start/goal pairs are placed randomly, therefore many start and goal configurations are separated by the obstacles. Consequently, the large goal-bias can help only for few start/goal pairs, which explains the decreasing success rate with the increasing goal-bias. An example of start/goal pair that is separated by the obstacles and the negative effect of the increasing goal-bias is depicted in Fig. 3.3.

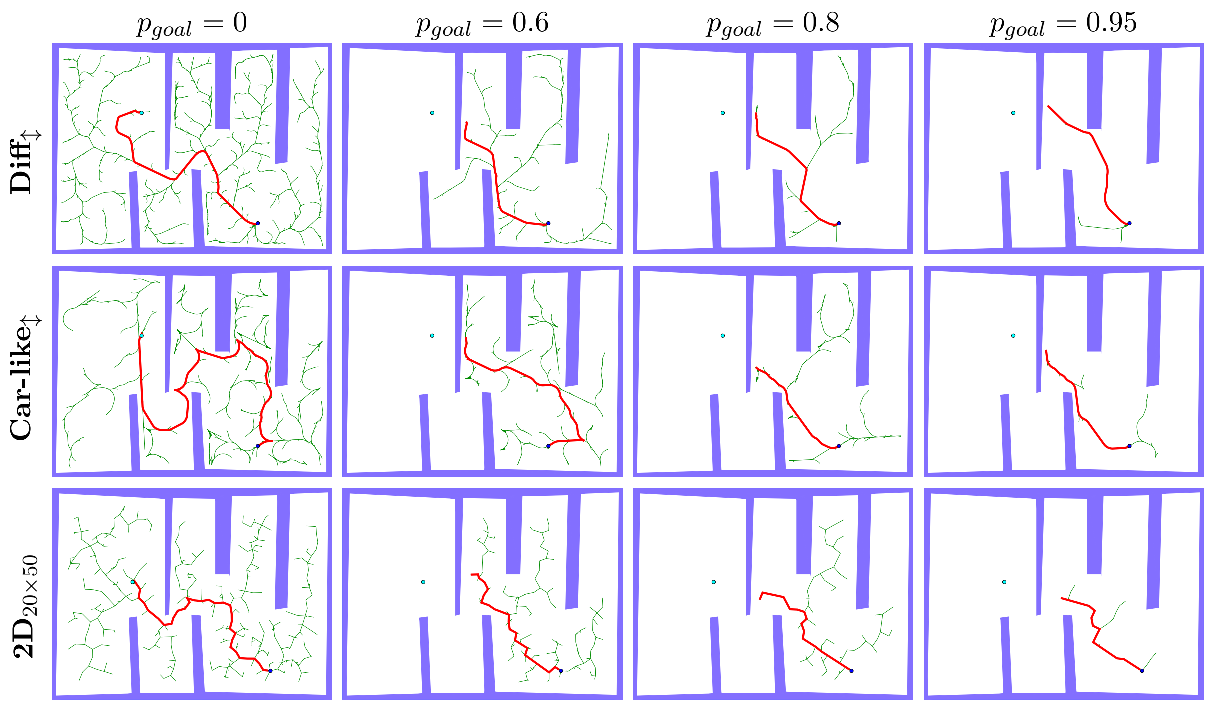


Figure 3.3: Examples of the configuration trees (green) and resulting trajectories (red) constructed with the goal-bias. All robots can move forward and backward.

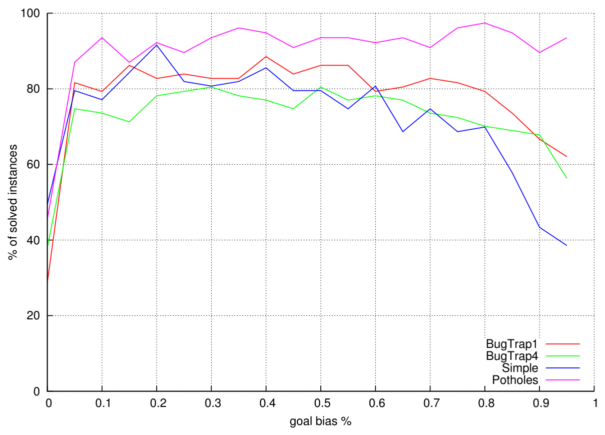
The results also show that the planning is more difficult for robots with reduced motion abilities (Car-like$_\uparrow$, Diff$_\uparrow$), because the percentage of solved stat/goal pairs decreases faster with increasing goal-bias than for Car-like$_\updownarrow$ and Diff$_\updownarrow$ robots. The reduced motion abilities decrease possibility of tree expansion, which consequently slows down the growth of the tree. Similar decrease can be observed with the 2D holonomic robots. The maneuverability of the 2D holonomic robots is determined by their size, therefore the smaller 2D$_{20\times50}$ robot is more agile than the larger 2D$_{20\times100}$ robot. The achieved results confirm this, because the percentage of solved start/goal pairs decreases faster for the 2D$_{20\times100}$ robot than for the 2D$_{20\times50}$ robot.

3.1.3 Motivation for a novel guiding schema

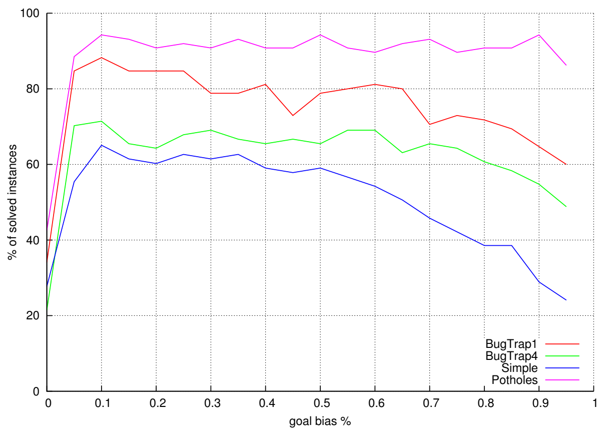
The previous motivation experiments have shown that the goal-bias principle can significantly improve growth of the tree towards a goal region and consequently, it can speeds up the planning process and increase the success rate, as was shown in Section 3.1.1. However, it is not easy to determine the proper amount of the goal-bias in environments with obstacles, where it can have positive and also negative effects, which has been discussed in Section 3.1.2.

The experiment described in Section 3.1.1 indicates, that the growth of the tree can be easily attracted towards a given region if the region is directly reachable from the tree, i.e, if there is no obstacle between the tree and the region. To maximize growth of the tree towards a given region, we propose to sample regions in the configuration space considering the ability of the tree to reach them. Let consider the situation depicted in Fig. 3.6a, where the goal-bias is used to generate more samples in the narrow passage area (denoted as np in the figure). The random samples q_{rand} generated in np will repeatedly select the same nodes for expansion (Fig. 3.6b), but the tree cannot be expanded due to the obstacle. The frequent selection of the same nodes therefore slows down the growth of the tree. To help the grow through the narrow passage, first the region A should be sampled, then the region B and then the narrow passage (Fig. 3.6c).

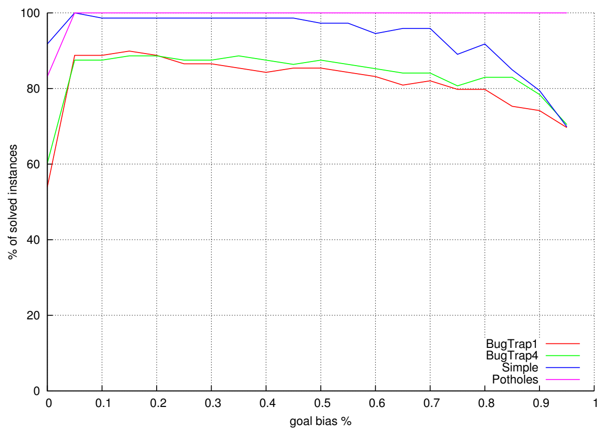
To improve the growth of the configuration tree, we propose to extend the goal-bias principle considering the following criteria.



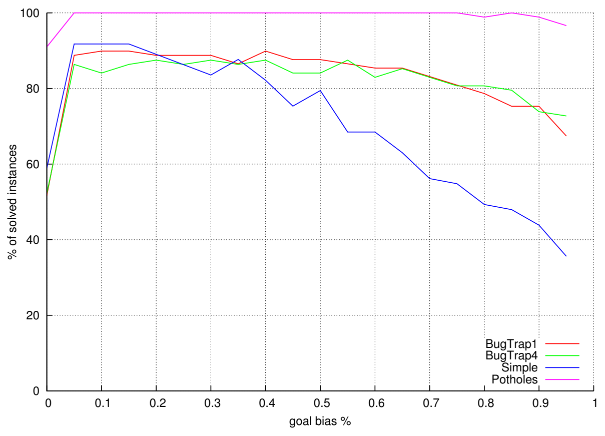
Car-like$_{\updownarrow}$



Car-like$_{\uparrow}$

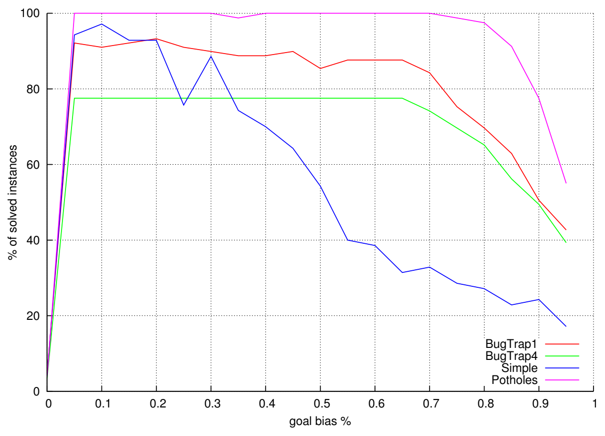


Diff$_{\updownarrow}$

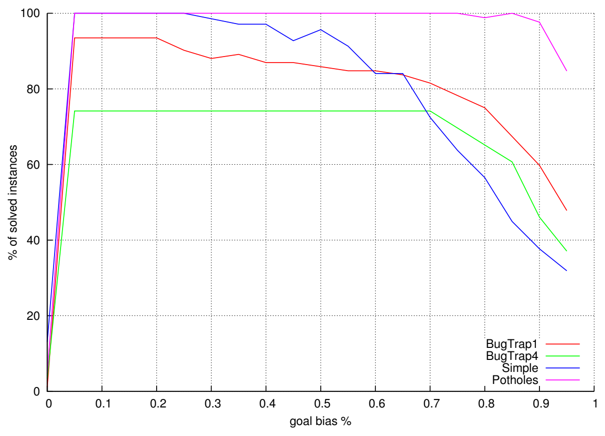


Diff$_{\uparrow}$

Figure 3.4: Performance of the RRT method with increasing goal-bias in the task of motion planning for mobile robots. The graphs show how many percents of start/goal pairs are connected by RRT at least once during 40 trials.



2D$_{20\times100}$



2D$_{20\times50}$

Figure 3.5: Performance of the RRT method with increasing goal-bias in the task of path planning for 2D holonomic robots. The graphs show how many percents of start/goal pairs are connected by RRT at least once during 40 trials.

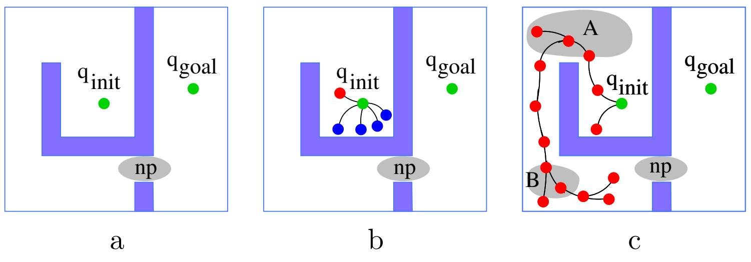


Figure 3.6: An environment with a narrow passage (denoted *np*). The biased sampling in the narrow passage slows down the growth of the tree, as the same nodes (blue in (b)) are frequently selected, but these cannot be expanded due to the U-shaped obstacle. To attract the tree towards the narrow passage, first the region *A* should be sampled followed by sampling in region *B*. After the tree reaches the region *B*, narrow passage *np* can be sampled more dense (c).

1. Biased sampling in a given region may become counterproductive if the tree is blocked by an obstacle. The ability of the tree to grow towards a given region has to be considered. To ensure that a tree can grown into a given region, the region should be sampled densely only if the tree is close to it.

2. Instead of the blind biased sampling in one or more regions, the tree should be guided through the configuration space. The guiding can be realized along a path computed in workspace leading from the start configuration to the goal configuration.

3. Besides sampling the configurations along the guiding path, sampling from whole $\mathcal{C}$ should be allowed as well. The probability of sampling along the guiding path should respect the ability of the tree to follow the path and it should be adapted automatically. If the tree cannot follow it, the sampling along the guiding path has to be suppressed in order to generate more samples from the whole configuration space. The aim of this adaptive sampling is to maintain growth of the tree even if the path cannot be properly followed.

3.2 RRT–Path: guided sampling of configuration space

The main idea of the proposed RRT–Path method [252] is to sample the configuration space along a path leading from the start configuration to the goal configuration. The random samples are not drawn along the whole path but only on such parts, that are not reached yet by the tree, which enables the tree to follow the path. As the path guides the growth of the tree, it is referred to as guiding path in the rest of the text.

Let $P = (q_1, \ldots, q_n)$, where $q_i \in \mathcal{C}_{free}, i = 1, \ldots, n$, $q_1 = q_{start}$ and $q_n = q_{goal}$, denote the guiding path. The guiding path represents a knowledge about the environment. To maximize the benefit of the guiding path, the configuration space should be sampled around such parts of the guiding path that have not been reached by the tree yet. Contrary, sampling along already reached parts of the path should be suppressed. This is achieved using a *virtual goal* $q_v \in P$, which is a point with index v. The virtual goal defines which part of the guiding path is still active, i.e., is used to attract the tree, and which part has no longer importance for the tree. At the beginning, the virtual goal is set to $v = 1$ and it moves towards the next points on the path if the tree approaches the virtual goal.

To generate random samples around the path, each point $q_i \in P$ has assigned a sampling radius r_i. The radius determines both an importance of the point as well as the region around the point to be sampled. It is computed as:

$$r_i = \begin{cases} 0 & \text{if} \quad i < v \\ d_{goal}(1 - \frac{c_i}{c_v}) + R'_{vg}\frac{c_i}{c_v} & \text{otherwise,} \end{cases} \tag{3.1}$$

where $R'_{vg} > d_{goal}$ is a radius around the virtual goal and d_{goal} is the final radius around q_{goal}. The radius r_i decreases linearly according the distance c_i between point $q_i \in P$ and the last point q_n measured along the path using the metric ϱ:

$$c_i = \sum_{j=i}^{n-1} \varrho(q_j, q_{j+1}). \tag{3.2}$$

The radius r_i determines the priority of the guiding path points, which is zero for points behind the virtual goal ($i < v$), it is largest for the virtual goal ($i = v$), and it linearly decreases towards end point of the guiding path.

To generate a random point along the guiding path, first a point $q_k \in P$ on the path is selected randomly according the priorities r_i of the points. As r_i is zero for $i < v$, the random point q_k will be selected in front of the virtual goal. The random sample $q_{rand} \sim N(q_k, \Sigma_k)$ is generated from Gaussian distribution around point q_k. In this work, we use symmetric Gaussian with $\Sigma_k = r_k\mathbf{I}$, where $\mathbf{I}$ denotes the identity matrix.

The proposed RRT–Path algorithm is listed in Alg. 3 and it works as follows. In each iteration, q_{rand} is generated along the guiding path with probability p_{goal} or it is generated randomly in the whole configuration space with probability $(1 - p_{goal})$. The nearest neighbor $q_{near} \in \mathcal{T}$ in the tree to q_{rand} is found and expanded using a motion model of the robot. The tree is extended by such a resulting configuration that is closest to q_{rand}. Then, the virtual goal is updated (Alg. 4). The virtual goal is moved forward if the tree approaches the path in front of the virtual goal. To determine whether the tree approached the path, nearest neighbors $q_{near,i}$ in the tree to points $q_i \in P, i = v, \ldots, n$ are found and distance $d_i = \varrho(q_i, q_{near,i})$ between the point q_i and its nearest neighbor is calculated. Point q_i is said to be approached, if the distance $d_i < d_{path}$. If no point on the path is approached, the virtual goal is not changed. Otherwise, the last point on the guiding path approached by the tree is found and the virtual goal is set to its successor on the path.

The virtual goal does not move sequentially on the guiding path, but it can "skip" some parts of the guiding path, because it is set as the successor of the last reached point on the path. The skipped part of the guiding path is not used to attract the tree. This brings a significantly advantage if a part of the guiding path cannot be easily followed by the tree, e.g. due to differential constraints or due to obstacles. An example of skipping part of the guiding path is depicted in Fig. 3.7.

The spread of samples around the guiding path is determined by the radius R'_{vg}. In an ideal case, the tree should be able to follow the guiding path precisely. As will be shown in the next section, the guiding is computed as a geometric path in the workspace and therefore, it might

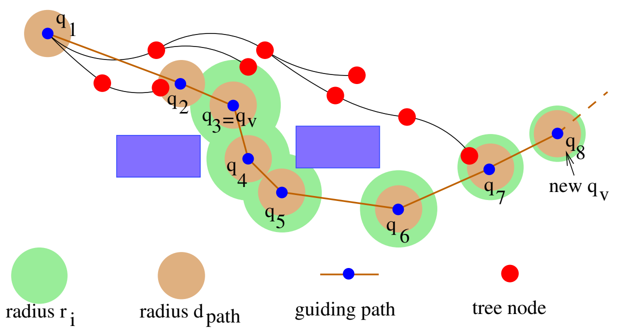


Figure 3.7: Example how the tree can skip part of the guiding path. The brown circles represent radius d_{path}, green circles represent decreasing sampling radius r_i. The probability of generating random sample around path is given by r_i. The actual virtual goal is set to $q_v = q_3$, so the guiding path is sampled starting from point q_3. If the tree approaches point q_7, the new virtual goal will be set to q_8.

Algorithm 3: RRT–Path: guiding a tree along a guiding path

```
Input: Configurations $q_{start}$, $q_{goal}$, maximum number of planning iterations $I_{max}$, goal-bias $p_{goal}$,
       initial value of radius $R_{vg}$, distance $d_{goal}$, guiding path $P = (q_1, \ldots, q_n)$
Output: Trajectory between $q_{start}$ and $q_{goal}$ or failure
1  $\mathcal{T}.add(q_{start})$;
2  $v = 1$; // index of the virtual goal is set to the first point on the guiding path
3  for $iteration$=$1$:$I_{max}$ do
4      if $rand(0,1) < p_{goal}$ then
5          update radius $r_i$; // Eq. (3.1)
6          $q_k$ = selected random point on guiding path according $r_i$;
7          $q_{rand}$ =random configuration around $q_k$ from $N(q_k, \Sigma_k)$;
8      else
9          $q_{rand}$ = random configuration from $\mathcal{C}$;
10     end
11     $q_{near}$ = nearestNeighbor($\mathcal{T}$,$q_{rand}$);
12     expandTree($q_{near}, q_{rand}$);
13     $(v, R'_{vg})$ = updateVirtualGoal($P$,$\mathcal{T}$,$v$,$R'_{vg}, R_{vg}$);
14     $d$ = distance from tree $\mathcal{T}$ to $q_{goal}$;
15     if $d < d_{goal}$ then
16         return extract trajectory from $q_{start}$ to $q_{goal}$;
17     end
18 end
19 return failure;
```

Algorithm 4: updateVirtualGoal($P, \mathcal{T}, v, R'_{vg}, R_{vg}$).

```
Input: Guiding path $P = (q_1, \ldots, q_n)$, configuration tree $\mathcal{T}$, index of virtual goal $v$, actual radius
       $R_{vg}$
Output: Index of new virtual goal $v$, attraction radius $R_{vg}$ of the virtual goal
1  $approached$ = false;
2  $v_{new} = v - 1$ ;
3  for $i = n, n-1, \ldots, v+1, v$ do //testing points from last point on the guiding path
4      $q_{near,i}$ = $\mathcal{T}$.nearestNeighbor($q_i$);
5      $d_i = \varrho(q_{near,i}, q_i)$;
6      if $d_i < d_{path}$ then
7          $v_{new} = i$;
8          $approached$=true;
9          break;
10     end
11 end
12 $v_{new} = \min(n, v_{new} + 1)$;
13 if $approached$=$true$ then
14     $R'_{vg} = R_{vg}$;
15 else
16     $R'_{vg} = R'_{vg}(1 + \alpha)$;
17 end
18 return $(v_{new}, R'_{vg})$;
```

be difficult to follow it exactly, especially with non-holonomic robots. In such a case, the radius needs to be automatically increased to explore more space around the guiding path. The radius R'_{vg} is adapted as

$$R'_{vg} = \begin{cases} R_{vg} & \text{if virtual goal is moved in current iteration} \\ R'_{vg}(1+\alpha) & \text{otherwise,} \end{cases} \tag{3.3}$$

where $0 \leq \alpha \leq 1$ is the adaptation rate, and R_{vg} is the initial value of the radius. Practically, R'_{vg} can be limited, so it is not greater than size of the environment.

The adaptation process is depicted in Fig. 3.8, where a trajectory has to be found through a narrow passage. The tree easily follows the guiding path until it approaches the narrow passage (iteration 143). The tree cannot enter the narrow passage during iterations 144–368, and R'_{vg} is automatically increased, because the virtual goal is not moved. The increase of the radius R'_{vg} causes that random samples are drawn also from the neighborhood of the narrow passage, which increases probability, that a tree will find the entrance to the narrow passage. In iteration 369, the tree finally traverses the narrow passage and the radius R'_{vg} is set back to the initial value R_{vg}.

RRT is probabilistically complete which is ensured by the sampling of the whole configuration space and the nearest-neighbor rule used to select the nodes for the expansions. To preserve probabilistic completeness of the RRT–Path method, it has to be ensured, that the whole configuration space can be sampled with a non-zero probability. This can be achieved either if $p_{goal} < 1$ or if $R_{vg} > 0$ and $\alpha > 0$. In the first case, the configuration space is sampled with probability $1 - p_{goal}$ (line 7 in Alg. 3). If this probability is zero ($p_{goal} = 1$), we need to ensure that q_{rand} can be generated from the whole configuration space. If the initial radius along the virtual goal $R_{vg} > 0$ and the adaptation mechanism is allowed ($\alpha > 0$), then this radius can be enlarged to cover the whole configuration space. This ensures, that the random samples can be generated from the whole configuration space.

3.2.1 Construction of a guiding path

An ideal guiding path should be constructed directly in the configuration space, which would require to solve the generalized path planning problem. A simple and faster approach is to construct the guiding path in the workspace. In the case of typical mobile robots with two or three DOFs, the guiding path can be computed as a geometric path in the workspace using basic path planning methods mentioned in Section 2.1. Examples of guiding paths computed by these methods are depicted in Appendix B.

These methods compute 2D guiding paths (in the case of 2D workspaces) as a sequence of n two-dimensional points $(p_1, \ldots, p_n), p_i \in \mathcal{W}$. For the purpose of RRT–Path, the points need to be extended to the configuration space of the robot. For example, a point $p_i = (x, y) \in \mathcal{W}$ can be converted to a configuration $q_i = (x, y, \varphi) \in \mathcal{C}$ of a Differential drive or Car-like robot by setting $\varphi = 0$. This path can guide the tree using x and y dimensions and the tree randomly samples the third dimension that describes heading of the robot. A more practical approach is to consider also the heading φ, e.g. by setting φ according the angle of the segment starting at the corresponding point.

The path planning methods computed in the polygonal domain consider either a point holonomic robot, or a disc robot. In the latter case, path planning is computed on the map where the obstacles are enlarged by the radius of the robot. To consider shape of a robot during the computation of the guiding path, the PRM path planner can be employed, because it considers full geometry of the robot during the roadmap construction process. Another advantage of PRM-based guiding path is that it can provide guiding path directly in the configuration space of the robot, therefore no conversion between workspace points and configuration space points is required. A disadvantage of PRM is that it can be slower than the basic path planning methods, therefore it is recommended to used it only for robots that cannot be approximated by a disc.

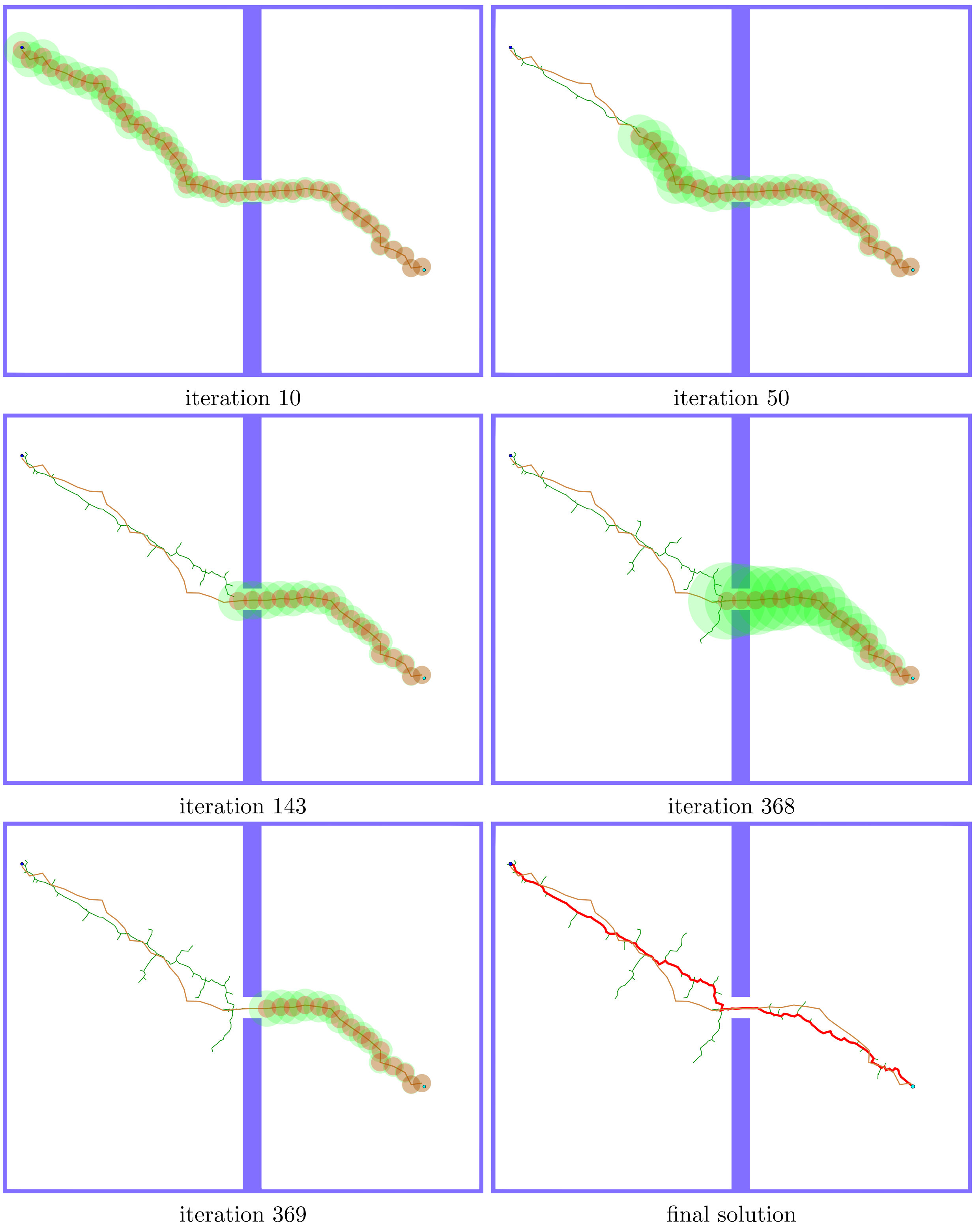


Figure 3.8: Example of guiding process in a narrow passage. The guiding path is depicted in brown and the green circles represent the radius r_i. During the iterations 143–368, the tree cannot enter the narrow passage, which leads to increase of R'_{vg} and consequently to increase of r_i. After the tree traverses the narrow passage, the radius R'_{vg} is decreased to its initial value R_{vg}.

3.3 Practical recommendations

Parameters of RRT–Path are the radius R_{vg}, goal-bias p_{goal}, the adaptation rate α and the threshold d_{path}. Beside that, the size of the goal region defined by d_{goal} needs to be specified. The radius R_{vg} defines how precisely is the guiding path followed by the configuration tree. Large value of R_{vg} speeds up progress of the virtual goal, but it may be counterproductive in the narrow passages, because it will generate the random samples from a large area instead from the narrow passage only. A small radius R_{vg} may slow down the tree growing process, especially with the non-holonomic robots, that cannot follow the guiding path precisely. To minimize the effect of wrongly set R_{vg}, the radius is automatically adapted according the ability of the tree to follow the guiding path. The rate of adaptation is controlled by the parameter α. Depending on the rate α, it may take several iterations to adapt the radius R_{vg} to a suitable value. As an initial setting, we recommend to set $R_{vg} = 2d_{goal}$. We also recommend to set the adaptation rate α to few percents, e.g. $\alpha = 0.01$ or $\alpha = 0.02$. This recommendation is based on many experiments, and it is also discussed in the next chapter in Section 4.4.

The parameter d_{path} determines how precisely the tree has to follow the path. Too small d_{path} means that the tree has to approach all points closely, which can increase the number of planning iterations. Contrary, too high d_{path} allows the tree to rather explore the configuration space without following the path precisely. We recommend to set this parameter as the maximum distance that the robot can traverse during the timestep Δt.

The guiding path is described by a set of its points. In each iteration of the algorithm, the nearest neighbors from the tree towards all path points need to be found. To speed this searching process, it is convenient to represent the guiding path using less number of points. We recommend to represent the guiding path by points so the distance of two consecutive points equals to the maximum distance that a robot is able to perform during the expansion step.

3.4 Discussion

The design of RRT–Path is motivated by the fact that the biased sampling of the configuration space can help to cope with the narrow passage problem and it can speed up the growth towards a given region. The effect of biased sampling in the presence of the narrow passages was studied mainly with PRM-based planners [191, 138, 247, 22, 173, 174, 172, 171, 24, 27, 98, 95]. The second motivation for RRT–Path is that knowledge about workspace can help to sample the configuration space. The knowledge can be represented e.g. by medial axis of the workspace [270, 68, 84, 90, 276]. These methods proposed for PRM cannot be used for RRT, because RRT constructs the configuration tree simultaneously with the sampling of the configuration space, and pure modification of sampling distribution does not ensure that a tree can grow towards the densely sampled areas.

The most relevant papers to our method are RRT-based planners [238] and [20]. In [238], a key-configuration is used to focus sampling to difficult areas of the configuration space. Authors suggest to place the key-configuration in a way that connects easy and difficult areas of the configuration space. For example, the key-configuration can be placed at the enter to a narrow passage. Detailed method for computing the key-configurations is however not presented in [238]. Modified RRT with key-configurations, called RRT–KC, uses two trees. The first tree $\mathcal{T}$ is rooted at q_{start} and the second tree $\mathcal{T}_{key}$ is rooted at the key-configuration. The method alternately generates random samples from the whole configuration space, around the key-configuration and around q_{goal} in order to extended the first tree $\mathcal{T}$. The second tree $\mathcal{T}_{key}$ is extended towards random configurations sampled from the whole configuration space or around q_{goal}. The planner terminates if one of the trees approaches q_{goal} close enough or if the maximum number of iterations exceeds.

During the sampling, RRT–KC attempts to join the trees. Usually, the connection is realized using the closest nodes $q_1 \in \mathcal{T}$ and $q_2 \in \mathcal{T}_{key}$. A feasible trajectory between the nodes q_1 and q_2 has to be computed, which leads to the two-point boundary problem that can be solved analytically only for limited class of system [40]. Therefore, the connection needs to be realized using numerical approaches, e.g. by perturbing control inputs of one tree [142, 40] or by shifting the second tree $\mathcal{T}_{key}$ so that $q_1 \equiv q_2$ [238]. The RRT–KC method suffers from the same issues as the bidirectional search [153, 135] or RRT-based methods with multiple trees [157, 46, 234, 266, 202, 224] as is discussed in Section 2.4.1.

RRT–KC is suitable for environments with a single narrow passage, because only a single key-configuration is used. In comparison to RRT–KC, the herein proposed RRT–Path method can guide the tree through multiple difficult areas due to the utilization of the guiding path. Moreover, RRT–Path builds only one tree, which is faster than building two trees and maintaining connections between them.

In [20] approach, the growth of the tree is biased using the configurations of the previously found trajectory. The goal configuration is sampled with probability p_{goal}, the trajectory is sampled with probability p_{traj} and the whole configuration space is sampled with probability $(1 - p_{goal} - p_{traj})$. To sample the trajectory, random point is selected on the trajectory from uniform distribution and this point is used as q_{rand}.

The approach [20] differs from RRT–Path in three main aspects. First, [20] requires to compute a trajectory in the configuration space in order to bias the sampling in the next iteration, while RRT–Path relies on a guiding path computed in the workspace. Second, the trajectory is sampled in [20] blindly without considering ability of the tree to reach it. The blind sampling along an old trajectory may become counterproductive in environments with obstacles, as is discussed in the motivation experiment in Section 3.1.2. Third, the trajectory is sampled in [20] exclusively using the trajectory points, while RRT–Path samples the configuration space around the guiding path, which is ensured by the parameter R_{vg}.

To conclude, RRT–Path utilizes the workspace knowledge more than RRT–KC, as the whole sequence of points of the guiding path is used to steer growth of the configuration tree through the configuration space. The sampling around the guiding path is not strict, as in [20], because RRT–Path generates the random samples in radius R_{vg} around the guiding path. Moreover, the sampling of the guiding path is adaptive (using the adaptation rate α) and it can therefore find a solution even if the guiding path cannot be followed precisely by the tree.

The RRT–Path method changes only sampling of the configuration space and it can therefore be combined with other RRT-based methods. A possible extension can be made for example with RRT–Blossom, that expands the tree using multiple nodes in each expansion step. This would provide more dense trees with higher chance to follow the path.

Chapter 4

Experimental verification of RRT–Path

4.1 Experiment setup

The performance of the proposed RRT–Path algorithm has been compared with several state-of-the-art RRT-based planners in the scenario of motion planning of mobile robots in 2D environments.

4.1.1 Motion models

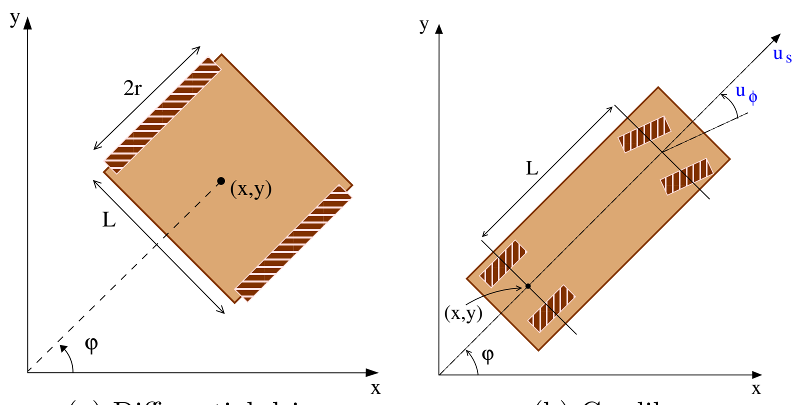
The experiments have been performed with the non-holonomic Car-like and Differential drive robots as well as with the holonomic non-points mobile robots. Configuration $q = (x, y, \varphi)$ describes position (x, y) and heading φ of the robots. The Differential drive robot (Fig. 4.1a) has two actuated independent wheels allowing the robot to turn on spot and its motion model is $\dot{x} = \frac{r}{2}(u_l + u_r)\cos\varphi$, $\dot{y} = \frac{r}{2}(u_l + u_r)\sin\varphi$, and $\dot{\varphi} = \frac{r}{L}(u_r - u_l)$, where u_l, u_r are control inputs — velocities of the left and right wheel respectively. The radius of the wheels is denoted r and their distance L. Motion of the Car-like robot (Fig. 4.1b) is described as $\dot{x} = u_s \cos\varphi$, $\dot{y} = u_s \sin\varphi$, and $\dot{\varphi} = \frac{u_s}{L}\tan u_\phi$, where u_s, u_ϕ are control inputs, u_s is the forward velocity, and u_ϕ is the steering angle. The distance between front and rear wheels is denoted L.

The experiments have been performed with two Differential drive robots (Diff$_\uparrow$ and Diff$_\updownarrow$), two Car-like robots (Car-like$_\uparrow$ and Car-like$_\updownarrow$) and with two holonomic 2D robots (2D$_{20\times100}$ and 2D$_{20\times50}$). The wheel velocities of Differential drive robots are $(-2 \leq u_l, u_r \leq 2)$, and radius of each wheel is $r = 10$, which gives us maximum forward velocity 20 mu.s^{-1}. The difference between Diff$_\uparrow$ and Diff$_\updownarrow$ is, that Diff$_\uparrow$ can move only forward, i.e., $u_l + u_r \geq 0$. The Differential drive robots are represented by a box of size 20×20 mu.

The Car-like robots are represented by rectangle 20×30 mu. The bidirectional Car-like robot, denoted as Car-like$_\updownarrow$ can move forward and backwards, therefore $-20 \leq u_s \leq 20$. The backward motion is prohibited for Car-like$_\uparrow$ robot, and $0 \leq u_s \leq 20$. The turning angle of Car-like robots in degrees is $-40 \leq u_\phi \leq 40$. The maximum forward speed of the used Car-like robots is same as the speed of the Differential drive robots. The 2D holonomic robots 2D$_{20\times100}$, 2D$_{20\times50}$ can move arbitrary and they are represented by 2D boxes. The subscripts denote size of the robots.

4.1.2 Algorithm setup

Three RRT-based planners are used for the comparison: RRT [150], RRT–ADD (RRT with Adaptive Dynamic Domain) [108], and RRT–Blossom [111]. RRT–Blossom can extend the tree using multiple configurations during each iteration. The trees constructed using RRT–Blossom have typically more nodes than the trees constructed by the other methods. RRT–Blossom is designed to cope with the robots under differential constraints and it is therefore more suitable



(a) Differential drive (b) Car-like

Figure 4.1: Illustration of Differential drive and Car-like mobile robots.

for the Differential drive and Car-like robots than for 2D holonomic robots. The goal-bias of RRT and RRT–Blossom $p_{goal} = 0.15$ is used according to experiment described in Sec. 3.1.2.

In RRT–ADD, each node in the tree has assigned an activation radius. A node can be selected for the expansion only if the random configuration q_{rand} lies in the distance defined by the activation radius. RRT–ADD was originally proposed for high-dimensional path planning of solid 3D objects, therefore this method should be suitable especially for the 2D holonomic robots. RRT–ADD is run with initial activation radius $R_{dd} = 75$ mu and learning rate $\alpha_{dd} = 0.01$.

The planners are implemented in C++ and compiled using gcc compiler version 4.2.1 with O2 flag. The experiments have been performed on PC Intel Pentium IV @ 2.2 GHz. All planners employ Rapid library [82] for collision detection and MPNN [280] library for the nearest-neighbor search. The forward motion models are integrated over time $\Delta t = 0.5$ s with 0.125 s time step of numerical integration during each expansion step. The resolution of the straight-line planner used in the case of the 2D robots is set to $\varepsilon = 0.5$ mu. Collision detection is evaluated three times per each edge. The experiments have been performed in four 2D maps (the maps are depicted in Fig. 3.2).

4.1.3 Performance evaluation

In literature, sampling-based planning algorithms are usually compared using the runtimes required to find a solution, or number of collision detection queries. A comparison based on quality of solutions is not suitable for the non-optimal planners like RRT and PRM, therefore it is not common. Many papers compare sampling-based planners only using single start/goal configuration, which can be skewed due to sensitivity of the planners to the relative position of start and goal configurations. This was shown in Section 3.1.1, where significantly different results were obtained for two goal configurations despite they are located in the same distance from the start configuration.

To investigate an overall performance of a planner in a given environment, we propose a novel testing setup using multiple start/goal configurations. The evaluation is motivated by applications of motion planning in mobile robotics, where multiple trajectories have to be computed either as a part of on-line replanning system (e.g. in warehouse systems [272] or in robotic soccer [20]) or as a part of a reasoning system for robotic exploration [62] or inspection [63, 136]. In these applications, many trajectories need to be computed between various places in the environments. A planner with an overall good performance is preferred than a planner that excels in few start/goal pairs, but fails otherwise.

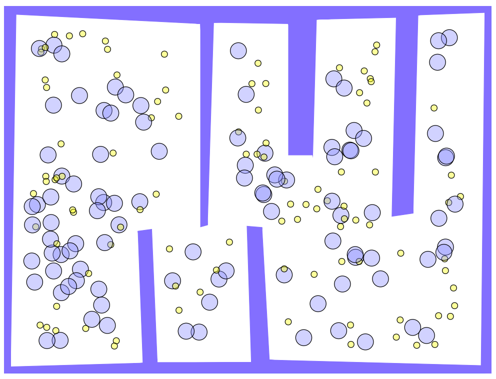
For a given robot and environment, g pairs of random start/goal configurations (q_{start}, q_{goal}), where $q_{start}, q_{goal} \in \mathcal{C}_{free}$ and $\varrho(q_{start}, q_{goal}) > 2d_{goal}$ are generated (an example is shown in Fig. 4.2a). A planner being tested then attempts to compute m plans (trials) for each start/goal pair. Therefore, each planner is run $g \cdot m$ times on each map. For each start/goal pair, a success rate (referred to as $s-$rate in the rest of the text) denotes the percentage of trials where the planner approached the goal configuration to a distance less than d_{goal}.

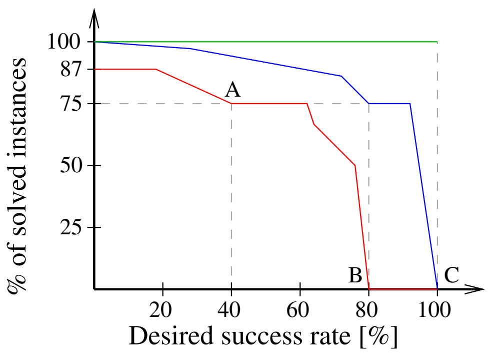
The overall performance of a planner on the multiple start/goal pairs can be described as the percentage of the start/goal pairs for which the planner achieved a desired minimal $s-$rate. An example is depicted in Fig. 4.2b. An ideal planner that is able to find plans between all tested start/goal pairs with 100 % probability, would result in the green line in Fig. 4.2b. A worse performance is indicated by the blue line, that can solve 75 % of start/goal pairs with at least 80 % probability, but there are instances, that cannot be solved with 100 % success rate with this algorithm (point C). The red line represents the performance of an even worse planner. The highest success rate for the red curve is 87 %. This indicates that 13 % start/goal pairs are not solved by this algorithm at all. The third algorithm can solve 75 % of instances with $s-$rate at least 40 % (point A), but none of the start/goal pairs can be solved with probability higher than 80 % (point B).

To compare several planners, a desired minimal reliability can be selected and the percentage of solved start/goal pairs is then used for the comparison. In this thesis, we compare performance of the tested methods using desired success rate 80 %. For example, the comparison of three curves depicted in Fig. 4.2b is: green=100 %, blue= 75% and red= 0%, and the planner with the green curve would be considered as the best one.

The start/goal configurations should be generated in such a way, that at least one tested planner can compute plans for all of them. The randomly placed start/goal pairs used in all experiments in this thesis are always reachable. It is ensured, that the narrow passages are wider than the size of the robots and the robots are placed into a sufficient distance from the obstacles. Moreover, the random start configurations for Diff$_{\uparrow}$ and Car-like$_{\uparrow}$ robots have orientation towards the free space in order to ensure that the robots can move from the configurations.

Also other variables such as runtime, number of planning iterations or size of constructed trees are measured. The runtimes and planning iterations are shown as boxplots with highlighted median. The boxplots are created from all $g \cdot m$ planning trials. It should be noted that these values contain measurements from close start/goal pairs as well as from distant start/goal pairs. Statistical comparison between runtimes or number of planning iterations of various planners is not provided, since the main aim of the test is to investigate an overall reliability of the planners in given environments. Other variables like number of collision detection queries of collision detection time are not reported. All planners use same supporting libraries (for collision detection and nearest-neighbor search) and the time spent in these routines is given by the number of realized planning iterations.


a

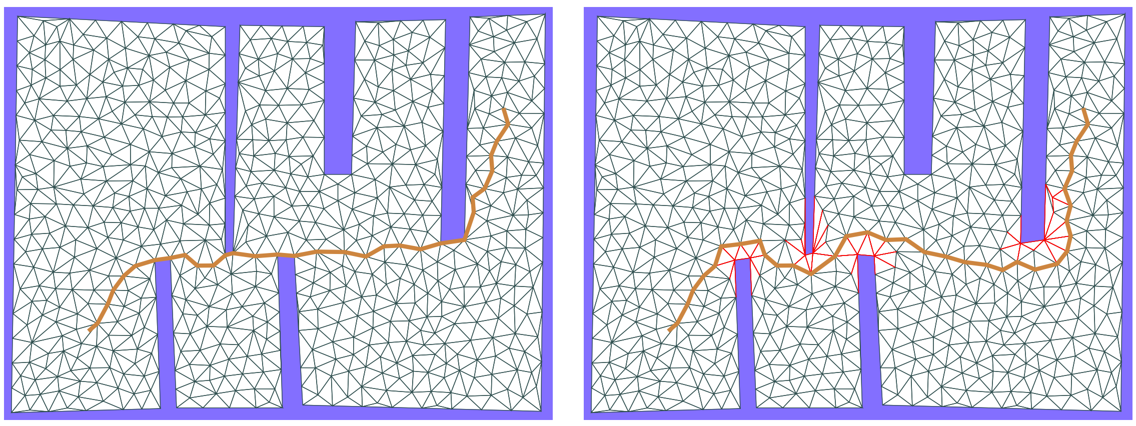


b

Figure 4.2: An example of random start (yellow) and goal configurations (gray) in the Simple map with d_{goal} = 20 mu (a). The graph (b) shows three examples of the progress of the percentage of solved instances with increasing $s-$rate.

4.2 Performance of RRT–Path with various guiding paths

The performance of RRT–Path with various guiding paths is investigated in the first set of experiments. The guiding paths are computed using Delaunay Triangulation (DT), Voronoi diagram (VD), Probabilistic roadmaps (PRM), triangle mesh (TMesh) and updated triangle mesh (TMesh2). The methods TMesh and TMesh2 decompose free regions of the workspace to triangles. The decomposition is described by a graph, where the nodes are the vertices of the triangles and edges connect the vertices of the triangles. The cost of an edge is given by the Euclidean distance of its end points. The Dijkstra's algorithm is used to find a shortest path in the graph. This path can touch obstacles. In order to increase clearance of this path, the method TMesh2 increases the cost of all edges incident with the path and touching the obstacles. After the cost is increased, another path is computed, which results in a path of higher clearance. This technique is similar to Lazy-PRM [17]. The difference between TMesh and TMesh2 paths is depicted in Fig. 4.3. The test has been performed with $g = 100$ start/goals pairs with $m = 40$ trials per start/goal. RRT–Path was run with maximum number of allowed planning iterations $I_{max} = 5,000$.



(a) TMesh (b) TMesh2

Figure 4.3: A path in TMesh touches the obstacles (a). By increasing cost of edges that touch the obstacles (red), a path with a higher clearance can be found (b).

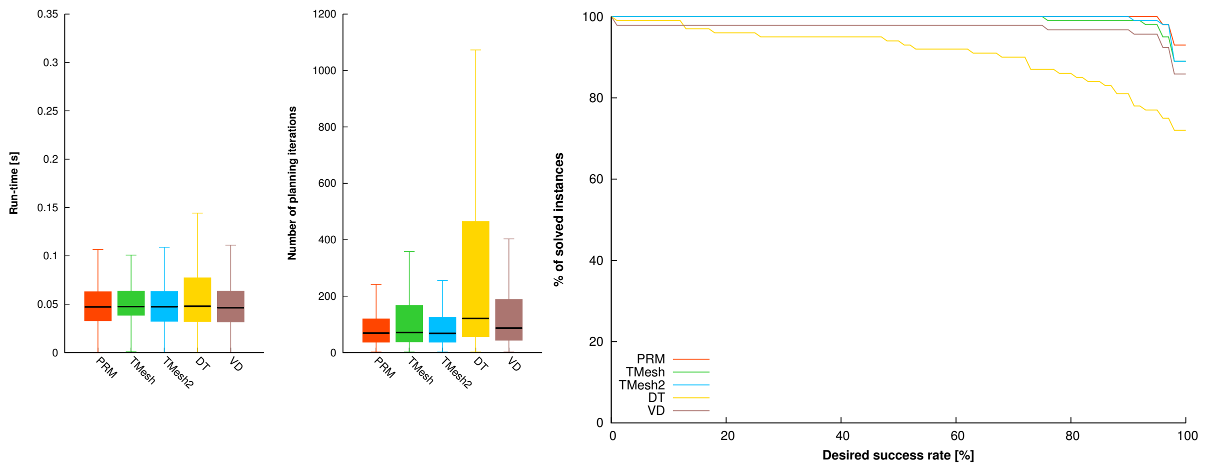


Figure 4.4: Performance of RRT–Path with the Diff$_{\updownarrow}$ robot with various guiding paths in the BugTrap4 map. Performance at 80% $s-$rate: **TMesh2=100 %**, TMesh=99 %, **PRM=100 %**, DT=86 %, VD=97 %.

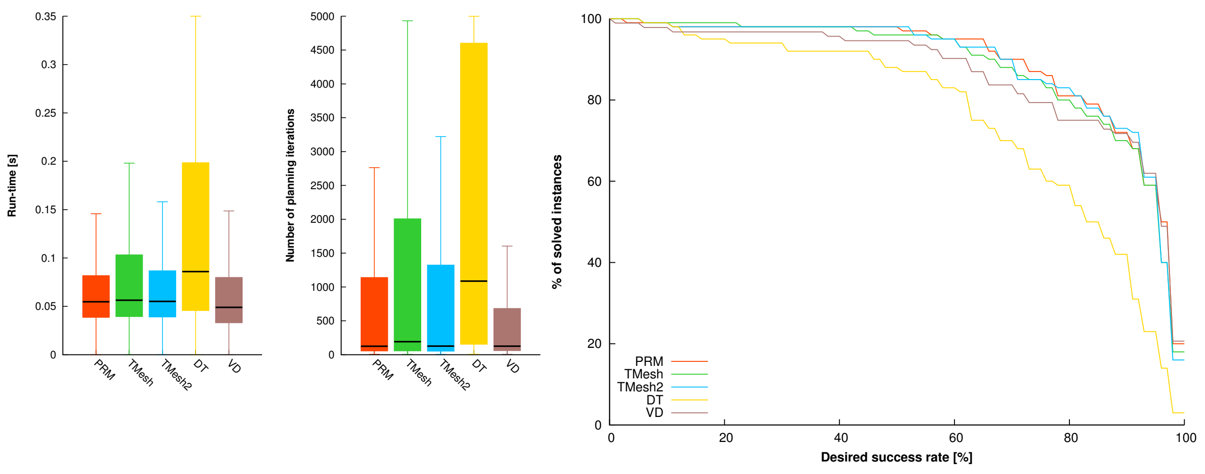


Figure 4.5: Performance of RRT–Path for the Diff$_{\uparrow}$ robot with various guiding paths in the BugTrap4 map. Performance at 80% s−rate: **TMesh2=83 %**, TMesh=80 %, PRM=81 %, DT=59 %, VD=75 %.

Results for the Differential drive robots in the BugTrap1 environment are shown in Fig. 4.4 and Fig. 4.5, and results in the Simple environment are depicted in Fig. 4.6 and Fig. 4.7 respectively. The graphs show boxplots of runtimes and the number of realized planning iterations, as well as the progress of s−rate.

All planners provide plans for most of the start/goals pairs with bidirectional robots. The number of solved start/goal pairs is lower if Diff$_{\uparrow}$ and Car-like$_{\uparrow}$ robots are used. In these cases, all planners show worse performance, but the lowest number of solved start/goal pairs is with DT-based guiding paths. The paths constructed using DT are too close to the obstacles and it is not easy to grow the tree along them, as expansions of many nodes possibly lead to collisions with the obstacles. More iterations are therefore needed in order to expand the tree properly. As the number of planning iterations I_{max} is limited, RRT–Path initialized with the DT-based path may fail to find useful solutions for certain start/goal pairs. Consequently, the percentage of solved pairs is lower than for other methods. The runtimes of RRT–Path with paths computed by VD, PRM, TMesh and TMesh2 are similar.

The ability of RRT-based planners to construct a tree is also influenced by the maneuverability of the robots. The comparison of the planning iterations (Fig. 4.4 vs. Fig. 4.5) shows that planning for the Diff$_{\updownarrow}$ robot requires less number of iterations than for the Diff$_{\uparrow}$ robot. This is caused by the limited maneuverability of the Diff$_{\uparrow}$ robot that can move only forward and it is therefore more difficult to follow the guiding path and to construct the tree during the limited amount of planning iterations. Consequently, solutions are provided for lower percentage of the start/goal pairs.

The guiding process for the 2D holonomic robots is influenced by the quality of guiding paths more than planning for the Differential drive robots. Although the 2D robots can move arbitrary in the environment, the quality of guiding paths is more important than for the Differential drive robots, because the 2D robots are larger than the Differential drive robots. The results are shown in Fig. 4.8 and Fig. 4.9. Similarly to the tests with the Differential drive robots, the worst performance is achieved with DT-based guiding path and the second worst performance is achieved with the TMesh method. These two methods produce guiding paths that touch the obstacles, therefore it is not easy to follow them by the 2D robots. Better results are achieved with the guiding paths computed on the updated mesh (TMesh2) that provides the guiding paths without touching the obstacles.

We can conclude, that the guiding paths computed by DT or TMesh are not suitable for RRT–Path, as these have low clearance. Sampling along such paths is more difficult, especially

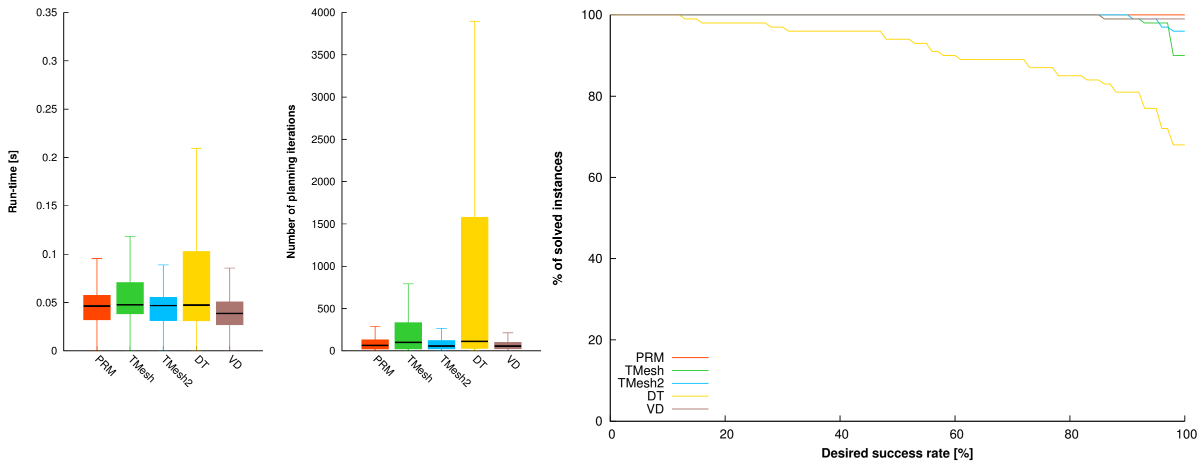


Figure 4.6: Performance of RRT–Path for the Diff$_\updownarrow$ robot with various guiding paths in the Simple map. Performance at 80% $s-$rate: **TMesh2=100 %**, **TMesh=100 %**, **PRM=100 %**, DT=85 %, **VD=100 %**.

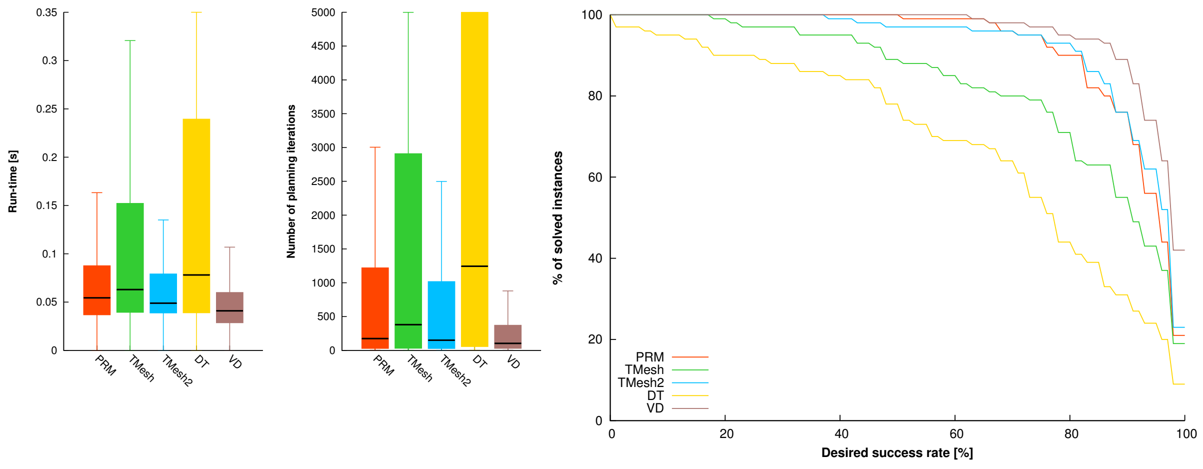


Figure 4.7: Performance of RRT–Path for the Diff$_\uparrow$ robot with various guiding paths in the Simple map. Performance at 80% $s-$rate: TMesh2=93 %, TMesh=71 %, PRM=90 %, DT=44 %, **VD=95 %**.

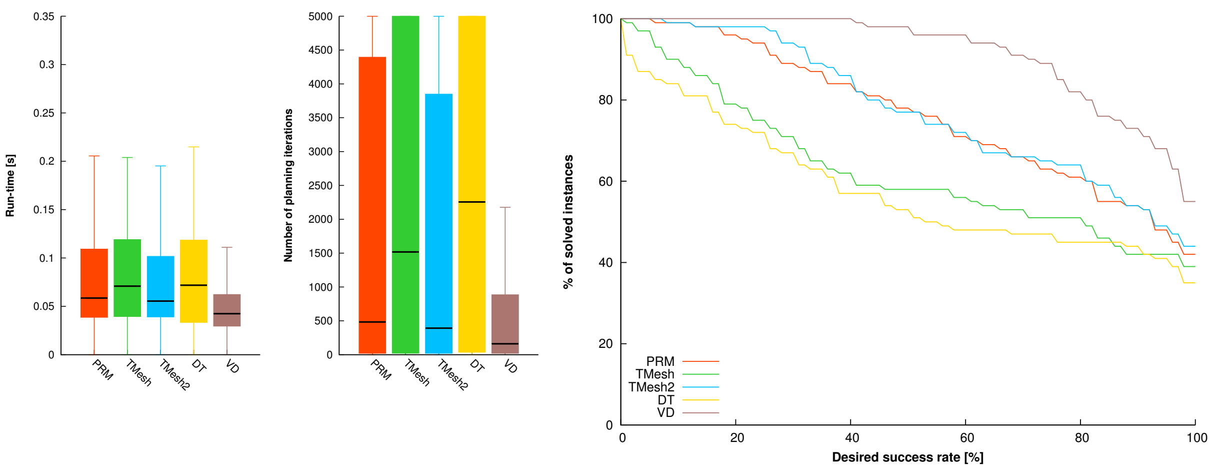


Figure 4.8: Performance of RRT–Path with $2D_{20\times100}$ robot in the Simple map. Performance at 80% $s-$rate: TMesh2=64 %, TMesh=51 %, PRM=61 %, DT=45 %, **VD=82 %**.

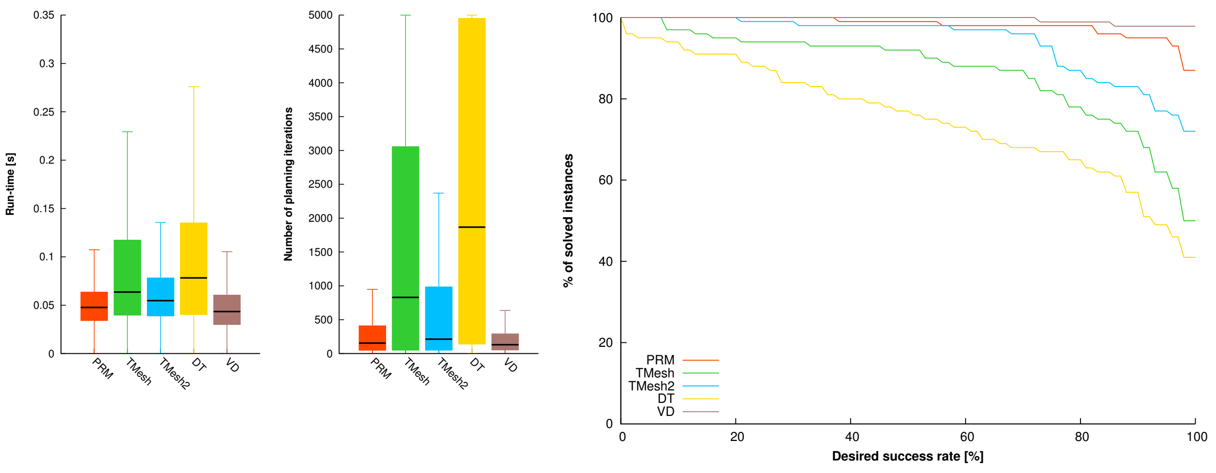


Figure 4.9: Performance of RRT–Path with $2D_{20\times100}$ robot in the BugTrap4 map. Performance at 80% $s-$rate: TMesh2=87 %, TMesh=78 %, PRM=98 %, DT=65 %, **VD=99 %**.

for robots with motion constraints. Guiding paths with suitable clearance are preferred. In the following experiments, guiding paths for RRT–Path are computed using the TMesh2 method.

4.3 Algorithms comparison

Performance of the tested planners in several 2D scenarios is investigated in this section. In each environment, $g = 200$ start/goal pairs have been randomly placed. Each planner has been run $m = 40$ times for each start/goal pair leading to total $200 \cdot 40 = 8,000$ planning trials. The number of planning iterations is set to $I_{max} = 5,000$ for all planners. The experiment has been performed with all mobile robots, i.e., $\text{Diff}_{\uparrow}$, $\text{Diff}_{\updownarrow}$, Car-like$_{\updownarrow}$, Car-like$_{\uparrow}$ and with 2D holonomic robots $2D_{20\times100}$ and $2D_{20\times50}$. Selected results are shown in this section, and the results achieved with remaining combinations of robots and maps are shown in Appendix D.

The performance with the Differential drive robots in the BugTrap4 map is depicted in Fig. 4.10 and 4.11. All planners provide better plans for the bidirectional robot ($\text{Diff}_{\updownarrow}$), which is indicated by the higher percentage of solved start/goal pairs. The better performance with the bidirectional robots is also indicated by the size of constructed trees that are smaller for the $\text{Diff}_{\updownarrow}$ robot. The biggest configuration trees are built by the RRT–Blossom method, because the method extends the tree with several nodes in each iteration. The size of the trees constructed by RRT–Blossom is significantly lower for the Diff$_\updownarrow$ robot than for the Diff$_\uparrow$ robot. The highest percentage of start/goal pairs is solved using RRT–Path.

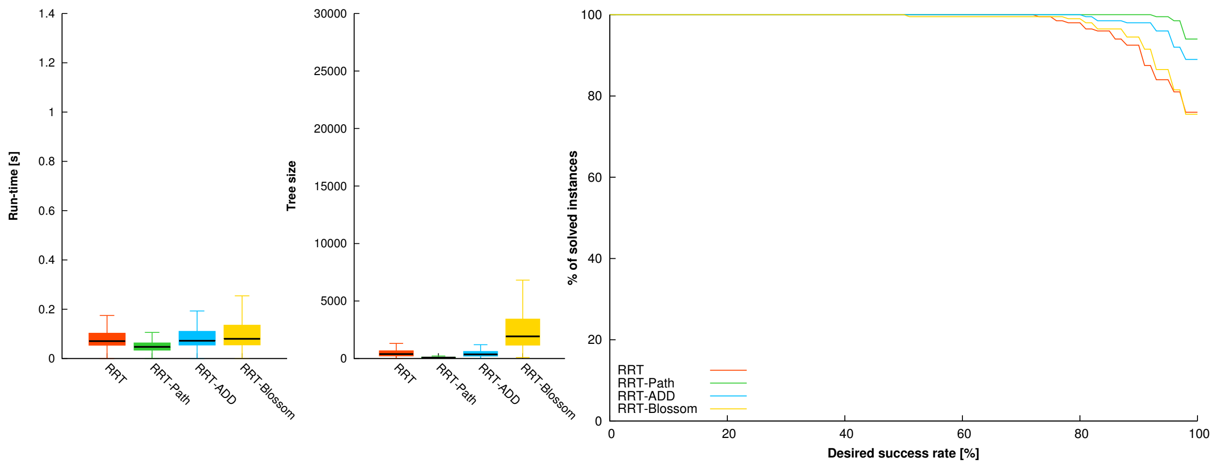


Figure 4.10: Comparison of motion planning for Diff$_\updownarrow$ robot in BugTrap4 environment. Performance at 80% s−rate: RRT=98 %, **RRT-Path=100 %**, RRT-Blossom=99 %, **RRT-ADD=100 %**.

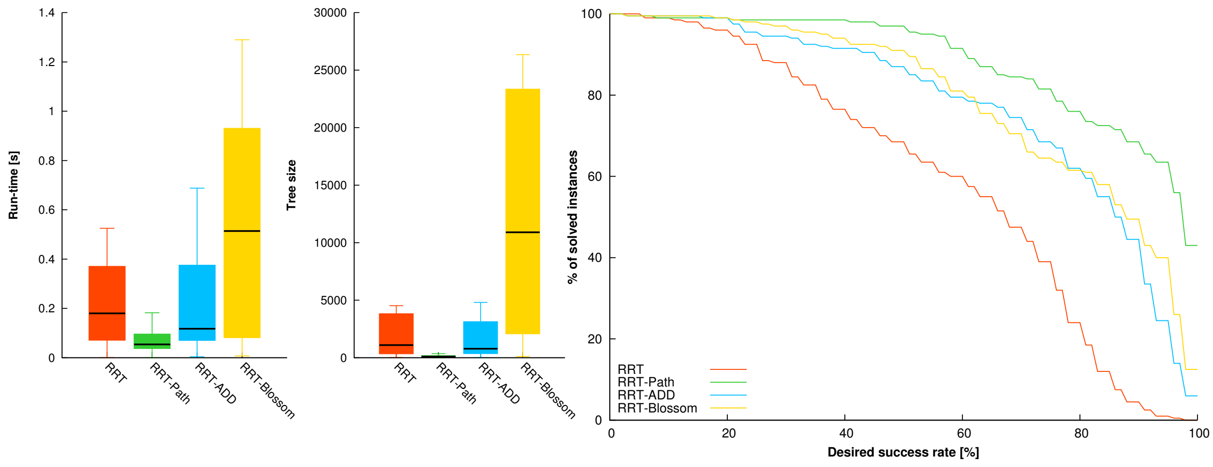


Figure 4.11: Comparison of motion planning for Diff$_\uparrow$ robot in BugTrap4 environment. Performance at 80% s−rate: RRT=24 %, **RRT-Path=76 %**, RRT-Blossom=62 %, RRT-ADD=62 %.

The comparison of motion planning for Car-like robots in BugTrap4 is depicted in Fig. 4.12 and 4.13. The highest percentage of solved start/goal pairs is provided by RRT–Path planner followed by RRT–Blossom planner.

Examples of constructed configuration trees for Car-like$_\updownarrow$ robot are depicted in Fig. 4.14 and for the Car-like$_\uparrow$ robot in Fig. 4.15. The configuration trees constructed for the Car-like$_\updownarrow$ robots contain more sharp turns and the resulting trajectories as well. This is caused by the ability of Car-like$_\updownarrow$ robot to move backward. All the motions, including the backward motions, are used in the expansion step and in certain situations, the backward motions are selected and added to the tree. The sharp turns can be seen also in the configuration trees for the Differential drive robots. Examples are depicted in C.

Results achieved on the Simple map with the Differential drive robots are shown in Fig. 4.16 and Fig. 4.17. All planners are able to construct a plan between almost all start/goal pairs for the Diff$_\updownarrow$ robot (Fig. 4.16), but the percentage of solved start/goal pairs is lower with the Diff$_\uparrow$

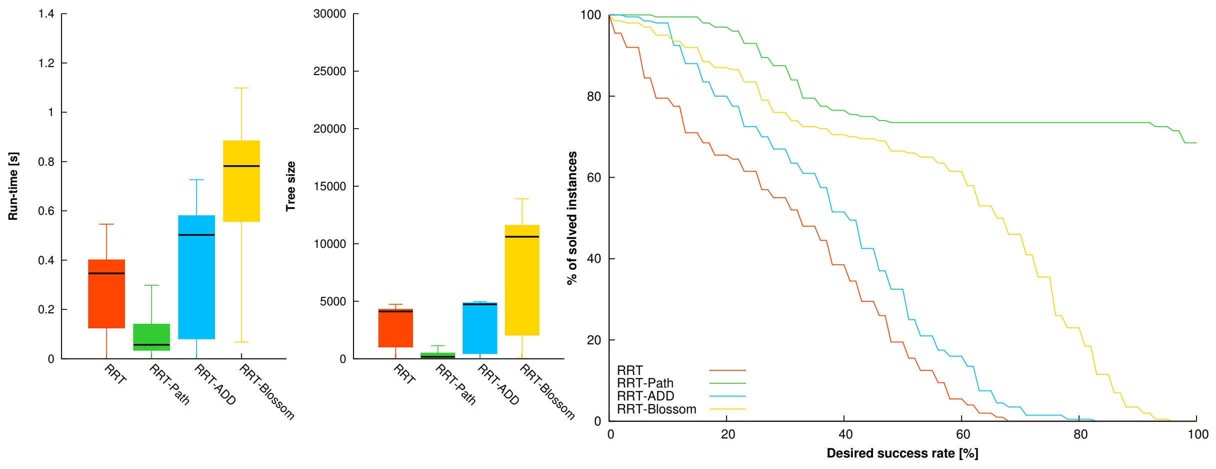


Figure 4.12: Comparison of motion planning for Car-like$_{\updownarrow}$ robot in BugTrap4 environment. Performance at 80% $s-$rate: RRT=0 %, **RRT-Path=74 %**, RRT-Blossom=23 %, RRT-ADD=0 %.

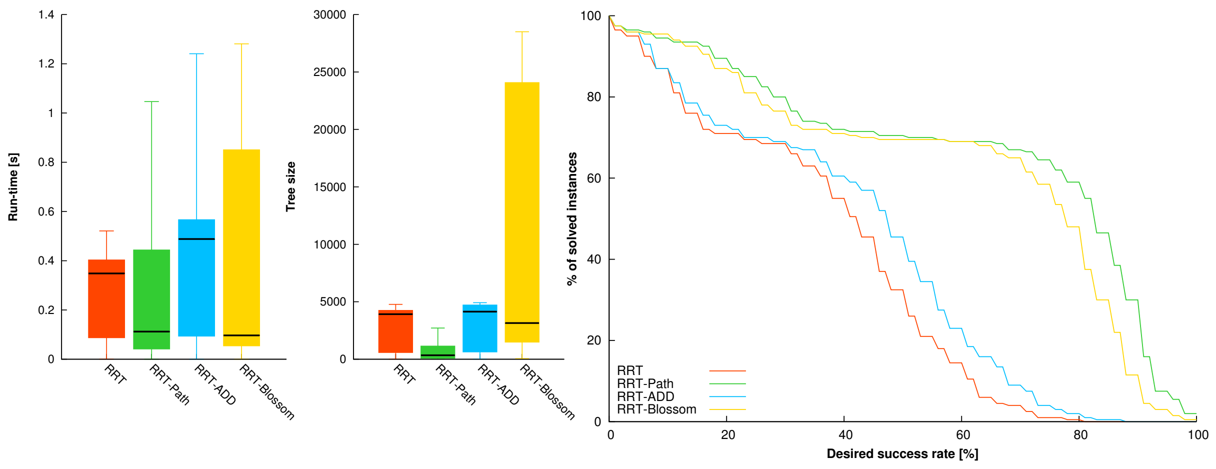


Figure 4.13: Comparison of motion planning for Car-like$_{\uparrow}$ robot in BugTrap4 environment. Performance at 80% $s-$rate: RRT=0 %, **RRT-Path=59 %**, RRT-Blossom=48 %, RRT-ADD=2 %.

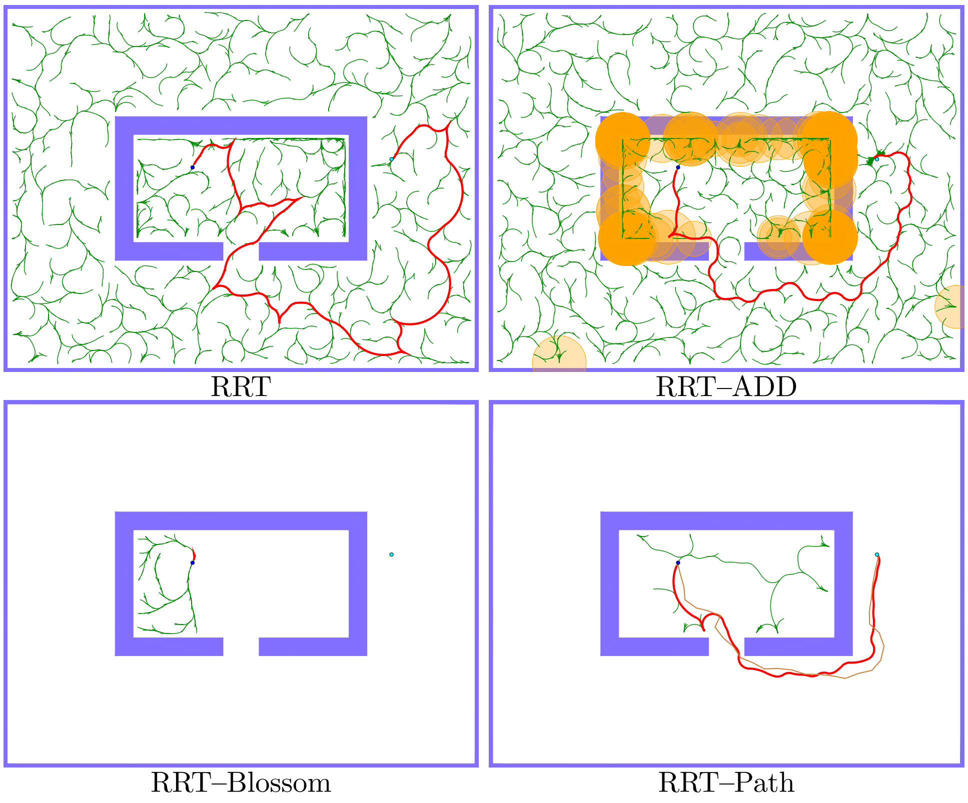


Figure 4.14: Example of motion plans for the Car-like$_{\updownarrow}$ robot in the BugTrap1 environment.

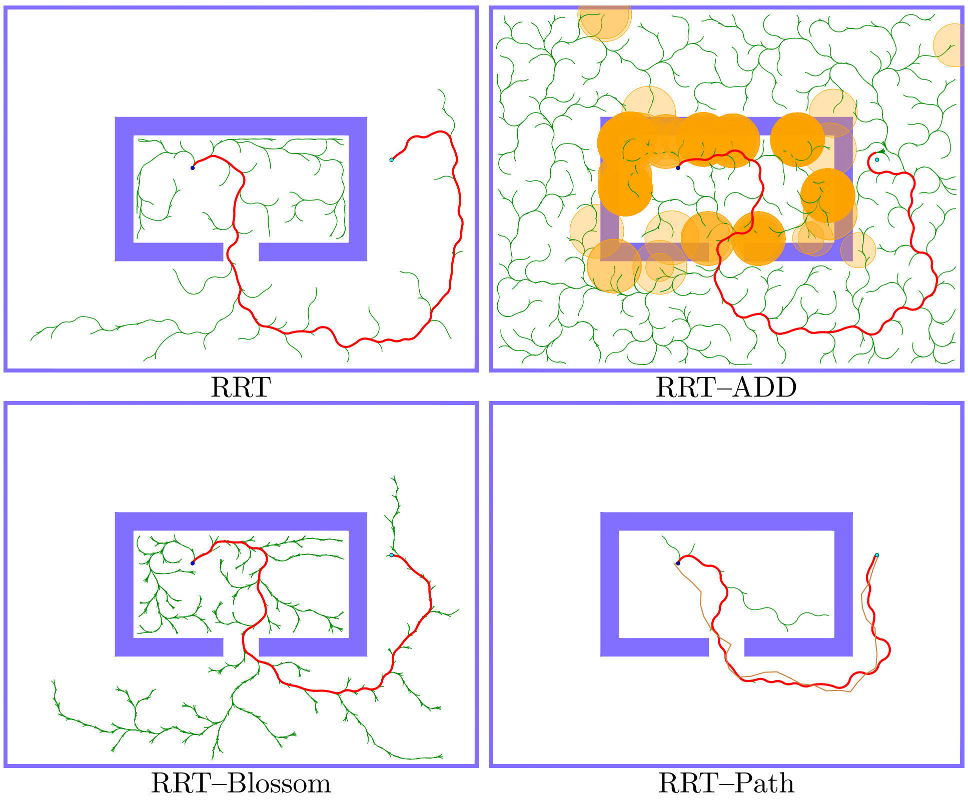


Figure 4.15: Examples of motion plans for the Car-like$_{\uparrow}$ robot in the BugTrap1 environment.

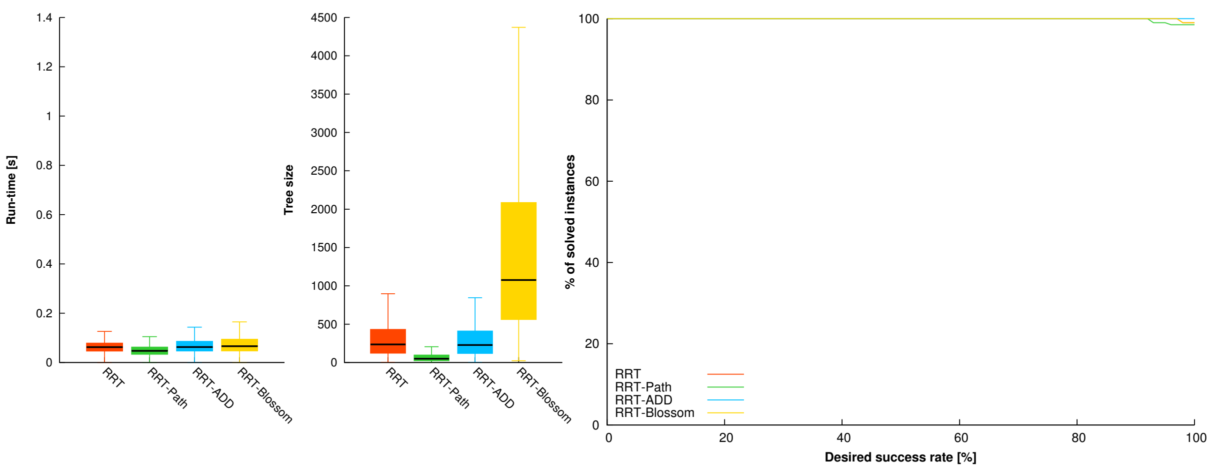


Figure 4.16: Comparison of motion planning for Diff$_\updownarrow$ robot in the Simple map environment. Performance at 80% $s-$rate: **RRT=100 %**, **RRT-Path=100 %**, **RRT-Blossom=100 %**, **RRT-ADD=100 %**.

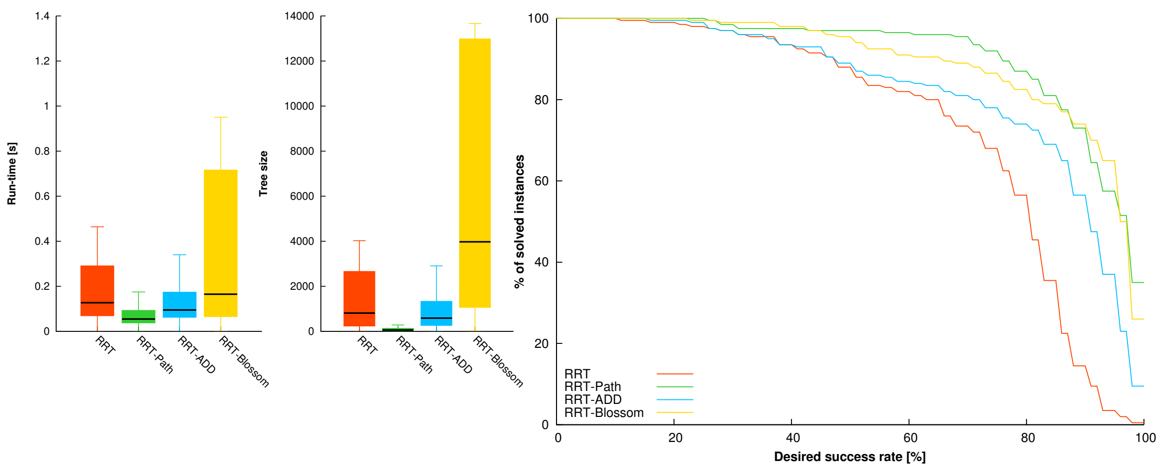


Figure 4.17: Comparison of motion planning for Diff$_\uparrow$ robot in the Simple map environment. Performance at 80% $s-$rate: RRT=56 %, **RRT-Path=87 %**, RRT-Blossom=82 %, RRT-ADD=74 %.

robot. Although there is no narrow passage in the Simple map, the environment is difficult for the Diff$_\uparrow$ robot, as the long walls require a robot to frequently change direction in order to drive around them. The robot has decreased maneuverability, because it can move only forward and it is more hard to move amongst the walls. According to the reliability, the second best results are achieved by RRT–Blossom. From the reliability point of view, RRT–Blossom is comparable to RRT–Path, but RRT–Blossom constructs significantly larger trees.

The planners have also been compared in the path planning task with the 2D holonomic robots. In this task, the RRT-based methods employ the straight-line planner for the expansion of the tree. RRT–Blossom is not considered in this test, as the straight-line planner does not allow to extend the tree by more than one configuration, which is required in RRT–Blossom.

The results achieved for the 2D$_{20\times50}$ and the 2D$_{20\times100}$ robots on the BugTrap4 map are shown in Fig. 4.18 and Fig. 4.19 respectively. According to the progress of $s-$rate, the best results are achieved by RRT–ADD method and RRT–Path provides the second best results. The size of robots influences performance of RRT and RRT–Path methods, because both methods solve less number of start/goal pairs with the 2D$_{20\times100}$ robot than with the 2D$_{20\times50}$ robot. Contrary, the

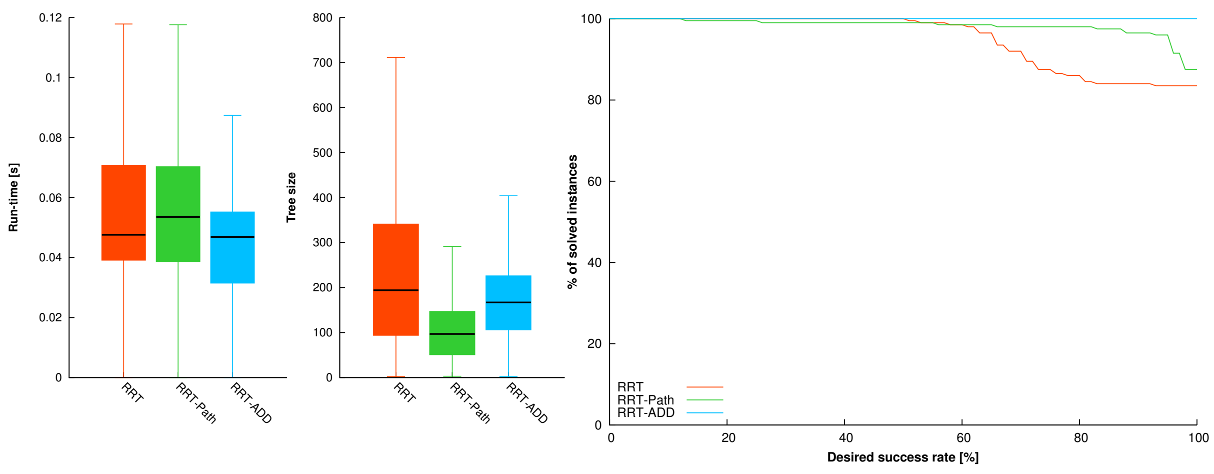


Figure 4.18: Comparison of motion planning for $2D_{20\times50}$ robot in the BugTrap4 environment. Performance at 80% $s-$rate: RRT=86 %, RRT-Path=98 %, **RRT-ADD=100 %**.

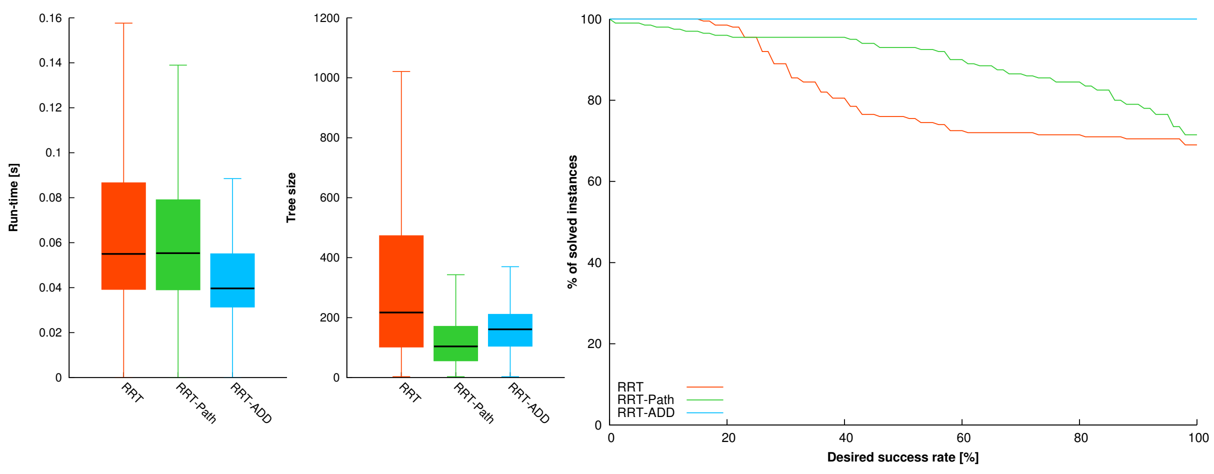


Figure 4.19: Comparison of motion planning for $2D_{20\times100}$ robot in the BugTrap4 environment. Performance at 80% $s-$rate: RRT=72 %, RRT-Path=84 %, **RRT-ADD=100 %**.

success rate of RRT–ADD is the same for both $2D_{20\times50}$ and $2D_{20\times100}$ robots: in both cases, 100 % of instances is solved.

The summary of the experiment is show in Tab. 4.1, where the percentage of start/goal pairs solved with success rate higher than 80 % are shown for all the planners. Out of total 24 tested scenarios, RRT provides the best results in 9 cases, RRT–Path in 17 cases, RRT–ADD in 12 cases. RRT–Blossom has been used only for motion planning of mobile robots and it provides the best results in 4 out of 16 scenarios. RRT–Path is superior especially in environments with the narrow passages (BugTrap1 and BugTrap4).

RRT–ADD is not suitable for motion planning of mobile robots, where it solved less amount of start/goal pairs than other methods, but it is suitable for path planning of the 2D holonomic robots. The method is able to find a feasible path for 2D robots for 100 % of start/goal pairs in all environments.

The basic RRT performs well in maps without difficult narrow passages like Simple, Potholes and BugTrap1, but its performance significantly decreases in BugTrap4 that contains the narrow passage.

The comparison shows that the proposed RRT–Path method improves path and motion planning of mobile robots in the 2D environments. The biggest advantage of the method can

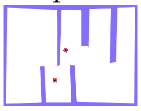
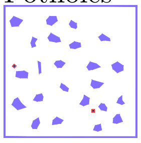
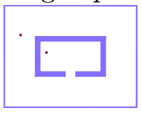
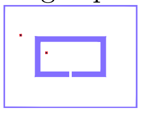
		Percentage of solved start/goal instances			
		RRT	**RRT–Path**	**RRT–Blossom**	**RRT–ADD**
	$\text{Car-like}_{\updownarrow}$	4 %	**90 %**	24 %	8 %
Simple	$\text{Car-like}_{\uparrow}$	1 %	51 %	**68 %**	3 %
	$\text{Diff}_{\updownarrow}$	**100 %**	**100 %**	**100 %**	**100 %**
	$\text{Diff}_{\uparrow}$	56 %	**87 %**	82 %	74 %
	$\text{2D}_{20\times 50}$	**100 %**	92 %	N/A	**100 %**
	$\text{2D}_{20\times 100}$	**100 %**	54 %	N/A	**100 %**
	$\text{Car-like}_{\updownarrow}$	0 %	**100 %**	32 %	1 %
Potholes	$\text{Car-like}_{\uparrow}$	0 %	**78 %**	64 %	2 %
	$\text{Diff}_{\updownarrow}$	**100 %**	**100 %**	**100 %**	**100 %**
	$\text{Diff}_{\uparrow}$	40 %	**93 %**	82 %	65 %
	$\text{2D}_{20\times 50}$	**100 %**	98 %	N/A	**100 %**
	$\text{2D}_{20\times 100}$	**100 %**	83 %	N/A	**100 %**
	$\text{Car-like}_{\updownarrow}$	0 %	**100 %**	29 %	0 %
Bugtrap1	$\text{Car-like}_{\uparrow}$	0 %	**70 %**	66 %	4 %
	$\text{Diff}_{\updownarrow}$	**100 %**	**100 %**	**100 %**	**100 %**
	$\text{Diff}_{\uparrow}$	30 %	**94 %**	78 %	65 %
	$\text{2D}_{20\times 50}$	**100 %**	**100 %**	N/A	**100 %**
	$\text{2D}_{20\times 100}$	**100 %**	98 %	N/A	**100 %**
	$\text{Car-like}_{\updownarrow}$	0 %	**74 %**	23 %	0 %
Bugtrap4	$\text{Car-like}_{\uparrow}$	0 %	**59 %**	48 %	2 %
	$\text{Diff}_{\updownarrow}$	98 %	**100 %**	99 %	**100 %**
	$\text{Diff}_{\uparrow}$	24 %	**76 %**	62 %	62 %
	$\text{2D}_{20\times 50}$	76 %	**100 %**	N/A	**100 %**
	$\text{2D}_{20\times 100}$	72 %	94 %	N/A	**100 %**
Overall		9 / 24	17 / 24	4 / 16	12 / 24

Table 4.1: The percentage of start/goal pairs solved with success rate higher than 80 %. The best performance in each row (for each robot) is in bold face. The last row summarize the number of scenarios where the given algorithm provided best performance out of all scenarios. The number of scenarios is 24 for RRT, RRT–Path and RRT–ADD robots. RRT–Blossom was not used for path planning of 2D robots therefore its total number of scenarios is 16.

be seen in maps with narrow passages (BugTrap1 and BugTrap4), where it solves most of the instances with the desired 80 % probability. The experiments have also shown, that the RRT–ADD method is better for path planning of 2D objects and it is therefore recommended for usage in this scenario.

4.4 Influence of adaptation rate α

The adaptation rate α determines how fast is the radius R'_{vg} increased if the tree cannot properly follow the guiding path. The influence of α to the performance of RRT–Path has been evaluated in all 2D environments. For each learning rate $\alpha = \{0, 0.01, 0.02, 0.05, 0.1\}$, each robot and each environment, RRT–Path has been run 50 times. The influence of α is evaluated using the percentage of solved start/goal pairs.

In all tested cases, the best results are obtained with $\alpha = 0.01$ or $\alpha = 0.02$. Higher adaptation rate ($\alpha = 0.05$ or even $\alpha = 0.1$) can however decrease the performance, because the sampling radius along the guiding path is increased too fast and the guiding path is not sampled as densely as necessary. Using even higher values turns off the guiding process because the radius R'_{vg} is increased too fast and it practically covers the whole configuration space, so the random samples are generated in the same manner as in RRT rather than around the guiding path. The same observation has been made also with other robots and maps, therefore the results are not shown here. An example of influence of the adaptation rate α is shown in Fig. 4.20.

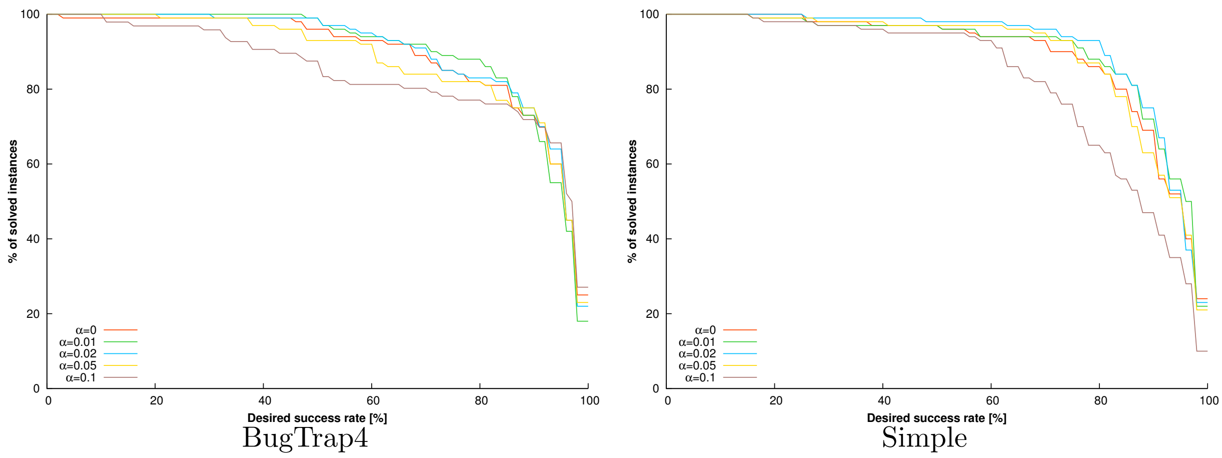


Figure 4.20: The influence of learning rate α to the performance of RRT–Path with the Differential drive robots.

4.5 Discussion

The RRT–Path method samples the configuration space using a predefined guiding path. The guiding path is an input to the algorithm, therefore it should be computed as fast as possible. In the case of motion planning for mobile robots with two or three degrees of freedom, the guiding path can be computed as a geometric path in the workspace using methods like Visibility graph and Voronoi graph.

The comparison of several methods for computing the guiding paths has shown, that the most successful guiding process is achieved with guiding paths computed by Voronoi diagram, while paths constructed using Delaunay triangulation are not suitable for attracting the tree as they are too close to obstacles. Other types of guiding paths, such as those found in PRM roadmap or TMesh2, are equally suitable for the guiding process.

The performance of RRT–Path has been experimentally verified and compared to the state-of-the-art methods RRT, RRT–Blossom and RRT–ADD. The comparison is based on the success rate of the planners to provide plans for multiple random start/goal pairs. The experiments have shown superior performance of RRT–Path in the task of motion planning for the non-holonomic mobile robots like Differential drive and Car-like. In these scenarios, RRT–Path solves more start/goal instances than other methods. The motion planning for these robots can be successfully solved also using RRT–Blossom method, which also provides solutions for most of the start/goal pairs. In comparison to RRT–Path, RRT–Blossom builds larger configuration trees.

Different results are observed in path planning for the 2D robots, where RRT–ADD is superior. The superior performance of RRT–ADD is not surprising, as it was designed primarily for path planning of 2D and 3D robots. Contrary, performance of RRT–ADD is significantly worse in the case of path planning for mobile robots.

The performance of the tested planners has been measured as the percentage of start/goal pairs that are visited with a given probability rather than using runtimes or memory consumption.

The presented multiple start/goal benchmark is designed to measure the overall performance of the planner in the given environments. Although the best results for the mobile robots are achieved with the novel RRT–Path, the method may suffer in certain configurations or environments. Before a motion planner is selected for a given task, it is always necessary to define conditions under which the planner will be used. This includes the specification of the start/goal pairs or regions. Performance of considered methods can be then measured only on these regions in order to select the best planner for the given task.

Similarly to other motion planners, RRT–Path has several pros and cons. The main advantage is the fast growth of the configuration tree towards the goal region. In comparison to other RRT-based planners, RRT–Path needs less number of iterations to reach the goal and consequently, its runtime is shorter. Moreover, the method constructs smaller configuration trees, which decreases memory requirements and increases speed of the nearest-neighbor search.

A disadvantage of RRT–Path is that its performance is directly influenced by the quality of the guiding path. As was shown in section 4.2, the configuration tree cannot follow paths that are too close to obstacles. Therefore methods like Visibility graphs are not suitable to generate the paths. Best results are achieved with the guiding paths constructed by Voronoi Diagram or using triangular representation of the workspace.

Chapter 5

Guided sampling for 3D objects

The guided sampling strategy proposed in the previous chapter utilizes a path in the configuration space that attracts the tree and guides its growth towards the goal configuration. In motion planning for mobile robots, the guiding path can be found using basic path planning methods (e.g. VD, VG, or decomposition-based approaches) as a geometric path in the workspace and it can be extended to the configuration space by setting the rotation dimension to zero.

The utilization of guiding paths found in the workspace may however become insufficient if the configuration space has significantly more dimensions than the workspace. This is studied in the generalized path planning problem which has applications e.g. in path planning for steerable needles in surgery [274, 6, 182], in assembly/disassembly studies [248, 36, 185, 227, 5] or in computational biology [230, 7, 170, 55, 199, 228]. In these applications, a collision-free path for a 3D solid object has to be found. This leads to search in six-dimensional configuration spaces. The generalized path planning problem is very challenging and it was proven to be PSPACE-hard [206].

Due to the high number of dimensions of the configuration space, it is not possible to discretize the space fully, e.g. into a grid, because the number of cells grows exponentially with the number of the dimensions. Due to high dimensionality of the configuration space, it is not easy to estimate locations of the narrow passages from the description of the workspace, which does not allow us to focus sampling to these difficult regions. In this chapter, we will investigate how to utilize the guiding principle for the purpose of path planning of 3D rigid objects in 3D environments.

5.1 Problem analysis

The configuration space of 3D objects (robots) has six dimensions, since each configuration $q = (x, y, z, \varphi, \theta, \psi) \in \mathcal{C}$ describes 3D position and also 3D rotation of the robot. To utilize RRT–Path for the generalized path planning problem in this six-dimensional configuration space, a guiding path is required. This guiding path cannot be computed in the workspace considering a point robot, because it would not provide information about rotation.

Let us consider a generalized path planning problem depicted in Fig. 5.1a, where the task is to remove the hedgehog from the cage. Due to many spikes, the hedgehog needs to be carefully rotated, while its 3D position changes slightly. An example of feasible solution is depicted in Fig. 5.2, where the phases with negligible translations but significant rotations are highlighted. Usage of a 3D guiding path computed in the workspace (Fig. 5.1b) is not helpful for RRT–Path, as it only suggests how to sample x, y and z dimensions, but the remaining three dimensions φ, θ and ψ need to be sampled randomly using the uniform distribution. As the solution of the hedgehog benchmark depends mainly on the rotation, utilization of a 3D guiding path is not suitable.

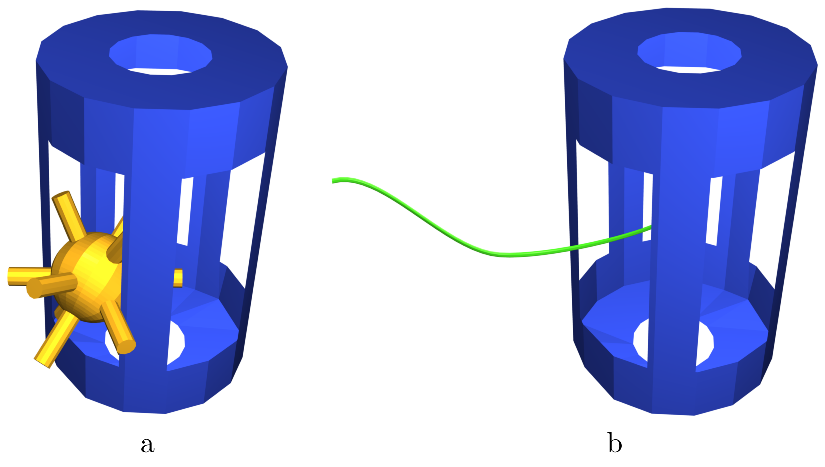


Figure 5.1: Hedgehog in the cage problem (a), where the task is to remove the spiky object from the cage. An example of 3D guiding path (b).

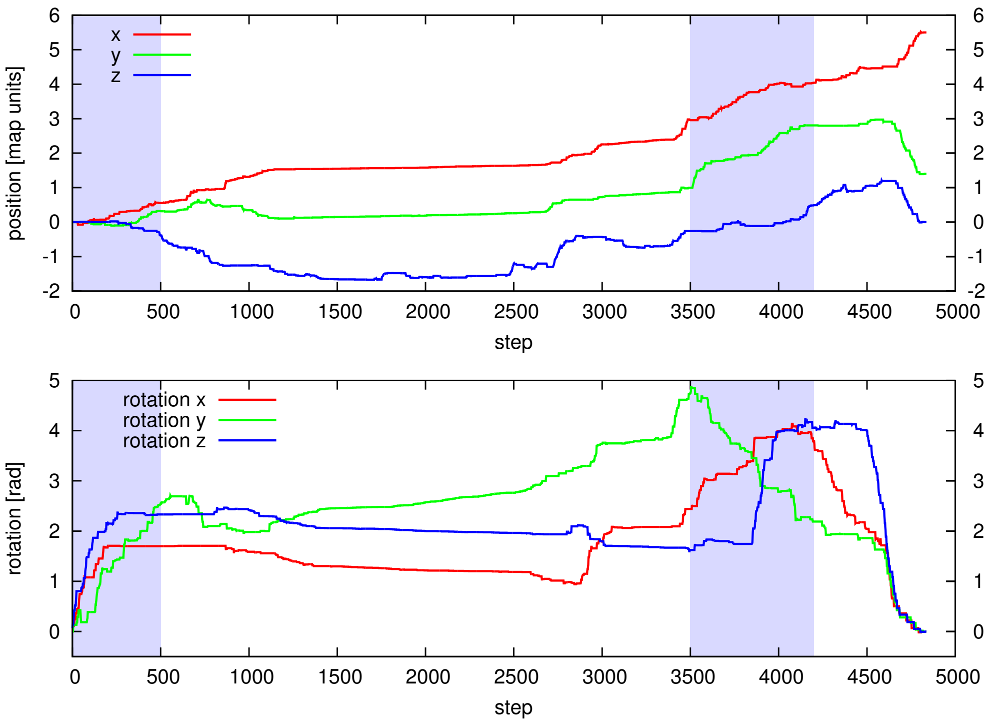


Figure 5.2: Visualization of a solution of the Hedgehog problem. The light-blue color highlights two situations where the 3D position (top) of the Hedgehog does not change significantly comparing to its rotation (bottom).

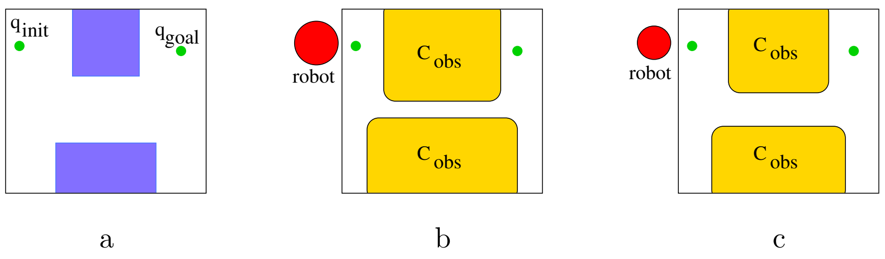


Figure 5.3: An example of a simple 2D workspace with two obstacles. The corresponding configuration space for a circular red robot contains a narrow passage (b). The width of the passage is increased if the radius of the robot is scaled-down (c).

5.1.1 Motivation for iterative scaling approach

To solve the challenging path planning problem in the six-dimensional configuration space, the RRT–Path method introduced in the Chapter 3 can be used. A suitable guiding path computed right in the configuration space is required to efficiently guide the configuration tree. This however leads to the generalized path planning problem. Computing the guiding path by solving the generalized path planning problem would be time consuming, because the problem is PSPACE-hard [206]. Moreover, it does not make sense to use a 6D path as a guiding path for RRT–Path, because the 6D guiding path itself would be the solution of the generalized path planning problem. A possible way to compute a 6D guiding path in a reasonable time is to relax the feasibility constraints in order to increase the relative volume of the configuration space.

The feasibility constraints in the path planning task can be relaxed by scaling down the geometry of the robot. Scaling down the geometry of the robot widens the narrow passages, because a smaller robot has more space to move amongst the obstacles (Fig. 5.3). Thanks to increased volume of the narrow passages, path planning for the relaxed problem (with a scaled-down robot) is easier than solving the original problem. After a path is found for the relaxed problem, it can be used as a guiding path for the original robot [253]. For the sake of simplicity, we use term *scaled robot* instead of *scaled geometry of the robot* in the following text.

A possible disadvantage of this approach is that the relaxation of some constraints can change connectivity of the configuration space. Therefore, a path computed for a scaled-down robot might not be executable by the original robot. This can happen e.g. if the robot is scaled-down too much and the guiding path is found through the regions of the configuration space that are not followable by the original robot. For example, the guiding path found for a circle robot (Fig. 5.4a) passes the narrow passage in the configuration space (Fig. 5.4b). The narrow passage however becomes obstacle for a large robot (Fig. 5.4c).

To cope with this issue, we propose to iteratively improve the quality of the solution obtained for a relaxed version of the problem. To achieve this, we propose to scale-up the geometry by small steps. As the scale is improved by the small steps, the connectivity of the configuration space should not change significantly and the previous solution can be used to guide the search for the larger robot. An example of iterative improvements of solution is depicted in Fig. 5.5.

5.2 RRT–IS: Rapidly Exploring Random Trees with Iterative Scaling

The main idea of RRT–IS (RRT with Iterative Scaling) [253] is to find a feasible path in the configuration space for a small (scaled-down) robot and to use this path as a guiding path for a larger robot (Fig. 5.6). A discrete set S of possible scales of the geometry of the robot is considered in RRT–IS, $S = \{s_{\min}, s_{\min} + s_\Delta, s_{\min} + 2s_\Delta, \dots, s_{\max}\}$, where $s_{\min}$ and $s_{\max}$ are

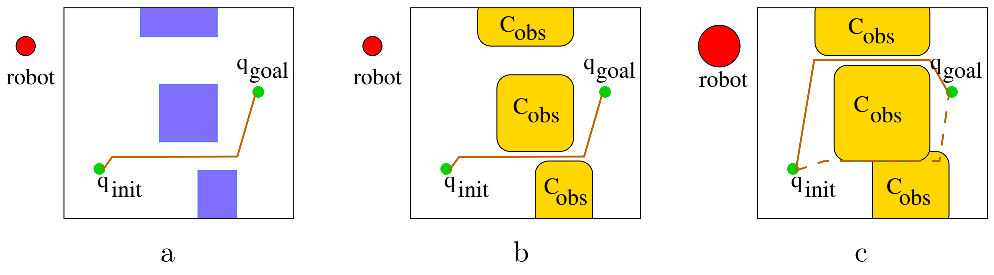


Figure 5.4: Connectivity of the configuration space depends on the size of the robot. Path in the workspace (a) and corresponding path in the configuration space (b). This path is not followable by the larger robot (dashed), because of the changed connectivity of the configuration space (c).

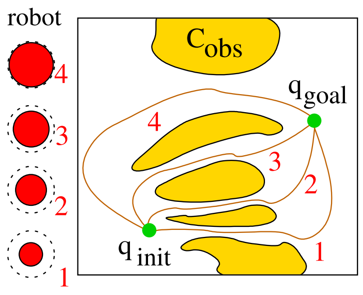


Figure 5.5: Motivation for iterative scaling approach. The robot is scaled down to a small scale 1, for which a path is found. Then, the size of the robot is increased to a scale 2, for which a new solution is found in the vicinity of path 1. The solution is close to the previous solution. This process repeats until a solution for the original robot (denoted by number 4) is found. The iterative improvement of the solution 1 to solution 2, etc. is necessary, as the path 1 traverses a narrow passage that is not feasible for the full robot. Moreover, the narrow passage of path 1 is distant from the feasible space of the full robot. Without the iterative improvement, it is not possible to sample vicinity of the path 1 in order to find a solution for the full robot.

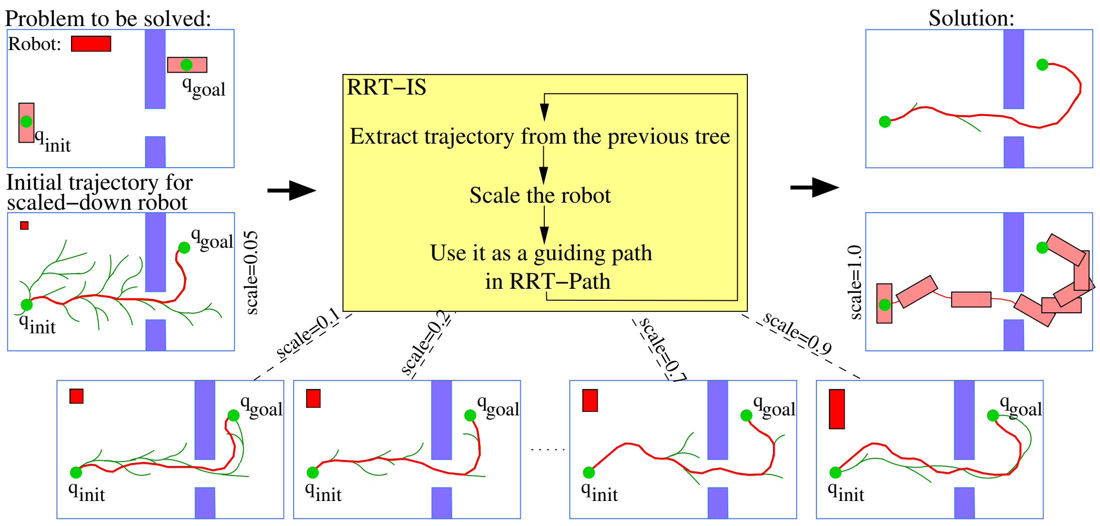


Figure 5.6: Schema of the RRT–IS algorithm.

minimal and maximal allowed scales respectively and s_Δ is the smallest considered difference between two scales. The geometry of the robot corresponding to a scale $s \in S$ at a configuration q is denoted $B_s(q) \subset \mathcal{W}$. A robot with original geometry is considered to has scale $s = 1$, while a scaled-down robot has scale $s < 1$. The maximal scale corresponds to the original size of the robot, therefore $s_{\max} = 1$. A configuration q is collision-free at scale s if there is no collision between $B_s(q)$ and the obstacles.

The guiding path $P = (q_1, \dots, q_n)$ is a path in the configuration space starting from the initial configuration $q_1 = q_{start} \in \mathcal{C}_{free}$ and leading to a configuration $q_n = q'_{goal} \in \mathcal{C}_{free}$, that is located in the vicinity of the goal configuration q_{goal}, i.e., $\varrho(q'_{goal}, q_{goal}) \le d_{goal}$.

For each point $q_i \in P$ and a given actual scale $\bar{s}$, we can identify the largest scale $\hat{s}_i \in S, \hat{s}_i \ge \bar{s}$ such that the configuration q_i is collision-free. A point $q_c \in P, c = 1, \dots, n$, with the smallest collision-free scale $\hat{s}_c$ is called the critical point. The procedure to determine the critical point is listed in Alg. 5.

Algorithm 5: findCriticalPoint(P,$\bar{s}$)

```
Input: guiding path $P = (q_1, q_2, \dots, q_n)$, actual scale $\bar{s}$, minimal scale $s_{\min}$, maximal scale $s_{\max}$
       and set of other scales $S$
Output: critical point $q_c \in P$ and its smallest collision-free scale $\hat{s}_c$
1  $\hat{s}_c = s_{\max}$;
2  $c = 0$;
3  for $i = 1 : n$ do
4      $\hat{s} = s_{\min}$;
5      for $s \in S$ do // find maximal collision-free scale $\hat{s}$ of point $q_i$
6          if ($s \ge \bar{s}$) and ($R_s(q_i)$ is collision-free) and ($s > \hat{s}$) then
7              $\hat{s} = s$;
8          end
9      end
10     if $\hat{s} < \hat{s}_c$ then
11         $\hat{s}_c = \hat{s}$; // collision-free scale of the critical point
12         $c = i$; // index of critical point
13     end
14 end
15 return $(q_c, \hat{s}_c)$;
```

The RRT–IS algorithm is listed in Alg. 6 and it works as follows. At the beginning, the robot is scaled down to scale $\bar{s} = s_{\min}$. In each iteration, first the critical point q_c with critical scale $\hat{s}_c$ is found and its predecessor q'_c on the path is determined. The point q'_c is set to the first point on the guiding path if the critical point is also the first point on the path, otherwise q'_c is the point before the critical point. The RRT–Path planner is called to find a path between q'_c and goal configuration q_{goal} for a robot with scale $s_c + s_\Delta$. If a path is found (i.e., it approaches q_{goal} to the distance d_{goal}), the guiding path is updated by replacing points $p_i \in P, i = c, \dots, n$ with the found path. This procedure repeats until the smallest collision-free scale $\hat{s}_c$ of the critical point is equal to $s_{\max}$ or the allowed number of attempts B is reached. The result of RRT–IS is the guiding path computed for the maximum scale $s_{\max}$. This guiding path also represents a solution of the original problem. The guiding path is represented as a sequence of waypoints with resolution ε, because the path is provided by RRT–Path that utilizes the straight-line planner with resolution ε.

RRT–IS listed in Alg. 6 assumes that an initial guiding path is provided. This can be computed using RRT–Path considering the small scale $s_{\min}$ of the robot and using $p_{goal} = 0$ in order to allow sampling from the whole configuration space.

Algorithm 6: RRT–IS

```
Input: Initial state $q_{start}$, goal state $q_{goal}$, minimal scale $s_{\min}$, maximal scale $s_{\max}$, scale
       increment $s_{\Delta}$, number of allowed unsuccessful iterations $B$, guiding path $P = (q_1, \dots, q_n)$
       computed for the scale $s_{\min}$
Output: path $P$
1  $\bar{s} = s_{\min}$;
2  $b = 0$; // counter for unsuccessful iterations
3  while $b < B$ do
4      $(q_c, \hat{s}_c)$ = findCriticalPoint($P$, $\bar{s}$);
5      $q'_{start}$ = predecessor of $q_c$ on the guiding path;
6      robot.scale($\hat{s}_c + s_{\Delta}$); // change scale of the robot
7      path = rrtPath($q'_{start}, q_{goal}, P$); // RRT–Path is listed in Alg. 3
8      $d$ = distance(path.end,$q_{goal}$);
9      if $d < d_{goal}$ then
10         replace points on $P$ from $c$ to $n$ by the found path;
11         $\bar{s} = \hat{s}_c + s_{\Delta}$;
12         $b = 0$;
13     else
14         $b = b + 1$;
15     end
16     if $\bar{s} \geq s_{\max}$ then
17         return $P$; // path $P$ contains solution for scale $s_{\max}$
18     end
19 end
20 return failure; // max. number of unsuccessful iterations $B$ reached
```

5.2.1 Analysis of the iterative scaling

The main task of RRT–IS is to maintain the guiding path. The algorithm terminates if the scale $\hat{s}_c$ of its critical point is equal to $s_{\max}$. This means that all points on the guiding path are collision-free at the scale at least $s_{\max}$. As $s_{\max}$ represents the original size of the robot (or the size for which a solution is required), a guiding path with $\hat{s}_c = s_{\max}$ represents the solution of the original problem.

In each iteration of RRT–IS, the guiding path is collision-free at a scale $\hat{s}_c$ that is determined by the bottleneck of the path — the critical point q_c. After the critical point is determined, RRT–Path is called to find a new solution in order to improve the guiding path. It is necessary to ensure that the new path is collision-free at a higher scale and therefore the RRT–Path algorithm is run with the robot of scale $\hat{s}_c + s_{\Delta}$. The path planning process cannot be initialized from the critical point q_c, because it is not collision-free already at the scale $\hat{s}_c + s_{\Delta}$, but only on the scale $\hat{s}_c$. Therefore, RRT–Path is initialized from the predecessor q'_c of the critical point q_c. The point q'_c is collision-free at the scale $\hat{s}_c + s_{\Delta}$, otherwise it would be selected as the critical point. If a new path from q'_c to q_{goal} is found, it is collision-free at scale $\hat{s}_c + s_{\Delta}$ and therefore, it can be used to replace the part of the old guiding path starting from the critical point q_c. This ensures iterative improvement of the guiding path, i.e., increasing the scale $\hat{s}_c$ of its critical point. Due to randomization, RRT–Path may fail to find a solution, therefore it is called at most B times.

The iterative improvements of the guiding path is depicted in Fig. 5.7. A guiding path with critical point q_c with scale 0.6 is depicted in Fig. 5.7a. The RRT–Path is constructed for a larger robot (with scale 0.7) from the predecessor of q_c (Fig. 5.7b). The resulting path then replaces the old guiding path starting from q'_c (Fig. 5.7c).

In RRT–IS it is assumed that if a configuration q is collision-free at the scale s it is collision-free also for smaller scales. Validity of this assumption depends both on the shape of the objects as well as on the utilized scaling method. It has to be ensured, that a scaled-down geometry is always contained in the geometry of the original robot. Besides the simple multiplication of

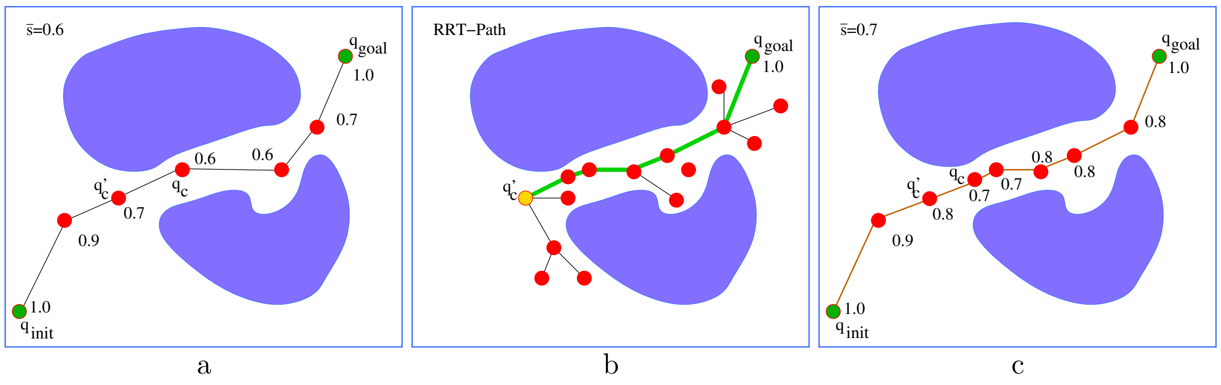


Figure 5.7: Example of path merging in RRT–IS. A guiding path that is collision-free at scale 0.6 (a). A new configuration tree is built by RRT–Path from the configuration q_c' that is predecessor of the critical point q_c (b). The result of the planning is the green path in the tree. This path replaces the old guiding path starting from q_c' (c). This improves the original path, because the critical point of the new path is collision free at scale 0.7.

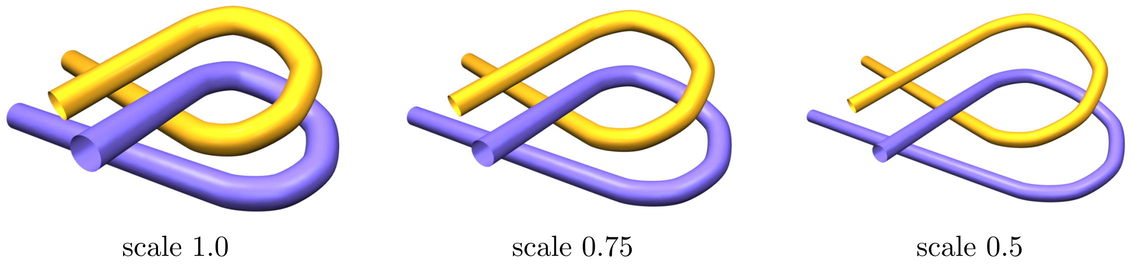


Figure 5.8: Example of scaling-down objects by thinning them around the medial axis. The objects are part of Alpha puzzle benchmark [141].

coordinates, thinning and shrinking of geometries is possible. The idea of object thinning is to shift the coordinates towards the medial axis of the object [214]. A method for object shrinking was presented in [94]. Example of object thinning suitable for Alpha puzzle is depicted in Fig. 5.8.

The number of iterations needed to find a feasible path for the robot of scale $s_{\max}$ is influenced by the minimal scale $s_{\min}$ and the smallest differences between scales s_Δ. Too small $s_{\min}$ increases the number of iterations, and it can also direct the guiding paths to such parts of the configuration space that are not feasible for higher scales. An example is shown in Fig. 5.4. If RRT–IS is initialized with a too small robot, it can find a solution through the bottom passage (Fig. 5.4b), but such a solution is not feasible for a larger robot (Fig. 5.4c) that needs to be manipulated through the top passage.

In each iteration of RRT–IS, the RRT–Path planner is called to construct a tree using at most I_{max} iterations, which can result in a configuration tree with at most I_{max} nodes. A path extracted from this tree is then used as a guiding path in the next iteration. The size of a configuration tree that is hold in memory at one time can therefore be at most I_{max} nodes. However, the total number of nodes explored during the whole run of RRT–IS can be larger. The theoretical maximum is $\frac{s_{\max}-s_{\min}}{s_\Delta}BI_{max}$, as RRT–Path is called at most B times for each scale of the robot and the number of scales is given by s_Δ.

5.3 Discussion

Motion planning for 3D solid objects is a challenging problem leading to search of six-dimensional configuration space. The high dimensionality of the problem prevents us to discretize the configuration space to a grid-based representation in order to employ state-space search methods to find a solution. A path can be found using sampling-based methods that build an approximation of the configuration space. The methods can however suffer from the existence of the narrow passages.

A novel sampling schema for RRT in the task of path planning of 3D solid objects has been proposed in this chapter to cope with the narrow passage problem. The method is based on RRT–Path and a process of iterative improvement of the guiding path, where the feasibility constraints are relaxed by scaling-down the robot geometry. At the beginning, a path is computed for a small robot and the obtained solution is used as a guiding path for a larger robot. The process is repeated until a solution of the original problem is found.

The most relevant work to RRT–IS is the IRC strategy [12], that defines general rules for solving challenging planning problems: first, some feasibility constraints are relaxed and solved by a sampling-based planner. The solution is then used to guide search for the original problem. The core of the IRC strategy is the procedure of relaxing the feasibility constraints, which is specific for each scenario. For example, motion planning for articulated robots can be simplified by fixing some of the links, which decreases the number of actuators that need to be considered by the planner.

For the purpose of path planning of 3D objects, the authors in [12] suggest to either scale-down the robots or to allow a little penetration between the robots and the obstacles. The latter approach requires to compute penetration depth between two geometries, which is computationally more demanding than a pure collision detection. The IRC method has been designed for PRM-based planners. For example, authors simply increase probability of sampling around the previous solution, which is realized by the OBPRM method [8].

Another PRM-based planner solving the original problem by scaling-down the robot is presented in [94]. The robot geometry is shrunken using a novel approach. The method is suitable also for multi-linkage robots like robotic manipulators. The planner utilizes PRM-based SBL planner [217] to find the solution of the relaxed problem. To repair an existing solution, neighborhood of the existing solution is sampled. If no new milestone is found within a given amount of iterations, the neighborhood is increased. This repeats until a solution for the original robot is found or a given maximum number of iterations is reached. To improve the approximate (relaxed) solution, the nodes on the path are validated using a larger robot and the unfeasible ones are removed. The removed segments are replaced by sampling around the invalid configurations. The technique is similar to Lazy PRM [17] and Fuzzy PRM [185] techniques.

The proposed RRT–IS planner and IRC strategy [12] share the same general idea: a solution for a relaxed problem, obtained using a scaled-down robot, can be used to find solution for the original problem. The main difference between RRT–IS and the IRC strategy [12] is that IRC is designed primarily for the PRM planner, while RRT–IS is based on the RRT planner. The IRC strategy simply assumes that it is enough to increase probability of sampling around a solution constructed for a relaxed version of a problem in order to find solution of the original problem. This strategy cannot be applied in RRT-based planners, where increase of probability of sampling in a given area does not ensure growth of the tree into the area.

Chapter 6

Experimental verification of RRT–IS

6.1 Algorithm setup

The performance of RRT–IS has been verified in the task of path planning for 3D rigid objects in 3D workspace. RRT–IS is compared to RRT [150], RRT–Retraction [286] and RRT–ADD [108]. The latter two methods have been selected as they represent two state-of-the-art modifications of RRT for path planning of 3D objects. Due to lack of implementation details of RRT–Retraction, the modification described in Section 2.4.2 is used. RRT–Retraction is run with $I_{ret} = 10$ retraction steps, the contact configurations are search amongst $N_{ret} = 32$ randomly placed points around the actual configuration in radius $r_{ret} = 0.5$ mu. RRT–ADD is run with parameters $\alpha_{dd} = 0.05$ and activation radius $R_{dd} = 0.5$ mu. Parameters of both methods have been obtained experimentally to achieve the best performance in the tested environments.

The performance of RRT–IS is given by the scaling parameters $s_{\min}, s_{\max}$ and s_{Δ} as well as by the parameters of RRT–Path, which is called by RRT–IS at most B times for each tested scale. In all scenarios, the parameters are set to $s_{\min} = 0.6$, $s_{\max} = 1.0$, $s_{\Delta} = 0.05$ and $B = 10$. RRT–Path is called with $R_{vg} = 0.5$ mu (the same radius as the action radius of RRT–ADD), adaptation rate $\alpha = 0.01$, and $p_{goal} = 0.6$.

The robots and the maps are represented using 3D triangle meshes and collision detection is solved using Rapid [82] library. Resolution of the straight-line planner is $\varepsilon = 0.2$ mu. Collisions are computed three times per edge. The distance ϱ between two configurations is computed using 6D Euclidean metric. Nearest-neighbor queries are solved using MPNN [280] library. The planners are implemented in C++ and compiled using gcc compiler version 4.2.1 with O2 flag. The experiments have been performed on PC Intel Pentium IV @ 2.2 GHz.

6.2 Cube scenario

In the first experiment, the task is to find a path for a cube robot of side size 1 mu in environments with one narrow passage. The tested environments consist of two rooms separated by a wall with a window of size 2 mu (Fig. 6.1). The size of the rooms is $25 \times 25 \times 6$ mu. The window represents a narrow passage, that has to be traversed in order to connect the rooms. Performance of the methods is evaluated using multiple start/goal pairs. The start/goal pairs are generated randomly but it is ensured that they are located in different rooms in order to force the planners to traverse the narrow passage. For each environment, 100 random start/goals is used and each planner is run 50 times on each pair. Examples of generated start/goal pairs are depicted in Fig. 6.2. The total number of trials per planner is $100 \cdot 50 = 5,000$. The maximum number of planning iterations is set to $I_{max} = 50,000$ for the Window and Tunnel environments, and $I_{max} = 100,000$ for the Long-tunnel environment.

The results achieved in the Window scenario are depicted in Fig. 6.3, results for the Tunnel map are depicted in Fig. 6.4 and results for Long-tunnel are shown in Fig. 6.5. The figures show

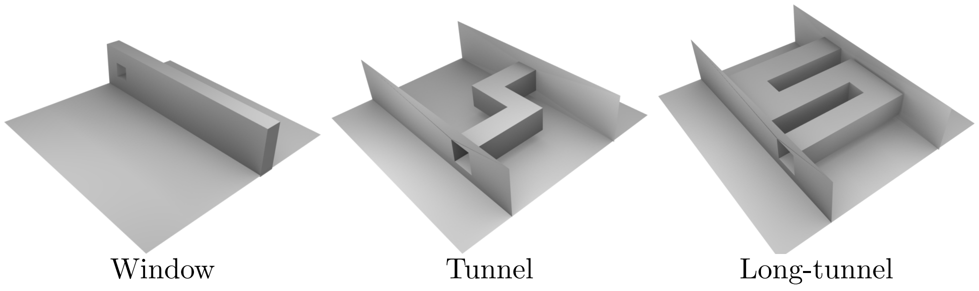


Figure 6.1: Scenarios for the generalized path planning problem.

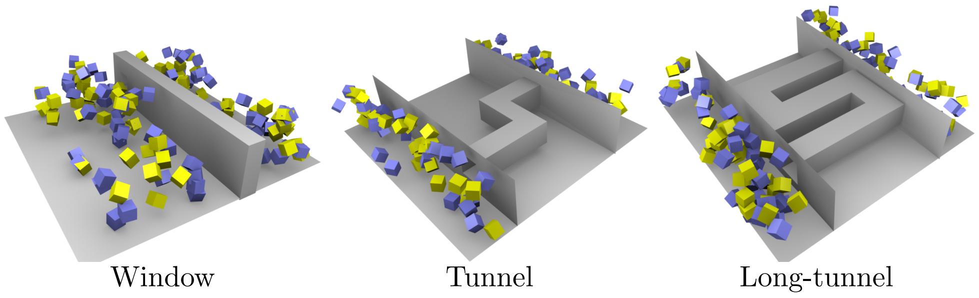


Figure 6.2: Examples of random start/goal pairs.

boxplots of runtimes and size of constructed trees and a graph with progress of percentage of solved start/goal instances. Each figure also shows percentage of start/goal pairs solved with at least 80 % $s-$rate.

Methods RRT–Retraction and RRT–IS compute solution for the highest amount of start/goal pairs. The boxplots of runtime show that RRT–IS is comparable to RRT–Retraction in the Window environment, but it is significantly slow in the case of Tunnel and Long-tunnel environments. In the Tunnel environment, both RRT–Retraction and RRT–IS solve the highest percentage of start/goal pairs, but RRT–IS requires significantly more time to find a solution.

The Long-tunnel environment is more challenging for the planners, therefore the number of maximal planning iterations is set to $I_{max} = 100,000$. Both RRT–Retraction and RRT–IS provide plans for the highest percentage of start/goal pairs, but from the runtime point of view, the RRT–IS method is significantly slower.

Although RRT provides good results in the Window and Tunnel scenarios, it fails in the Long-tunnel scenario. The boxplots of size of constructed trees show, that RRT does not reach the maximum allowed tree size (which equals to the number of planning iterations, i.e., 100,000). This indicates, that the algorithm does not expand the tree during each iteration. The inability to expand the tree is caused by frequent selection of boundary nodes for the expansion. The boundary nodes cannot be expanded because these are located too close to the obstacles. The constructed trees usually fill the whole room, where the start configuration is located, but due to the small entrance to the narrow passage, they do not expand to the second room. The situation in Long-tunnel is also complicated due to the zig-zag narrow passage, that requires more manipulation with the object than the narrow passages in the Window or Tunnel environments.

Performance of RRT–ADD method is similar to RRT in the Long-tunnel scenario. A possible reason is that the activation radius R_{dd} of RRT–ADD is too small, which does not allow the tree to be expanded properly. RRT–ADD solved almost all start/goal pairs in the Window and Tunnel scenarios with the same radius R_{dd}, but a different radius needs to be used for the Long-tunnel environment. Although all scenarios use the same robots and the narrow passages of same widths, the results indicated that the activation radius R_{dd} should be set also considering the shape of the obstacles and the narrow passage. This shows the possible disadvantage of RRT–ADD, which is sensitive to the proper setting of R_{dd}.

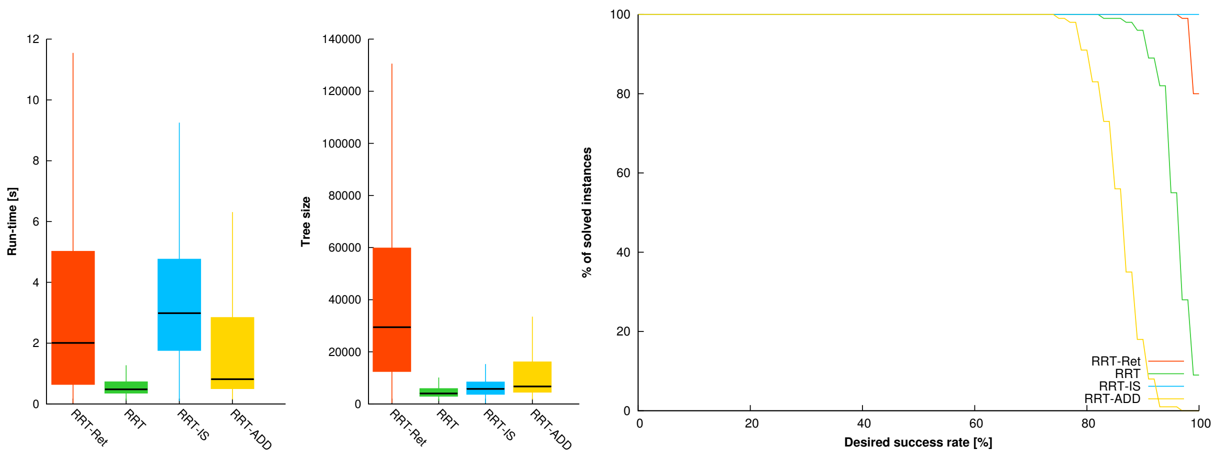


Figure 6.3: Performance of path planning for cube robot in the Window environment. Percentage at 80 % $s-$rate: **RRT=100 %**, **RRT-Ret=100 %**, **RRT-IS=100 %**, RRT-ADD=91 %.

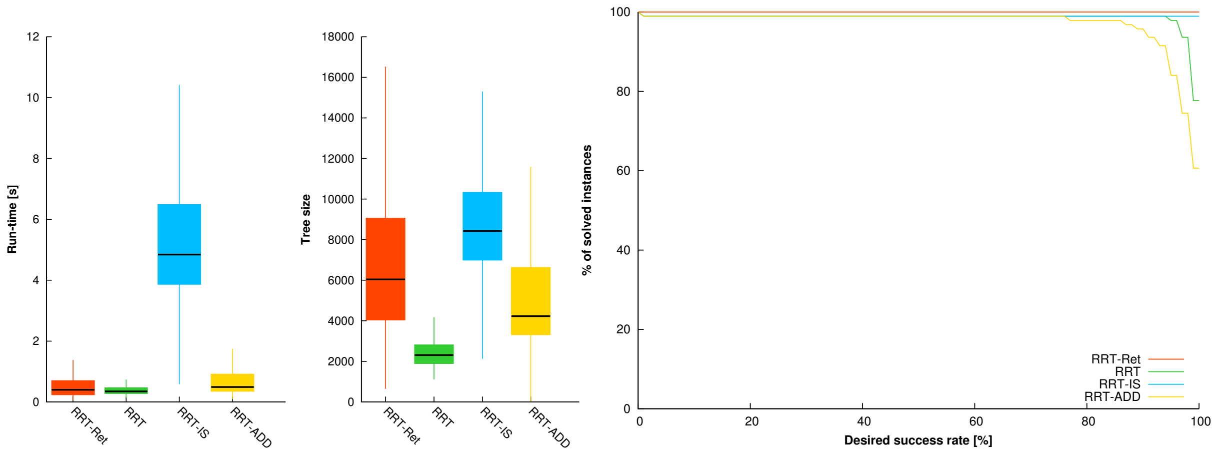


Figure 6.4: Performance of path planning for cube robot in the Tunnel environment. Percentage at 80 % $s-$rate: RRT=99 %, **RRT-Ret=100 %**, RRT-IS=99 %, RRT-ADD=98 %.

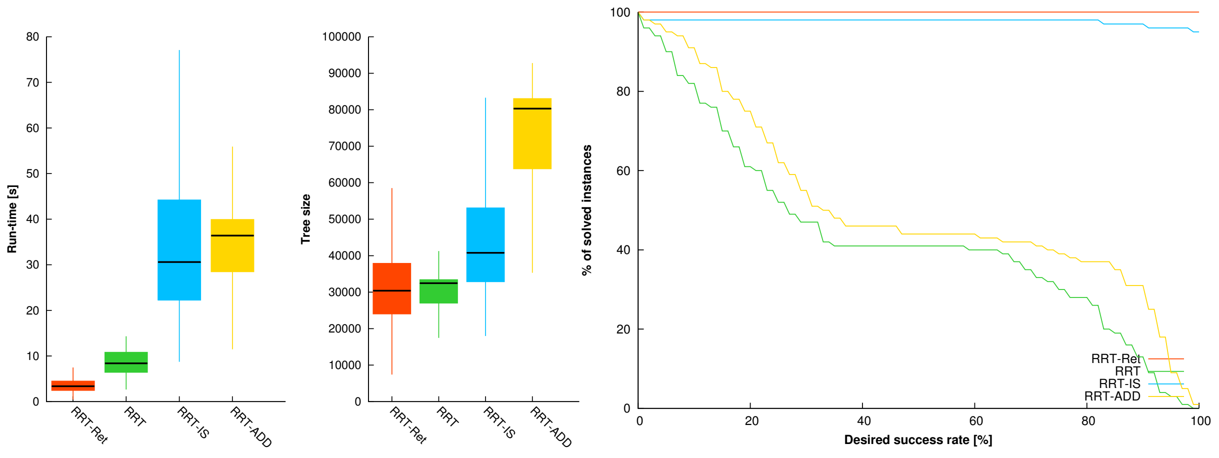


Figure 6.5: Performance of path planning for cube robot in the Long-tunnel environment. Percentage at 80 % $s-$rate: RRT=28 %, **RRT-Ret=100 %**, RRT-IS=98 %, RRT-ADD=37 %.

6.3 Stick scenario

The aim of the second experiment is to test the performance of the planners for path planning of non-cubic, yet simple, robots. Three robots are used: Stick, Stick$_c$, and L-shape robot (Fig. 6.6). The difference between the stick robots is the origin of their coordinate system: Stick has the origin at one end of the stick and Stick$_c$ has the origin in the geometric center. The experiments have been performed in the Window environment with 50 random start/goal pairs. The start and goal configurations are placed into different rooms in order to test the ability of the planners to find paths through the narrow passage. The examples of start/goal pairs are shown in Fig. 6.7. The maximum number of planning iterations is set to $I_{max} = 50,000$ for all algorithms and robots except $I_{max} = 150,000$ for RRT–ADD and the L-shape scenario. For each start/goal pair, the planners have been run 50 times leading to $50 \cdot 50 = 2,500$ trials.

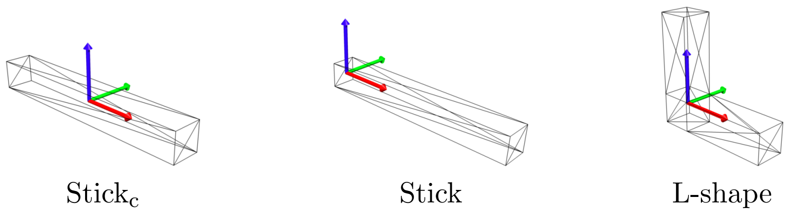


Figure 6.6: The 3D objects used in the second experimental setup. The cross-section of each object is 0.6 map units.

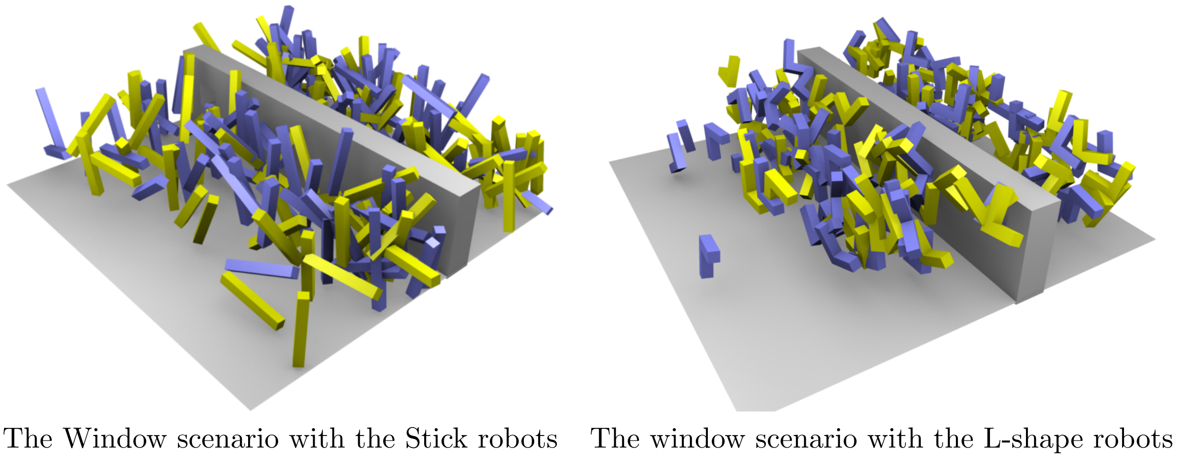

The Window scenario with the Stick robots
The window scenario with the L-shape robots

Figure 6.7: Examples of random start/goal configurations of Stick and L-shape robots.

The results are depicted in Fig. 6.9 for the Stick$_c$ robot, in Fig. 6.8 for the Stick robot and in Fig. 6.10 for the L-shape robot. In all scenarios, the best results in the term of percentage of start/goal pairs are obtained by the RRT–IS method. The experiment have shown unexpected sensitivity of the planners to the geometric center of the Stick robots. The percentage of solved start/goal pairs is significantly higher for the Stick robot (Fig. 6.8) than for the Stick$_c$ robot (Fig. 6.9). RRT–Retraction solves high percentage of start/goal pairs only for the Stick robot, but it fails to solve the same planning instances with the Stick$_c$ robot.

RRT–IS shows superior performance also for the L-shape robot (Fig. 6.10), where other planners are not able to solve most of the instances. Examples of the configuration trees built by RRT–IS for various scales of the L-shape robot are depicted in Fig. 6.11.

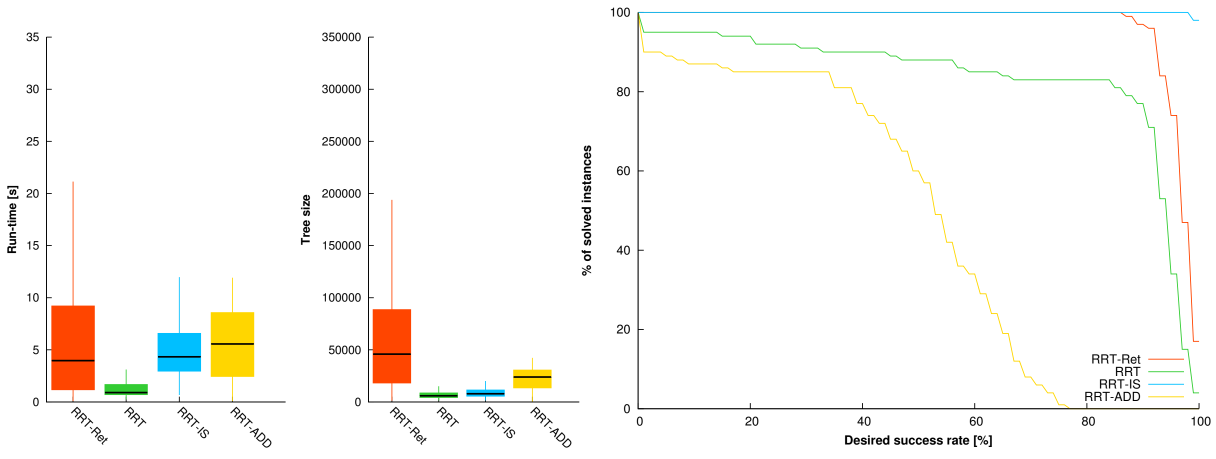


Figure 6.8: Performance of the planners with the Stick robot. Percentage at 80 % $s-$rate: RRT=83 %, **RRT-Ret=100 %**, **RRT-IS=100 %**, RRT-ADD=0 %.

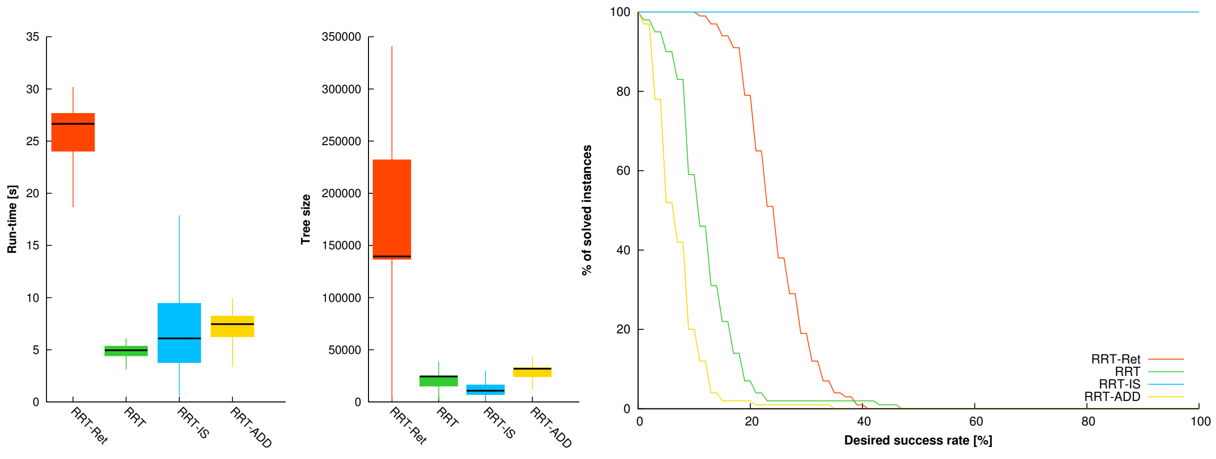


Figure 6.9: Performance of the planners with the Stick$_c$ robot. Percentage at 80 % $s-$rate: RRT=0 %, RRT-Ret=0 %, **RRT-IS=100 %**, RRT-ADD=0 %.

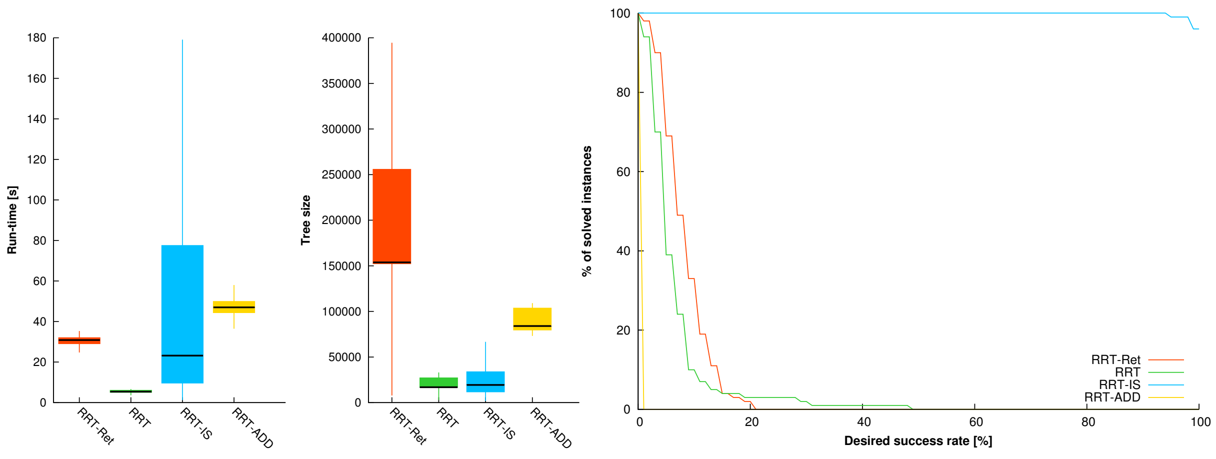


Figure 6.10: Performance of the planners with the L-shape robot. Percentage at 80 % $s-$rate: RRT=0 %, RRT-Ret=0 %, **RRT-IS=100 %**, RRT-ADD=0 %.

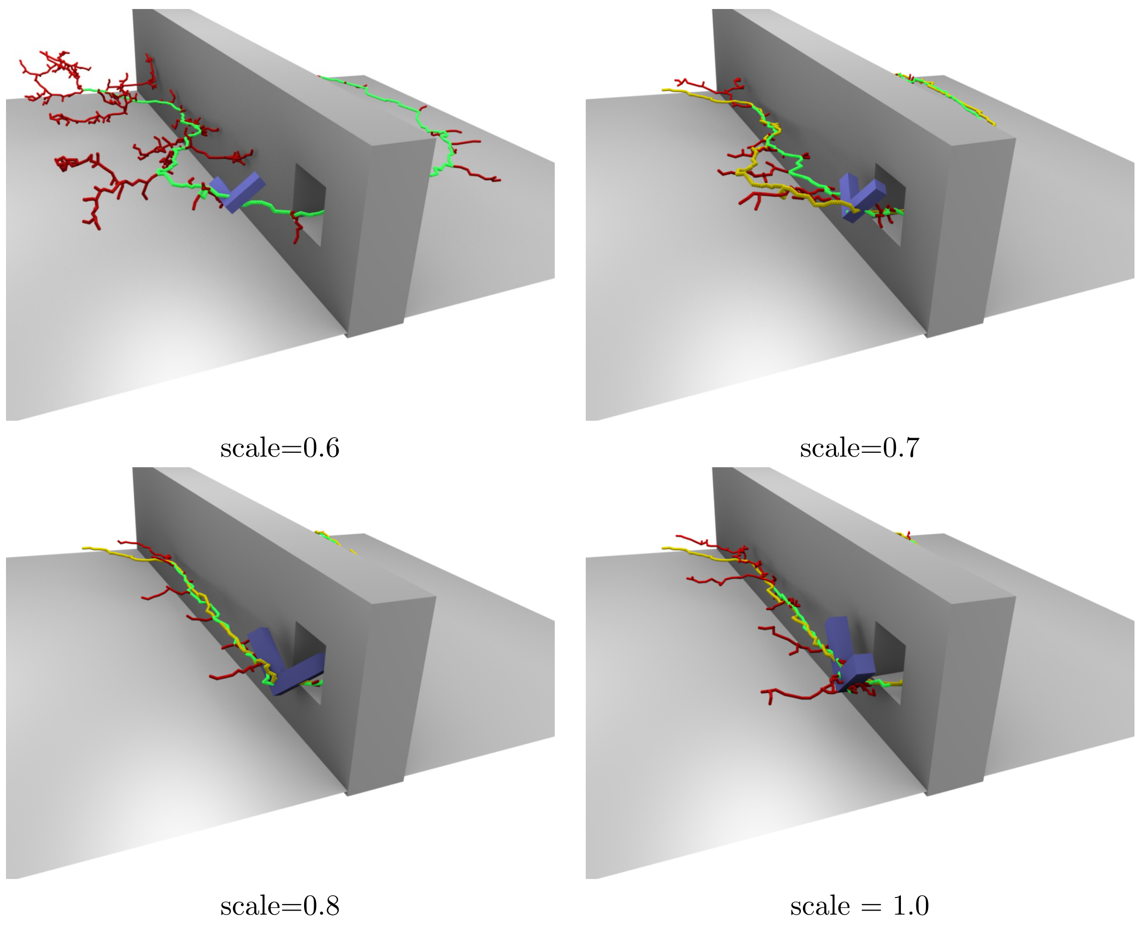


Figure 6.11: Example of the guiding paths (yellow) and the configuration trees (red) constructed by RRT–IS for the L-shape robot. The path found in the configuration tree (green) becomes a guiding path in the next iteration.

6.4 3D BugTrap

The third experiment has been performed on the 3D BugTrap motion planning benchmark [141]. The objective is to remove the stick robot from the trap obstacle through the narrow passage. The benchmark is challenging not only due to the narrow passage, but also due to time consuming collision detection. Collision detection is realized using AABB (Axis-Aligned Bounding Box) hierarchical data structure, but because the robot is located inside the shell, the bounding boxes of both objects always overlap. More levels of the hierarchy need to be traversed before the collision is properly determined, which increases time of the collision detection system [56].

The planners have been run with $I_{max} = 100,000$, $I_{max} = 200,000$ and $I_{max} = 500,000$ of the planning iterations. Parameters of RRT–IS are $s_{\min} = 0.3$, $s_{\Delta} = 0.1$ and $p_{goal} = 0.7$. Small goal-bias $p_{goal} = 0.01$ is used also by other methods. RRT–Retraction is run with $I_{ret} = 10$ retraction steps, contact configurations are searched amongst $N_{ret} = 32$ random samples on a sphere of radius $r_{ret} = 1$ mu. The initial radius of RRT–ADD is set to $R_{dd} = 0.5$ and $\alpha_{dd} = 0.01$. Each algorithm is run 50 times. In this scenario, only one start/goal configuration is used. The start configuration is inside the capsule and the goal configuration is defined as a region outside the capsule.

The results are summarized in Tab. 6.1. Within allowed $I_{max} = 100,000$ of the planning iterations, RRT–IS provides solution in 100 % of trials, RRT and RRT–ADD failed (their $s-$rate is 0 %), and RRT–Retraction solves the problem in 24 % of trials. By increasing the number of allowed iterations I_{max} from 100,000 to 500,000 the performance of RRT–IS remains same (it provides solutions in 100 % of trials), but performance of other planners improves. The success rate of RRT–Retraction increases from 24 % to 100 %, and the success rate of RRT increases from 0 % to 64 %. This is the usual method to increase probability of finding a solution with sampling-based planners, but it does not work always. For example, the success rate of RRT–ADD increases from 0 % to only 18 % after I_{max} is increased from 100,000 to 500,000. The worse performance of RRT–ADD is probably caused by improper setting of the activation radius R_{dd}. The sensitivity of RRT–ADD to the parameter R_{dd} may disadvantage the method in scenarios, where the radius cannot be easily estimated and where its experimental setting would be time consuming.

The size of the constructed trees of RRT–IS does not increase significantly with increasing I_{max} (it is 437×10^3 for $I_{max} = 100,000$ and 477×10^3 for $I_{max} = 500,000$). RRT–IS can build larger trees than is the number of allowed planning iterations, as it calls RRT–Path upto B times for each scale. The increasing number of planning iterations I_{max} leads to the larger trees constructed by RRT and RRT–ADD that build trees of maximal allowed size, which equals to the number of iterations I_{max}. The size of constructed trees build by RRT–Retraction does not increase so much with increased I_{max}. RRT–Retraction always builds larger trees than RRT and RRT–ADD, because the retraction procedure can push the samples deeply into the narrow passages.

The visualization of a solution found by RRT–IS is depicted in Fig. 6.12. The video with the final solution is available at [249].

6.5 Hedgehog in the cage

In the Hedgehog problem (Fig. 5.1), the task is to remove the spiky object from the cage with five identical windows. The problem requires gentle manipulation of the object, as the radius of its main sphere is just few percents smaller than the width of the windows. The configuration space probably consists of many narrow dead-end passages and only few traversable ones. The cage is a cylinder of radius 2.5 mu and height 7.5 mu and the size of the whole 3D workspace is $8 \times 8 \times 10$ (width $\times$ depth $\times$ height) and the radius of the Hedgehog body is 0.95 mu.

The size of the environment is the same as the 3D BugTrap problem, so the algorithms have

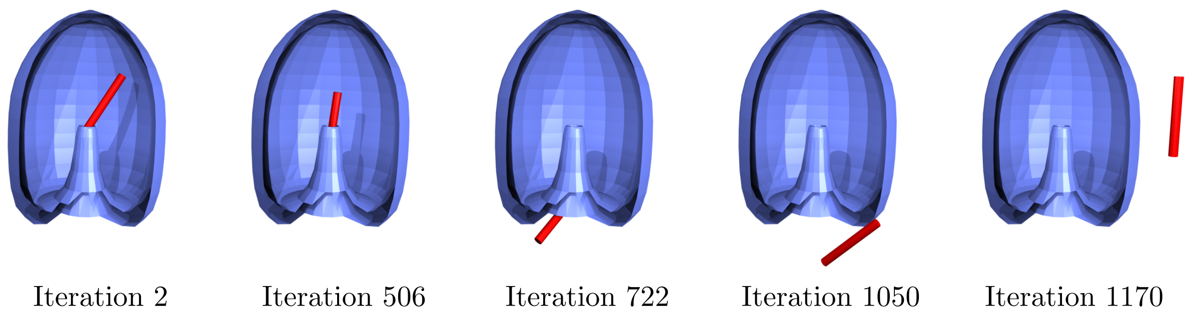


Figure 6.12: Solution of 3D BugTrap benchmark.

Table 6.1: Results on 3D BugTrap benchmark.

	RRT-IS		**RRT**		**RRT-ADD**		**RRT-Ret**	
	Mean	Dev	Mean	Dev	Mean	Dev	Mean	Dev
$I_{max} = 100,000$								
Run-time [s]	232.17	153.49	13.37	0.29	16.68	1.23	60.93	6.91
Tree size $(\times 10^3)$	437	230	99	0	99	0	95	11
Success rate	100 %		0 %		0 %		24 %	
$I_{max} = 200,000$								
Run-time [s]	227.37	160.25	28.75	1.82	36.16	0.93	93.01	33.01
Tree size $(\times 10^3)$	433	257	199	0	199	0	140	49
Success rate	100 %		0 %		0 %		66 %	
$I_{max} = 500,000$								
Run-time [s]	225.74	176.05	71.83	12.75	106.21	9.57	123.98	23.21
Tree size $(\times 10^3)$	477	346	446	69	486	32	183	34
Success rate	100 %		64 %		18 %		100 %	

been run with the same setting as in the previous section. The number of planning iterations is set to $I_{max} = 5 \times 10^5$ for RRT–IS. Due to low success rate of other methods, they have been tested also with $I_{max} = 5 \times 10^6$. The start position is located in the center of the cage, while the goal position is defined as a region outside the cage (the whole hedgehog must be out of the cage). The video of a final solution found by RRT–IS is available at [250].

The results are shown in Tab. 6.2. The RRT–IS method is the only planner that provides solution of the benchmark. The difficulty of the object manipulation is indicated by the size of the constructed trees. With $I_{max} = 5 \times 10^5$, RRT–IS builds trees with $\sim 174 \cdot 10^3$ nodes, but the other methods build significantly smaller trees.

The reason is that there is not much space to manipulate the object within the cage. After several thousands of planning iterations, the configuration tree fills free space of the cage and in order to extend it more, random samples q_{rand} need to be generated inside the cage. However, due to uniform sampling of configuration space used in RRT, RRT–ADD and RRT–Retraction, most of the samples q_{rand} are generated outside the cage. By considering the cage as a cylinder and by neglecting the thorns, the volume of the cage is ~ 22 % of the workspace. Therefore, approximately 78 % of random samples cannot extend the tree. From the rest of 22 % of random samples, most of them do not lead to successful extension of the tree due to collisions of the spiky robot with the cage. This is one of the possible reasons why increasing of planning iterations from $I_{max} = 5 \times 10^5$ to $I_{max} = 5 \times 10^6$ does not increase the success rate of RRT, RRT–ADD and RRT–Retraction methods.

Table 6.2: Results on Hedgehog in the cage benchmark.

	RRT-IS		**RRT**		**RRT-ADD**		**RRT-Ret**	
	Mean	Dev	Mean	Dev	Mean	Dev	Mean	Dev
$I_{max} = 5 \times 10^5$								
Run-time [s]	1,619.86	2,179.19	118.77	14.45	70.30	6.99	463.87	12.07
Tree size ($\times 10^3$)	174	213	7	0	42	3	41	0
Success rate	90 %		0 %		0 %		0 %	
$I_{max} = 5 \times 10^6$								
Run-time [s]			1,710.52	80.79	1,546.34	68.00	3,600.00	0.00
Tree size ($\times 10^3$)			60	1	276	12	317	6
Success rate			0 %		0 %		0 %	

6.6 Discussion

The presented experiments have shown performance of herein proposed RRT–IS method in the task of path planning for the simple cubic objects, stick robots as well as in 3D BugTrap and Hedgehog benchmarks. The method has been compared to RRT, RRT–ADD and RRT–Retraction.

The experiments have shown, that RRT–Retraction and RRT–ADD are superior for the simple cubic robots, but their performance decreases if robots of other shapes are used. The experiment with two Stick robots has shown, that the planners are also sensitive to the geometric center of the robots. Path planning for the Stick$_c$ robot is more challenging for RRT, RRT–Retraction and RRT–ADD than for the Stick robot. The proposed RRT–IS method is able to find paths for both Stick and Stick$_c$ robots.

The biggest difference between the performance of RRT–IS and the other planners has been observed in the challenging Hedgehog and 3D BugTrap scenarios. These benchmarks contain challenging narrow passages, that requires gentle manipulation of the objects. The percentage of solved trials is significantly lower for the RRT, RRT–ADD and RRT–Retraction methods. The planning iterations I_{max} can be increased to achieve better results with these methods, but this helps only in the 3D BugTrap scenario.

There is no universal method capable of solving all instances of the considered path planning problem. The experiments shown, that the problems involving convex geometries are solved faster by the state-of-the-art methods RRT–ADD and RRT–Retraction. The proposed RRT–IS method is computationally more intensive in these scenarios. However, RRT–IS has been shown to be superior in path planning involving complex geometries and in scenarios with complicated narrow passages. Finally, the most challenging benchmark of the experiments, the Hedgehog problem, has been solved only by the RRT–IS planner.

Chapter 7

Motion planning for modular robots

Modular robots are composed of many basic building blocks — modules. The modules can be connected in many ways and they can form robots of various shapes (Fig. 7.1). In comparison to fixed-shape robots, modular robots are more versatile as they can adapt their shape for a given task and environment.

Motion planning is required for self-reconfiguration as well as for navigation in an environment. In the second task, which is considered in this chapter, a feasible trajectory needs to be found for the whole robot in order to reach a desired place. Modular robots typically have many degrees of freedom and motion planning leads to search of a high-dimensional configuration space.

Motion planning for modular robots operating in large flat environments is investigated in this chapter. The task is motivated by Grand Challenge scenario proposed in Symbrion/Replicator EU projects [156]. The scenario starts by autonomous exploration of a 2D environment with tens of modules that are supposed to form larger robots in order to overcome obstacles and reach distant power sockets. An integral part of the scenario is motion planning for the organisms (modular robots), where the main task is to find trajectory for the whole robot to a distant place.

Motion planning for modular robots is challenging due to many motion constraints, necessity to control many actuators and the need to search a high-dimensional configuration space. Here, the number dimensions of the configuration space grows with the number of modules. The RRT method is a suitable candidate for the motion planning as it can cope with all the constraints. Motion model of modular robots needs to be simulated rather than described analytically. This, together with the necessity to control many actuators, significantly slows down construction of the configuration tree.

During implementation of the RRT-based planner, it was identified that performing motions of the whole robot is more challenging that the narrow passage problem. To enable RRT-based motion planning for modular robots, we propose to employ locomotion generators to realize basic motions of the robots. To decrease the number of evaluations of simulated motion model, the guiding principle can be used. The novel motion planner, called RRT–MP (RRT with Motion Primitives) [263] is described in this chapter.

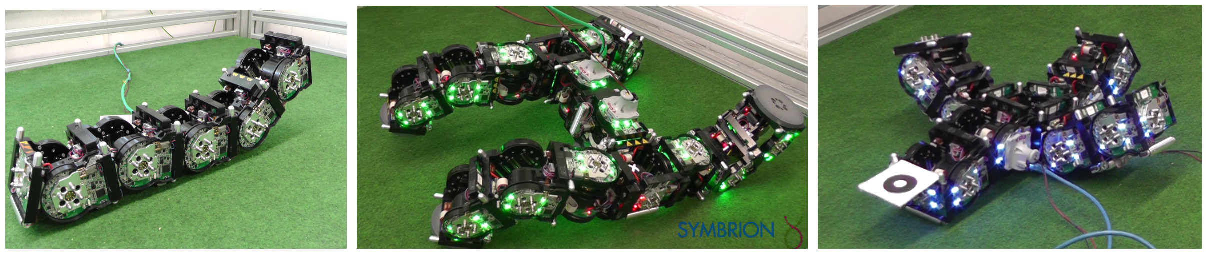


Figure 7.1: Example of modular robots composed of CoSMO [158] platform.

7.1 Overview of modular robotics

Modular robotics is motivated by challenging scenarios like space exploration [282, 283], search and rescue missions [58], or maintenance of scientific facilities [192, 193]. In these scenarios, robots have to perform various tasks from locomotion to object manipulation. The idea of modular robotics is to make robots on demand according to a given task instead of using many specialized robots. Modular robots are composed of many mechatronic modules that can be connected to form bodies of various shapes [70]. For example, a lizard-like robot can be formed to traverse a rough terrain and it can be reconfigured to a snake-like robot in order to traverse a narrow passage.

According to geometrical arrangement of the modules, modular robots can be classified into two types, chain-type or lattice-based systems. The chain-type systems are composed of modules that form single- or multi-branch linkages so they can form robots with legs or arms similar to living creatures. The modules can be positioned arbitrary in space, which allows to move the whole robot using joint-control locomotion and to perform object manipulation. The joint-control locomotion is achieved by controlling angles of individual joints. However, it is difficult to realize self-reconfiguration, as it requires a precise cooperation of several modules. In lattice-based systems, modules occupy discrete locations similarly as atoms in a grid [44, 43, 265]. As the modules move in the grid, reconfiguration is easier than with the chain-type robots, but it is difficult to generate dynamic robotic motions like walking or crawling. Instead, lattice-based robots can move via self-reconfiguration [195, 66, 285, 242, 204, 213].

Many modular robotic systems have been developed. We refer to [176] for a comprehensive survey. Examples of homogeneous platforms that consist of modules of same type are CONRO [222, 212], PolyBot [281, 283], M-TRAN [179, 178], ATRON [109], SuperBot [215], Molecubes [288] and ICubes [244]. Homogeneous platforms have gain special interest, as they simply scale in size by adding new modules. Heterogeneous systems employ several types of modules that are specialized for certain tasks. Examples of heterogeneous systems are ModRed platform [21] and recent Symbrion/Replicator robots [156].

The Symbrion/Replicator platform consists of three types of modules called Scout, CoSMO and ActiveWheel (Fig. 7.2). The modules use the same electrical design, power and communication buses, and docking units. Power is provided by an internal battery or it can be shared within the organism through the power bus. The modules can communicate via ZigBee interface and, if connected, also using Ethernet. Each module is also equipped with a Blackfin processor, 32 MB of RAM and μClinux operating system. The connection between the modules is achieved using an active docking mechanism with locks, i.e., at least one of the connected modules has to lock the docking mechanisms in order to ensure the proper connection.

Scout robots support fast 2D locomotion using a pair of tracks and they are also equipped with a simple hinge for 3D locomotion [123]. The main task of Scout modules is to explore 2D environments in order to collect information about obstacles and to localize other modules. CoSMO modules are equipped with a more powerful hinge for 3D locomotion, which allows them to lift up to four other modules. The CoSMO modules are intended as backbone modules for large robots (organisms). 2D locomotion is achieved using a screw-drive mechanism that provides precise, albeit slow motion [158]. ActiveWheels are hybrid robots, as they are equipped with three omnidirectional wheels for 2D locomotion and they can be also connected with other two platforms [156]. ActiveWheel robots are designed as rescue robots, as these can pickup failed modules and deliver them autonomously to a repair station.

Motion planning methods proposed in this thesis has been developed and tested on the CoSMO platform [158]. The modules are cubes of size $10 \times 10 \times 10$ cm and they weight ~ 1.2 kg. The modules are equipped with four docking mechanisms. Beside the powerful joint for 3D locomotion and the pair of screw-drives, each module is equipped with basic sensors (camera, IR distance sensor, accelerometers). Examples of various robots made of CoSMO modules are depicted in Fig. 7.1.

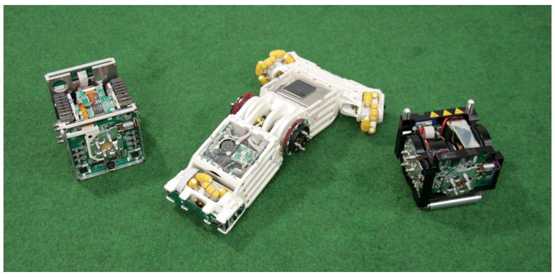


Figure 7.2: Examples of Symbrion/Replicator heterogeneous modules from left to right: Scout, ActiveWheel and CoSMO.

7.2 Problem analysis

In this thesis, chain-type modular robots without loops are considered. A configuration of a modular robot with n modules is described as $q = (x, y, z, \varphi, \theta, \psi, a_1, \dots, a_n) \in \mathcal{C}$, where (x, y, z) and (φ, θ, ψ) denote the 3D position and orientation of the pivot module respectively, a_i denotes the angles of the joints and $\mathcal{C}$ is the configuration space. A configuration is feasible, if the robot does not collide with any obstacle and if no two modules collide. The task of motion planning is to find a feasible trajectory from a start configuration q_{start} to a goal configuration q_{goal}. In this thesis, the self-reconfiguration capabilities are not considered, therefore the topology of the robot is not changed during the motion. Although the self-reconfiguration is supported by the CoSMO modules, it is not used because of the design of Grand Challenge scenario. The motion planning problem requires to search the high-dimensional configuration space considering the motion constraints. We propose to solve this task using RRT method, as it allows to handle motion constraints.

A crucial procedure of RRT is the expansion step, that generates new configurations reachable from a selected node q_{near}. Traditionally, the expansion is realized by examining all possible combinations of discretized control inputs. This leads to k^n combinations, where k is the number of discrete levels in each actuator and n is the number of actuators. This approach is not useful for modular robots, because the number of combinations grows exponentially with the number of actuators, and consequently, also with the number of modules. To decrease the complexity of the expansion step, the hinges can be moved to randomly chosen positions [255] or they can be controlled using random time signals [284]. However, motions achieved using these techniques are clumsy and inefficient. A robot controlled in such a random manner performs many small steps, which increases the number of planning iterations required to find a solution.

7.2.1 Motivation for motion planning with motion primitives

Instead of generating the control signals randomly by the motion planner, the joint-control locomotion can be achieved using locomotion generators like Central Pattern Generators (CPG). CPGs are inspired by observations from nature, where coupled neuro-oscillators provide rhythmic signals for muscles [177]. By changing parameters and coupling of the CPGs, various gaits like crawling or caterpillar-like motions can be achieved. The parameters can be found automatically using optimization tools like genetic algorithms (GA) [165, 239, 102, 261, 113, 114, 103], evolutionary algorithms (EA) [260, 264, 263, 258] or reinforcement learning [181]. CPGs have been used to generate locomotion of snake-like robots [48, 188, 113], quadruped robots [128], humanoid robots [175, 208, 61], swimming robots [102, 101] and modular robots [113, 114, 47]. A comprehensive review of CPGs and their usage in robotics can be found in [100].

It should be noted, that despite the ability of CPGs to generate various gaits for wide range of robots, they are aimed to provide locomotion rather than solving high-level tasks as collision avoidance or navigation to a predefined location. A more complex behavior can be achieved by introducing sensory feedback [183, 167, 137, 42], but these approaches are intended mainly to cope with altering terrains.

To achieve a more high-level behavior (i.e., to navigate the robot to a goal region in a large environment), we propose to equip the robot with several gaits — motion primitives. Each primitive p is realized by a suitable locomotion generator and it is represented by a vector $\mathbf{x}^p$ of parameters of its generator. The generator provides control signals $\mathbf{u}^p(t) = (a_1(t), \dots, a_n(t)), 0 < t \leq \tau^p$, where $a_i(t)$ are desired positions of joints. Depending on the type of CPGs, the signals are distributed to individual modules from a central unit, or they are computed on each module separately. The centralized control is easier for implementation, but it can be prone to communication failures. A more robust approach is to run a cyclic function on each module separately. To synchronize actions of the modules, each module sends a message to its neighbors after a fraction of the period is completed [233, 232]. Synchronization can also be achieved using artificial hormones [223, 216], which flow though the modules and trigger their actions. In this thesis, the control signals are generated on the pivot module and they are distributed using messages to other modules. Each module then moves to the desired position using an internal PD controller.

The robot can be equipped with several motion primitives such as 'walk-forward', 'rise-head' or 'turn-left'. To control the robot using a primitive p, first the parameters $\mathbf{x}^p$ are retrieved from a look-up table and the control signal $\mathbf{u}^p(t)$ is generated and distributed to the individual modules.

7.3 RRT–MP: RRT with Motion Primitives

The RRT–MP (Rapidly Exploring Random Trees with Motion Primitives) [263] algorithm is designed to solve the motion planning problem for joint-controlled modular robots. A robot is equipped with a set of motion primitives, that are realized using CPGs. These primitives are considered as atomic actions in the motion planning level and the task of the planner is to find a sequence of the primitives to reach the goal configuration.

RRT–MP builds configuration tree rooted at q_{start} and it works as follows. In each iteration, a random configuration $q_{rand} \in \mathcal{C}$ is generated and its nearest neighbor $q_{near} \in \mathcal{T}$ is found in the tree. To obtain new configurations reachable from q_{near}, input signal $\mathbf{u}^p(t)$ is generated from the settings $\mathbf{x}^p$ for each of the employed motion primitives. A new configuration reachable using the primitive p is determined using a simulated motion model, which is run over time interval τ^p. From the obtained set of reachable configurations, the nearest configuration q_{new} to q_{rand} is selected and added to the tree. The information about the utilized primitive is stored in the edge (q_{near}, q_{new}). The algorithm terminates if q_{new} approaches the q_{goal} to a predefined distance d_{goal}. The RRT–MP method differs from RRT only in the expansion step, therefore only the expansion step is listed in Alg. 7. An example of the expansion step is visualized in Fig. 7.3.

A trajectory is then found in the tree similarly as in normal RRT: the nearest node to $q'_{goal} \in \mathcal{T}$ towards the goal configuration is found and the path towards q_{start} is retrieved from the tree. The resulting trajectory is a sequence of k segments (q_i, p_i, τ_i, q'_i), where q_i is the initial state of the segment, p_i is the primitive that needs to be performed in this segment over time τ_i, and q'_i is the expected end configuration of this segment. The first segment starts from q_{start}, therefore $q_1 = q_{start}$. The last segment ends in the vicinity of the goal, and so $q_k = q'_{goal}$. The last configuration q'_i of segment i is the same as the first configuration q_{i+1} of the next segment.

The robot then executes this sequence of primitives by generating signals $\mathbf{u}^{p_i}(t)$ according to setting $\mathbf{x}^{p_i}$ of primitive p_i and these signals are applied to the actuators over time τ_i. The expected end configurations q'_i of each segments can be used to detect deviation of the robot from the designed trajectory. A new plan is computed if the robot deviates from the planned trajectory.

It has to be ensured that a primitive can be switched to another one. The current applicability of a primitive depends on the actual state of the robot, and also on the previously used primitive.

For example, the primitive ‘stand-up’ can be applied if the robot is lying on the ground. However, it may not be straightforward to determine the proper states for complex shapes and it is more comfortable to decide the applicability based on the previously used primitive. For example, we can specify, that ‘stand-up’ can be applied after ‘lie-down’ has been applied without the necessity to define all possible states in which it can be applied. Let the predicate $isApplicableAt(p, q)$ be true if a primitive p can control the robot from state q. Also, let $isApplicableAfter(p, p_{prev})$ be true, if a primitive p can be applied after a primitive p_{prev}. The predicates are used in the RRT–MP expansion step (line 3 in Alg. 7).

Algorithm 7: expandConfiguration(q_{rand}, q_{near}): Expansion procedure of RRT–MP

```
Input: random configuration $q_{rand}$, configuration to expand $q_{near}$, previously used primitive $p_{prev}$
Output: Extended tree $\mathcal{T}$
1  $R = \emptyset$;
2  foreach $p \in motionPrimitives$ do
3      if $isApplicableAt(p, q_{near}) \wedge isApplicableAfter(p, p_{prev})$ then
4          $\mathbf{x}^p$ = parameters of the motion primitive $p$;
5          $\mathbf{u}^p(t) = (a_1(t), \dots, a_n(t)), 0 < t \le \tau^p$; // control signal using the locomotion generator with
           setting $\mathbf{x}^p$;
6          $q$ = simulatedMotionModel($\mathbf{u}^p(t)$,$q_{near}$);
7          $R = R \cup \{(q, p)\}$;
8      end
9  end
10 if $R \neq \emptyset$ then
11     $(q_{new}, p)$ = closest configuration from $R$ to $q_{rand}$ and its primitive p;
12     $\mathcal{T}$.addNode($q_{new}$);
13     $\mathcal{T}$.addEdge($q_{near}, q_{new}, p$);
14 end
```

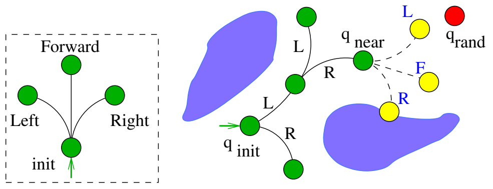


Figure 7.3: Example of a tree constructed using three motion primitives (Left, Forward, Right). In the expansion step, the nearest node q_{near} in the tree towards q_{rand} is found and the motion primitives are applied in this node (resulting in the yellow configurations).

The utilization of motion primitives significantly decreases the complexity of the expansion step, that is given by the number of the primitives rather than by the number of actuators in the robot. However, due to utilization of the simulated motion model, the expansion step is still a time consuming procedure. To speed up the planning process, the number of expansion steps should be decreased. We propose to employ the guiding principle designed in Chapter 3 to achieve this. In the considered task of motion planning for modular robot in flat-like large environments, where the task is to move the robot to a distant place, the main task of the guiding path is to suppress exploration of the whole configuration space and to focus the sampling to a region, where a solution can be found.

The guiding path can be computed using the basic path planning methods in the 3D workspace. We propose two basic strategies to compute the guiding path. First, the environment can be represented using 2.5D grid, in which a path from start position to goal position is found. Second, the environment can be described using a 3D triangular mesh. The mesh can be converted to a graph, where nodes are vertices, the edges correspond to edges of the triangles, and the cost of edges is the distance of related vertices. If the mesh also contains inaccessible areas, such a

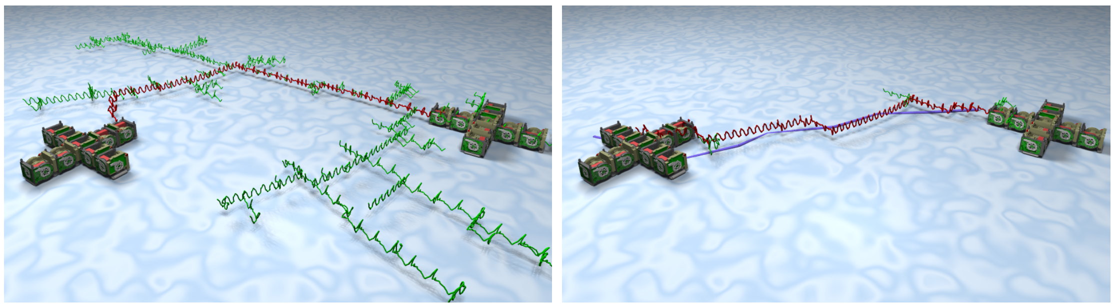


Figure 7.4: Example of motion planning for modular robot with and without the guiding path.

obstacles, the corresponding edges are not used in the graph. The graph is then searched for the guiding path.

The result of both approaches is a workspace path represented by a sequence of 3D waypoints. To utilize this workspace path as a guiding path for the modular robot, it has to be converted to configuration space. A possible way to convert a workspace point (x, y, z) to a configuration $q(x', y', z', \varphi, \theta, \psi, a_i), i = 1, \dots, n$, is to use the 3D position from the workspace (i.e., $x' = x$, $y' = y$ and $z' = z$), and set the remaining variables to zero. This is suitable if the robot moves in a large area, where it is not required to follow the path precisely. As the workspace path is computed on the surface of the ground, but the pivot module moves above the ground, it is possible to add offset, i.e., $z' = z + \text{offset}$. The offset can be set e.g. as the average altitude of the pivot module during a motion. Example of a guiding path is depicted in Fig. 7.4.

7.3.1 Analysis of motion planning with motion primitives

The utilization of motion primitives brings several advantages. First, we can assume that each primitive generates the desired motion efficiently. Such primitives hence move the robot faster than randomly generated control signals. Planning with motion primitives is thus faster and less memory consuming than naive sampling-based planning with random control inputs [255, 284].

Second, the input signals of the motion primitives do not need to be stored in the tree, because these can be generated on demand by the utilized CPG. Therefore, the edges in RRT–MP keep only information about the utilized primitives, which decreases the memory requirements.

Third, the complexity of the expansion step is not influenced by the number of actuators, but only by the number of available motion primitives (size of the set *motionPrimitives* on line 2 in Alg. 7). RRT–MP is scalable and can also be used for systems with many actuators.

The ability of RRT–MP to find a plan for a modular robot depends on two key properties: motion primitives and the metric. Both have to be chosen by human operator considering the actual mission. The primitives should be selected considering capabilities of the robot as well as suitability of the primitives for the given task. For example, primitives like ‘crawl’ or ‘turn’ are enough for moving on a terrain, but ‘stand-up’ primitive may be necessary if the robot has to reach a place above the ground. As the complexity of RRT–MP is given by the number of the motion primitives, the user should select suitable primitives, that are believed to be necessary to accomplish the task.

The ability of the tree to grow in the configuration space also depends on the employed metric measuring distances between the configurations. The metric has to be designed considering also effects of the primitives, so the primitives are distinguishable by the metric. The 3D Euclidean metric measured between 3D positions of pivot modules is suitable for motion planning in large environments, because it supports fast motion of the robot in the 3D subspace of the configuration space. This metric is used for motion over large distances. Different metric, possibly with rotation, should be used to plan shorter trajectories, e.g. during the docking to a power socket.

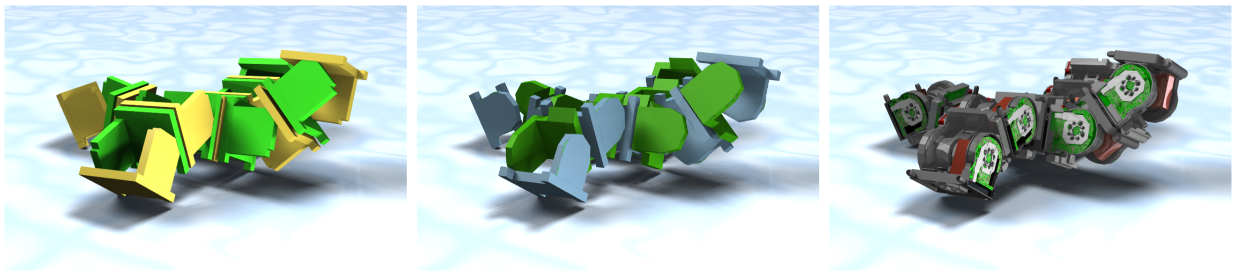


Figure 7.5: Simulation models of the CoSMO modules.

7.4 Implementation details

The sampling-based motion planning for modular robots using the concept of motion primitives differs from motion planning for mobile robots in two main aspects. First, motion model is not described analytically, but it is realized using a physical simulation. Second, an optimization of the motion primitives is required. These two components are described in this section.

7.4.1 Simulated motion model

The forward motion model describes the change of position of the robot after control inputs are applied. While the motion of mobile robots can be described analytically, it is not easy to derive a closed-form motion model for modular robots due to many constraints caused by kinematics and necessity to consider contacts between the robots and underneath surface [205]. A black-box motion model can be realized using physical simulation [205, 255, 263, 235]. Physical simulations are widely used in robotics e.g. for modeling complex robots (e.g. snake-like, legged or modular robots), particular devices [278] or deformable objects [74]. In comparison to closed-form motion models that are prepared for a specified robot, physical simulations allow us to model robots with changing shape and topology, which is very important in modular robotics, where robots can reconfigure to various shapes. On the other hand, evaluation of simulated motion models is usually slower than computation of a closed-form model.

In the physical simulation, a robot is represented using a set of physical bodies with associated masses, mass densities and geometric shapes. The connection between the bodies (connection between two modules) can be realized using several types of joints, which restrict relative motions of the bodies. Static obstacles including ground or terrain, are modeled only using their geometries. In each simulation step, first collisions between the geometries are computed in order to determine contact forces. These forces are caused e.g. by contacts between the robot and the ground. In the next step, the contact forces, gravitational force and forces caused by actuators are integrated to determine new position of the bodies. Due to restrictions caused by the joints, new positions of the bodies are computed numerically.

The speed of the physical simulation is determined by the speed of its two main components: collision detection and numerical solver. The task of the collision detection is to report penetration vectors (or penetration depth [56]) that describe how to move two bodies in order to remove the collision. The quality of penetration vectors is crucial for the stability of the physical simulation. The vectors can be computed analytically between geometric primitives like boxes, spheres and cylinder, and it is therefore recommended to approximate robots by these primitives. Usage of detailed geometric models from CAD systems is also possible, but the computation of penetration vectors between general (non-convex) triangle meshes is not trivial. Computation of penetration vector may become ambiguous because of missing information about orientation of faces. Therefore, it cannot be determined how to move the bodies in order to remove the collision and the probable vector has to be estimated heuristically. Incorrectly determined penetration

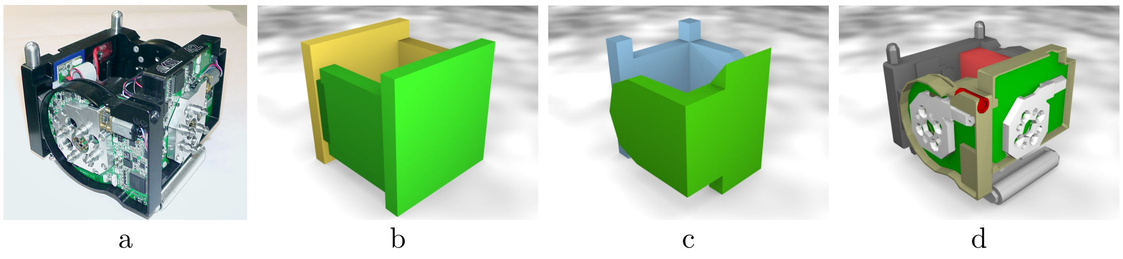


Figure 7.6: Simulation models of CoSMO modules. The HW module is depicted in (a). In the simulation, the module is represented by four bodies (two green and two yellow) (b). Each body has assigned different mass and mass density. Collision detection is realized using more detailed geometry (c) and the visualization is realized using 3D CAD models with $\sim$ 10k triangles (d).

vectors can cause leaking of objects to each other, which consequently decreases precision and also stability of the simulation.

The task of the numerical solver is to determine new accelerations impacting the bodies based on actual velocities/moments, forces caused by contacts between the bodies, moments of joints and additional forces like gravitation. A time complexity of each simulation step of ODE physical engine [2] for a group of n bodies with m_1 joints is $O(k_1 m_1 + k_2 m_2^3 + k_3 n)$, where m_2 is the total number of DOFs removed by all m_1 joints. Here, the number joints also includes the contact joints between two touching bodies and it is therefore always higher than the number of real joints of the modular robot being modeled. Due to the presence of many 1-DOF joints, that are used to model connection between modules, the physical simulation of modular robots is considerably time consuming. Each simulation step integrates the forces over a time t_{sim}, that determines speed, precision and also stability of the simulation. In the unstable simulation, objects can penetrate each other and the joints do not properly limit relative motion of the objects. The instability can even lead to "explosion" of the objects, that is caused by violation of the assumptions of the numerical solver. With smaller t_{sim}, positions of objects are determined more precisely and the stability of simulation is increased, but the runtime of the overall simulation is slower, as more physical steps need to be performed.

To decrease the time complexity and to increase the stability of the simulation, the simulated CoSMO modules (Fig. 7.6a) are modeled using multiple boxes (Fig. 7.6b). Each body has assigned different mass density and mass in order to make precise model of mass distribution within the modules that is necessary for the accurate simulation. Collision detection is computed using a more detailed shape (Fig. 7.6c). Visualization is achieved using detailed 3D meshes obtained from CAD models (Fig. 7.6d). An example of simulated Quadropod robot with various levels of details is depicted in Fig. 7.5. The physical simulation of Symbrion/Replicator robots is provided by Robot3D [271] and Sim [254] simulators.

The physical simulation is used in the expansion step of RRT–MP to determine how the robot moves after a motion primitive p is applied. In each iteration of the planner, the tree can be expanded from different configuration q_{near}. This requires to reset positions of all bodies and joints according to configuration q_{near}. This non-linear utilization of the physical simulation may cause instability. It is necessary to maintain also other variables like accelerations and velocities of the internal objects in order to ensure its stability. These additional variables need to be stored in the configuration tree together with the configurations. We refer to our paper [255] where technical details about integration of physical simulation into RRT-based planners are described.

7.4.2 Optimization of motion primitives

To achieve a desired motion primitive, parameters $\mathbf{x}^p$ need to be optimized according to a fitness function $f^p(\mathbf{x}^p)$ that differs for each primitive p. The success of the optimization process depends on the definition of the fitness (cost) function. The optimization of tens or even hundreds of parameters of CPGs is necessary to achieve motion of modular robots, because of many actuators to be tuned. This high-dimensional optimization can be solved using genetic algorithms [165, 239, 102] or other evolutionary methods. These methods rely only on the fitness function. Therefore, the fitness function should allow the optimization process to start the evolution, which requires to distinguish between good and bad solutions in the early stages of the evolution although the solution itself does not provide the expected behavior [184]. Many works suggest using the traveled distance as the fitness. However, the traveled distance cannot distinguish between an oscillation around a fixed point and a real motion outward the point. Let us imagine a robot moving forward and backward, whose average position is zero. Such a motion would be rewarded by a high fitness according to the traveled distance, but the motion itself is useless if a robot has to visit a distant place.

We propose a different approach to define the fitness function, which is motivated by the purpose of motion planning with the motion primitives. As the main task of the robot is to move in a possibly large environment, fast motions to various directions are preferred. The fitness function of one primitive to be maximized is $f^p(\mathbf{x}^p) = D - d$, where D is the initial distance of the robot from a virtual goal and d is the distance from the goal after primitive $\mathbf{x}^p$ has been applied. The distances D and d are measured as 3D Euclidean distance between robot's pivot module and the virtual goal, that is placed in a sufficient distance D from the robot in the desired direction. The distance D should be large enough to ensure that the robot cannot overcome it during the time τ^p.

To evaluate the fitness function using the physical simulation, the simulated robot is created at the initial configuration $q(0)$ and it is driven by control signals $\mathbf{u}^p(t)$ defined by the primitive $\mathbf{x}^p$ being tested. The robot reaches configuration $q(\tau^p)$ after time τ^p. The fitness is visualized in Fig. 7.7. This fitness provides enough pressure to force the robot to move in the given direction as fast as possible.

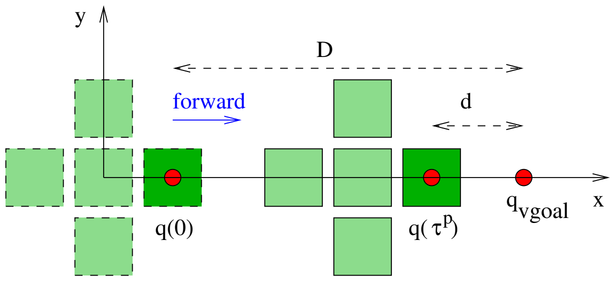


Figure 7.7: Example of a fitness function for the motion pattern 'move forward along x axis'. At the beginning, the robot is placed at position $q(0)$ and the motion primitive being tested is applied to the actuators for time τ^p. The initial distance D and the final distance d from the virtual goal q_{vgoal} are then used to compute the fitness $f^p(\mathbf{x}^p) = D - d$.

We employ the Particle Swarm Optimization technique (PSO) to find the suitable parameters $\mathbf{x}^p$. PSO is a population-based stochastic optimization technique, where each particle represents a candidate solution $\mathbf{x}^p$. Movements of the particles are inspired by bird flocking and fish schooling. In comparison to GA, PSO maintains the population using mutation only i.e., the crossing operators are not used. Movement of a particle in the search space is influenced by the particle's experience, and also by the experience of other particles. Each particle maintains coordinates $\bar{\mathbf{x}}_i^p$ in the search space associated with the best solution that it has achieved so far. Moreover, the position $\hat{\mathbf{x}}^p$ of the best solution over all particles in the past is shared between the particles. The quality of the particles is evaluated using a fitness function. Each particle has assigned a velocity $\mathbf{v}_i$. In each iteration of the PSO algorithm, a new velocity $\mathbf{v}'_i$ is computed and the new position $\mathbf{x}'^p_i$ is updated using the old position $\mathbf{x}_i^p$ and the velocity $\mathbf{v}'_i$. The velocity and position are updated in each dimension j separately:

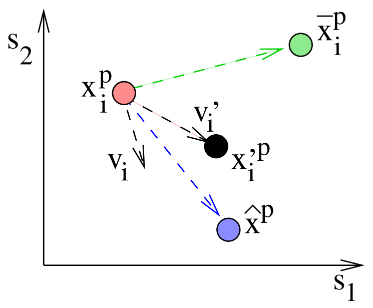


Figure 7.8: The PSO rule applied to a particle in a two-dimensional case. The old position of the particle is $\mathbf{x}_i^p$ and its velocity is $\mathbf{v}_i$. The particle is attracted to its best solution $\bar{\mathbf{x}}_i^p$ (green) and to the best solution found in the whole swarm $\hat{\mathbf{x}}^p$ (blue). The new velocity $\mathbf{v}'_i$ is computed and the particle changes position from $\mathbf{x}_i^p$ to $\mathbf{x}_i'^p$ (black).

$$\begin{aligned} v'_{i,j} &= \omega v_{i,j} + \phi_l r_p(\bar{x}^p_{i,j} - x^p_{i,j}) + \phi_g r_g(\hat{x}^p_j - x^p_{i,j}) \\ x'^p_{i,j} &= x^p_{i,j} + v'_{i,j}, \end{aligned} \tag{7.1}$$

where ω is an inertia factor, and r_p and r_g are random numbers from the uniform distribution $U(0, 1)$. The parameters ϕ_l and ϕ_g provide balance between exploration and exploitation of the solution space [122]. In the next step, $\mathbf{x}_i'^p$ becomes $\mathbf{x}_i^p$. The velocity of the particles is bound by a vector v_{max} so that the particles cannot overpass the search space. Practically, it is useful to set v_{max} at 10–20 % of the dynamic range of the variable in each dimension [54] or it can be a linear or nonlinear function of time [225]. Despite its simplicity, PSO has been shown to be fast and effective in various optimization problems. A comprehensive survey of PSO-based methods can be found in [59, 124].

Optimization using PSO brings several advantages that are important for optimizing motion primitives. As the particles flow through the solution space, the technique is suitable for optimization in continuous spaces, which is the case for the optimization of CPG parameters. PSO can provide a solution even if only a few particles are used [262]. This is important for PSO-based adaptation of the motion primitives, as the possibility to utilize few particles significantly speeds up the adaptation process. As the searching of the solution space is based purely on a mutation of existing solutions (particles), there is no need to design specialized crossing operators, which would be necessary for a GA-based optimization. The PSO-based adaptation method can be used with different CPGs without the need to tune the adaptation method itself.

7.5 Discussion

Modular robotics traditionally employ planning methods to study the self-reconfiguration task [195, 66, 285, 242, 204, 213, 145, 269], rather than for motion planning of the whole robots in environments. The joint-control locomotion is mainly studied using the concept of Central Pattern Generators, but the research is usually focused to the locomotion control only without aim to avoid obstacles and visit a distant goal.

This chapter proposed a novel motion planner for modular robots considering joint-control locomotion. The task of the planner is to design motion plans in large environments, which is motivated by Grand Challenge scenario of Symbrion/Replicator EU projects. The proposed method, called RRT–MP, is based on a sampling-based principle that solves the motion planning problem by random sampling of the configuration space of the robot. Due to the high number of actuators, it becomes intractable to examine all possible combinations of control inputs of the modular robot. We therefore suggest to employ local motion primitives realized by Central Pattern Generators.

The most relevant work to our RRT–MP system is the RRT–CPG planner [284]. The RRT–CPG generates basic motions of modular robots using a CPG. To reduce the number of CPG parameters, modules are divided to several functional blocks representing legs or the main body. The corresponding actuators in the same blocks are then controlled using same parameters. The parameters of CPGs are selected randomly using RRT. Each node in the tree represents a configuration of the robot. The nodes are expanded by examining several random parameters of the used CPG.

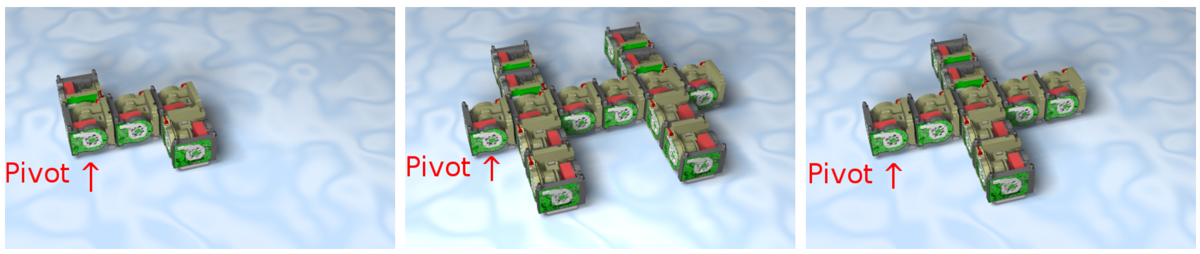
Both RRT–MP and RRT–CPG generate the low-level motions of the modular robots using CPGs. The main difference between the methods is that RRT–MP utilizes a set of fixed motion primitives that are optimized for a given task before the planning process. Contrary, RRT–CPG utilizes single CPGs, but its parameters are selected randomly during the planning process without evaluating efficiency of the motions. The motions achieved by RRT–CPG are thus more inefficient and many planning iterations are required to reach the goal configuration. Consequently, motion planning with RRT–MP is faster than with RRT–CPG.

Chapter 8

Experimental verification of RRT–MP

8.1 Simulated Experiments

The simulated experiments have been executed on Intel Core2@2.2 GHz with 4GB of RAM under FreeBSD 8.2. Three types of modular robots are used in the experiments: S-Bot (a five-module organism with an asymmetric one-module leg), Quadropod (nine modules with four legs) and Lizard (14 modules with four legs). The robots are depicted in Fig. 8.1. The simulations are based on the Robot3D simulator [271], a specialized simulator for modular robots co-developed by the author of the thesis within the Symbrion/Replicator projects. The time step of the physical engine is $t_{sim} = 10$ ms, which is a compromise between the speed of the simulation and its stability. For the purposes of simplicity, the size of each module is 1 map unit (mu) and other lengths are described in this unit. The nearest neighbors are searched using the KD-tree data structure provided by the MPNN library [280]. The distances between the configurations of the robots are measured using the 6D Euclidean metric computed between position and rotation of pivot modules.



S-bot Lizard Quadropod.

Figure 8.1: Simulated robots with highlighted pivot modules.

8.1.1 Optimization of motion primitives

The robots are equipped with four motion primitives $p \in \{$‘move-forward’,‘move-back’,‘move-left’,‘move-right’$\}$. The primitives are modeled using sine-CPG, i.e., $a_i(t) = A_i + \sin(\omega_i t + \varphi_i) + B_i$. The parameters $x^p = (A_i, \omega_i, \varphi_i, B_i)$ of the CPGs are found using the Particle Swarm Optimization approach described in Section 7.4.2. The parameters space to be searched is defined by physical properties of the hinges, and the ranges are: $0 < A_i < \pi/2$, $0.1 < \omega_i < 5$ and $0 < \varphi_i < 2\pi$. The offsets B_i are set to zero for the purposes of this work. The sine-CPG has been chosen as it can model the desired ‘move-*’ primitives efficiently with less parameters in comparison with other CPGs [258].

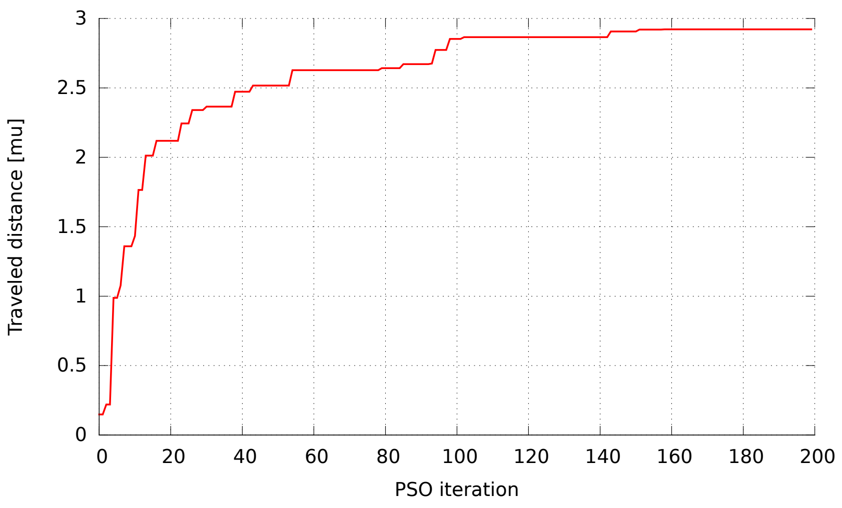


Figure 8.2: Progress of fitness function for 'move-forward' primitive for the Quadropod robot.

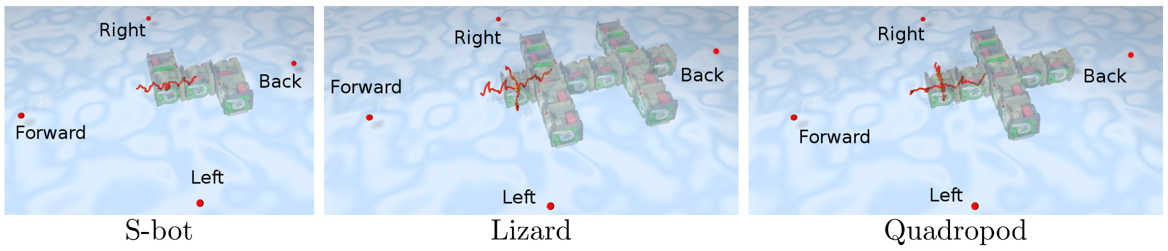


Figure 8.3: Visualization of motion primitives 'move-left', 'move-right', 'move-forward' and 'move-back'. The primitives are visualized as the trajectory of the pivot module. The red points represent goal points used for computing the fitness function in the PSO algorithm.

The motion primitives have been found using PSO with parameters $\phi_l = \phi_g = 2.05$ [122], 30 particles and 200 iterations. Duration of each primitive is $\tau^p = 10$ s. The primitives are optimized according to the fitness function depicted in Fig. 7.7, where $D = 7$ mu. Example of progress of the fitness function for the 'move-forward' motion is depicted in Fig. 8.2. The resulting motion primitives are visualized in Fig. 8.3.

8.1.2 Algorithm setup

Motion planning using motion primitives is compared with the methods RRT–CPG [284] and RRT–K [255]. In RRT–CPG, the actuators are also driven by the sine-CPG. However, their parameters are not predefined as in RRT–MP, but are generated randomly in each expansion step. In the RRT–K approach, the actuators are not controlled by a periodic signal. Instead, they are moved to a fixed angle a_i, which is generated randomly in each expansion step within the range $(-\pi/2, \pi/2)$. To enable a time comparison of all three planners, the complexity of their expansion steps should be equivalent. In RRT–MP, the complexity is determined by the number of utilized motion primitives. Therefore, RRT–CPG generates four random CPG settings in each iteration. RRT–K expands the tree using random motions, therefore four sequences of 1000 random angles are tested in each expansion of RRT–K to match the 10 s long motion primitives evaluated using $t_{sim} = 0.01$ s step. The maximum number of planning iterations is $I_{max} = 5,000$ for all algorithms, and the goal region is defined by radius $d_{goal} = 1$ mu. To show the efficiency of the motion planning with motion primitives, RRT–MP is run without the guiding path, unless noted otherwise.

Table 8.1: Results in the Plane scenario.

		RRT–MP		**RRT–CPG**		**RRT–K**	
		Mean	Dev	Mean	Dev	Mean	Dev
S-bot	**Path length**	17.20	4.90	63.13	18.46	54.35	22.87
	Iterations	6.60	4.28	80.23	61.30	1,412.17	1,174.51
	Path time [s]	4.53	1.63	23.33	7.33	73.63	37.10
	Runtime [s]	1.78	0.96	44.05	33.49	108.30	93.17
	Success rate	100 %		100 %		100 %	
Quadropod	**Path length**	26.54	6.05	120.95	14.90	186.84	105.73
	Iterations	18.85	25.06	130.30	73.41	5,000	0.00
	Path time [s]	6.15	1.53	50.75	6.43	201.40	109.14
	Runtime [s]	12.85	16.04	239.34	138.35	1,401.15	530.27
	Success rate	100 %		100 %		15 %	
Lizard	**Path length**	35.15	4.24	382.72	77.97	103.54	26.77
	Iterations	13.93	3.79	2,270.29	1,375.92	943.00	459.32
	Path time [s]	9.10	1.40	211.95	41.60	191.05	46.58
	Runtime [s]	30.31	7.66	12,488.80	7,553.03	696.09	388.38
	Success rate	100 %		90 %		100 %	

8.1.3 Plane scenario

In the first experiment, the task is to find a trajectory between two fixed positions in a flat environment of size 24 × 14 mu. The average results from 30 trials are shown in Tab. 8.1, where *Path length* (measured in map units) denotes the length of the found trajectory, and its time duration is denoted by *Path time*. *Runtime* is the time needed to find the solution. The row *Iterations* denotes the number of iterations that are needed to find the solution.

Based on the number of iterations, we can see that almost all algorithms solve the problem in less than the allowed number of iterations ($I_{max} = 5,000$), which is also indicated by success rates close to 100 %. The only exception is the RRT–K algorithm, which fails to find a solution for the Quadropod robot, and the success rate is only 15 % in this case. The RRT–MP algorithm outperforms the other two methods in all measured aspects: it solves the problem in the shortest time and it generates the fastest trajectories. RRT–MP is able to find a solution for all robots in less than 20 iterations, which indicates that its motion primitives are very efficient. Although RRT–CPG can also provide a solution, its runtime is significantly worse than the runtime of RRT–MP, especially for the Lizard and Quadropod robots. While RRT–MP is able to move the robot over long distances in each expansion step, the movements achieved by RRT–CPG are less effective. This is indicated by the highest number of iterations over all algorithms. The runtime and the number of required planning iterations of RRT–CPG increase with the size of the robot which indicates that robots controlled by random signals move slower, and it requires more iterations before the goal is reached. Examples of constructed trees are depicted in Fig. 8.4.

8.1.4 Plane scenario with multiple start/goals

The proposed RRT–MP planner is intended as the global planner, i.e., it should provide trajectory between any two positions in an environment. This is tested using the multiple start/goal test. The test consists of 120 start/goal pairs randomly placed in the flat environment of size 24 × 14 mu. For each pair, each planner is run 30 times.

The results are shown in Fig. 8.5 (S-Bot), Fig. 8.6 (Lizard) and Fig. 8.7 (Quadropod). The results obtained using the multiple start/goal pairs confirm results achieved in the Plane scenario. The motion plans are provided for most of the start/goal pairs by RRT–CPG and RRT–MP. Significantly less amount of start/goal pairs is solved by the RRT–K planner. From the time point of view, the slowest motion planning is realized using RRT–K and RRT–CPG planners

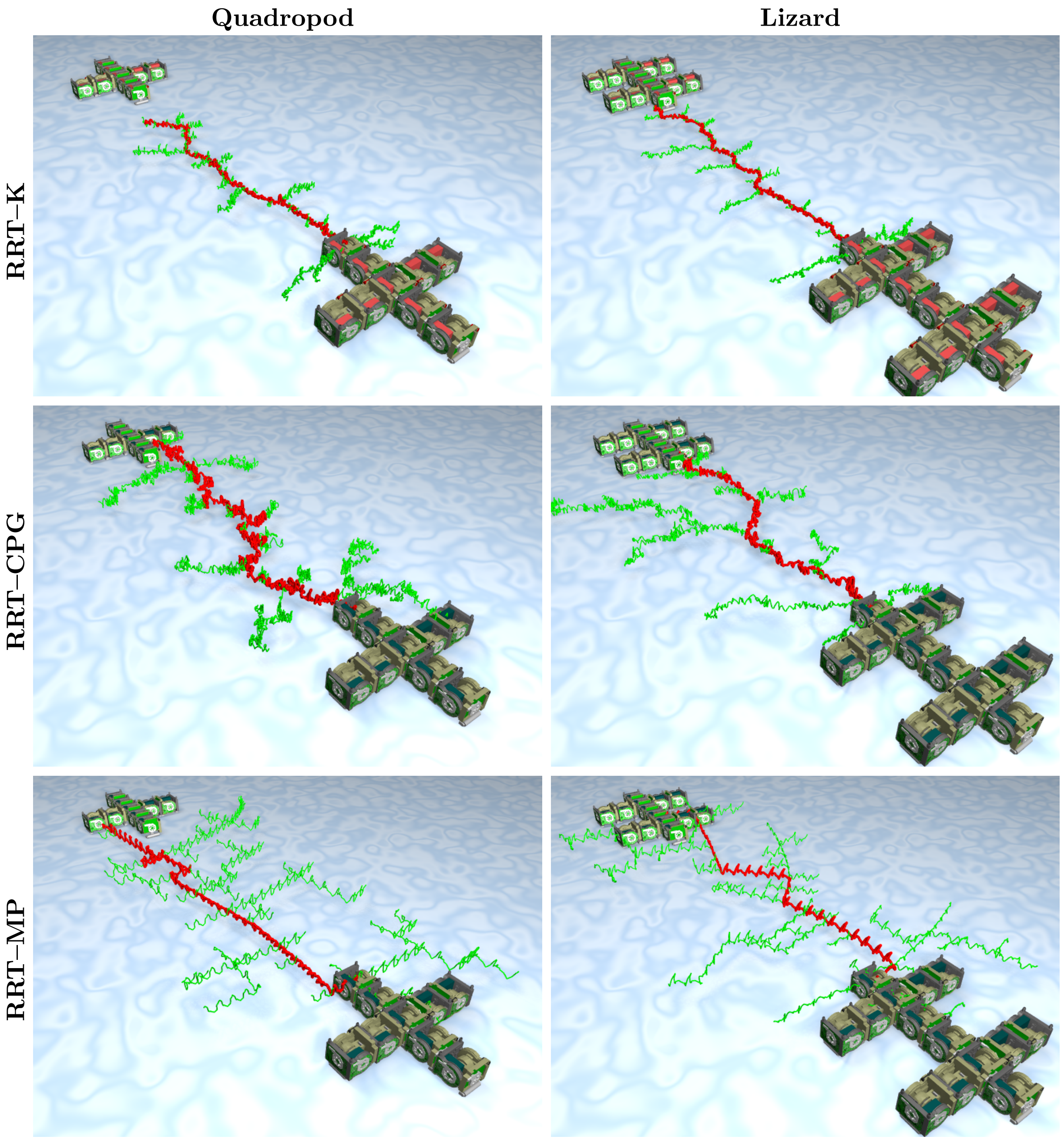


Figure 8.4: Examples of configuration trees (green) and final trajectories (red) for the Quadropod and Lizard robots in the Plane scenario. The trees and trajectories are visualized by the positions of the pivot module. The structure of motion primitives can be clearly seen in the configuration trees built by RRT–MP.

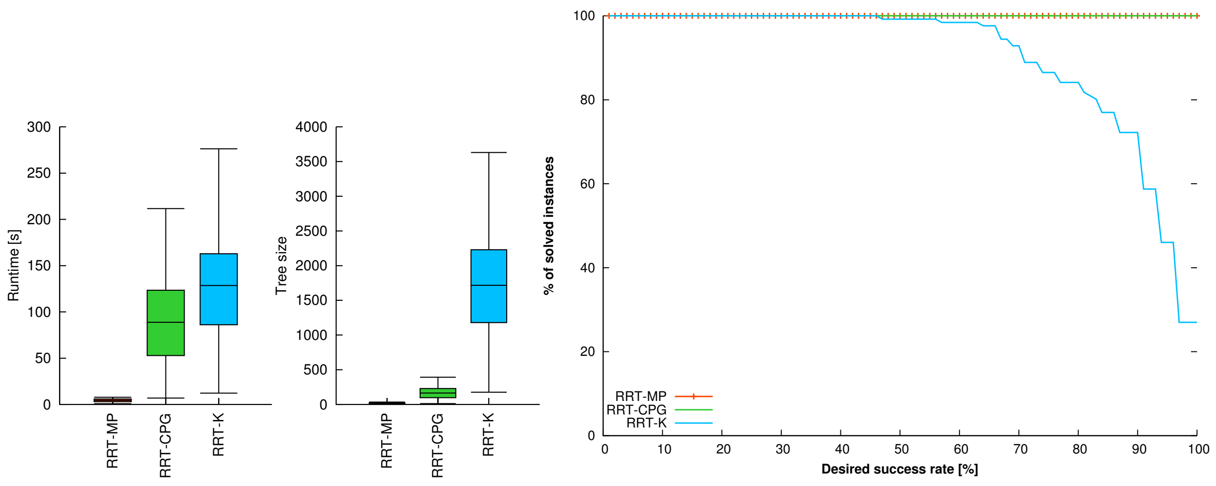


Figure 8.5: Performance of motion planning for the S-Bot robot in the Plane scenario with multiple start/goal pairs.

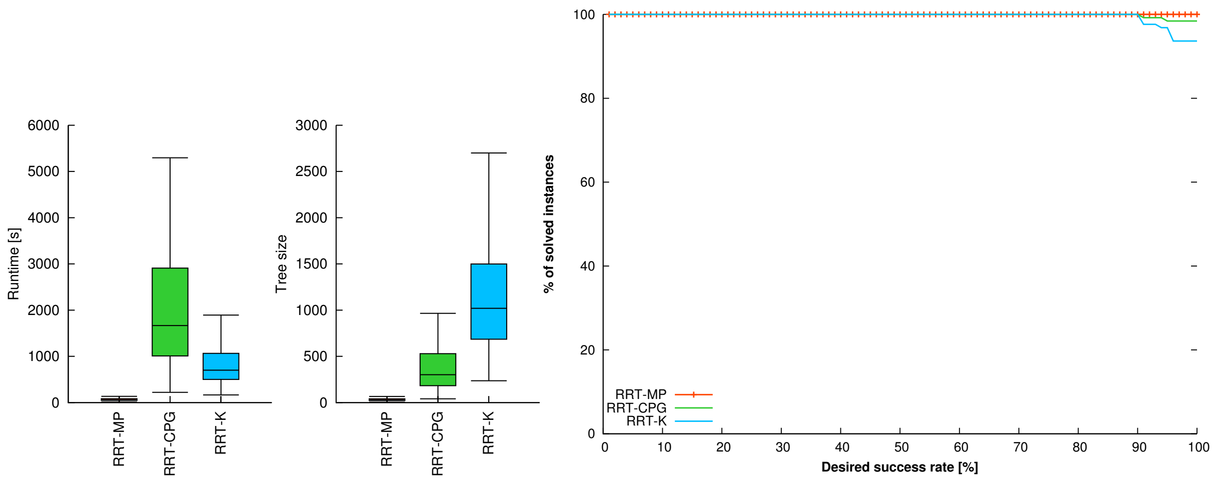


Figure 8.6: Performance of motion planning for the Lizard robot in the Plane scenario with multiple start/goal pairs.

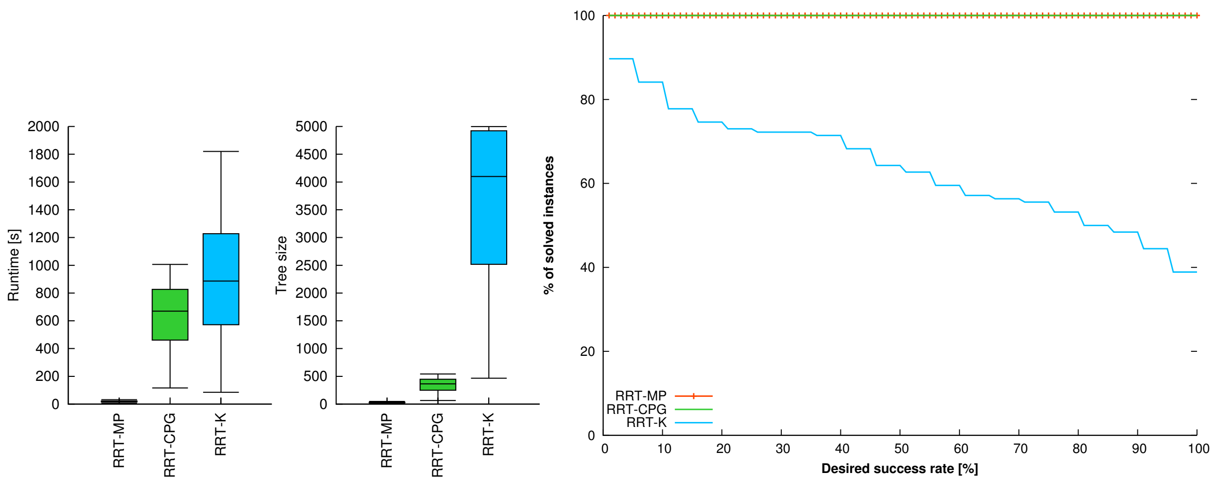


Figure 8.7: Performance of motion planning for the Quadropod robot in the Plane scenario with multiple start/goal pairs.

that require hundreds and even thousands of seconds, while RRT–MP can compute a plan in tens of seconds.

The experiments show the strength of the nearest-neighbor rule and implicit Voronoi-bias used to search the configuration space. Both RRT–K and RRT–CPG explores the configuration spaces using randomly controlled robot. Despite inefficiencies of this random control, that leads to slow and clumsy motions, the configuration tree can grow in the environment.

8.1.5 Uneven terrain with multiple start/goals

In this scenario, the robots move in an uneven terrain of size (40×25 mu). The terrain is modeled as 3D triangle mesh with $2,058$ triangles. The motion primitives used in this scenario are learned using the same approach as described in Section 8.1.1, only the fitness function is evaluated on a randomly chosen position on the terrain. The planners are tested using 100 randomly placed start/goal pairs. Each planner is run 30 times for each pair.

The progress of $s-$rate together with boxplots of runtimes and Fig. 8.10 (S-bot), in Fig. 8.8 (Lizard) and in Fig. 8.9 (Quadropod).

The planners RRT–K and RRT–CPG are able to provide solutions only for certain start/goal pairs, but the generally, they are not able to find a solution with 100 % $s-$rate for all the start/goal pairs. Contrary, RRT–MP provides solutions for all start/goal pairs with 100 % success rate for Quadropod and Lizard robots.

The motion planning with motion primitives is worse for the S-Bot robot, where other planners are able to provide solutions for more start/goal pairs with higher success rate. The S-Bot is the smallest robot and it has problems to move on the undulating surface especially if the start configuration is located in a dip. The motion primitives optimized for RRT–MP are learned on a randomly selected part of the terrain, so it can happen, that these primitives are not suitable for the whole terrain. The methods RRT–K and RRT–CPG move the S-Bot randomly, which increases chance that the robot will escape the dip.

From runtime point of view, the fastest motion planning is provided by RRT–MP, which also constructs the smallest configuration trees.

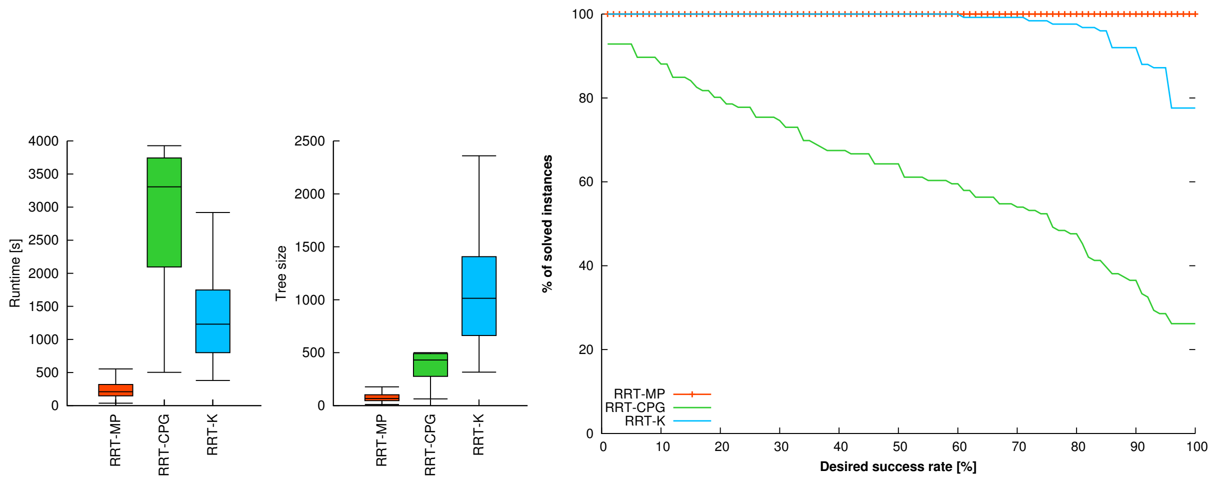


Figure 8.8: Performance of motion planning for the Lizard robot in the Surface map with multiple start/goal pairs.

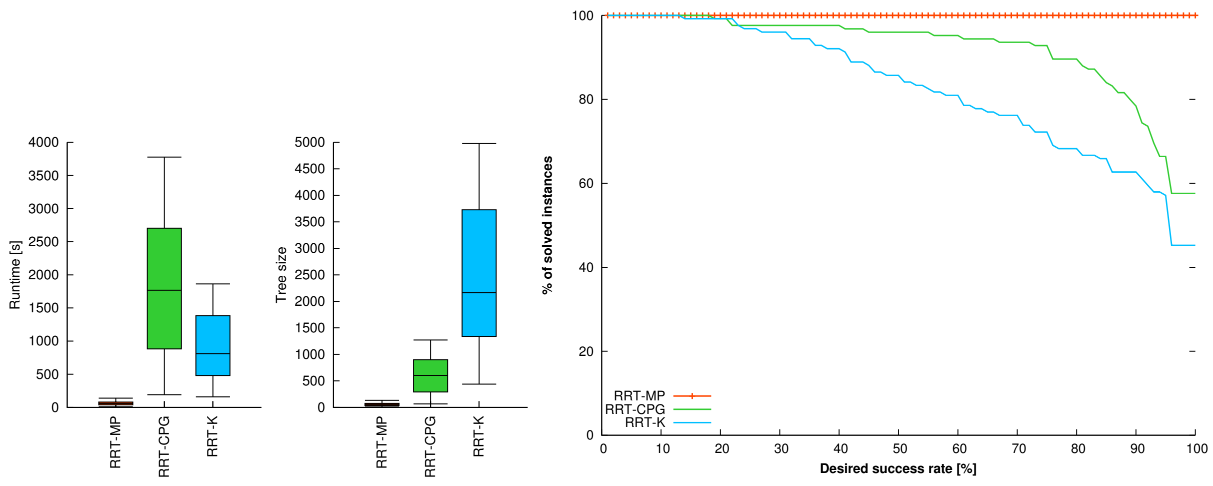


Figure 8.9: Performance of motion planning for the Quadropod robot in the Surface map with multiple start/goal pairs.

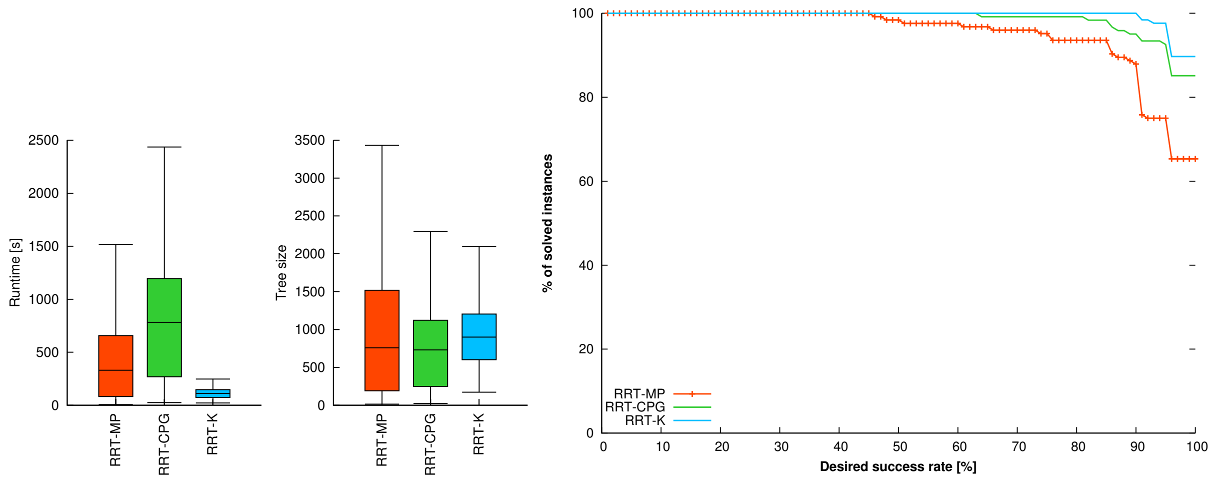


Figure 8.10: Performance of motion planning for the S-bot robot in the Surface map with multiple start/goal pairs.

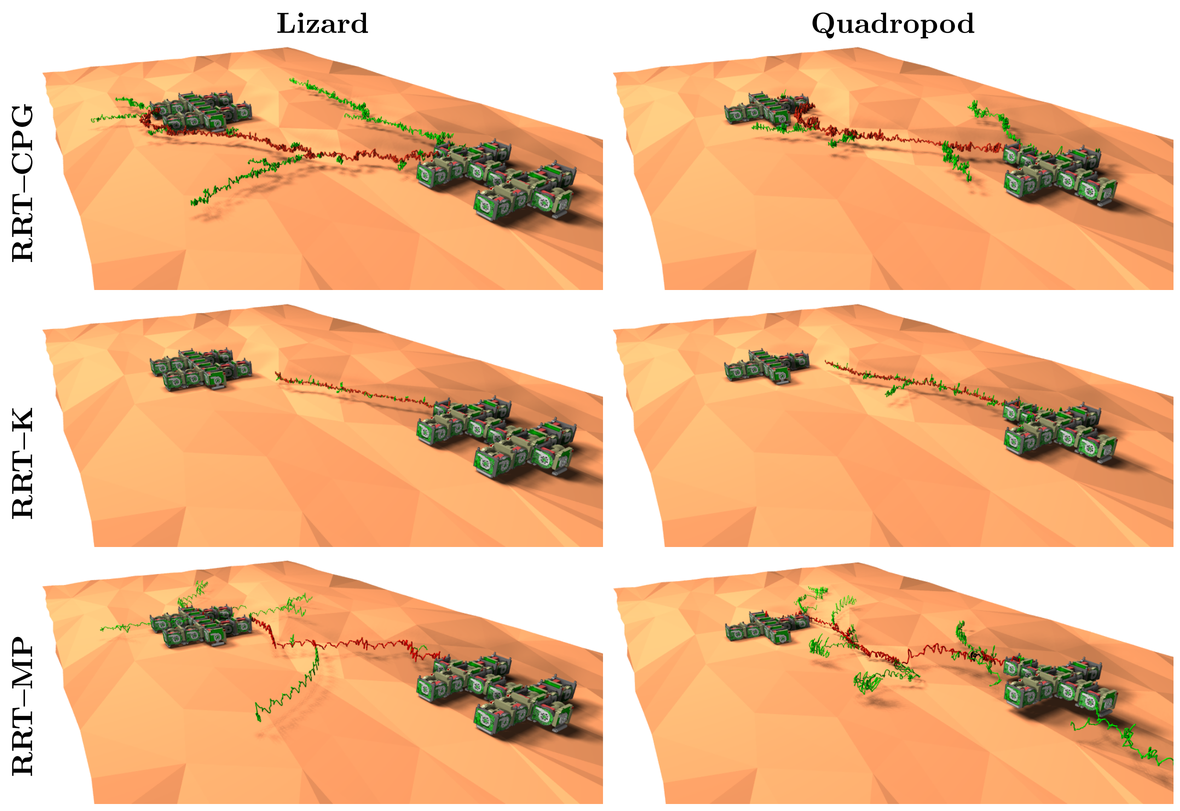


Figure 8.11: Configuration trees built for modular robots operating on terrain.

8.1.6 RRT–MP with guided sampling

The combination of motion planning with motion primitives realized by RRT–MP and guided sampling has been investigated in this experiment. The task of the planner is to find a trajectory for modular robots on uneven terrain of size 40×25 mu. Motion primitives have been optimized on the terrain using the PSO process. The guiding path is computed in the surface represented by 3D triangle mesh. The RRT–MP planner with guiding process has been run with parameters $p_{goal} = 0.6$, $d_{goal} = 2$ mu, $\alpha = 0.01$, and $R_{vg} = 3$ mu. The number of planning iterations $I_{max} = 300$.

The average results over 50 trials are depicted in Tab. 8.2. Motion planning with the guiding path outperforms the basic RRT–MP both in runtime, size of the constructed trees as well as in the quality of the resulting paths. Examples of constructed configuration trees with and without guiding paths are shown in Fig. 8.12.

8.1.7 Step scenario

In this scenario, the task is to verify the applicability of the RRT–K and RRT–MP methods in environments with obstacles. Size of the environment is 24×14 mu and it contains three steps in the middle. Size of each step is 0.7 mu. The task of the motion planning is to find a trajectory for Quadropod to overcome these steps. The influence of the vocabulary of motion primitives on the quality of the planning process is studied. Therefore, RRT–MP is tested in two variants: RRT–MP-r and RRT–MP-a. Both variants employ the same primitives: 'move-left', 'move-right', 'move-back' and they differ in the fourth primitive. The RRT–MP-r variant is equipped with a 'raise-head' primitive; while RRT–MP-a uses the 'move-forward' primitive instead. The 'raise-head' primitive can lift up the pivot module to a position necessary for climbing the step. The difference between RRT–MP-r and RRT–MP-a is depicted in Fig. 8.13. Two solutions are depicted in Fig. 8.14.

Table 8.2: Comparison of guided sampling of configuration space for modular robots with CPG-based motion patterns.

		RRT–MP		**RRT–MP + guiding path**	
		Mean	Dev	Mean	Dev
Lizard	**Run-time** [s]	832.74	0.00	188.22	104.27
	Tree size	272.00	0.00	61.04	32.03
	Trajectory length [mu]	290.02	0.00	204.05	28.72
	Success ratio	100.00 %		100.00 %	
S-Bot	**Run-time** [s]	347.80	250.96	252.86	259.03
	Tree size	267.04	181.14	174.84	166.07
	Trajectory length [mu]	272.18	54.27	271.36	64.61
	Success ratio	94.00 %		98.00 %	
Quadrop.	**Run-time** [s]	156.17	85.70	56.07	23.02
	Tree size	101.32	52.34	33.02	12.19
	Trajectory length [mu]	152.79	28.49	152.50	15.25
	Success ratio	100.00 %		100.00 %	

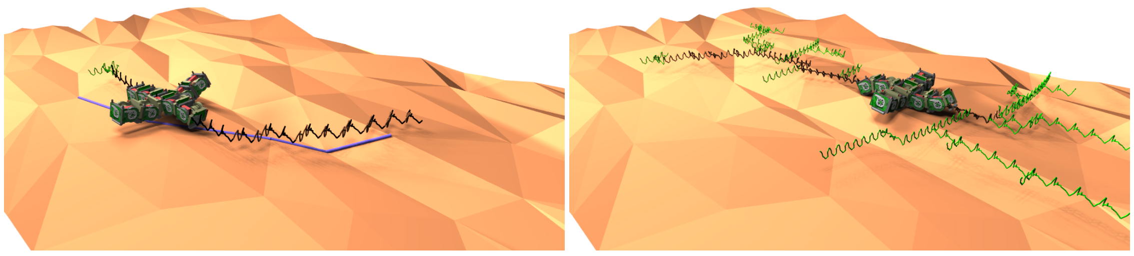
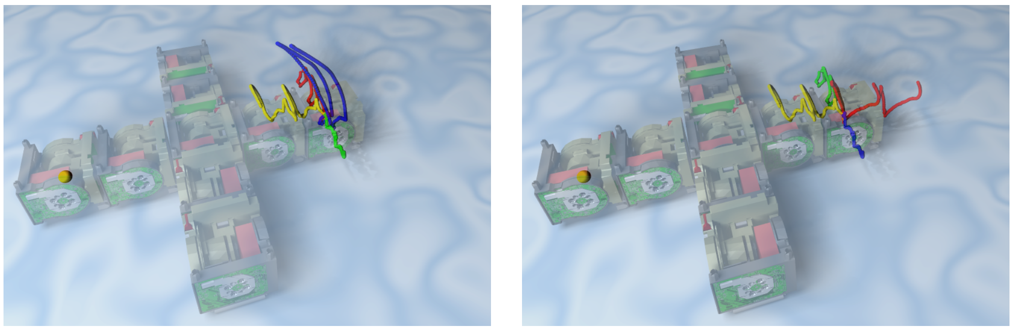

(a) RRT–MP + guiding path RRT–MP

Figure 8.12: Examples of constructed trees (green) with highlighted trajectories (red) constructed with and without the guiding path. The guiding path is depicted in blue. The robots are shown during execution of the plans. The tree constructed with the guiding path is significantly smaller than the tree constructed with basic RRT–MP.

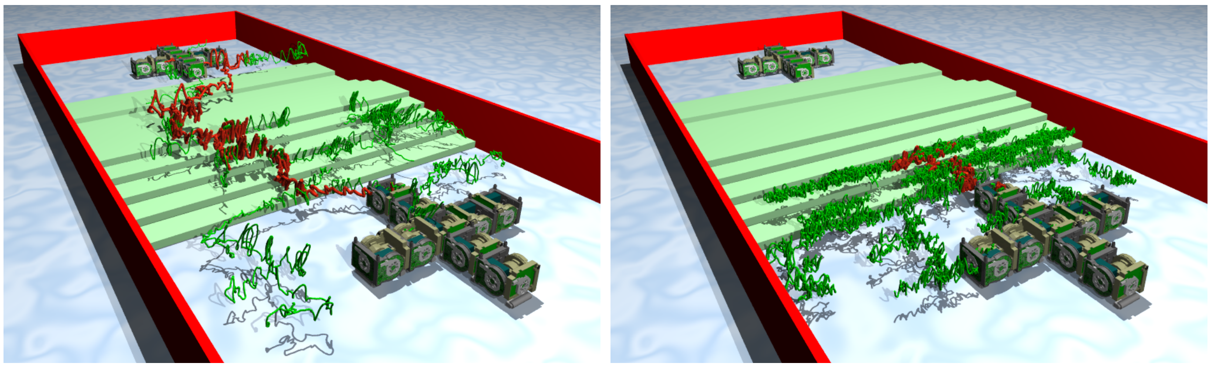


RRT–MP-r RRT–MP-a

Figure 8.13: Vocabulary of motion primitives for the Step scenario. RRT–MP-r contains 'raise-head' primitive (blue), which moves the pivot module higher in order to enter the step.

The average results from 30 trials can be found in Tab. 8.3. RRT–K and RRT–MP-r find a solution in all 30 trials, which is indicated by the 100% success ratio. Due to the large deviation of the results, the methods are compared using T-test statistics. The p-values, listed in the last column, show that the runtimes are same (under significance level $\alpha = 0.05$), however the other measures differ as the p-values are zero. We can conclude, that RRT–MP-r has better performance, because its runtime is statistically the same as the runtime of RRT–K, but it provides better trajectories than the RRT–K.

The results show that the performance of RRT–MP is determined by the capabilities of the utilized motion primitives. RRT–MP-a fails to overcome the steps, because it utilizes primitives suitable only for flat environments. By contrast, RRT–MP-r is able to reach the goal position and to overcome the steps, because it utilizes one primitive, that allows it to climb the steps. This shows the importance of using primitives suitable for a given situation.



RRT–MP-r RRT–MP-a

Figure 8.14: Example of configuration trees in the Step scenario.

Table 8.3: Results of the Step scenario. The last column is the p-value of the T-test computed between the RRT–MP-r and RRT–K columns.

		RRT–MP-r		**RRT–MP-a**		**RRT–K**		**p-val**
		Mean	Dev	Mean	Dev	Mean	Dev	
Quadropod	**Path length**	138.64	32.18	74.40	27.97	226.08	17.54	0.0000
	Iterations	624.93	517.63	3,780.00	40.00	2,985.67	640.10	0.0000
	Path time [s]	32.33	7.07	18.20	6.32	381.00	29.10	0.0000
	Runtime [s]	749.52	623.79	3,656.30	26.19	981.29	210.07	0.0662
	Success rate	100 %		0 %		100 %		

8.2 Hardware verification

The proposed RRT–MP planning has been used in the preparation of Grand Challenge scenario in Symbrion/Replicator EU projects. Before motion planning, motion primitives were optimized for several modular robots composed of CoSMO modules.

8.2.1 Optimization on HW caterpillar robots

The performance of PSO-based optimization of motion primitives has been verified with Snake$_3$ and Snake$_4$ robots, where the subscript denotes the number of modules. The locomotion is realized using sine-CPG described in Section 8.1.1. Ranges of individual parameters are same as in the simulation. All parameters $(A_i, f_i, \varphi_i), i = 1, \dots, n$ are optimized for Snake$_3$ robot, therefore 9 parameters have to be optimized.

As have been observed in many simulated experiments, the motion of snake robots is strongly influenced by the phase shift of the individual joints. Therefore, only four parameters $(\varphi_i), i = 1, \dots, 4$ have to be found in the case of Snake$_4$ and the other parameters are set according to best solution found for Snake$_3$.

The fitness function on real robots is evaluated after 25 s of motion. The position of the robot in the experimental arena is tracked using a top-view camera that detects a circular pattern [131]. The progress of the fitness functions is depicted in Fig. 8.15. The optimization for Snake$_4$ is faster than for Snake$_3$, as only four parameters are tuned for Snake$_4$. This shows the importance of selection of proper locomotion generators, that are able to perform desired motions, but have reasonable number of parameters. The dimension of the search space can be further decreased by optimizing only those parameters, that influence the behavior of the robot.

The results achieved on real robots confirm the simulated results. The optimization quickly finds desired primitives for both robots. The achieved primitives move the Snake$_3$ about ~ 20 cm and Snake$_4$ about ~ 30 cm. In the case of Snake$_3$ robot, first suitable pattern is found in 40th iteration (~ 16 minutes of optimization). The convergence is even faster in the case of Snake$_4$ robot, which achieves $f = 20$ cm in 8th iteration (~ 4 minutes). Finally, primitive with $f = 30$ cm is found in 15th iteration (~ 7 minutes).

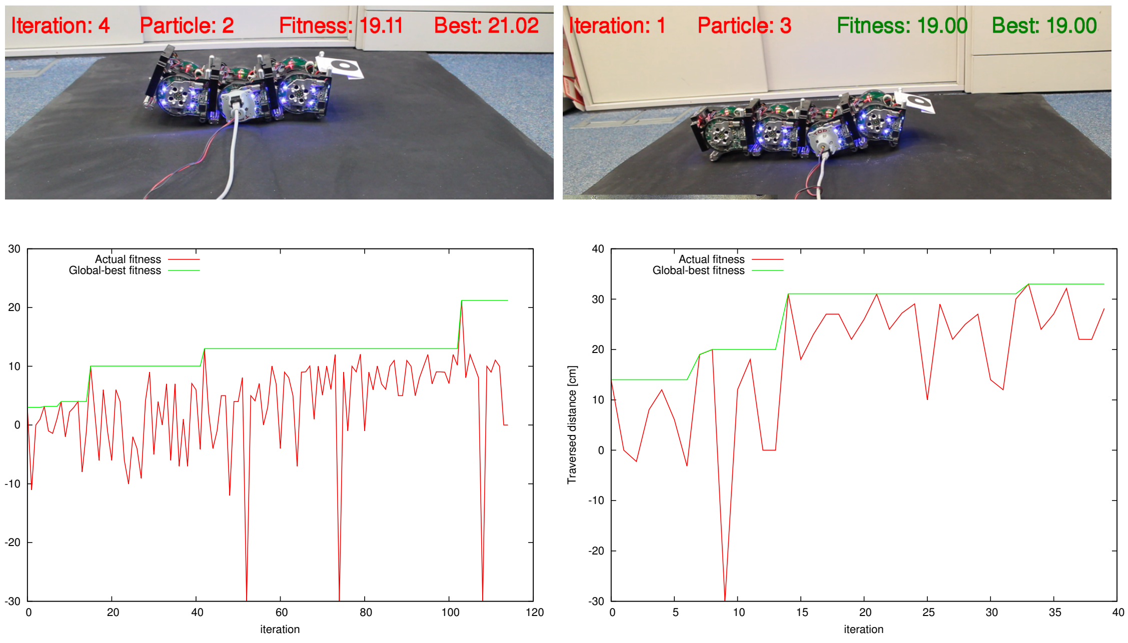
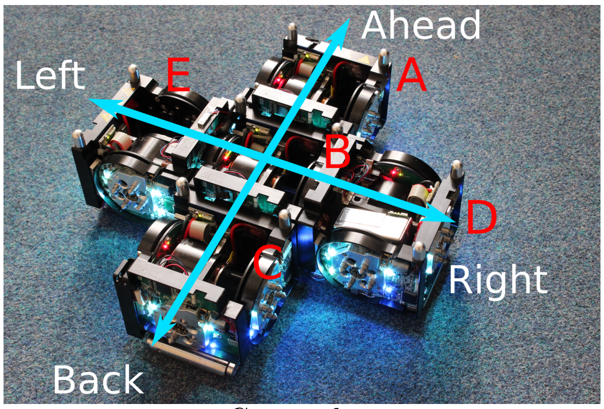


Figure 8.15: Snapshots from the experiments with real robots (top). The bottom graphs show progress of the fitness function of 'Move-forward' for Snake$_3$ (left) and Snake$_4$ (right). The fitness is computed as traveled distance after 25 s (higher is better).

8.2.2 Motion planning with Quadropod robots

Motion primitives have been also optimized for a small Cross robot made of five CoSMO modules (Fig. 8.16). The robot moves using the same four motion primitives as in the simulated scenarios (‘move-forward’,‘move-left’,‘move-right’ and ‘move-back’), that are optimized in simulation using the process described in Section 8.1.1. After the primitives are optimized in the simulation, the best primitives are further optimized on real robot. The execution of each primitive takes 30 seconds on the real robot. The control signals $a_i(t), i = 1, \dots, n$, where n is the number of modules, define the desired position of each joint. The joints are controlled by a PD regulator. The physical simulation used in the motion planner uses the same PD controller which allows us to control the robot using the motion primitives obtained from the simulation. The examples of control signals for the Cross robot are depicted in Fig. 8.18.



Cross robot

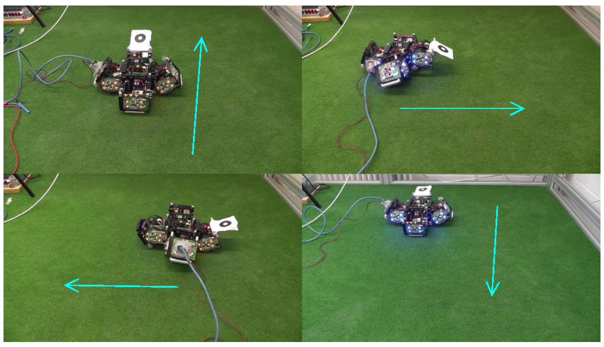
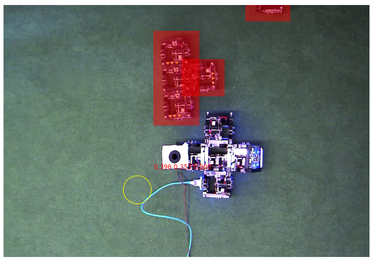
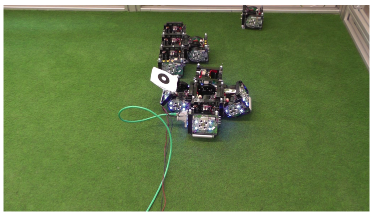

Screenshots from primitives optimization

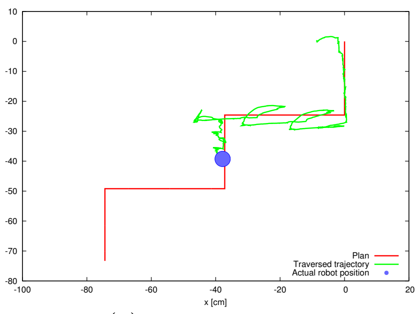
Figure 8.16: Cross robot made of five Cosmo modules.

The task of the planner is to find a path in an environment 3 × 5 meters in size with static obstacles (Fig. 8.17a). After a plan is created, the robot starts to execute it. To allow the robot to replan when it deviates from the planned trajectory, its position is determined using a camera-based localization system [132]. Screenshots from the navigation phase are shown in Fig. 8.17 and the corresponding control signals are shown in Fig. 8.18.

Planning between two fixed positions is repeated 10 times. The average planning time in this environment is 4.5 seconds, while the time needed to execute the trajectory is of the order of minutes. The planning time is thus negligible in comparison to the execution time, thus the replanning is sufficiently fast. A video from the experiment can be found at [251].


(a) top view


(b) side view



(c) motion plan

Figure 8.17: Screenshot from execution of the plan (a,b). The robot is localized using the circular pattern from the top camera. The graph (c) shows predefined plan (red) together with tracked position of the robot’s pivot module with the pattern (green).

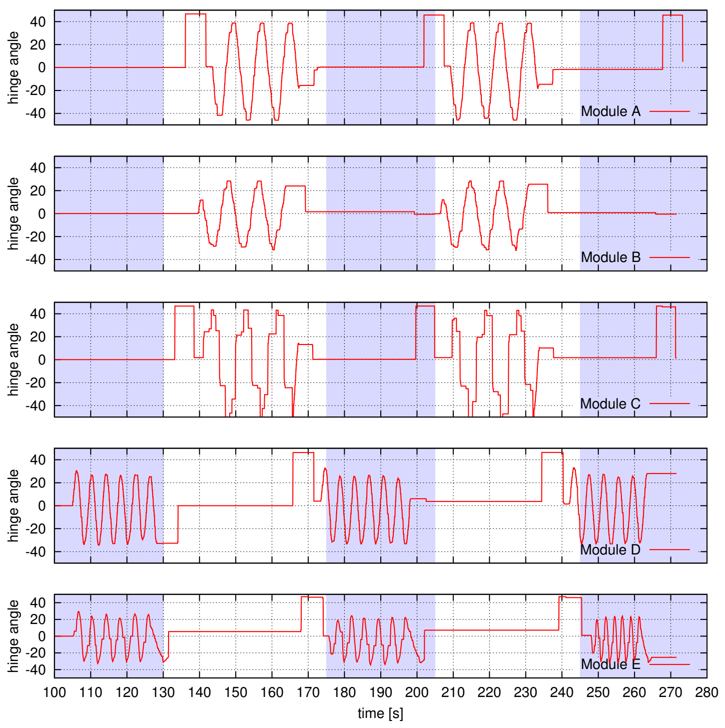


Figure 8.18: Progress of hinge angles (in degrees) on the real Cross robot. In the 'move-right' pattern, which is highlighted by the blue background, only modules D and E are used, while the remaining modules remain in zero position. The 'move-forward' primitive is achieved by modules A,B and C, while the side modules D and E are fixed in the zero position.

8.3 Discussion

The proposed motion planning system for modular robots has been verified in simulations, as well as on real robots. The simulated experiments also employed other RRT-based planner, namely RRT–CPG [284] and RRT–K [255]. The simulated experiments showed that the proposed RRT–MP motion planner significantly outperforms both RRT–CPG and RRT–K in planning time as well as in the quality of the solutions.

RRT–MP explores the configuration space only on the basis of the motion primitives, so the performance of RRT–MP is significantly influenced by the quality of the primitives. This has been demonstrated on the Step scenario, where the task is to climb the steps. RRT–MP-a equipped with unsuitable motion primitives, which provide only forward/left/right/backward motions, is not able to find any solution. However, the motion plans are found if another primitive providing 'raise-head' is added, which is utilized in the RRT–MP-r variant. This primitive is responsible for lifting up the pivot module of the robot, so it can enter the steps.

The simulated experiments have also shown the strength of the RRT motion planner itself: both methods RRT–CPG and RRT–K explore the configuration space using randomly generated control inputs. Although the robot controlled by these approaches move clumsy and inefficiently, the robot are able to move in the environment, which allows RRT planner to reach the goal position.

The motion planner was used also on HW CoSMO robots. To speed up the optimization of motion primitives, the optimization has been started in simulation to find best candidates that are further optimized on real robots.

Chapter 9

Conclusion

9.1 Conclusion

The thesis addresses the motion planning problem. Motion planning finds applications mainly in robotics, but also in other fields like surgery, CAD/CAM design, and even in computational biology. Wide range of motion planning problems can be solved using the concept of configuration space. The sampling-based motion planners that have been studied in this thesis, randomly sample the configuration space in order to build a roadmap describing free regions of the space. A path in the roadmap then corresponds to a motion in the workspace. A well known bottleneck of the sampling-based planners is the narrow passage problem, where a relatively small region of the configuration space cannot be sampled properly, which prevents to construct a path through it.

The overall goal of the thesis was to design a novel sampling-based motion planner that is able to cope with the narrow passage problem. The proposed solution, called RRT–Path is based on the widely used Rapidly Exploring Random Tree method and it uses a guiding path to help the sampling of the configuration space. The proposed guiding principle has been studied in three different scenarios, from low-dimensional motion planning of mobile robots, in the task of path planning for 3D objects in 3D workspace, and finally in the task of motion planning for many-DOF modular robots.

The main contributions of the thesis are summarized as follows.

- A novel guided strategy for Rapidly Exploring Random Tree planner was proposed. The proposed method, called RRT–Path, utilizes a guiding path to guide the sampling of the configuration space from the start configuration to the goal configuration. For the purpose of motion planning of mobile robots with three degrees of freedom, path planning methods can be used to compute the guiding path.

- The guiding principle has been extended for path planning of 3D solid objects in 3D workspaces. This problem requires searching in the six-dimensional configuration space. The proposed method, called RRT–IS, approaches the problem by relaxation of feasibility constraints, which is achieved by scaling-down the geometry of the robot. A solution found for a scaled-down robot is used as the guiding path for a larger robot. The process is repeated for increasing scales of the robot until the solution for the original problem is found.

- Finally, motion planning for modular robots has been studied. Modular robots are composed from many modules, therefore motion planning requires to work in high-dimensional configuration spaces. Motion planning for modular robots was motivated by needs of Symbrion/Replicator EU projects and the task of motion planning of whole modular robots in large environments was considered. Contrary to motion planning of mobile robot and path planning for 3D objects, motion planning for modular robots is challenging due to necessity to control motion of the whole robot using many actuators. The generation of these low-level motions is difficult as it requires cooperation of many modules and it cannot be solved on the motion planning level. To cope with this problem, we proposed to combine locomotion generators with the sampling-based motion planning techniques. The resulting planner, called RRT–MP, provides high-level plans for modular robots, while the low-level motions like 'crawl-forward' or 'climb-step' are realized using the locomotion generations like Central Pattern Generator. The RRT–MP planner tightly cooperates with a physical simulation that implements motion model of the modular robot. To decrease computational burden of RRT–MP, the number of calls of the physical simulation can be significantly decreased by utilizing the guiding principle.

All the proposed planners have been experimentally verified and compared to state-of-the-art methods. The guiding principle helps to find motion plans faster and with less amount of random samples than other tested methods. The RRT–MP planners was further utilized in Symbrion/Replicator project to derive motion plans for physical CoSMO modular robots. Besides the above mentioned algorithmic results, the following contributions have been achieved.

- A novel evaluation procedure for comparison of sampling-based planner utilizing multiple start/goal pairs was proposed. This comparison has been already used for comparing quality of motion planners for modular robots [263], and for comparing optimization of motion primitives under failures [257, 256].
- The RRT–MP planner relies on the physical simulation of modular robots. Two robotic simulators were co-developed by the author. The Robot3D simulator [271] provides physical simulation for Symbrion/Replicator modular robots and it has been used during these projects [155]. A more lightweight simulator dedicated for fast simulation on robot's internal PCs or on computational grids, called Sim [254], was developed.
- A novel optimization of CPG-based locomotion using Particle Swarm Optimization. The PSO method provides fast optimization to suitable solutions without need to tune the method itself. The optimization start in physical simulation and the best candidates can be finalized using HW robots [258]. The PSO-based optimization further allows to consider failures [262, 256, 257].
- The Hedgehog benchmark was accepted to the list of 3D planning benchmarks [1]. A visualization of our solution of the benchmark is available at [250].

9.2 Future work

The proposed guiding principle can be further improved and it can be combined with other modifications proposed for RRT. To achieve better behavior in the narrow passages, the principle of the activation radius proposed in the RRT–DD and RRT–ADD planners can be used. A faster sampling of narrow passages can be achieved by extending the tree using multiple nodes as was proposed in RRT–Retraction or RRT–Blossom methods.

The guiding principle seems to be useful especially in the path planning task for 3D solid objects. The proposed RRT–IS strategy that iteratively improves the quality of solution, is based on simple scaling of the geometries of the robots. Various types of geometries require different scaling strategies. For example, methods for object thinning can provide better relaxation of feasibility constraints for certain types of robots. Moreover, not all parts of the objects need to be scaled. For example, an alternative scaling strategy for the Hedgehog robot is to scale only the thorns, while preserving the radius of the sphere. One of the possible research directions is the design of automatic tools for a more advanced scaling process.

Motion planning for modular robots with motion primitives require operators to prepare the motion primitives based on the solved task and abilities of the robot. Due to possibility of modular robots to reconfigure into various shapes, human-designed primitives might not be achievable by robots, as human usually tend to prefer simple and nice primitives. Motion planning system can be extended by a tool to automatically derive suitable motion primitives based on the configuration of the modules and the task being solved.

Appendices

Appendix A

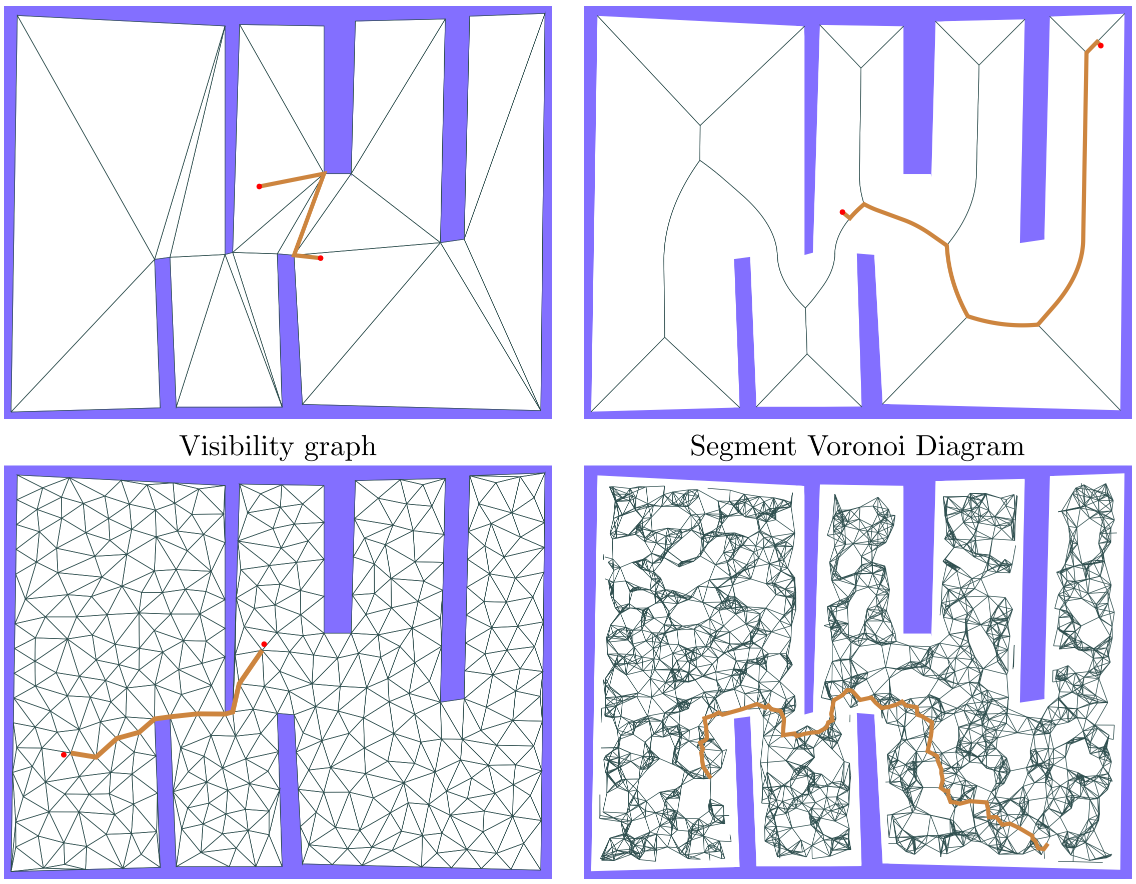
Algorithms

Algorithm 8: expandTree(q_{rand}, q_{near}): Expansion procedure of RRT-Retraction

```
Input: Random configuration $q_{rand} \in \mathcal{C}$, nearest node in the tree $q_{near} \in \mathcal{T}$, local planner
       resolution $\varepsilon$, radius $r_{ret}$ for sampling vicinity of contact configurations, number of samples
       $N_{ret}$, number of retraction steps $I_{ret}$
Output: extended tree $\mathcal{T}$
1  $S = \emptyset$;
2  if $(q_{near}, q_{rand})$ is collision-free then
3      $\mathcal{T}$.addNode($q_{rand}$);
4      $\mathcal{T}$.addEdge($q_{near}, q_{rand}$);
5  else
6      $L = (p_1, \ldots, p_m)$ = discretize line segment $(q_{near}, q_{rand})$ with resolution $\varepsilon$;
7      $q$ = first contact configuration from $L$;
8      $S = \emptyset$;
9      if $q$ is feasible then
10         $S = S \cup \{q\}$;
11     end
12     for $i = 1 : I_{ret}$ do
13         $C$=set of $N_{ret}$ random samples from on a $n$-dimensional sphere centered at $q$ with radius
           $r_{ret}$;
14         $C_c$ =set of contact configuration from $C$;
15         $q = \mathrm{argmin}_{q \in S_c} \varrho(q_{rand}, q)$;
16         if $q$ is feasible then
17             $S = S \cup \{q\}$;
18         end
19     end
20     for $q \in S$ do
21         $\mathcal{T}$.addNode($q$);
22     end
23 end
```

Appendix B

2D guiding paths


Visibility graph

Segment Voronoi Diagram

Triangle decomposition

2D Probabilistic roadmaps

Figure B.1: Examples of 2D guiding paths.

Appendix C

Example of motion plans

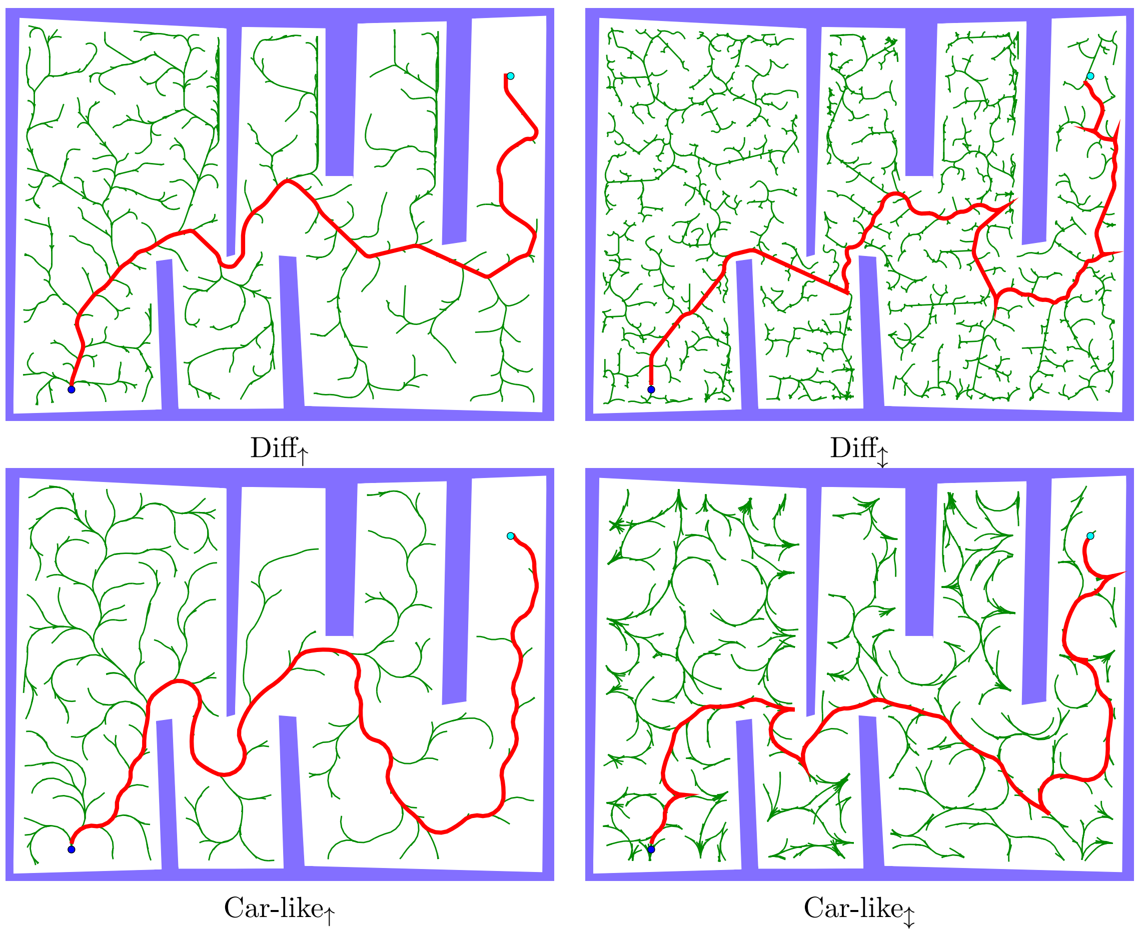

$\text{Diff}_{\uparrow}$
$\text{Diff}_{\updownarrow}$
$\text{Car-like}_{\uparrow}$
$\text{Car-like}_{\updownarrow}$

Figure C.1: Examples of motion plans constructed by RRT for the Differential drive and Car-likerobots with/without backward motions.

Appendix D

Comparison of RRT–Path with state-of-the-art methods

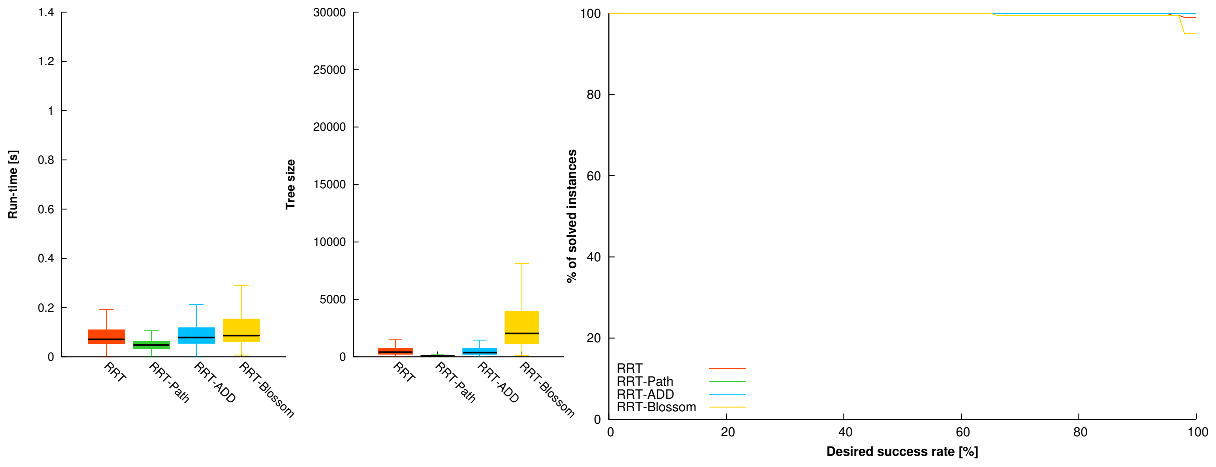


Figure D.1: Comparison of motion planning for the Diff$_{\updownarrow}$ robots in the BugTrap1 environment. Performance at 80% $s-$rate: **RRT=100 %**, **RRT-Path=100 %**, **RRT-Blossom=100 %**, **RRT-ADD=100 %**.

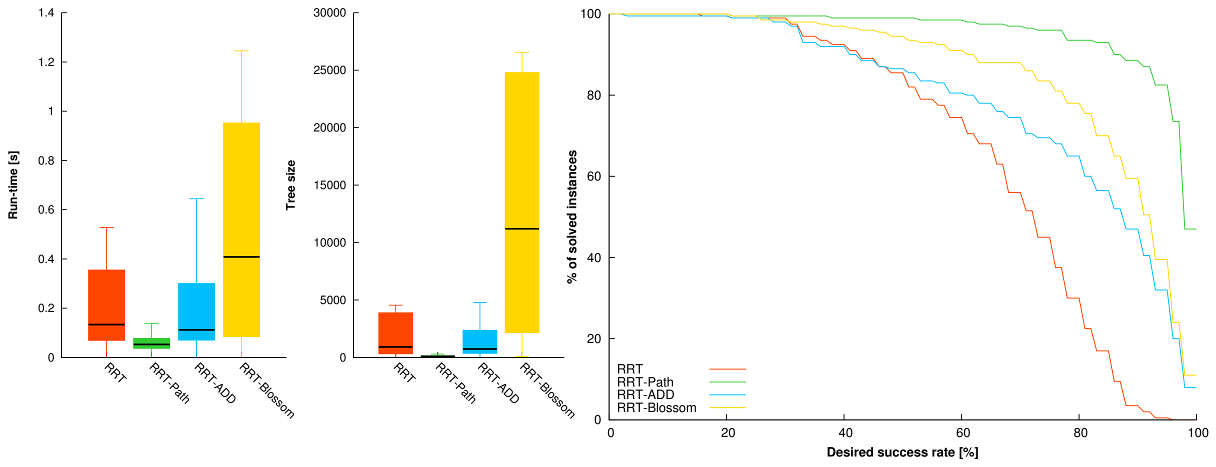


Figure D.2: Comparison of motion planning for the Diff$_{\uparrow}$ robots in the BugTrap1 environment. Performance at 80% $s-$rate: RRT=30 %, **RRT-Path=94 %**, RRT-Blossom=78 %, RRT-ADD=65 %.

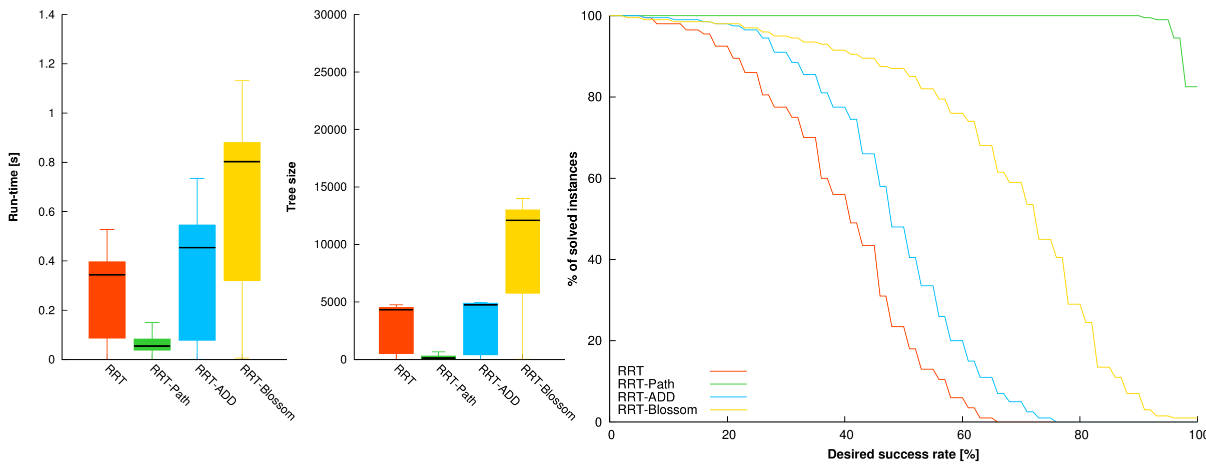


Figure D.3: Comparison of motion planning for the Car-like$_{\updownarrow}$ robots in the BugTrap1 environment. Performance at 80% $s-$rate: RRT=0 %, **RRT-Path=100 %**, RRT-Blossom=29 %, RRT-ADD=0 %.

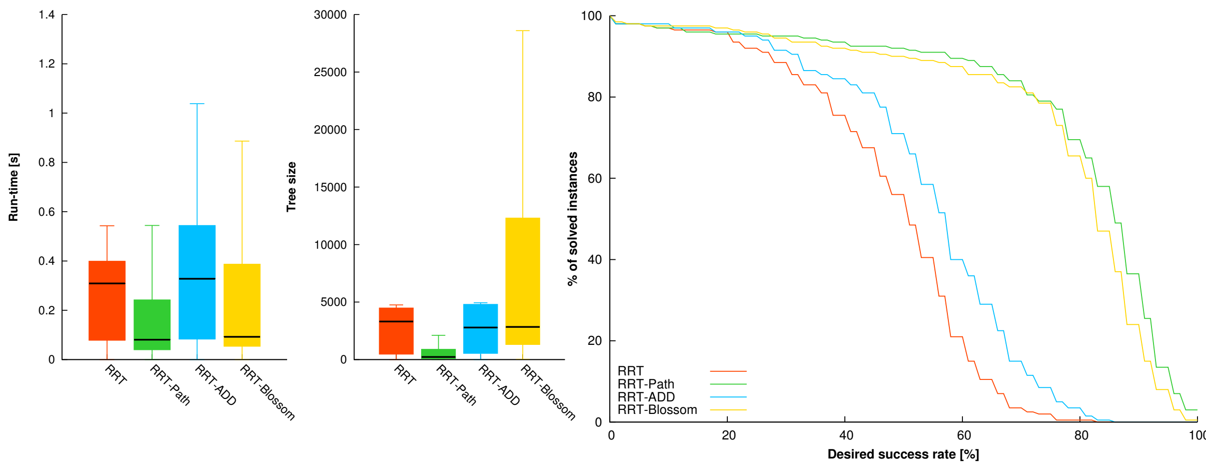


Figure D.4: Comparison of motion planning for the Car-like$_{\uparrow}$ robots in the BugTrap1 environment. Performance at 80% $s-$rate: RRT=0 %, **RRT-Path=70 %**, RRT-Blossom=66 %, RRT-ADD=4 %.

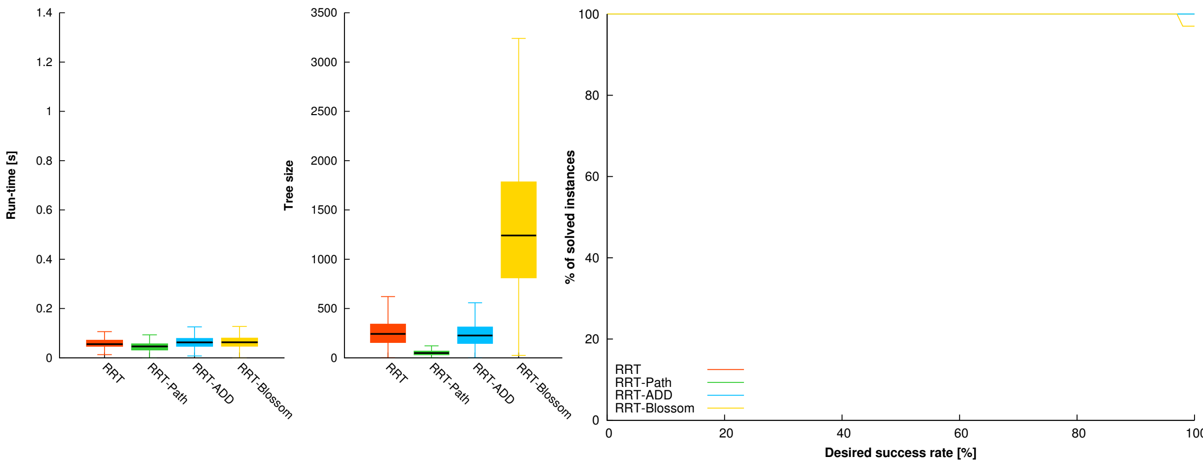


Figure D.5: Comparison of motion planning for the Diff$_{\updownarrow}$ robot in the Potholes environment. Performance at 80% $s-$rate: **RRT=100 %**, **RRT-Path=100 %**, **RRT-Blossom=100 %**, **RRT-ADD=100 %**.

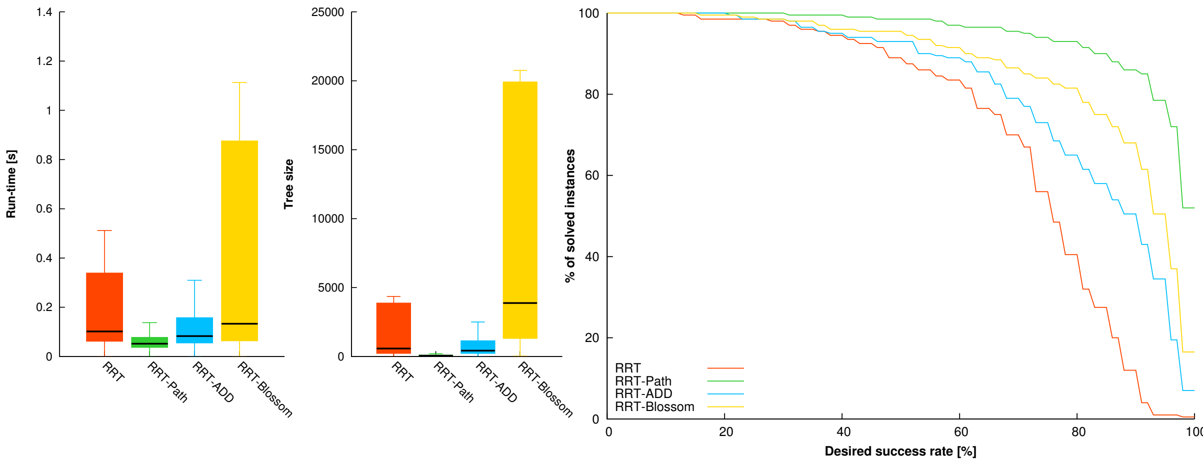


Figure D.6: Comparison of motion planning for the Diff$_{\uparrow}$ robot in the Potholes environment. Performance at 80% $s-$rate: RRT=40 %, **RRT-Path=93 %**, RRT-Blossom=82 %, RRT-ADD=65 %.

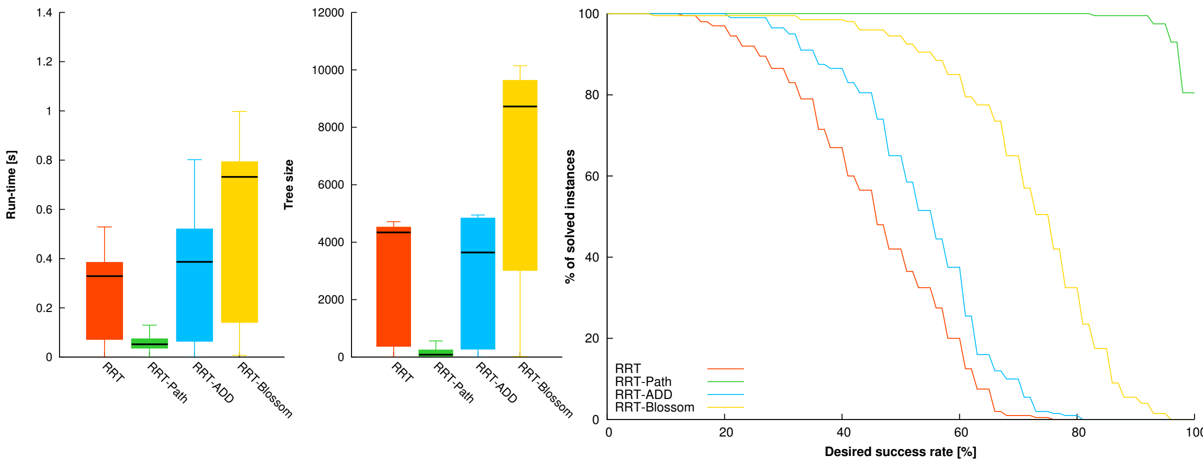


Figure D.7: Comparison of motion planning for the Car-like$_{\updownarrow}$ robot in the Potholes environment. Performance at 80% $s-$rate: RRT=0 %, **RRT-Path=100 %**, RRT-Blossom=32 %, RRT-ADD=1 %.

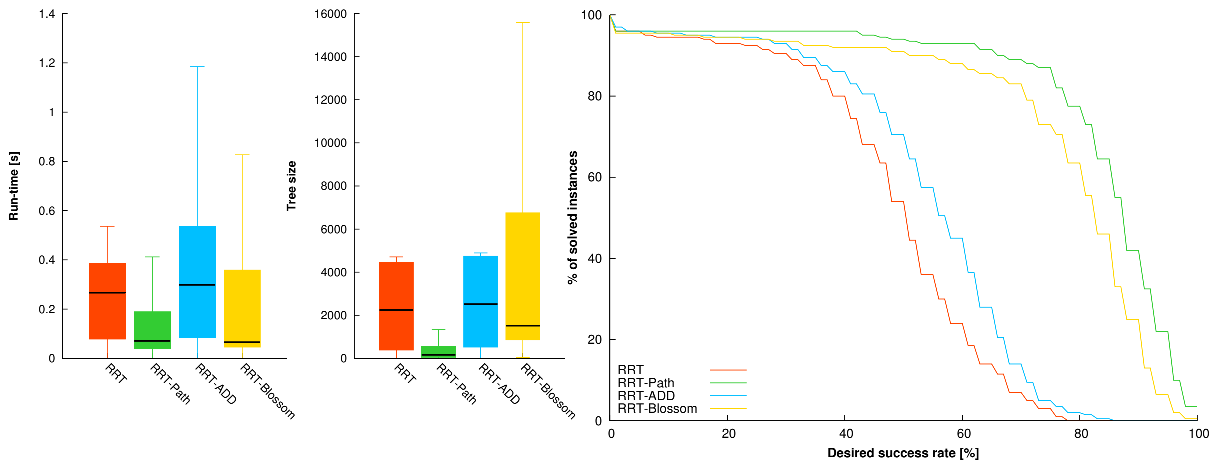


Figure D.8: Comparison of motion planning for the Car-like$_{\uparrow}$ robot in the Potholes environment. Performance at 80% $s-$rate: RRT=0 %, **RRT-Path=78 %**, RRT-Blossom=64 %, RRT-ADD=2 %.

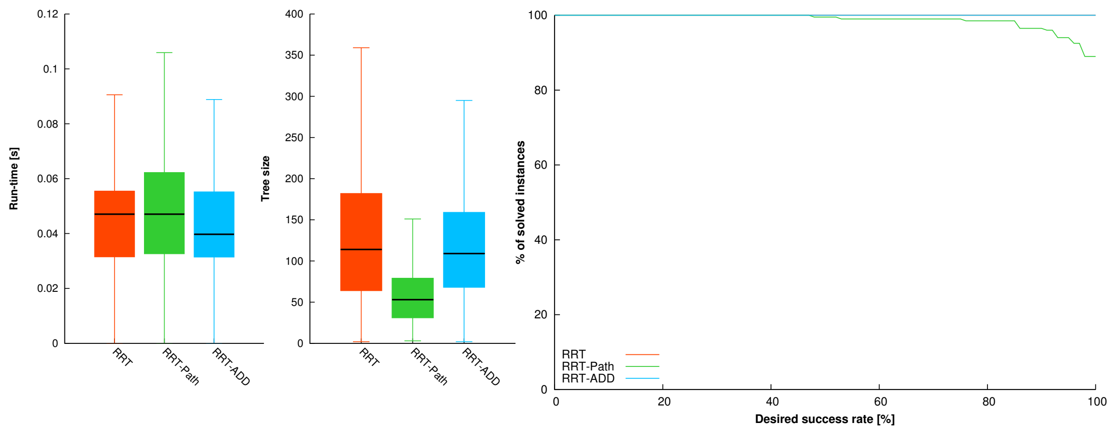


Figure D.9: Comparison of motion planning for the 2D$_{20\times50}$ robot in the Potholes environment. Performance at 80% $s-$rate: **RRT=100 %**, RRT-Path=98 %, **RRT-ADD=100 %**.

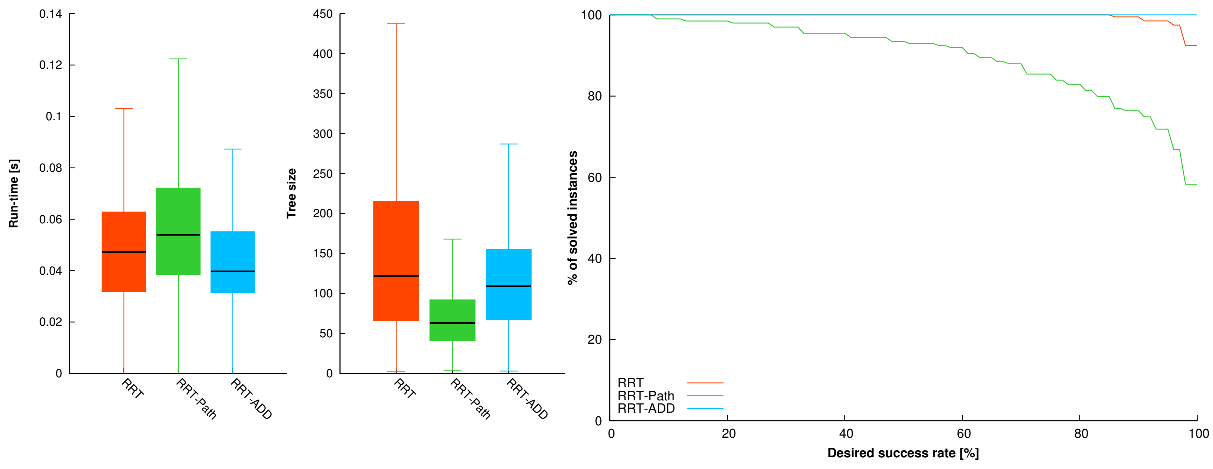


Figure D.10: Comparison of motion planning for the 2D$_{20\times100}$ robot in the Potholes environment. Performance at 80% $s-$rate: **RRT=100 %**, RRT-Path=83 %, **RRT-ADD=100 %**.

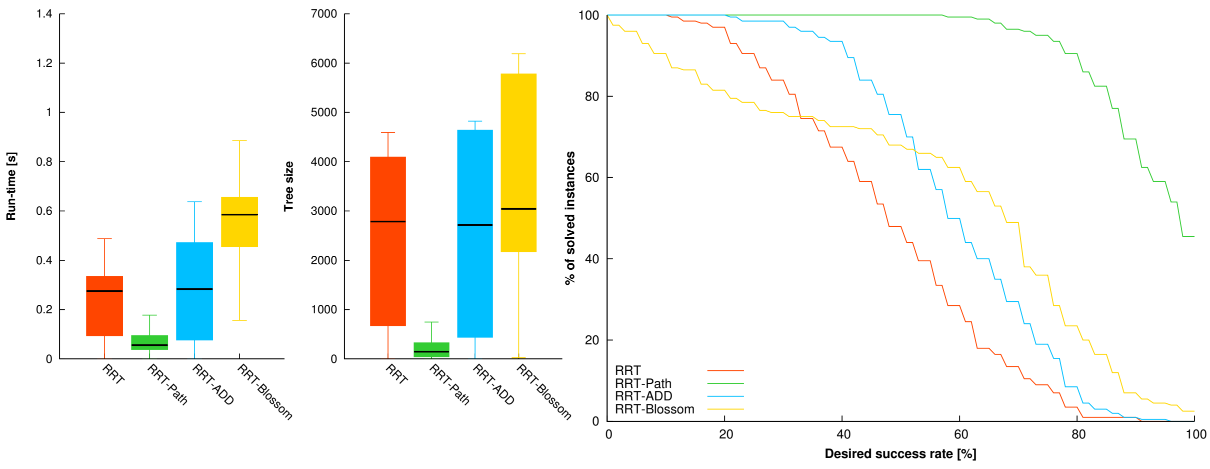


Figure D.11: Comparison of motion planning for the Car-like$_{\updownarrow}$ robot in the Simple map environment. Performance at 80% $s-$rate: RRT=4 %, **RRT-Path=90 %**, RRT-Blossom=24 %, RRT-ADD=8 %.

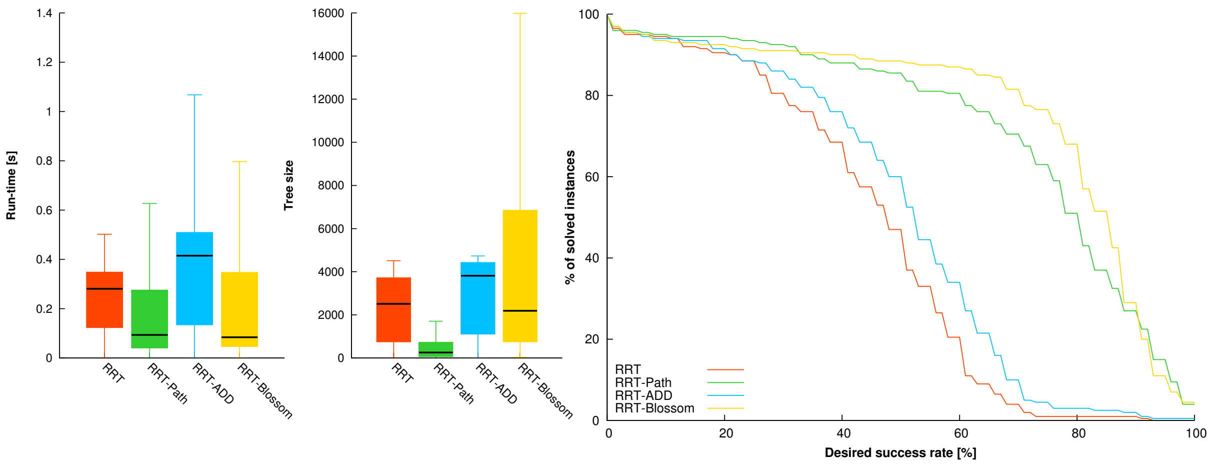


Figure D.12: Comparison of motion planning for the Car-like$_{\uparrow}$ robot in the Simple map environment. Performance at 80% $s-$rate: RRT=1 %, RRT-Path=51 %, **RRT-Blossom=68 %**, RRT-ADD=3 %.

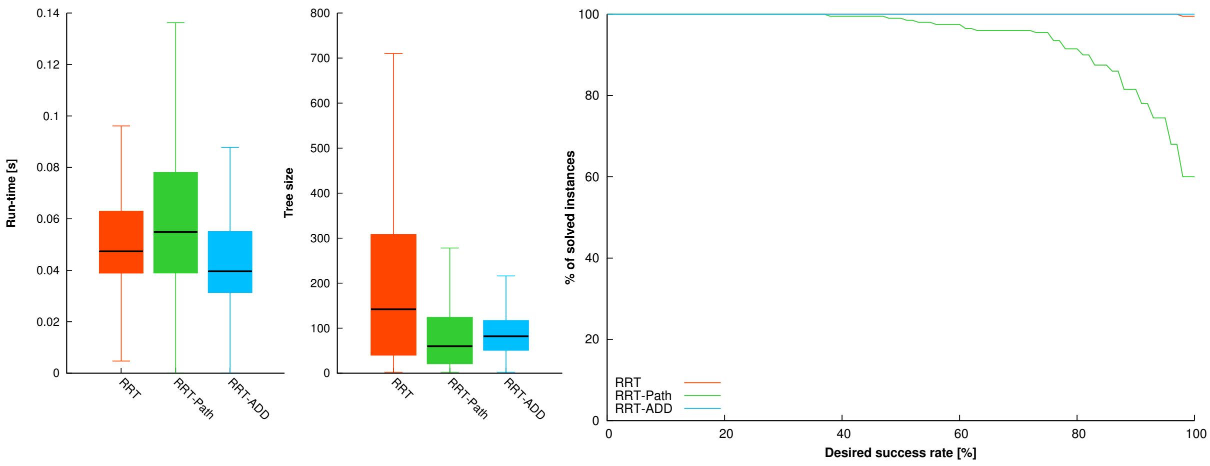


Figure D.13: Comparison of motion planning for the 2D$_{20\times50}$ robot in the Simple map environment. Performance at 80% $s-$rate: **RRT=100 %**, RRT-Path=92 %, **RRT-ADD=100 %**.

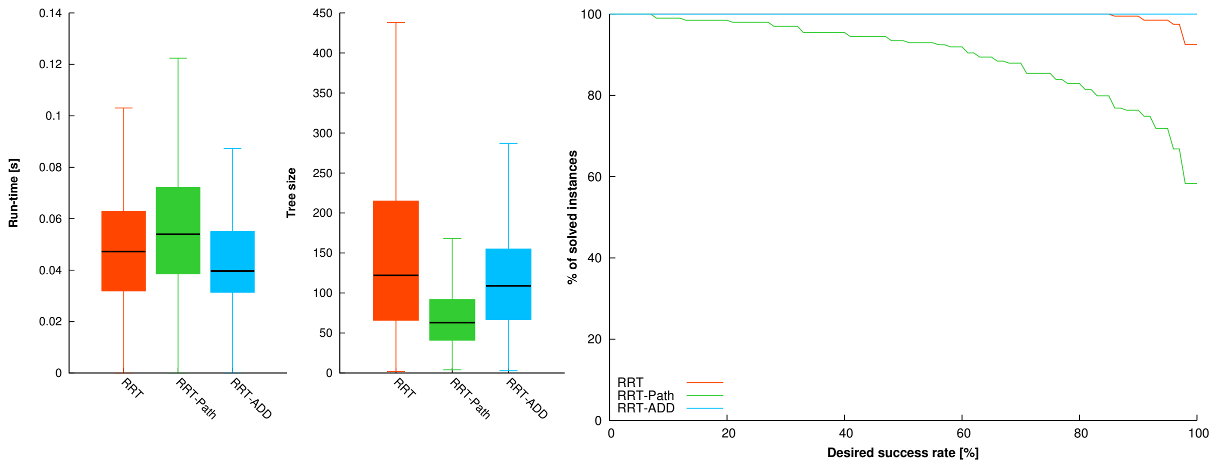


Figure D.14: Comparison of motion planning for the $2D_{20\times100}$ robot in the Simple map environment. Performance at 80% $s-$rate: **RRT=100 %**, RRT-Path=54 %, **RRT-ADD=100 %**.

Appendix E

Author's publications

Publications related to the topic of the thesis

Impacted journal papers

- Vonásek, V. - Saska, M. - Winkler, L. - Přeučil, L.: High-level Motion Planning for CPG-driven Modular Robots. *Robotics and Autonomous Systems.* 2015, vol. 68, no. 68, p. 116-128. ISSN 0921-8890. (Vonásek 50 %, Saska: 20 %, Winkler: 15 %, Přeučil: 15 %)

- Saska, M. - Vonásek, V. - Přeučil, L.: Trajectory Planning and Control for Airport Snow Sweeping by Autonomous Formations of Ploughs. Journal of Intelligent and Robotic Systems. 2013, vol. 72, no. 2, p. 239-261. ISSN 0921-0296. (Saska: 70 %, Vonásek: 15 %, Přeučil: 15 %)

- Saska, M. - Vonásek, V. - Krajník, T. - Přeučil, L.: Coordination and Navigation of Heterogeneous MAV-UGV Formations Localized by a hawk-eye-like Approach Under a Model Predictive Control Scheme. *International Journal of Robotics Research.* 2014, vol. 33, no. 10, p. 1393-1412. ISSN 0278-3649. (Saska: 70 %, others: 10 %)

- Saska, M. - Krajník, T. - Vonásek, V. - Kasl, Z. - Spurný, V. - et al.: Fault-Tolerant Formation Driving Mechanism Designed for Heterogeneous MAVs-UGVs Groups. *Journal of Intelligent and Robotic Systems.* 2014, vol. 73, no. 1-4, p. 603-622. ISSN 0921-0296. (Saska: 70 %, Přeučil: 10 %, others: 5 %)

Indexed by Web Of Sciences

- Vonásek, V. - Faigl, J. - Krajník, T. - Přeučil, L.: RRT-Path: a guided Rapidly exploring Random Tree. *In Robot Motion and Control 2009.* Heidelberg: Springer, 2009, p. 307-316. ISSN 0302-9743. ISBN 978-1-84882-984-8. (Vonásek: 50 %, Faigl: 30 %, Krajník: 10 %, Přeučil: 10 %)

- Vonásek, V. - Fišer, D. - Košnar, K. - Přeučil, L.: A Light-Weight Robot Simulator for Modular Robotics. *In Modelling and Simulation for Autonomous Systems.* Cham: Springer, 2014, p. 206-216. ISSN 0302-9743. ISBN 978-3-319-13822-0. (Vonásek: 50 %, Fiser: 25 %, Košnar: 15 %, Přeučil: 10 %)

- Vonásek, V. - Penc, O. - Přeučil, L.: Guided motion planning for modular robots. *In Modelling and Simulation for Autonomous Systems.* Cham: Springer, 2014, p. 217-230. ISSN 0302-9743. ISBN 978-3-319-13822-0. (Vonásek: 75 %, Penc: 15 %, Přeučil: 10 %)

- Vonásek, V. - Penc, O. - Košnar, K. - Přeučil, L.: Optimization of Motion Primitives for High-Level Motion Planning of Modular Robots. *In Mobile Service Robotics: CLAWAR*

2014: 17th International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines. Singapore: World Scientific, 2014, p. 109-116. ISBN 978-981-4623-34-6. (Vonásek: 50 %, Penc: 25 %, Košnar: 15 %, Přeučil: 10 %)

- Vonásek, V. - Saska, M. - Košnar, K. - Přeučil, L.: Global Motion Planning for Modular Robots with Local Motion Primitives. *In ICRA2013: Proceedings of 2013 IEEE International Conference on Robotics and Automation.* Piscataway: IEEE, 2013, ISSN 1050-4729. ISBN 978-1-4673-5641-1. (Vonásek: 35 %, Saska: 35 %, Košnar: 20 %, Přeučil: 20 %)

- Vonásek, V. - Saska, M. - Přeučil, L.: Motion Planning for a Cable Driven Parallel Multiple Manipulator Emulating a Swarm of MAVs. *In ROBOT MOTION AND CONTROL (RoMoCo)* . Pistacaway, NJ: IEEE Robotics and Automation Society, 2013, art. no. 6614577, p. 13-18. ISBN 978-1-4673-5511-7. (Vonásek: 60 %, Saska: 25 %, Přeučil: 15 %)

- Saska, M. - Krajník, T. - Vonásek, V. - Vaněk, P. - Přeučil, L.: Navigation, Localization and Stabilization of Formations of Unmanned Aerial and Ground Vehicles. *In Proceedings of 2013 International Conference on Unmanned Aircraft Systems.* New York: Springer, 2013, p. 831-840. ISBN 978-1-4799-0817-2.

- Kulich, M. - Vonásek, V. - Přeučil, L.: Simulation-Based Goal-Selection for Autonomous Exploration. *In Modelling and Simulation for Autonomous Systems.* Cham: Springer, 2014, p. 173-183. ISSN 0302-9743. ISBN 978-3-319-13822-0. (Kulich: 50 %, Vonásek: 35 %, Přeučil: 15 %)

- Saska, M. - Vonásek, V. - Krajník, T. - Přeučil, L.: Coordination and Navigation of Heterogeneous UAVs-UGVs Teams Localized by a Hawk-Eye Approach. *In Proceedings of 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems.* Piscataway: IEEE, 2012, vol. 1, p. 2166-2171. ISBN 978-1-4673-1735-1. (Saska: 50 %, Vonásek: 20 %, Krajník: 15 %, Přeučil: 15 %)

- Saska, M. - Vonásek, V. - Přeučil, L.: Roads Sweeping by Unmanned Multi-vehicle Formations. *In ICRA2011: Proceedings of 2011 IEEE International Conference on Robotics and Automation.* Madison: Omnipress, 2011, p. 631-636. ISSN 1050-4729. ISBN 978-1-61284-386-5. (Saska: 60 %, Vonásek: 30 %, Přeučil: 10 %)

- Saska, M. - Vonásek, V. - Krajník, T.: Airport snow shoveling. *In IEEE/RSJ International Conference on Intelligent Robots and Systems* , 2010, vol. 1, p. 2531-2532. ISSN 2153-0858. ISBN 978-1-4244-6675-7. (Saska: 70 %, Vonásek: 20 %, Přeučil: 10 %)

- Saska, M. - Vonásek, V. - Přeučil, L.: Control of ad-hoc formations for autonomous airport snow shoveling. *In IEEE/RSJ International Conference on Intelligent Robots and Systems* 2010, vol. 1, p. 4995-5000. ISSN 2153-0858. ISBN 978-1-4244-6675-7. (Saska: 70 %, Vonásek: 20 %, Přeučil: 10 %)

Other publications

- Vonásek, V. - Kulich, M. - Fišer, D. - Krajník, T. - Saska, M. - et al.: Techniques for Modeling Simulation Environments for Modular Robotics. *In Proccedings of International Conference on Mathematical Modelling.* Vienna: Vienna University of Technology, 2012, p. 1-6. ISBN 978-3-902823-23-6. (Vonásek: 28 %, others: 12 %)

- Vonásek, V. - Faigl, J. - Krajník, T. - Přeučil, L.: A Sampling Schema for Rapidly Exploring Random Trees Using a Guiding Path. *In Proceedings of the 5th European Conference on Mobile Robots.* AASS Research Centre, 2011, p. 201-206. (Vonásek: 50 %, Faigl: 35 %, Krajník: 10 %, Přeučil: 5 %)

- Vonásek, V. - Oertel, D. - Neumann, S. - Worn, H.: Failure Recovery for Modular Robot Movements without Reassembling Modules. *In Proceedings of 10th International Workshop on Robot Motion and Control (RoMoCo)* Piscataway: IEEE, 2015, (Vonásek: 50 %, Oertel: 20 %, Neuman: 20 %, Worn: 10 %)

- Vonásek, V. - Neumann, S. - Oertel, D. - Worn, H.: Online Motion Planning for Failure Recovery of Modular Robotic Systems. *In Proceedings of 2014 IEEE International Conference on Robotics and Automation.* Piscataway: IEEE, 2015, p. 1905-1910. ISSN 1050-4729. ISBN 978-1-4799-6923-4. (Vonásek: 50 %, Oertel: 20 %, Neuman: 20 %, Worn: 10 %)

- Saska, M. - Vonásek, V. - Báča, T. - Přeučil, L.: Ad-hoc Heterogeneous (MAV-UGV) Formations Stabilized Under a Top-View Relative Localization. *In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems.* Piscataway: IEEE, 2013, (Saska: 65 %, Vonásek: 10 %, Baca: 10 %, Přeučil: 15 %)

- Vonásek, V. - Košnar, K. - Fišer, D. - Přeučil, L.: Sim: A Light-Weight Robot Simulator for Modular Robots. *In Workshop Proceedings on Unconventional Approaches to Robotics.* Piscataway: IEEE, 2013, ISBN 978-1-4673-5642-8. (Vonásek: 30 %, Košnar: 30 %, Fiser: 25 %, Přeučil: 15 %)

- Levi, P. - Meister, E. - van Rossum, A. - Krajník, T. - Vonásek, V. - et al.: A Cognitive Architecture for Modular and Self-Reconfigurable Robots. *In Proceedings of 8th Annual IEEE Systems Conference.* Piscataway: IEEE, 2014, p. 465-472. ISSN 1944-7620. ISBN 978-1-4799-2086-0.

- Saska, M. - Chudoba, J. - Přeučil, L. - Thomas, J. - Loianno, G. - Třešňák, A. - Vonásek, V. - Kumar, V.: Autonomous Deployment of Swarms of Micro-Aerial Vehicles in Cooperative Surveillance. *In Proceedings of 2014 2014 International Conference on Unmanned Aircraft Systems (ICUAS).* Danvers: IEEE Computer society, 2014, vol. 1, art. no. 6842301, p. 584-595. ISBN 978-1-4799-2376-2. (Saska: 55 %, Chudoba: 10 %, Přeučil: 10 %, others: 5 %)

- Vonásek, V. - Neumann, S. - Winkler, L. - Košnar, K. - Woern, H. - et al.: Task-Driven Evolution of Modular Self-Reconfigurable Robots. *In From Animals to Animats 13.* Heidelberg: Springer, 2014, vol. 8575, p. 240-249. ISSN 0302-9743. ISBN 978-3-319-08863-1. (Vonásek: 50 %, Neuman: 15, Winkler: 10 %, Košnar: 10 %, Přeučil: 10 %, Woern: 10 %)

- Vonásek, V. - Winkler, L. - Liedke, J. - Saska, M. - Košnar, K. - et al.: Fast On-Board Motion Planning for Modular Robots. *In ICRA2014: Proceedings of 2014 IEEE International Conference on Robotics and Automation* Piscataway: IEEE, 2014, p. 1215-1220. ISBN 978-1-4799-3684-7. (Vonásek: 40 %, Winkler: 20 %, others: 10 %)

- Winkler, L. - Vonásek, V. - Worn, H. - Přeučil, L.: Robot3D — A Simulator for Mobile Modular Self-Reconfigurable Robots. *In Proceedings of 2012 IEEE International Conference on Multisensor Fusion and Information Integration.* Piscataway: IEEE, 2012, p. 464-469. ISBN 978-1-4673-2511-0.

- Vonásek, V. - Košnar, K. - Přeučil, L.: Motion Planning of Self-reconfigurable Modular Robots Using Rapidly Exploring Random Trees. *In Joint Proceedings of the 13th Annual TAROS Conference and the 15th Annual FIRA RoboWorld Congress.* Dordrecht: Springer, 2012, p. 279-290. ISSN 0302-9743. ISBN 978-3-642-32526-7. (Vonásek: 60 %, Košnar: 20 %, Přeučil: 20 %)

- Saska, M. - Vonásek, V. - Přeučil, L.: Formation Coordination with Path Planning in Space of Multinomials. *In Artificial Intelligence and Soft Computing 2011.* Calgary: IASTED, 2011, p. 348-355. ISBN 978-0-88986-885-4. (Saska: 60 %, Vonásek: 30 %, Přeučil: 10 %)

- Faigl, J. - Vonásek, V. - Přeučil, L.: A Multi-Goal Path Planning for Goal Regions in the Polygonal Domain. *In Proceedings of the 5th European Conference on Mobile Robots.* rebro: AASS Research Centre, 2011, p. 171-176. (Faigl: 70 %, Vonásek: 20 %, Přeučil: 10 %)

Publications not related to the topic of the thesis

Impacted journal papers:

- Saska, M. - Mejía, J.S. - Stipanovic, D.M. - Vonásek, V. - Schilling, K. - et al.: Control and Navigation in Manoeuvres of Formations of Unmanned Mobile Vehicles. European Journal of Control. 2013, vol. 19, no. 2, p. 157-171. ISSN 0947-3580. (Schiling: 16 %, Přeučil: 16 %, others: 17 %)

- Krajník, T. - Faigl, J. - Vonásek, V. - Košnar, K. - Kulich, M. - et al.: Simple, Yet Stable Bearing-Only Navigation. *Journal of Field Robotics.* 2010, vol. 27, no. 5, p. 511-533. ISSN 1556-4959. (Krajník: 50 %, Faigl: 20 %, Vonásek: 10 %, others: 5 %)

- Faigl, J. - Vonásek, V. - Přeučil, L.: Visiting Convex Regions in a Polygonal Map. *Robotics and Autonomous Systems.* 2013, vol. 61, no. 10, p. 1070-1083. ISSN 0921-8890. (Faigl: 80 %, Vonásek: 19 %, Přeučil: 1 %)

- Krajník, T. - Faigl, J. - Vonásek, V. - Szücsová, H. - Fišer, O. - et al.: A Monocular Navigation System for RoboTour Competition. *AT&P journal PLUS 2.* 2010, vol. 18, no. 1, p. 57-63. ISSN 1336-5010. (Krajník: 55 %, Faigl: 20 %, Vonásek: 10 %, others: 5 %)

Publications indexed by WOS

- Košnar, K. - Vonásek, V. - Kulich, M. - Přeučil, L.: Comparison of Shape Matching Techniques for Place Recognition. *In Proceedings of 6th European Conference on Mobile Robots.* 2013, p. 107-112. ISBN 978-1-4799-0263-7. (Košnar: 50 %, Vonásek: 20 %, Kulich 20 %, Přeučil: 10 %)

- Saska, M. - Krajník, T. - Faigl, J. - Vonásek, V. - Přeučil, L.: Low Cost MAV Platform AR-Drone in Experimental Verifications of Methods for Vision Based Autonomous Navigation. *In Proceedings of 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems.* Piscataway: IEEE, 2012, vol. 1, p. 4808-4809. ISBN 978-1-4673-1735-1. (Saska: 60 %, others: 10 %)

- Faigl, J. - Krajník, T. - Vonásek, V. - Přeučil, L.: On Localization Uncertainty in an Autonomous Inspection. *In Proceedings of 2012 IEEE International Conference on Robotic and Automation.* Piscataway: IEEE, 2012, p. 1119-1124. ISBN 978-1-4673-1405-3. (Faigl: 40 %, Krajník: 30 %, Vonásek: 20 %, Přeučil: 10 %)

- Faigl, J. - Kulich, M. - Vonásek, V. - Přeučil, L.: An Application of the Self-Organizing Map in the non-Euclidean Traveling Salesman Problem. *Neurocomputing.* 2011, vol. 74, no. 5, p. 671-679. ISSN 0925-2312. (Faigl: 60 %, Kulich: 20 %, Vonásek: 10 %, Přeučil: 10 %)

- Krajník, T. - Vonásek, V. - Fišer, D. - Faigl, J.: AR Drone as a Platform for Robotic Research and Education. *In RESEARCH AND EDUCATION IN ROBOTICS: EUROBOT*

2011, p. 172-186. ISSN 1865-0929. ISBN 978-3-642-21974-0. (Krajník: 50 %, Vonásek: 25 %, Fiser: 15 %, Faigl: 10 %)

- Fišer, O. - Szücsová, H. - Grimmer, V. - Popelka, J. - Vonásek, V. - et al.: A Mobile Robot for Small Object Handling. *In EUROBOT 2009 — International Conference on Research and Education in Robotics.* 2009, p. 47-60. ISSN 1865-0929. ISBN 978-3-642-16369-2. (Krajník: 50 %, Vonásek 20 %, Chudoba:10 %, others: 5 %)

Other publications

- Košnar, K. - Krajník, T. - Vonásek, V. - Přeučil, L.: LaMa - Large Maps Framework. *In Proceedings of Workshop on Field Robotics, Civilian-European Robot Trial 2009.* 2009, p. 9-16. ISBN 978-951-42-9176-0. (Košnar: 40 %, Krajník: 25 %, Vonásek: 25 %, Přeučil: 10 %)

- Fišer, O. - Szücsová, H. - Grimmer, V. - Popelka, J. - Vonásek, V. - et al.: A Mobile Robot for Small Object Handling. *In EUROBOT 2009 — International Conference on Research and Education in Robotics.* 2009,

- Košnar, K. - Vonásek, V. - Kulich, M. - Přeučil, L.: Combining Multiple Shape Matching Techniques with Application to Place Recognition Task. *In Computer Vision - ACCV 2014 Workshops.* 2015, ISBN 978-3-319-16627-8. (Košnar: 50 %, Vonásek: 20 %, Kulich: 20 %, Přeučil: 10 %)

- Saska, M. - Vonásek, V. - Přeučil, L.: Navigation and Formation Control Employing Complementary Virtual Leaders for Complex Maneuvers. *In 7th international Conference on Informatics in Control, Automation and Robotics.* 2010, vol. 2, p. 141-146. ISBN 978-989-8425-01-0. (Faigl: 70 %, Vonásek: 20 %, Přeučil: 10 %)

Appendix F

SCI Citations of author's work

- Vonásek, Vojtch, Jan Faigl, Tomáš Krajník, and Libor Přeučil. Rrt-patha guided rapidly exploring random tree." *In Robot Motion and Control (RoMoCo)* pp. 307-316. Springer London, 2009. **Cited: 1x**

  - Belter, Dominik, Przemyslaw Labecki, and Piotr Skrzypczynski. An exploration-based approach to terrain traversability assessment for a walking robot. *In IEEE International symposium on Safety, Security, and Rescue Robotics (SSRR)*, pp. 1-6. IEEE, 2013.

- Saska, Martin, Vojtch Vonásek, Tomáš Krajník, and Libor Přeučil. Coordination and navigation of heterogeneous UAVs-UGVs teams localized by a hawk-eye approach. *In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* pp. 2166-2171. IEEE, 2012. **Cited: 3**

  - Mathew, Neil, Stephen L. Smith, and Steven L. Waslander. Multirobot Rendezvous Planning for Recharging in Persistent Tasks. *IEEE Transactions on Robotics*, 31, no. 1 (2015): 128-142.
  - Aghaeeyan, A.; Abdollahi, F.; Talebi, H. A. UAV-UGVs cooperation: With a moving center based trajectory, *ROBOTICS AND AUTONOMOUS SYSTEMS* , Volume: 63 , Pages: 1-9, Part: 1, 2015
  - Lugo, Jacobo Jimenez; Masselli, Andreas; Zell, Andreas. Following a quadrotor with another quadrotor using onboard vision. *Conference: 6th European Conference on Mobile Robots (ECMR)*, 2013

- Vonásek, V. - Saska, M. - Přeučil, L.: Motion Planning for a Cable Driven Parallel Multiple Manipulator Emulating a Swarm of MAVs. *In ROBOT MOTION AND CONTROL (RoMoCo)* IEEE Robotics and Automation Society, 2013, art. no. 6614577, p. 13-18. ISBN 978-1-4673-5511-7. **Cited: 1x**

  - Gravish, Nick; Chen, Yufeng; Combes, Stacey A.; et al. High-throughput study of flapping wing aerodynamics for biological and robotic applications, *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* Pages: 3397-3403, 2014

- Vonásek, V. - Saska, M. - Košnar, K. - Přeučil, L.: Global Motion Planning for Modular Robots with Local Motion Primitives. *In ICRA2013: Proceedings of 2013 IEEE International Conference on Robotics and Automation.* Piscataway: IEEE, 2013, ISSN 1050-4729. ISBN 978-1-4673-5641-1. **Cited: 1x**

  - Van-Dung Hoang; Hernandez, Danilo Caceres; Hariyono, Joko; et al., Global Path Planning for Unmanned Ground Vehicle based on Road Map Images, *IEEE 7th International Conference on Human System Interactions (HSI)* Pages: 82-87, 2014

- Saska, M. - Mejía, J.S. - Stipanovic, D.M. - Vonásek, V. - Schilling, K. - et al.: Control and Navigation in Manoeuvres of Formations of Unmanned Mobile Vehicles. *European Journal of Control.* 2013, vol. 19, no. 2, p. 157-171. ISSN 0947-3580. **Cited: 3x**

  - Ramos Turci, Luiz Felipe; Ramos Simoes, Mateus Mendonca, Adaptive pinning of mobile agent network, *COMMUNICATIONS IN NONLINEAR SCIENCE AND NUMERICAL SIMULATION* Volume: 26 Issue: 1-3 Pages: 75-86, 2015
  - Franco, Carlos; Stipanovic, Dusan M.; Lopez-Nicolas, Gonzalo; et al., Persistent coverage control for a team of agents with collision avoidance, *EUROPEAN JOURNAL OF CONTROL* Volume: 22 Pages: 30-45, 2015
  - Lamburn, Darren J.; Gibbens, Peter W.; Dumble, Steven J. Efficient constrained model predictive control, *EUROPEAN JOURNAL OF CONTROL* Volume: 20 Issue: 6 Pages: 301-311, 2014

Bibliography

[1] Algorithms & applications group motion planning puzzles. set of benchmarks. http://parasol-www.cs.tamu.edu/dsmft/benchmarks/mp/.

[2] ODE — Open Dynamics Engine. `http://www.ode.org/`. Accessed: 2015-20-07.

[3] P. K. Agarwal, L. Arge, A. Danner, and B. Holland-Minkley. Cache-oblivious data structures for orthogonal range searching. In *Proceedings of the nineteenth annual symposium on Computational geometry*, pages 237–245. ACM, 2003.

[4] P. K. Agarwal, L. J. Guibas, H. Edelsbrunner, J. Erickson, M. Isard, S. Har-Peled, J. Hershberger, Ch. Jensen, L. Kavraki, P. Koehl, M. Lin, D. Manocha, D. Metaxas, B. Mirtich, D. Mount, S. Muthukrishnan, D. Pai, E. Sacks, J. Snoeyink, S. Suri, and O. Wolefson. Algorithmic issues in modeling motion. *ACM Computing Surveys*, 34(4):550–572, 2002.

[5] I. Aguinaga, D. Borro, and L. Matey. Parallel RRT-based path planinng for selective disassembly planning. *International Journal of Advanced Manufacturing Technology*, 36:1221–1233, 2008.

[6] R. Alterovitz and K. Goldberg. Planning for steerable bevel-tip needle insertion through 2D soft tissue with obstacles. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1652–1657, 2005.

[7] N. M. Amato, K. A. Dill, and G. Song. Using motion planning to map protein folding landscapes and analyze folding kinetics of known native structures. In *International Conference on Computational biology (RECOMB)*, pages 2–11, New York, NY, USA, 2002. ACM.

[8] N. M. Amato, L. K. Dale O. B. Bayazit, Ch. Jones, and D. Vallejo. OBPRM: an obstacle-based PRM for 3D workspaces. In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, pages 155–168, Natick, MA, USA, 1998. A. K. Peters, Ltd.

[9] A. Atramentov and S. M. LaValle. Efficient nearest neighbor searching for motion planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 632–637, 2002.

[10] B. Baginski. Efficient motion planning in high dimensional spaces: The parallelized Z3-method. In *International Workshop on Robotics in the Alpe-Adria-Danube Region*, pages 247–252, 1997.

[11] J. Barraquand and J.-C. Latombe. Robot motion planning: a distributed representation approach. *International Journal on Robotics Research*, 10(6):628–649, 1991.

[12] O. B. Bayazit, D. Xie, and N. M. Amato. Iterative relaxation of constraints: a framework for improving automated motion planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3433–3440, 2005.

[13] S. Berchtold and B. Glavina. A scalable optimizer for automatically generated manipulator motions. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 1796–1802, 1994.

[14] P. Bhattacharya and M. L. Gavrilova. Roadmap-based path planning using the Voronoi diagram for a clearance-based shortest path. *IEEE Robotics & Automation Magazine*, 15(2):58–66, 2008.

[15] J. Bialkowski, S. Karaman, and E. Frazzoli. Massively parallelizing the RRT and the RRT*. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3513–3518, 2011.

[16] R. Bohlin and E.E. Kavraki. Path planning using Lazy PRM. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 1, pages 521–528, 2000.

[17] R. Bohlin and L. E. Kavraki. Path planning using Lazy PRM. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 521–528, 2000.

[18] G. Bottesi, J.-P. Laumond, and S. Fleury. A motion planning based video game. Technical report, Technical Report 04576, LAASCNRS, 2004.

[19] M. S. Branicky, M. M. Curtiss, J. A. Levine, and S. B. Morgan. RRTs for nonlinear, discrete, and hybrid planning and control. In *IEEE Conference on Decision and Control*, pages 9–12, 2003.

[20] J. Bruce and M. Veloso. Real-time randomized path planning for robot navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 2383–2388, 2002.

[21] A. Brunete, M. Hernando, E. Gambao, and J. E. Torres. A behaviour-based control architecture for heterogeneous modular, multi-configurable, chained micro-robots. *Robotics and Autonomous Systems*, 60(12):1607–1624, 2012.

[22] B. Burns and O. Brock. Information theoretic construction of probabilistic roadmaps. In *IEEE/RSJ International Conference on Intelligent Robots and Systems(IROS)*, volume 1, pages 650–655, 2003.

[23] B. Burns and O. Brock. Model-based motion planning. *Computer Science Department Faculty Publication Series*, pages 1–22, 2004.

[24] B. Burns and O. Brock. Sampling-based motion planning using predictive models. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3120–3125, 2005.

[25] B. Burns and O. Brock. Single-query entropy-guided path planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2124–2129, 2005.

[26] B. Burns and O. Brock. Toward optimal configuration space sampling. In *Proceedings of Robotics: Science and Systems*, Cambridge, USA, June 2005.

[27] B. Burns and O. Brock. Single-query motion planning with utility-guided random trees. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3307–3312, 2007.

[28] D. Calisi and D. Nardi. Performance evaluation of pure-motion tasks for mobile robots with respect to world models. *Autonomous Robots*, 27(4):465–481, 2009.

[29] J. Canny. A Voronoi method for the piano-movers problem. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 530–535, 1985.

[30] J. Canny, A. Rege, and J. Reif. An exact algorithm for kinodynamic planning in the plane. In *Symposium on Computational geometry (SCG)*, pages 271–280, New York, NY, USA, 1990. ACM.

[31] J. Canny and J. Reif. New lower bound techniques for robot motion planning problems. In *Symposium on Foundations of Computer Science*, pages 49–60, 1987.

[32] S. Carpin and E. Pagello. On parallel RRTs for multi-robot systems. In *Conference of Italian Association for Artificial Intelligence*, pages 834–841, 2002.

[33] A. Casal. *Reconfiguration planning for modular self-reconfigurable robots.* PhD thesis, 2002. Adviser J.-C. Latombe.

[34] S. Caselli and M. Reggiani. ERPP: An experience-based randomized path planner. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 1002–1008, 2000.

[35] D. J. Challou, M. Gini, and V. Kumar. Parallel search algorithms for robot motion planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 46–51, 1993.

[36] H. Chang and L. Tsai-Yen. Assembly maintainability study with motion planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 3, pages 1012–1019, 1995.

[37] B. Chazelle. *Advances in Robotics 1: Algorithmic and Geometric Aspects of Robotics.* Lawrence Erlbaum Associates, 1987.

[38] P.C. Chen and Y.K. Hwang. SANDROS: a dynamic graph search algorithm for motion planning. *IEEE Transactions on Robotics and Automation*, 14(3):390–403, 1998.

[39] X. Chen and Y. Li. Smooth Path Planning of a Mobile Robot Using Stochastic Particle Swarm Optimization. In *IEEE International Conference on Mechatronics and Automation*, pages 1722–1727, 2006.

[40] P. Cheng, E. Frazzoli, and S. M. LaValle. Improving the performance of sampling-based motion planning with symmetry-based gap reduction. *IEEE Transactions on Robotics*, 24(2):488–494, 2008.

[41] P. Cheng and S. LaValle. Reducing metric sensitivity in randomized trajectory design. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 43–48, 2001.

[42] H. J. Chiel, L. H. Ting, O. Ekeberg, and M. J. Z. Hartmann. The brain in its body: motor control and sensing in a biomechanical context. *Journal of Neuroscience*, 29(41):12807–12814, 2009.

[43] G. Chirikjian and A. Pamecha. Evaluating efficiency of self-reconfiguration in a class of modular robots. *Journal of Robotic Systems*, 13(5):317–338, 1996.

[44] G. S. Chirikjian. Kinematics of a metamorphic robotic system. In *IEEE International Conference on Robotics and Automation*, pages 449–455, 1994.

[45] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations.* MIT Press, Cambridge, MA, 2005.

[46] M. Clifton, G. Paul, N. Kwok, D. Liu, and D.-L. Wang. Evaluating performance of multiple RRTs. In *IEEE/ASME International Conference on Mechtronic and Embedded Systems and Applications*, pages 564–569, 2008.

[47] J. Conradt and P. Varshavskaya. Distributed central pattern generator control for a serpentine robot. In *International Conference on Artificial Neural Networks*, 2003.

[48] R. Crespi, A. Badertscher, A. Guignard, and A. J. Ijspeert. Amphibot I: an amphibious snake-like robot. *Robotics and Autonomous Systems*, 50:163–175, 2005.

[49] I. A. Şucan and L. E. Kavraki. On the implementation of single-query sampling-based motion planners. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2005–2011, Anchorage, Alaska, 2010.

[50] M. de Berg, O. Cheong, M. Kreveld, and M. van Overmars. *Computational Geometry: Algorithms and Applications.* Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd edition, 2008.

[51] J. Denny, E. Greco, S. Thomas, and N. Amato. MARRT: Medial Axis biased Rapidly-exploring Random Trees. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[52] D. Devaurs, T. Siméon, and J. Cortés. Parallelizing RRT on distributed-memory architectures. *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2261–2266, 2011.

[53] M. Du, J. Chen, P. Zhao, H. Liang, Y. Xin, and T. Mei. An improved RRT-based motion planner for autonomous vehicle in cluttered environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4674–4679, 2014.

[54] R. C. Eberhart and Y. Shi. Particle swarm optimization: developments, applications and resources. In *Proceedings of the Congress on Evolutionary Computation*, volume 1, pages 81–86, 2001.

[55] A. Enosh, B. Raveh, O. Furman-Schueler, D. Halperin, and N. Ben-Tal. Generation, Comparison, and Merging of Pathways between Protein Conformations: Gating in K-Channels. *Biophysical Journal*, 95(8):3850–3860, 2008.

[56] C. Ericson. *Real-Time Collision Detection (The Morgan Kaufmann Series in Interactive 3-D Technology).* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[57] G. Erinc and S. Carpin. A genetic algorithm for nonholonomic motion planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1843–1849, 2007.

[58] I. Erkmen, A.M. Erkmen, F. Matsuno, R. Chatterjee, and T. Kamegawa. Snake robots to the rescue! *IEEE Robotics Automation Magazine*, 9(3):17–25, sep 2002.

[59] M. Eslami, H. Shareef, M. Khajehzadeh, and A. Mohamed. A survey of the state of the art in Particle Swarm Optimization. *Research Journal of Applied Sciences, Engineering and Technology*, 4:1181–1197, 2012.

[60] A. Ettlin and H. Bleuler. Rough-terrain robot motion planning based on obstacleness. In *International Conference on Control, Automation, Robotics and Vision*, pages 1–6, 2006.

[61] C. Armando F. de Pina, M. S. Dutra, and L. Raptopoulos. Modeling of a bipedal robot using mutually coupled Rayleigh oscillators. *Biological cybernetics*, 92(1):1–7, 2005.

[62] J. Faigl, M. Kulich, and L. Přeučil. Goal Assignment using Distance Cost in Multi-Robot Exploration. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 1, pages 3741–3746, Piscataway, 2012.

[63] J. Faigl, M. Kulich, V. Vonásek, and L. Přeučil. An Application of the Self-Organizing Map in the non-Euclidean Traveling Salesman Problem. *Neurocomputing*, 74(5):671–679, 2011.

[64] D. Ferguson, T. M Howard, and M. Likhachev. Motion planning in urban environments. *Journal of Field Robotics*, 25(11–12):939–960, 2008.

[65] D. Ferguson, N. Kalra, and A. Stentz. Replanning with RRTs. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1243–1248, 2006.

[66] R. Fitch and Z. Butler. Million module march: Scalable locomotion for large self-reconfiguring robots. *International Journal of Robotic Research*, 27(3–4):331–343, 2008.

[67] S. Fortune. A sweepline algorithm for Voronoi diagrams. In *Symposium on Computational Geometry (SCG)*, pages 313–322, 1986.

[68] M. Foskey, M. Garber, M. C. Lin, and D. Manocha. A Voronoi-based hybrid motion planner for rigid bodies. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 55–60, 2001.

[69] E. Frazzoli, M. Dahleh, and E. Feron. Real-time motion planning for agile autonomous vehicles. In *American Control Conference*, volume 1, pages 43–49. IEEE, 2001.

[70] T. Fukuda and S. Nakagawa. Dynamically reconfigurable robotic system. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1581–1586, 1988.

[71] C. Fulgenzi, C. Tay, A. Spalanzani, and C. Laugier. Probabilistic navigation in dynamic environment using Rapidly-exploring Random Trees and Gaussian processes. In *IEEE/RSJ International Conference on Intelligent Robotics and Systems (IROS)*, pages 1056–1062, 2008.

[72] I. Garcia and J. P. How. Improving the efficiency of Rapidly-exploring Random Trees using a potential function planner. In *IEEE Conference on Decision and Control and Europena Control Conference (CDC-ECC)*, pages 7965–7970, 2005.

[73] M. A. P. Garcia, O. Montiel, O. Castillo, R. Sepúlveda, and P. Melin. Path planning for autonomous mobile robot navigation with Ant Colony Optimization and fuzzy cost function evaluation. *Applied Soft Computing*, 9(3):1102–1110, 2009.

[74] R. Gayle, S. Redon, A. Sud, M. C. Lin, and D. Manocha. Efficient motion planning of highly articulated chains using physics-based sampling. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3319–3326, 2007.

[75] R. Geraerts and M. Overmars. A comparative study of probabilistic roadmap planners. In *Workshop on the algorithm foundations of robotics (WAFR)*, pages 43–57, 2002.

[76] R. Geraerts and M. H. Overmars. Clearance based path optimization for motion planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2386–2392, 2004.

[77] R. Geraerts and M H Overmars. Sampling techniques for probabilistic roadmap planners. In *International Conference on Intelligent Autonomous Systems (IAS)*, pages 600–609, 2004.

[78] R. Geraerts and M. H. Overmars. Reachability analysis of sampling based planners. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 406–412, 2005.

[79] R. Geraerts and M. H. Overmars. Creating high-quality paths for motion planning. *International Journal on Robotics Research*, 26(8):845–863, 2007.

[80] R. Geraerts and Mark H. Overmars. Reachability-based analysis for Probabilistic Roadmap planners. *Robotics and Autonomous Systems*, 55(11):824–836, 2007.

[81] B. Glavina. Solving findpath by combination of goal-directed and randomized search. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1718–1723, 1990.

[82] S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: a hierarchical structure for rapid interference detection. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 171–180, 1996. Also available at: `http://gamma.cs.unc.edu/OBB/`.

[83] R. Guernane and N. Achour. Generating optimized paths for motion planning. *Robotics and Autonomous Systems*, 59(10):789–800, 2011.

[84] L. J. Guibas, C. Holleman, and L. E. Kavraki. A probabilistic roadmap planner for flexible objects with a workspace medial-axis-based sampling approach. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 1, pages 254–259, 1999.

[85] J. Guitton, J.-L. Farges, and R. Chatila. Cell-RRT: Decomposing the environment for better plan. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5776–5781, 2009.

[86] D. Halperin. Robust geometric computing in motion. *International Journal of Robotics Research*, 21(3):219–232, 2002.

[87] C. S. Han, K. H. Law, J.-C. Latombe, and J. C. Kunz. A performance-based approach to wheelchair accessible route analysis. *Advanced Engineering Informatics*, 16(1):53–71, 2002.

[88] K. Hauser, T. Bretl, K. Harada, and J.-C. Latombe. Using motion primitives in probabilistic sample-based planning for humanoid robots. In *In Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2006.

[89] P. S. Heckbert and M. Garland. Survey of polygonal surface simplification algorithms. Technical report, Carnegie Mellon University, School of Computer Science, Pittsburgh, May 1997. Multiresolution Surface Modeling Course, SIGGRAPH.

[90] K. III Hoff, T. Culver, J. Keyser, M. Lin, and D. Manocha. Interactive motion planning using hardware-accelerated computation of generalized Voronoi diagrams. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 3, pages 2931–2937, 2000.

[91] C. Holleman and L. E. Kavraki. A framework for using the workspace medial axis in PRM planners. In *International Conference on Robotics and Automation (ICRA)*, volume 2, pages 1408–1413, 2000.

[92] T. Horsch, F. Schwarz, and H. Tolle. Motion planning with many degrees of freedom-random reflections at C-space obstacles. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3318–3323, 1994.

[93] D. Hsu. The bridge test for sampling narrow passages with probabilistic roadmap planners. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4420–4426, 2003.

[94] D. Hsu, H. Cheng, and J.-C. Latombe. Multilevel free-space dilation for sampling narrow passages in PRM planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1255–1260, 2006.

[95] D. Hsu, L. E. Kavraki, J.-C. Latombe, R. Motwani, S. Sorkin, et al. On finding narrow passages with probabilistic roadmap planners. In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, pages 141–154, 1998.

[96] D. Hsu, J.-C. Latombe, and H. Kurniawati. On the probabilistic foundations of probabilistic roadmap planning. *International Journal of Robotics Research*, 25(7):627–643, 2006.

[97] D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *International Journal of Computational Geometry and Applications*, volume 3, pages 2719–2726, 1997.

[98] D. Hsu, G. Sánchez-Ante, and Z. Sun. Hybrid PRM sampling with a cost-sensitive adaptive strategy. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3874–3880, 2005.

[99] J. Ichnowski, J. F. Prins, and R. Alterovitz. Cache-aware asymptotically-optimal sampling-based motion planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5804–5810, 2014.

[100] A. J. Ijspeert. Central pattern generators for locomotion control in animals and robots: A review. *Neural Networks*, 21(4):642–653, 2008.

[101] A. J. Ijspeert, A. Crespi, D. Ryczko, and J.-M. Cabelguen. From swimming to walking with a salamander robot driven by a spinal cord model. *Science*, 315(5817):1416–1420, 2007.

[102] A. J. Ijspeert, J. Hallam, and D. Willshaw. Evolving swimming controllers for a simulated lamprey with inspiration from neurobiology. *Adaptive Behavior*, 7(2):151–172, 1999.

[103] H. Inada and K. Ishii. Behavior generation of bipedal robot using central pattern generator (CPG) (1st report: CPG parameters searching method by genetic algorithm). In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 2179–2184, 2003.

[104] F. Islam, J. Nasir, U. Malik, Y. Ayaz, and O. Hasan. RRT-smart: Rapid convergence implementation of RRT* towards optimal solution. In *International Conference on Mechatronics and Automation (ICMA)*, pages 1651–1656, 2012.

[105] S. A. Jacobs, N. Stradford, C. Rodriguez, S. L. Thomas, and N. M. Amato. A scalable distributed RRT for motion planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5088–5095, 2013.

[106] L. Jaillet, J. Cortes, and T. Simeon. Transition-based RRT for path planning in continuous cost spaces. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2145–2150, 2008.

[107] L. Jaillet, J. Cortes, and T. Simeon. Sampling-based path planning on configuration-space costmaps. *IEEE Transactions on Robotics*, 26(4):635–646, 2010.

[108] L. Jaillet, A. Yershova, S. M. LaValle, and T. Simeon. Adaptive tuning of the sampling domain for dynamic-domain RRTs. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2851–2856, 2005.

[109] M. W. Jorgensen, E. H. Ostergaard, and H. H. Lund. Modular ATRON: modules for a self-reconfigurable robot. In *IEEE/RSJ International Conference On Intelligent Robots and Systems (IROS)*, pages 2068–2073, 2004.

[110] M. Kalisiak and M. van de Panne. Grasp-based motion planning algorithm for character animation. In *Eurographics Workshop on Computer Animation and Simulation*, 2000.

[111] M. Kalisiak and M. van de Panne. RRT-blossom: RRT with a local flood-fill behavior. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1237–1242, 2006.

[112] M. Kalisiak and M. van de Panne. Faster motion planning using learned local viability models. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2700–2705, 2007.

[113] A. Kamimura, H. Kurokawa, E. Toshida, K. Tomita, S. Murata, and S. Kokaji. Automatic locomotion pattern generation for modular robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 1, pages 714–720, 2003.

[114] A. Kamimura, H. Kurokawa, E. Yoshida, K. Tomita, S. Kokaji, and S. Murata. Distributed adaptive locomotion by a modular robotic system, M-TRAN II. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2370–2377, 2004.

[115] A. Kamphuis, M. Mooijekind, D. Nieuwenhuisen, M. H. Overmars, and Games Camera Movement Groups. Automatic construction of roadmaps for path planning in games. In *International Conference on Computer Games: Artificial Intelligence, Design and Education*, pages 285–292, 2004.

[116] S. Karaman and E. Frazzoli. Incremental sampling-based algorithms for optimal motion planning. *arXiv preprint arXiv:1005.0416*, abs/1005.0416, 2010. `http://arxiv.org/abs/1005.0416`.

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

[118] Z. Kasl, M. Saska, and L. Přeučil. Rapidly Exploring Random Trees-Based Initialization of MPC Technique Designed for Formations of MAVs. In *Proceedings of the 11th International Conference on Informatics in Control, Automation and Robotics*, volume 2, pages 436–443, 2014.

[119] L. E. Kavraki. Geometry and the discovery of new ligands. In *Algorithms for Robotic Motion and Manipulation (WAFR)*, pages 435–448, 1997.

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

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

[122] J. Kennedy and R. Eberhart. Particle swarm optimization. In *IEEE International Conference on Neural Networks*, pages 1942–1948, 1995.

[123] S. Kernbach, O. Scholz, K. Harada, S. Popesku, J. Liedke, R. Humza, W. Liu, F. Caparrelli, J. Jemai, J. Havlik, E. Meister, and P. Levi. Multi-robot organisms: State of the art. In *IEEE International Conference on Robotics and Automation (ICRA), workshop on "Modular Robots: State of the Art"*, pages 1–10, 2010.

[124] A. Khare and S. Rangnekar. A review of Particle Swarm Optimization and its applications in Solar Photovoltaic system. *Applied Soft Computing*, 13(5):2997–3006, 2013.

[125] O Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5(1):90–98, 1986.

[126] J. T. Kider, M. Henderson, M. Likhachev, and A. Safonova. High-dimensional planning on the gpu. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2515–2522, 2010.

[127] J. Kim and Joel M. Esposito. An RRT-based algorithm for testing and validating multi-robot controllers. In *Robotics: Science and Systems*, pages 249–256, 2005.

[128] H. Kimura, Y. Fukuoka, and A. H. Cohen. Adaptive dynamic walking of a quadruped robot on natural ground based on biological concepts. *International Journal of Robotic Research*, 26(5):475–490, 2007.

[129] R. Kindel, D. Hsu, J.-C. Latombe, and S. Rock. Kinodynamic motion planning amidst moving obstacles. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 1, pages 537–543, 2000.

[130] Y. Koga, K. Kondo, J. Kuffner, and J.-C. Latombe. Planning motions with intentions. In *Conference on Computer graphics and interactive techniques (SIGGRAPH)*, pages 395–408, 1994.

[131] T. Krajník, M. Nitsche, J. Faigl, P. Vaněk, M. Saska, L. Přeučil, T. Duckett, and M. Mejail. A practical multirobot localization system. *Journal of Intelligent & Robotic Systems*, 76(3-4):539–562, 2014.

[132] T. Krajník, M. Nitsche, J. Faigl, P. Vaněk, M. Saska, L. Přeučil, T. Duckett, and M. Mejail. A practical multirobot localization system. *Journal of Intelligent and Robotic Systems*, 76(3–4):539–562, 2014.

[133] J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue. Motion planning for humanoid robots. In *International Symposium on Robotic Research*, pages 365–374. Springer, 2005.

[134] J. J. Kuffner. *Autonomous agents for realtime animation.* PhD thesis, Stanford, CA, USA, 2000.

[135] J. J. Kuffner and S. M. LaValle. RRT-Connect: An efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 995–1001, 2000.

[136] M. Kulich, V. Vonásek, and L. Přeučil. Simulation-Based Goal-Selection for Autonomous Exploration. In *Modelling and Simulation for Autonomous Systems (MESAS)*, pages 173–183. Springer, 2014.

[137] Y. Kuniyoshi and S. Suzuki. Dynamic emergence and adaptation of behavior through embodiment as coupled chaotic field. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 2, pages 2042–2049, 2004.

[138] H. Kurniawati and D. Hsu. Workspace importance sampling for Probabilistic Roadmap Planning. In *International Conference on Intelligent Robots and Systems (IROS)*, volume 2, pages 1618–1623, 2004.

[139] Y. Kuwata, A. Elfes, M. Maimone, A. Howard, M. Pivtoraiko, T. M. Howard, and A. Stoica. Path planning challenges for planetary robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS): 2nd workshop on planning, perception and navigation for intelligent vehicles*, 2008.

[140] Y. Kuwata, G. A. Fiore, J. Teo, E. Frazzoli, and J. P. How. Motion planning for urban driving using RRT. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 1681–1686, 2008.

[141] Parasol Lab. Motion planning bechmark. `https://parasol.tamu.edu/dsmft/benchmarks/`. Visited at 17.7.2015.

[142] F. Lamiraux, E. Ferre, and E. Vallee. Kinodynamic motion planning: connecting exploration trees using trajectory optimization methods. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 4, pages 3987–3992, 2004.

[143] F. Lamiraux and J.-P. Laumond. On the expected complexity of random path planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 4, pages 3014–3019, 1996.

[144] F. Lamiraux, J.-P. Laumond, C. Van Geem, D. Boutonnet, and G. Raust. Trailer truck trajectory optimization: the transportation of components for the airbus A380. *IEEE Robotics Automation Magazine*, 12(1):14–21, 2005.

[145] T. Larkworthy and S. Ramamoorthy. An efficient algorithm for self-reconfiguration planning in a modular robot. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5139–5146, 2010.

[146] L. J. Latecki and R. Lakämper. Convexity rule for shape decomposition based on discrete contour evolution. *Computer Vision and Image Understanding*, 73(3):441–454, 1999.

[147] J.-C. Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Norwell, MA, USA, 1991.

[148] J.-C. Latombe. Motion planning: A journey of robots, molecules, digital actors, and other artifacts. *International Journal of Robotics Research*, 18:1119–1128, 1999.

[149] J.-P. Laumond and T. Simeon. Notes on visibility roadmaps and path planning. In *In Proceedings of the Workshop on the Algorithmic Foundations of Robotics*, pages 317–328, 2000.

[150] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. `http://coitweb.uncc.edu/~xiao/itcs6151-8151/RRT.pdf`, 1998. Technical report 98-11.

[151] S. M. LaValle. *Planning Algorithms.* Cambridge University Press, Cambridge, U.K., 2006. Available at `http://planning.cs.uiuc.edu/`.

[152] S. M. LaValle. Motion planning, part II: Wild frontiers. *IEEE Robotics Automation Magazine*, 18(2):108–118, 2011.

[153] S. M. LaValle and J. J. Kuffner. Rapidly-Exploring Random Trees: Progress and Prospects. In *Algorithmic and Computational Robotics: New Directions*, pages 293–308, 2000.

[154] J. Lee, C. Pippin, and T. Balch. Cost based planning with RRT in outdoor environments.

In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 684–689, 2008.

[155] P. Levi and S. Kernbach, editors. *Symbiotic Multi-Robot Organisms: Reliability, Adaptability, Evolution.* Springer-Verlag, 2010.

[156] P. Levi, E. Meister, A. van Rossum, T. Krajník, V. Vonásek, P. Štěpán, W. Liu, and F. Caparrelli. A Cognitive Architecture for Modular and Self-Reconfigurable Robots. In *Proceedings of 8th Annual IEEE Systems Conference*, pages 465–472, Piscataway, 2014.

[157] T.-Y. Li and Y.-C. Shie. An incremental learning approach to motion planning with roadmap management. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 4, pages 3411–3416, 2002.

[158] J. Liedke, R. Matthias, L. Winkler, and H. Woern. The collective self-reconfigurable modular organism (CoSMO). In *IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pages 1–6, 2013.

[159] Yu-Te Lin. The Gaussian PRM sampling for dynamic configuration spaces. In *International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pages 1–5, 2006.

[160] S. R. Lindemann and S. M. LaValle. Incrementally reducing dispersion by increasing voronoi bias in RRTs. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 4, pages 3251–3257, 2004.

[161] S. R. Lindemann and S. M. LaValle. Steps toward derandomizing RRTs. In *International Workshop on Robot Motion and Control (RoMoCo)*, pages 271–277, 2004.

[162] Stephen R. Lindemann and S. M. LaValle. Current issues in sampling-based motion planning. In *Robotics Research: The Eleventh International Symposium*, pages 36–54, 2005.

[163] T. Lozano-Pérez and P. A. O'Donnell. Parallel robot motion planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1000–1007, 1991.

[164] J. Mačák. Multi-Robot Cooperative Inspection Task. Master's thesis, Czech Technical University in Prague, Czech Republic, 2009.

[165] D. Marbach and A. J. Ijspeert. Online optimization of modular robot locomotion. In *IEEE International Conference on Mechatronics and Automation (ICMA)*, pages 248–253, 2005.

[166] E. Masehian and D. Sedighizadeh. Classic and heuristic approaches in robot motion planning — a chronological review. *World Academy of Science, Engineering and Technology*, 23:101–106, 2007.

[167] L. Matthey, L. Righetti, and A. J. Ijspeert. Experimental study of limit cycle and chaotic controllers for the locomotion of centipede robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1860–1865, 2008.

[168] E. Mazer, J. M. Ahuactzin, and P. Bessiere. The ariadne's clew algorithm. *Journal of Artificial Intelligence Research*, 9:295–316, 1998.

[169] M. Moll and L. E. Kavraki. Path planning for variable resolution minimal-energy curves of constant length. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2130–2135, 2005.

[170] M. Moll, D. Schwarz, and L. E. Kavraki. Roadmap methods for protein folding. In *Protein Structure Prediction*, pages 219–239. Springer, 2008.

[171] M. Morales, R. Pearce, and N. M. Amato. Analysis of the evolution of C-space models built through incremental exploration. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1029–1034, 2007.

[172] M. Morales, S. Rodriguez, and N. M. Amato. Improving the connectivity of prm roadmaps. In *International Conference on Robotics and Automation (ICRA)*, volume 3, pages 4427–4432, 2003.

[173] M. Morales, L. Tapia, R. Pearce, S. Rodriguez, and N. M. Amato. A machine learning approach for feature-sensitive motion planning. In *International Workshop on Algorithmic Foundations of Robotics (WAFR)*, pages 361–376, 2004.

[174] M. Morales, L. Tapia, R. Pearce, S. Rodriguez, and N. M. Amato. C-space subdivision and integration in feature-sensitive motion planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3114–3119, 2005.

[175] J. Morimoto, G. Endo, J. Nakanishi, S. Hyon, G. Cheng, D. Bentivegna, and C. G. Atkeson. Modulation of simple sinusoidal patterns by a coupled oscillator model for biped walking. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1579–1584, 2006.

[176] P. Moubarak and P. Ben-Tzvi. Modular and reconfigurable mobile robotics. *Robotics and Autonomous Systems*, 60(12):1648–1663, 2012.

[177] T. Mulder, J. Duysens, and Henri W.A.A Van De Crommert. Neural control of locomotion: sensory control of the central pattern generator and its relation to treadmill training. *Gait & Posture*, 7(3):251–263, 1998.

[178] S. Murata, K. Kakomura, and H. Kurokawa. Toward a scalable modular robotic system. *IEEE Robotics & Automation Magazine*, 14(4):56–63, 2007.

[179] S. Murata, E. Yoshida, A. Kamimura, H. Kurokawa, K. Tomita, and S. Kokaji. M-TRAN: Self-reconfigurable modular robotic system. *IEEE/ASME Transactions on Mechatronics*, 7(4):431–441, 2002.

[180] F. Nageotte, P. Zanne, M. de Mathelin, and C. Doignon. A circular needle path planning method for suturing in laparoscopic surgery. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 514–519, 2005.

[181] Y. Nakamura, T. Mori, M. Sato, and S. Ishii. Reinforcement learning for a biped robot based on a cpg-actor-critic method. *Neural Networks*, 20(6):723–735, 2007.

[182] L. Napalkova, J. W. Rozenblit, G. Hwang, A. J. Hamilton, and L. Suantak. An optimal motion planning method for computer-assisted surgical training. *Applied Soft Computing*, 24(0):889–899, 2014.

[183] J. Nassour, P. Hénaff, F. B. Ouezdou, and G. Cheng. A study of adaptive locomotive behaviors of a biped robot: patterns generation and classification. In *International conference on Simulation of adaptive behavior: from animals to animats (SAB)*, pages 313–324, 2010.

[184] A. L. Nelson, G. J. Barlow, and L. Doitsidis. Fitness functions in evolutionary robotics: A survey and analysis. *Robotics and Autonomous Systems*, 57(4):345–370, 2009.

[185] C. Nielsen and L. E. Kavraki. A two-level Fuzzy PRM for manipulation planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 1716–1722, 2000.

[186] D. Nieuwenhuisen and M. H. Overmars. Motion planning for camera movements. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 4, pages 3870–3876, 2004.

[187] D. Nieuwenhuisen and M.H. Overmars. Useful cycles in probabilistic roadmap graphs. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 1, pages 446–452, 2004.

[188] N. M. Nor and S. Ma. CPG-based locomotion control of a snake-like robot for obstacle avoidance. In *IEEE International Conference on Robotics ad Automation (ICRA)*, pages 347–352, 2014.

[189] C. Ó'Dúnlaing. Motion planning with inertial constraints. *Algorithmica*, 2:431–475, 1987.

[190] M. Otte and N. Correll. C-FOREST: Parallel shortest path planning with superlinear speedup. *IEEE Transactions on Robotics*, 29(3):798–806, 2013.

[191] M. H. Overmars. The Gaussian Sampling strategy for probabilistic roadmap planners. In *International Conference on Robotics and Automation (ICRA)*, pages 1018–1023, 1999.

[192] P. Pagala, M. Ferre, and M. Armada. Design of modular robot system for maintenance tasks in hazardous facilities and environments. In *ROBOT2013: First Iberian Robotics Conference*, volume 253 of *Advances in Intelligent Systems and Computing*, pages 185–197. Springer International Publishing, 2014.

[193] Prithvi Sekhar Pagala, Jos Baca, Manuel Ferre, and Rafael Aracil. Modular robot system for maintenance tasks in large scientific facilities. *International Journal of Advanced Robotic System*, 10(394), 2013.

[194] L. Palmieri and K. O. Arras. Distance metric learning for RRT-based motion planning with constant-time inference. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[195] A. Pamecha, I. Ebert-Uphoff, and G. S. Chirikjian. Useful metrics for modular robot motion planning. *IEEE Transaction on Robotics and Automation*, 13(4):531–545, 1997.

[196] J. Pan, C. Lauterbach, and D. Manocha. g-Planner: Real-time motion planning and global navigation using GPUs. In *AAAI*, 2010.

[197] J. Pan, L. Zhang, and D. Manocha. Retraction-based RRT planner for articulated models. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2529–2536, 2010.

[198] X. Pan and C. S. Han. Using motion-planning to determine the existence of an accessible route in a cad environment. *Assistive Technology*, 22:32–45, 2010.

[199] D. Parsons and J. Canny. Geometric problems in molecular biology and robotics. In *International Conference on Intelligent Systems for Molecular Biology*, pages 322–330, 1994.

[200] H. Peng, F. Su, Y. Bu, G. Zhang, and L. Shen. Cooperative area search for multiple UAVs based on RRT and decentralized receding horizon optimization. In *Asian Control Conference*, pages 298–303, 2009.

[201] J. Pettré, J.-P. Laumond, and T. Siméon. A 2-stages locomotion planner for digital actors. In *ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 258–264, 2003.

[202] E. Plaku, Kostas E. Bekris, B. Y. Chen, A. M. Ladd, and Lydia E. Kavraki. Sampling-based roadmap of trees for parallel motion planning. *IEEE Transactions on Robotics*, 21(4):597–608, 2005.

[203] E. Plaku and L. E. Kavraki. Quantitative analysis of nearest neighbors search in high-dimensional sampling-based motion planning. In *Workshop on Algorithmic Foundations of Robotics (WAFR)*, 2006.

[204] K. C. Prevas, C. Unsal, M. O. Efe, and P. K. Khosla. A hierarchical motion planning strategy for a uniform self-reconfigurable modular robotic system. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 787–792, 2002.

[205] R. Primerano, D. Wilkie, and W. C. Regli. A case study in system-level physics-based simulation of a biomimetic robot. *IEEE Transactions on Automation Science and Engineering*, 8(3):664–671, 2011.

[206] J. H. Reif. Complexity of the mover's problem and generalizations. In *Symposium on Foundations of Computer Science (SFCS)*, pages 421–427, Washington, DC, USA, 1979. IEEE Computer Society.

[207] GAMMA research group. Collision detection and proximity queries. `http://gamma.cs.unc.edu/research/collision/`. Visited at 17.7.2015.

[208] L. Righetti and A. J. Ijspeert. Programmable Central Pattern Generators: an application to biped locomotion control. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1585–1590, 2006.

[209] C. Rodriguez, J. Denny, S. A. Jacobs, S. Thomas, and N. M. Amato. Blind RRT: A probabilistically complete distributed RRT. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1758–1765, 2013.

[210] S. Rodriguez and N. M. Amato. Behavior-based evacuation planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 350–355, 2010.

[211] S. Rodriguez, X. Tang, J. Lien, and N. M. Amato. An obstacle-based Rapidly-Exploring Random Tree. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 895–900, 2006.

[212] M. Rubenstein and W.-M. Shen. Scalable self-assembly and self-repair in a collective of robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1484–1489, 2009.

[213] D. Rus and M. Vona. Crystalline robots: Self-reconfiguration with compressible unit modules. *Autonomous Robots*, 10(1):107–124, 2001.

[214] M. Saha, J.-C. Latombe, Y. Chang, and F. Prinz. Finding narrow passages with probabilistic roadmaps: The small-step retraction method. *Autonomous robots*, 19(3):301–319, 2005.

[215] B. Salemi, M. Moll, and W.-M. Shen. SUPERBOT: a deployable, multi-functional, and modular self-reconfigurable robotic system. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3636–3641, 2006.

[216] B. Salemi, W.-M. Shen, and P. Will. Hormone-controlled metamorphic robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4194–4199, 2001.

[217] G. Sanchez and J.-C. Latombe. On delaying collision checking in PRM planning – application to multi-robot coordination. *International Journal of Robotics research*, 21:5–26, 2002.

[218] M. Saska, T. Krajník, V. Vonásek, Z. Kasl, V. Spurný, and L. Přeučil. Fault-Tolerant Formation Driving Mechanism Designed for Heterogeneous MAVs-UGVs Groups. *Journal of Intelligent and Robotic Systems*, 73(1-4):603–622, January 2014.

[219] M. Saska, V. Vonásek, T. Krajník, and L. Přeučil. Coordination and Navigation

of Heterogeneous MAV-UGV Formations Localized by a hawk-eye-like Approach Under a Model Predictive Control Scheme. *International Journal of Robotics Research*, 33(10):1393–1412, September 2014.

[220] M. Saska, V. Vonásek, and L. Přeučil. Trajectory Planning and Control for Airport Snow Sweeping by Autonomous Formations of Ploughs. *Journal of Intelligent and Robotic Systems*, 72(2):239–261, November 2013.

[221] M. Saska, V. Vonásek, and L. Přeučil. Roads sweeping by unmanned multi-vehicle formations. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 631–636, 2011.

[222] W.-M. Shen, Y. Lu, and P. Will. Hormone-based control for self-reconfigurable robots. In *International Conference on Autonomous Agents*, pages 1–8. ACM, 2000.

[223] W.-M. Shen, Y. Lu, and P. Will. Hormone-based control for self-reconfigurable robots. In *International Conference on Autonomous Agents, 2000*, pages 1–8, 2000.

[224] K. Shi, J. Denny, and N. M. Amato. Spark PRM: Using RRTs within PRMs to efficiently explore narrow passages. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4659–4964, 2014.

[225] Y. Shi and R. Eberhart. A modified particle swarm optimizer. In *IEEE World Congress on Computation Intelligence*, pages 69–73, 1998.

[226] A. Shkolnik, M. Walter, and R. Tedrake. Reachability-guided sampling for planning under differential constraints. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2859–2865, 2009.

[227] T. Simeon, J.-P. Laumond, and F. Lamiraux. Move3D: A generic platform for path planning. In *IEEE International Symposium on Assembly and Task Planning*, pages 25–30, 2001.

[228] A. P. Singh, J.-C. Latombe, and D. L. Brutlag. A motion planning approach to flexible ligand binding. In *International Conference on Intelligent Systems for Molecular Biology*, pages 252–61, 1999.

[229] J. Solano and D. I. Jones. Generation of collision-free paths, a genetic approach. In *IEEE Colloquium on Genetic Algorithms for Control Systems Engineering*, pages 1–6, 1993.

[230] G. Song and N. M. Amato. A motion planning approach to folding: From paper craft to protein folding. In *IEEE International Conference Robotics Automation (ICRA)*, pages 948–953, 2001.

[231] G. Song, S. Miller, and N. M. Amato. Customizing PRM roadmaps at query time. In *IEEE International Conference Robotitcs and Automation (ICRA)*, pages 1500–1505, 2000.

[232] K. Stoy, W.-M. Shen, and P. M. Will. Using role-based control to produce locomotion in chain-type self-reconfigurable robots. *IEEE/ASME Transactions on Mechatronics*, 7(4):410–417, 2002.

[233] K. Stoy, W.-M. Shen, and P. M. Will. A simple approach to the control of locomotion in self-reconfigurable robots. *Robotics and Autonomous Systems*, 44(3):191–199, 2003.

[234] M. Strandberg. Augmenting RRT-planners with local trees. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 4, pages 3258–3262, 2004.

[235] I. Sucan and L. E. Kavraki. A sampling-based tree planner for systems with complex dynamics. *IEEE Transactions on Robotics*, 28(1):116–131, 2012.

[236] Z. Sun, D. Hsu, T. Jiang, H. Kurniawati, and J. H. Reif. Narrow passage sampling for probabilistic roadmap planning. *IEEE Transactions on Robotics*, 21(6):1105–1115, 2005.

[237] S. Sundaram, I. Remmler, and N. M. Amato. Disassembly sequencing using a motion planning approach. In *IEEE International Conference Robotics and Automation (ICRA)*, pages 1475–1480, 2001.

[238] E. Szadeczky-Kardoss and B. Kiss. Extension of the Rapidly Exploring Random Tree algorithm with key configurations for nonholonomic motion planning. In *IEEE International Conference on Mechatronics*, pages 363–368, 2006.

[239] G. Taga, Y. Yamaguchi, and H. Shimizu. Self-organized control of bipedal locomotion by neural oscillators in unpredictable environment. *Biological Cybernetics*, 65(3):147–159, July 1991.

[240] L. Tapia, S. Thomas, B. Boyd, and N. M. Amato. An unsupervised adaptive strategy for constructing probabilistic roadmaps. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4037–4044, 2009.

[241] R. Z. Tombropoulos, J. R. Adler, and J.-C. Latombe. CARABEAMER: A treatment planner for a robotic radiosurgical system with general kinematics. *Medical Image Analysis*, 3:3–3, 1998.

[242] K. Tomita, S. Murata, H. Kurokawa, E. Yoshida, and S. Kokaji. Self-assembly and self-repair method for a distributed mechanical system. *IEEE Transactions on Robotics and Automation (ICRA)*, 15(6):1035–1045, 1999.

[243] K. I. Tsianos and L. E. Kavraki. Replanning: A powerful planning strategy for hard kinodynamic problems. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1667–1672, 2008.

[244] C. Ünsal, H. Kilite, and P. K. Khosla. A modular self-reconfigurable bipartite robotic system: Implementation and motion planning. *Autonomous Robots*, 10:23–40, 2000.

[245] C. Urmson and R. Simmons. Approaches for heuristically biasing RRT growth. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 2, pages 1178–1183, 2003.

[246] D. R. Vallejo, C. Jones, and N. M. Amato. An adaptive framework for 'single shot' motion planning. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 1722–1727, 2000.

[247] J. P. van den Berg and M. H. Overmars. Using workspace information as a guide to non-uniform sampling in probabilistic roadmap planners. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 1, pages 453–460, 2004.

[248] C. Van Geem, T. Simeon, J.-P. Laumond, J.-L. Bouchet, and J. F. Rit. Mobility analysis for feasibility studies in cad models of industrial environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 3, pages 1770–1775, 1999.

[249] V. Vonásek. 3D Bugtrap benchmark problem: solution. `https://www.youtube.com/watch?v=qci_AktcrD4`. Accessed: 2015-20-07.

[250] V. Vonásek. Hedgehog in the cage problem: solution. `https://www.youtube.com/watch?v=BqD77OXVOmo`. Accessed: 2015-20-07.

[251] V. Vonásek. Video from HW verification on CoSMO modular robot. `http://www.youtube.com/watch?v=fCy3grSRC9k`, 2014. Visited: 17.7.2015.

[252] V. Vonásek, J. Faigl, T. Krajník, and L. Přeučil. RRT-path — a guided rapidly exploring random tree. In *Robot Motion and Control (RoMoCo)*, pages 307–316. Springer, 2009.

[253] V. Vonásek, J. Faigl, T. Krajník, and L. Přeučil. A sampling schema for rapidly exploring random trees using a guiding path. In *European Conference on Mobile Robots (ECMR)*, pages 201–206, 2011.

[254] V. Vonásek, D. Fišer, K. Košnar, and L. Přeučil. A Light-Weight Robot Simulator for Modular Robotics. In *Modelling and Simulation for Autonomous Systems (MESAS)*, pages 206–216. Springer, 2014.

[255] V. Vonásek, K. Košnar, and L. Přeučil. Motion planning of self-reconfigurable modular robots using rapidly exploring random trees. In *Advances in Autonomous Robotics — Joint Proceedings of the 13th Annual TAROS Conference and the 15th Annual FIRA RoboWorld Congress*, pages 279–290, 2012.

[256] V. Vonásek, S. Neumann, D. Oertel, and H. Wörn. Online Motion Planning for Failure Recovery of Modular Robotic Systems. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1905–1910, 2015.

[257] V. Vonásek, D. Oertel, S. Neumann, and H. Wörn. Failure recovery for modular robot movements without reassembling modules. In *10th International Workshop on Robot Motion and Control (RoMoCo)*, Poznan, Poland, 2015.

[258] V. Vonásek, O. Penc, K. Košnar, and L. Přeučil. Optimization of Motion Primitives for High-Level Motion Planning of Modular Robots. In *Mobile Service Robotics: CLAWAR 2014: 17th International Conference on Climbing and Walking Robots and the Support Technologies*, pages 109–116, Singapore, 2014. World Scientific.

[259] V. Vonásek, O. Penc, and L. Přeučil. Guided motion planning for modular robots. In *Modelling and Simulation for Autonomous Systems (MESAS)*, pages 217–230. Springer, 2014.

[260] V. Vonásek, M. Saska, K. Košnar, and L. Přeučil. Global Motion Planning for Modular Robots with Local Motion Primitives. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2465–2470, Piscataway, 2013.

[261] V. Vonásek, M. Saska, K. Košnar, and L. Přeučil. Global motion planning for modular robots with local motion primitives. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

[262] V. Vonásek, M. Saska, K. Košnar, and L. Přeučil. Motion planning with adaptive motion primitives for modular robots. *Applied Soft Computing*, 34:678–692, 2015.

[263] V. Vonásek, M. Saska, L. Winkler, and L. Přeučil. High-level motion planning for CPG-driven modular robots. *Robotics and Autonomous Systems*, 68(0):116–128, 2015.

[264] V. Vonásek, L. Winkler, J. Liedke, M. Saska, K. Košnar, and L. Přeučil. Fast On-Board Motion Planning for Modular Robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1215–1220, Piscataway, 2014.

[265] J. E. Walter, E. M. Tsai, and N. M. Amato. Algorithms for fast concurrent reconfiguration of hexagonal metamorphic robots. *IEEE Transactions on Robotics*, 21(4):621–631, 2005.

[266] W. Wang and Y. Li. A multi-RRTs framework for robot path planning in high-dimensional configuration space with narrow passages. In *International Conference on Mechatronics and Automation*, pages 4952–4957, 2009.

[267] W. Wang, Y. Li, X. Xu, and S. X. Yang. An adaptive roadmap guided Multi-RRTs strategy for single query path planning. pages 2871–2876, 2010.

[268] R. Wein. The Visibility-Voronoi complex and its applications. In *ACM Symposium Computational Geometry*, pages 63–72, 2005.

[269] M. P. Weller, M. E. Karagozler, B. Kirby, J. Campbell, and S. S. Goldstein. Movement primitives for an orthogonal prismatic closed-lattice-constrained self-reconfiguring module. In *Workshop on Self-Reconfiguring Modular Robotics at the IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2007.

[270] S. A. Wilmarth, N. M. Amato, and P. F. Stiller. MAPRM: A probabilistic roadmap planner with sampling on the medial axis of the free space. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1024–1031, 1999.

[271] L. Winkler, V. Vonásek, H. Worn, and L. Přeučil. Robot3D — a simulator for mobile modular self-reconfigurable robots. In *IEEE International Conference on Multisensor Fusion and Information Integration*, pages 464–469, 2012.

[272] P. R. Wurman, R. D'Andrea, and M. Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI magazine*, 29(1):9, 2008.

[273] M. Wzorek and P. Doherty. Reconfigurable path planning for an autonomous unmanned aerial vehicle. In *International Conference on Hybrid Information Technology (ICHIT)*, volume 2, pages 242–249, 2006.

[274] J. Xu, V. Duindam, R. Alterovitz, and K. Goldberg. Motion planning for steerable needles in 3D environments with obstacles using Rapidly-Exploring Random trees and backchaining. In *International Conference on Automation Science and Engineering*, pages 41–46, 2008.

[275] K. Yang and S. Sukkarieh. 3D-smooth path planning for a UAV in cluttered natural environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 794–800, 2008.

[276] Y. Yang and O. Brock. Adapting the sampling distribution in PRM planners based on an approximated medial axis. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 5, pages 4405–4410, 2004.

[277] Y. Yang and O. Brock. Elastic roadmaps–motion generation for autonomous mobile manipulation. *Autonomous Robots*, 28(1):113–130, 2010.

[278] Y. Ye and C. K. Liu. Synthesis of detailed hand manipulations using contact sampling. *ACM Transactions on Graphics (TOG)*, 31(4):41, 2012.

[279] A. Yershova, L. Jaillet, T. Simeon, and S. M. LaValle. Dynamic-domain RRTs: Efficient exploration by controlling the sampling domain. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3856–3861, 2005.

[280] A. Yershova and S. M. LaValle. Improving motion-planning algorithms by efficient nearest-neighbor searching. *IEEE Transactions on Robotics*, 23(1):151–157, 2007.

[281] M. Yim, D. G. Duff, and K. D. Roufas. PolyBot: a modular reconfigurable robot. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 1, pages 514–520, 2000.

[282] M. Yim, K. Roufas, D. Duff, Y. Zhang, C. Eldershaw, and S. Homans. Modular reconfigurable robots in space applications. *Autonomous Robots*, 14(2–3):225–237, 2003.

[283] M. Yim, W.-M. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. S. Chirikjian. Modular self-reconfigurable robot systems [grand challenges of robotics]. *IEEE Robotics & Automation Magazine*, 14(1):43–52, 2007.

[284] E. Yoshida, H. Kurokawa, A. Kamimura, K. Tomita, S. Kokaji, and S. Murata. Planning behaviors of a modular robot: an approach applying a randomized planner to coherent structure. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 2, pages 2056–2061, 2004.

[285] E. Yoshida, S. Murata, H. Kurokawa, K. Tomita, and S. Kokaji. A distributed reconfiguration method for 3D homogeneous structure. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 2, pages 852–859, 1998.

[286] L. Zhang and D. Manocha. An efficient retraction-based RRT planner. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3743–3750, 2008.

[287] Q.-H. Zhang, X.-H. Liu, and X.-S. Ge. Nonholonomic motion planning with PSO and spline approximation. In *IEEE International Conference on Control and Automation*, pages 2600–2604, 2007.

[288] V. Zykov, P. Williams, N. Lassabe, and H. Lipson. Molecubes extended: Diversifying capabilities of open-source modular robotics. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS): Workshop: Self-reconfigurable robots, systems and applications*, 2008.