Sampling-based Motion Planning in Theory and Practice

Wolfgang A. Pointner[1]

June 4, 2013

[1]W. Pointner is with Department of Telecooperation, Johannes Kepler University, 4020 Linz, Austria, and Department of Aeronautics and Astronautics, Stanford University, Stanford, CA 94305, USA. {wolfgang.pointner@jku.at, pointner@stanford.edu}

Contents

List of Figures

List of Algorithms

Abstract

Motion planning is one of the main topics when it comes to the development of mobile systems like e.g. robots or autonomous vehicles. The ability to perform fast and efficient planning is essential for the practicality of a system. It directly influences characteristics like performance and efficiency as well as usability and safety. Content of this work are the authors experiences and conclusions in the field of motion planning with special focus on Sampling-Based Motion Planning. Thereby, both theoretical and practical findings are going to be presented.

The practical experience in the mentioned field could be gained within a research project based on a mechanical arm. This robot had to perform a series of maneuvers under the influence of randomly placed thermal hazards. Main focus was the development of an efficient and robust planning algorithm that generated paths in order to avoid these obstacles.

The theoretical aspect in the field of motion planning algorithms was covered by implementation and numerical analysis of a modified algorithm. To be more precise this was the enhancement of the relatively popular RRT* algorithm which is based on Rapidly-Exploring Random Trees and uses a single-level rewiring strategy to achieve asymptotic optimality. The mentioned enhancement was the introduction of a multi-level rewiring strategy and this algorithm is further referred to as RRT^{N*}.

All results presented in this work were achieved during the authors five months stay as a Visiting Student Researcher at the Department of Aeronautics & Astronautics at Stanford University.

Kurzfassung

Pfad- und Bewegungsplanung gehören zu den wichtigsten Themen bei der Entwicklung von beweglichen Systemen wie beispielsweise Roboter oder autonomen Fahrzeugen. Die Möglichkeit effiziente und schnelle Planung ist essenziell für den praktischen Einsatz eines Systems und wirkt sich direkt auf Charakteristika wie Leistungsfähigkeit und Effizienz aber auch Sicherheit und Benutzbarkeit aus. Inhalt dieser Arbeit sind die Erfahrungen und Erkenntnisse des Autors im Zusammenhang mit Pfad- und Bewegungsplanung speziell auf dem Gebiet des sogenannten Sampling-Based Motion Planning.

Praktische Erfahrungen mit dieser Thematik konnten im Rahmen eines Projekts auf Basis eines mechanischen Arms gesammelt werden. Aufgabe dieses Roboters war die Durchführung mehrere bestimmter unter der Einwirkung von detektierbaren Hitzequelle. Hauptaugenmerkt lag dabei auf der Entwicklung eines effizienten und robusten Planungsalgorithmus zur Generierung von Pfaden welche willkürlich platzierten Hindernissen umgehen können.

Der theoretische Aspekt im Zusammenhang mit Pfadplanungsalgorithmen wurde Anhand der Implementierung und numerischen Analyse eines neuartigen Algorithmmus behandelt. Es handelt sich dabei um eine Weiterentwicklung des relative populären RRT* Algorithmus der auf sogenannten Rapidly-Exploring Random Trees, also Baumstrukturen die zufällig platzierte Punkte in ihre Struktur integrieren, basiert. RRT* verwendet ein einschichtiges Optimierungsverfahren zur Gewährleistung das der Algorithmus asymtotisch optimal ist. Bei der erwähnten Weiterentwicklung handelt es sich um ein mehrstufiges Verfahren zur Optimierung der Verbindungen innerhalb des resultierenden Baumes das in weiterer Folge als RRT^{N*} bezeichnet wird.

Alle in dieser Arbeit präsentierten Ergebnisse entstanden im Rahmen eines mehrmonatigen Forschungsaufenthalts am Institut für Luft- und Raumfahrt der Stanford Universität.

Acknowledgements

I gratefully acknowledge the support and generosity of the Austrian Marshall Plan Foundation as well as the Federal State Government of Upper Austria, without which the presented results achieved during my five months stay at Stanford University could not have been achieved. I also want to thank the Department of Aeronautics & Astronautics for the support during my time as a Visiting Student Researcher. Special thanks go to, Prof. Marco Pavone and his team of graduate students for their support and hospitality and Dr. Michael Naderhirn for his networking efforts. Furthermore, I want to acknowledge the support of the Stanford Structures and Composites Laboratory (SACL) under the lead of Prof. Fu-Kuo Chang. The presented material related to the robotic arm project is based upon work supported by the Air Force Office of Scientific Research (AFOSR) through the Multidisciplinary University Research Initiative (MURI) under Award No. FA9550-09-1-0677. The program monitor is Dr. Les Lee (AFOSR). Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author and do not necessarily reflect the views of AFOSR.

1. Introduction

This report covers the authors research at the Department of Aeronautics & Astronautics at Stanford University as a Visiting Student Researcher. The main focus of this program was the analysis and development of algorithms in the field of sampling-based motion planning in order to extend the authors expertise in the field of software systems design based on hybrid systems and formal methods.

Basically the program was dominated by two projects in the field of motion planning and control which covered both practical and theoretical aspects of planning in high-dimensional configuration spaces. The first one was a project that covered the implementation and demonstration of a control strategy for a robotic arm. This arm is equipped with skin-like sensor network that is capable of detecting thermal hazards that have to be avoided by the arm while it's performing a fairly simple pick-and-place task. The second project was the implementation and evaluation of a sampling-based motion planning algorithm that produces rapidly-exploring random trees. The main improvement of this algorithm is the application of a multi-level rewiring strategy that shall allow the algorithm to approach the optimum solution for given planning problem faster than the original algorithm.

This report is basically split into three parts: At first the fundamentals of motion planning and especially sampling-based motion planning are introduced in Chapter 2. This chapter also introduces some of the most popular algorithms in this field as well as the Open Motion Planning Library (OMPL) which was used to implement most of the introduced work. The second part introduces the mentioned project with the robotic arm in Chapter 3. It illustrates the algorithm and the control strategy and also introduces the sensor system. Finally, in Chapter 4 the mentioned improvement of the RRT* algorithm, called RRT^{N*}, is being discussed. The algorithm itself and also the results of various tests and simulations are presented in this chapter.

2. Motion Planning

Motion planning is a field of computer science that covers a variety of concepts and techniques. The basic idea of motion planning is the transformation of a certain task that shall be performed by a mobile system into a sequence of discrete motions. Such a task could e.g. the motion of a vehicle from a certain position to another one or a pick-and-place task that has to be performed by a robot.

Motion planing has a very wide field of applications which covers not only the already mentioned robotics, and navigational planning, but also control theory, artificial intelligence, computer graphics, and drug design. Although, all these fields might seem more or less related to each other they share some common characteristics:

Configuration space The configuration space C covers the set of configurations that shall be considered for motion planning. Typically the dimensions of this space are the degrees of freedom of the affected system. While e.g. navigational planning for a driving vehicle might be performed in a two-dimensional configuration space the control of a quadrotor helicopter might require the investigation of a much higher dimensionality.

Free space The free space $C_{free} = C \backslash C_{obst}$ is the subset of the configuration space that is not occupied by obstacles. Typically motion planning is performed in C_{free}, hence, it is often not trivial to determine whether a state within the configuration space collides with an obstacle or not.

Problem definition The problem that has to be solved by a motion planning algorithm is typically to reach a certain goal within the configuration space. This goal can either be a single state or a region which represents a configuration of the system that satisfies certain constraints.

Solution The solution of motion planning problem is often a sequence of concrete operations or control inputs that brings the system towards the planning goal.

The motion planning task has to overcome a fair amount of difficulties that arise from the specific planning problem. As already mentioned the determination of collisions with obstacles is a non-trivial task. Furthermore the complexity of the configuration space as well as it's limitations due to constraints can make it hard to find a feasible solution for a certain problem. Furthermore many problems in motion planning include the involvement of dynamics which makes them even more difficult.

Motion planning covers a variety of sectors that use different techniques or handle different problems. As described in [1] there are a couple of main categories:

- Discrete Planning
- Combinatorial Motion Planning
- Sampling-Based Motion Planning
- Feedback Motion Planning

The main focus of this work lies in the field of sampling-based motion planning since the results presented in Chapter 3 utilizes this kind of planning algorithms to control the motion of a robotic arm. Furthermore the numerical analysis in Chapter 4 are also based on this type of motion planning strategy.

2.1 Sampling-based Motion Planning

Sampling-based motion planning has become one of the most popular strategies to solve various planning tasks in recent years. It utilizes randomly picked samples within the configuration space and tries to use them to find a solution for a certain planning problem. This methodology has some main advantages over other motion planning techniques:

- The sampling strategy distinguishes between valid and invalid states the moment they are generated and therefore the construction of the configuration space that is obstructed by obstacles C_{obst} doesn't have to be performed explicitly. This would especially be challenging in configuration spaces with a very high number of obstacles.
- Depending on the sampling strategy it's likely that the quality of a solution found by a sampling-based motion planner improves while the number of samples grows. This makes this type of algorithm particularly scalable for various planning problems.

- Single components of the motion planning strategy can be encapsulated and algorithms can be developed independent of particular underlying models or specific configuration spaces. Typically these encapsulated parts cover the tasks of state sampling and validation, collision checking, and distance measurement.

The following section introduces some of the most popular sampling-based motion planning algorithms. As mentioned in [1] they basically can be separated into two groups:

Roadmap methods for multiple queries Roadmaps represent a graph that covers the configuration space. This approach is typically used to perform multiple queries within the configuration space without the need to reconstruct the graph structure each time. In the following section PRM and PRM* are introduced but there are also algorithms that produce so-called Visibility Roadmaps that will not be covered in this work.

Rapidly exploring dense trees for single-query applications RDTs are a family of algorithms that cover the configuration space by a tree structure that roots at the single-query initial state of the motion planning problem. Algorithms of this category that will be presented in the following section are RRT, RRT-Connect, and RRT*.

2.1.1 Motion Planning Algorithms

The family of sampling-based motion planning algorithms covers both single-query as well as multiple-query applications. In this work the main focus set on rapidly exploring dense trees but also probabilistic roadmaps will be presented in detail.

PRM

As mentioned before one form of sampling-based motion planning algorithms are Probabilistic Roadmaps (PRM). As introduced in [2] the PRM algorithm uses a strategy based on two phases:

Learning phase In this preprocessing phase a so called roadmap is built by iteratively adding random samples within the obstacle free configuration space. These newly added vertices are connected to the growing graph based on a fast local planner. In order to restrict the number of connections within the roadmap new connections within the same

connected component of the graph are avoided. The learning phase of the PRM algorithm is illustrated in Algorithm 2.1.

Query phase In the query phase of PRM a path between an initial configuration x_{init} and the goal configuration x_{goal} is determined. This is achieved by connecting both of them to the roadmap and searching for the shortest path through it that links them.

Algorithm 2.1: PRM Learning Phase

Result: Roadmap of vertices.
$V \leftarrow 0; E \leftarrow 0;$
for $i \leftarrow 0$ **to** n **do**
  $x_{rand} \leftarrow$`SampleFree`$_i$;
  $U \leftarrow$`Near`(G=(V,E),x_{rand},r);
  $V = U \cup \{x_{rand}\}$;
  $U_{ordered}$ = `SortByDistance`(U,x_{rand});
  foreach $u \in U_{ordered}$ **do**
    if `NotConnected`(x_{rand},u) **then**
      if `CollisionFree`(x_{rand},u) **then**
        $E \leftarrow E \cup \{(x_{rand}, u), (u, x_{rand})\}$;
return G=(V,E);

The learning phase and the query phase of PRM do not necessarily have to be executed subsequently. Alternation between those two phases can e.g. be enforced if the query phase is not particularly successful and a higher density of the roadmap would be required.

Figure 2.1 shows the typical development of a roadmap that is constructed by PRM within a two dimensional configuration space. The two sequences were constructed in an obstacle free space and a configuration space partially occupied by obstacles, respectively.

Related algorithms based on PRM are e.g. PRM* which will be discussed in detail later and Lazy PRM (see [3]). The Lazy PRM algorithm avoids the learning phase described for PRM and assumes that all vertices and edges within the roadmap are collision free. During the query phase the shortest path search investigates the graph and either finds a feasible already existing solution or updates the roadmap by adding new edges or removing invalidated vertices and edges.

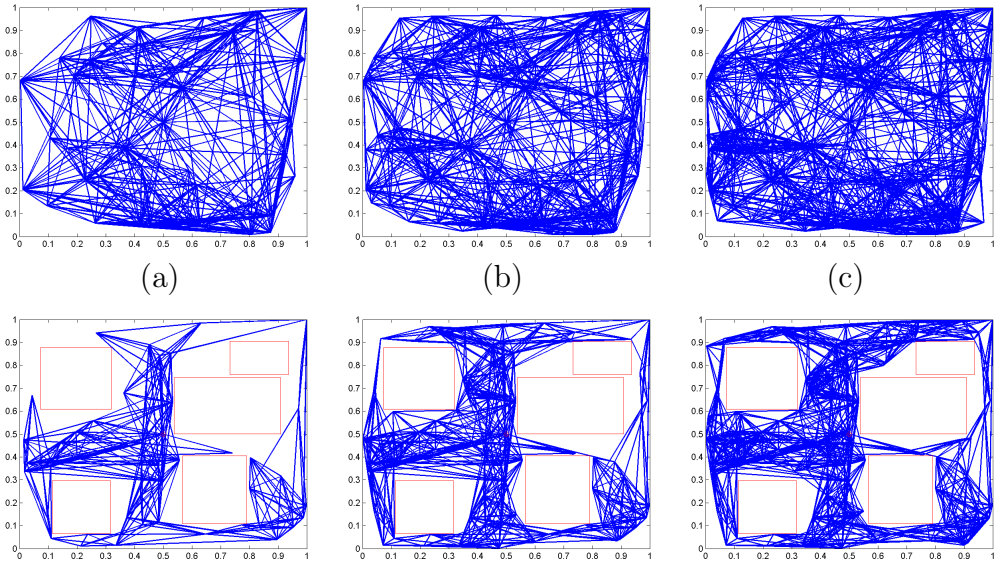


Figure 2.1: The sequence of plots shows the development of a roadmap with 50, 100, and 150 vertices, constructed by PRM in a space without (a)-(c) and with obstacles (d)-(f), respectively.

PRM*

As already mentioned the PRM* algorithm introduced in [4] is a modification of the PRM algorithm. Similar to a simplified version of PRM (sPRM) discussed in [5], it initializes the set of n vertices in the initialization and allows connections within already connected components. In contrast to sPRM, PRM* choses the connection radius r as a function of n. According to [4] this leads to an average number of connections attempted for a single vertex in the roadmap that is proportional to $log(n)$. Similar to PRM this algorithm is probabilistically complete. However, in contrast to PRM, PRM* is also asymptotically optimal since the probability not to find the optimal solution between two vertices within the roadmap converges to zero while n approaches infinity. Algorithm 2.2 illustrates the structure of PRM* including the identification of near nodes based on the radius $r = \gamma_{PRM}(log(n)/n)^{1/d}$, where $\gamma_{PRM} = 2(1+1/d)^{1/d}(\mu(X_{free})/\zeta_d)^{1/d}$, d is the dimension of the configuration space X, $\mu(X_{free})$ is the Lebesgue measure of the obstacle-free space, and ζ_d is the volume of the d-dimensional unit ball.

Similar to the previous section Figure 2.2 illustrates typical roadmap development of the PRM* algorithm in a two dimensional configuration space with and without obstacles.

Algorithm 2.2: PRM* Algorithm

Result: Roadmap of vertices.
$V \leftarrow \{x_{init}\} \cup \{\texttt{SampleFree}_i\}_{i=1,..,n}$;
$E \leftarrow 0$;
foreach $v \in V$ **do**
  $U \leftarrow$`Near`(G=(V,E),v,$\gamma_{PRM}(log(n)/n)^{1/d}) \setminus \{v\}$;
  foreach $u \in U$ **do**
    if `CollisionFree`(v,u) **then**
      $E \leftarrow E \cup \{(v,u),(u,v)\}$;
return G=(V,E);

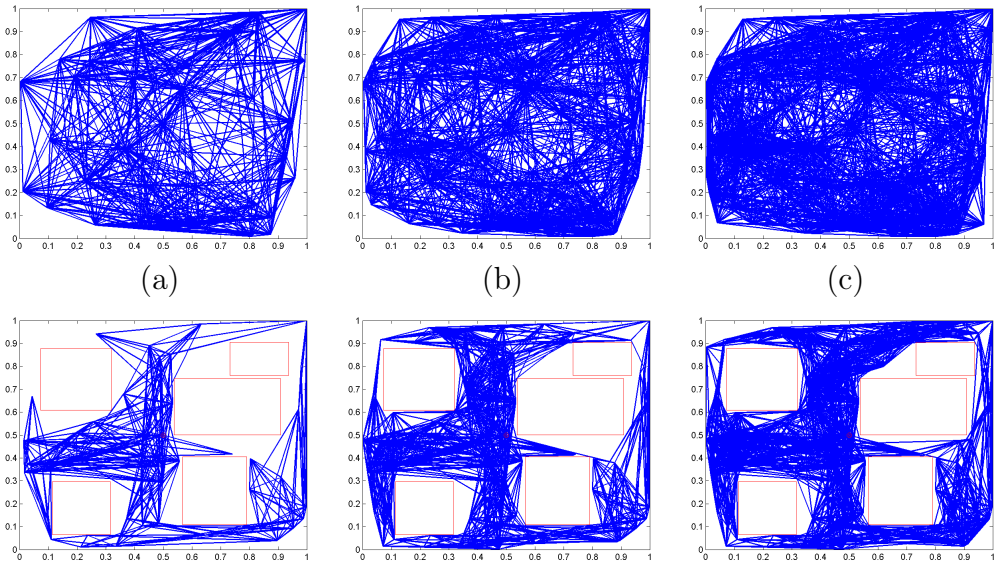


Figure 2.2: The development of a PRM* roadmap with 50, 100, and 150 vertices, in a space without (a)-(c) and with obstacles (d)-(f), respectively.

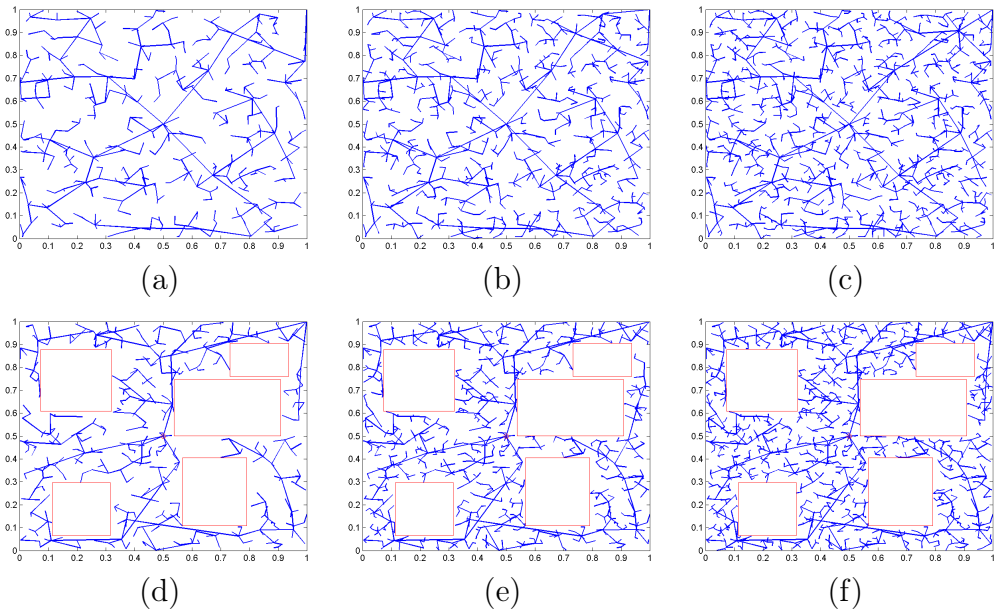
RRT

Rapidly-exploring random trees (RRT) are a popular method to perform sampling-based motion planning for single-query applications. They were first described in [6] and are further discussed in [7]. In contrast to the previously presented algorithms the result of RRT is a graph in form of a tree structure the stems from the initial state x_{init} and does not contain circuits. A randoml sample x_{rand} is chosen in X and the nearest vertex $x_{nearest}$ in the tree is steered towards it by adding a new vertex x_{new} if the connection

between $x_{nearest}$ and x_{new} does not cause a collision with an obstacle. While Algorithm 2.3 illustrates this behavior, the growth of the tree produced by RRT is illustrated in Figure 2.3.

Algorithm 2.3: RRT Algorithm

Result: Tree that stems from x_{init}.
$V \leftarrow \{x_{init}\}$;
$E \leftarrow 0$;
for $i \leftarrow 1$ **to** n **do**
  $x_{rand} \leftarrow$`SampleFree`$_i$;
  $x_{nearest} \leftarrow$`Nearest`(G=(V,E),x_{rand});
  $x_{new} \leftarrow$`Steer`($x_{nearest}$,x_{rand});
  if `ObstacleFree`($x_{nearest}$,x_{new}) **then**
    $V \leftarrow V \cup \{x_{new}\}$;
    $E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$;
return G=(V,E);



(a) (b) (c)

(d) (e) (f)

Figure 2.3: The growth of the tree produced by RRT shown for 100, 200, and 300 vertices, in a space without (a)-(c) and with obstacles (d)-(f), respectively.

RRT-Connect

An algorithm that is based on RRT is RRT-Connect which was first introduced in [8]. It basically improves the concept of rapidly-exploring random trees by implementing two additional concepts:

- It uses a heuristic that tries to establish connections over longer distances than the regular steering of RRT would do that.
- The algorithm grows two tree structures that stem from the initial state x_{init} and the goal configuration x_{goal}, respectively. These trees a intended to be connected so that a feasible path from the initial state to the goal can be established.

Algorithm 2.4 illustrates the general structure of RRT-Connect including the initialization of two trees T_a and T_b. In contrast to RRT the RRT-Connect algorithm also uses the function `Connect` to perform the extension function `Extend` multiple times for a certain tree. While one tree is extended in the regular way know from RRT, by steering x_{new} towards the randomly chosen sample, the other tree tries to connect to the newly added vertex by using `Connect`. This greedy behavior stops when either the x_{new} is reached or the algorithm is trapped when no valid new configuration x_{new} within the second tree can be found. As long as the status of the algorithm is either *Trapped* or *Advanced* it proceeds by swapping the trees and repeating the previously described behavior until the two trees could be connected, hence, a feasible path between x_{init} and x_{goal} has been found.

Algorithm 2.4: RRT-Connect

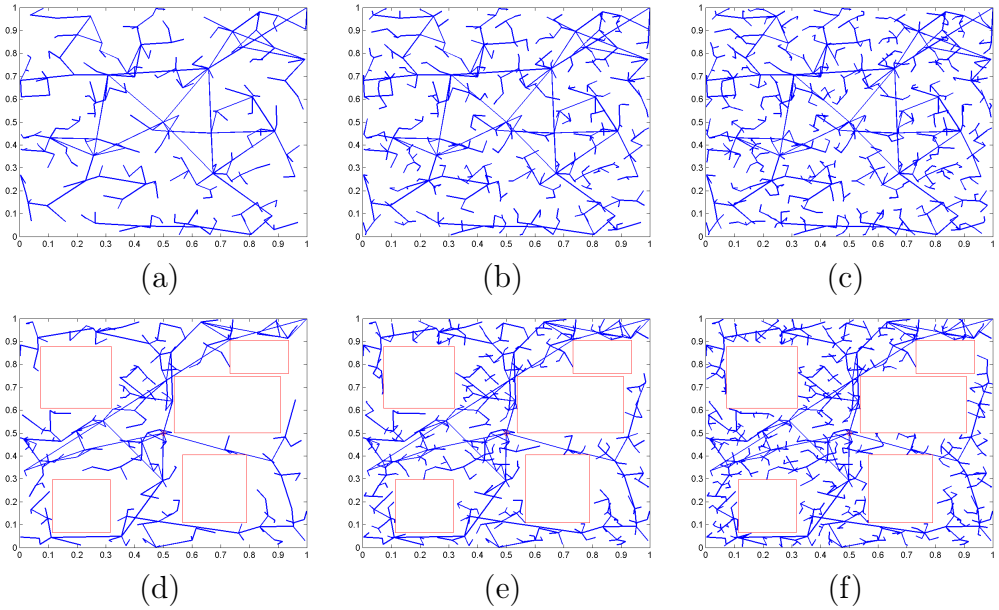
Input: Initial state x_{init} and goal state x_{goal}.
Output: Path from x_{init} to x_{goal} or *Failure*.
$T_a \leftarrow$`InitTree`(x_{init});
$T_b \leftarrow$`InitTree`(x_{goal});
for $i \leftarrow 1$ **to** n **do**
  $x_{rand} \leftarrow$SampleFree$_i$;
  if `Extend`(T_a, x_{rand}) != Trapped **then**
    if `Connect`(T_b, x_{new}) = Reached **then**
      return `Path`(T_a, T_b);
  `Swap`(T_a, T_b);
return *Failure*;

Function: Extend(T=(V,E),x)

Input: Tree T and sample x.
Output: Status of tree extension (*Reached*, *Advanced* or *Trapped*).
$x_{nearest}$ ←`Nearest`(T=(V,E),x);
x_{new} ←`Steer`($x_{nearest}$,x);
if `ObstacleFree`($x_{nearest}$,x_{new}) **then**
    $V \leftarrow V \cup \{x_{new}\}$;
    $E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$;
    if $x = x_{new}$ **then return** *Reached*;
    else return *Advanced*;
return *Trapped*;

Function: Connect(T=(V,E), x)

Input: Tree T and sample x.
Output: Status of tree connection.
repeat
    S ←`Extend`(T, x);
until S != Advanced;
return S;



(a) (b) (c)

(d) (e) (f)

Figure 2.4: The growth of the tree produced by RRT-Connect shown for 100, 200, and 300 vertices, in a space without (a)-(c) and with obstacles (d)-(f), respectively.

RRT*

The RRT* algorithm is also based on RRT and was first introduced in [4]. The basic idea of this algorithm is the extension of RRT by adding a rewiring strategy to it. The basic RRT algorithm adds nodes to the rapidly-exploring random tree without checking whether the newly added vertex would be beneficial to already existing paths within the tree. RRT* overcomes this drawback by applying a rewiring step within the local neighborhood of a newly added sample. This strategy ensures that vertices within the tree are reached trough a minimum-cost path. Algorithm 2.5 illustrates this strategy including the rewiring of nodes that are within a certain connection radius r, where r depends on the current number of vertices within the tree n.

Algorithm 2.5: RRT*

Input: Initial node x_{init}, number of samples n.
Output: Tree that stems from x_{init}.
$V \leftarrow \{x_{init}\}; E \leftarrow 0;$
for $i \leftarrow 1$ **to** n **do**
  $x_{rand} \leftarrow$ `SampleFree`$_i$;
  $x_{nearest} \leftarrow$ `Nearest`(G=(V,E),x_{rand});
  $x_{new} \leftarrow$ `Steer`($x_{nearest}$,x_{rand});
  if `ObstacleFree`($x_{nearest}$,x_{rand}) **then**
    $X_{near} \leftarrow$ `Near`(G=(V,E),x_{new}, $\min\{\gamma_{RRT^*}(log(card(V))/card(V))^{1/d}, \eta\}$);
    $V \leftarrow V \cup \{x_{new}\}$;
    $x_{min} \leftarrow x_{nearest}$;
    $c_{min} \leftarrow$ `Cost`($x_{nearest}$)+c(`Line`($x_{nearest}, x_{new}$));
    foreach $x_{near} \in X_{near}$ **do**
      if `CollisionFree`(x_{near},x_{new})$\wedge$ `Cost`(x_{near})+$c(Line(x_{near}, x_{new}))< c_{min}$ **then**
        $x_{min} \leftarrow x_{near}$;
        $c_{min} \leftarrow$ `Cost`(x_{near})+c(`Line`(x_{near}, x_{new}));
    $E \leftarrow E \cup \{(x_{min}, x_{new})\}$;
    foreach $x_{near} \in X_{near}$ **do**
      if `CollisionFree`(x_{new},x_{near})$\wedge$ $Cost(x_{new})+c($`Line`$(x_{new}, x_{near}))<Cost(x_{near})$ **then**
        $x_{parent} \leftarrow$ `Parent`(x_{near});
        $E \leftarrow (E \backslash \{(x_{parent}, x_{near})\}) \cup \{(x_{new}, x_{near})\}$;
return G=(V,E);

Basically the rewiring is performed in two phases:

- At first, all selected neighbors in X_{near} are investigated whether the cumulated costs to reach x_{new} through them might be a minimum. In that case x_{new} is connected to that vertex rather than $x_{nearest}$ which would be the nearest node.
- In the second phase all nodes in X_{near} are investigated if a obstacle free path through x_{new} might lower their costs-to-come. If that's the case x_{new} becomes the new parent of the affected vertices and their costs as well as the costs of their children get updated recursively.

In that way RRT* guarantees that vertices within the tree are reached via minimum-costs paths. Furthermore this rewiring strategy makes RRT* asymptotically optimal since the probability that the optimal solution path to any point within the configuration space C is found goes to 1 while the number of samples n approaches infinity.

A modified version of RRT* which is also presented in [4] is called k-nearest RRT*. In this version the rewiring strategy is performed for the k nearest nodes in the neighborhood of x_{new}.

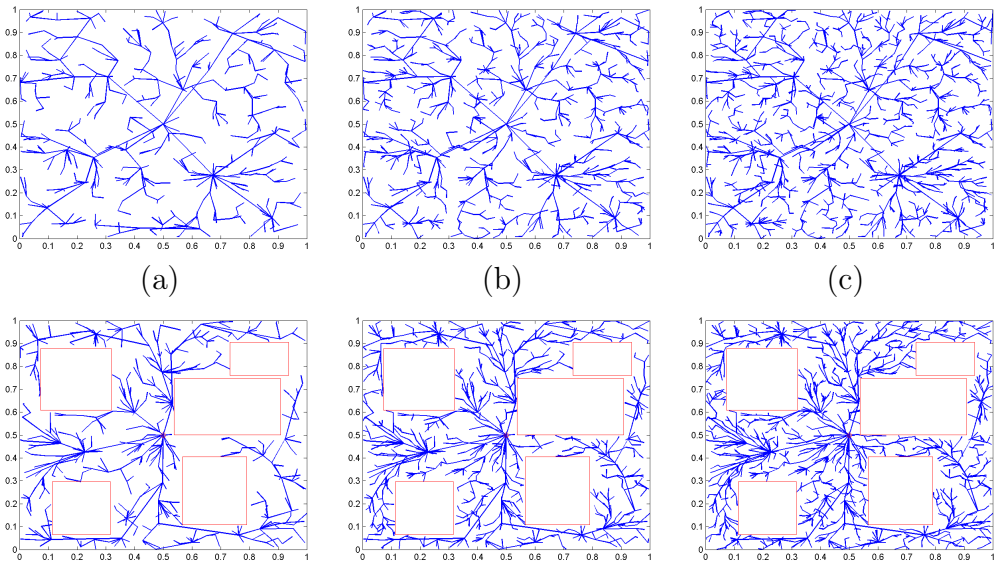


Figure 2.5: The growth of the tree produced by RRT* shown for 100, 200, and 300 vertices, in a space without (a)-(c) and with obstacles (d)-(f), respectively.

Further Algorithms

Additionally to the previously described algorithm there are various others in the field of sampling-based motion planning. A few of them will be briefly described as follows.

KPIECE Kinodynamic Planning by Interior-Exterior Cell Exploration is a kinodynamic motion planner, specifically designed for systems with complex dynamics, which was first introduced in [9]. It uses a grid-based discretization strategy on multiple levels that helps it to determine the coverage of the configuration space and focus exploration on less covered areas.

EST Expansive Space Trees were introduced in [10]. Similar to RRT-Connect this algorithm uses a strategy where two trees are constructed rooted at the initial configuration and the goal configuration, respectively. In EST random samples are connected to the the trees based on their visibility regions. A valid connection thus a feasible path is found when the visibility regions of the two trees intersect. A modified version of ESTs are guided EST which were introduced in [11]. They try to find a relatively straight path through the tree in order to minimize the costs.

SBL Single-Query Bi-Directional Probabilistic Roadmap Planner with Lazy Collision Checking (see [12]) is another motion planning algorithm that grows two trees rooted at the initial state and the goal state, respectively. While the expansion strategy is the same as for EST the validity of a found solution path is not checked until a connection is established. In case the result includes invalid parts these are removed from the trees and the exploration of the state space continues until a new solution is found. In order to guide the exploration an additional grid structure is maintained that keeps track of the states that have been part of a possible solution in previous attempts.

The currently still evolving progress in the field of sampling-based motion planning continuously produces new algorithms that are more or less related or based on those previously presented. They are based on random or quasi-random selection of samples within the configuration space which are somehow connected to one or multiple existing graphs. Since the basic methods are pretty similar for all of them new developments often distinguish themselves by combination of various components or modification of certain concepts. One of this improvements based on RRT* is RRT^{N*} which is going to be presented in Chapter 4.

2.2 Open Motion Planning Library (OMPL)

The Open Motion Planning Library (OMPL)[1] is an open source library that can be used to solve a variety of motion planning problems. It provides various sampling-based motion planning algorithms and furthermore it's architecture and components allow the implementation and integration of self-developed algorithms. It was developed by robotics company Willow Garage[2] in cooperation with the Kavraki Lab of the Department of Computer Science at Rice University[3].

A brief introduction on the installation and integration of OMPL is given in [13]. The API is implemented in C++ and provides Python bindings and there are also packages available that allow the integration into the Robot Operating System (ROS)[4].

The general architecture of the OMPL and it's main components are briefly described in [14]. As mentioned before sampling-based motion planning algorithms commonly depend on a couple of basic components which are all covered by OMPL:

State space The description of the systems state covers all configurations that have to be considered during motion planning. As mentioned earlier it's also referred to as configuration space.

Control space For systems that incorporate dynamics the set of parameters that are considered for planning are covered by the control space.

Problem definition The definition of the planning problem is basically the goal that should be reached by the planning algorithm. Such a goal can be a single state or multiple states as well as a goal region within the state space.

Sampler The sampler generates states within the state space that shall be investigated during planning. In case of a planning environment that includes controls there is also a control sampler required.

State validity checker The validation of the sampled states distinguishes states that shall be considered for planning from those that might be invalidated by violating some constraints. These constraints might be

[1] The Open Motion Planning Library. `ompl.kavrakilab.org`
[2] Willow Garage, Menlo Park, CA 9402, USA. `www.willowgarage.com`
[3] Kavraki Lab, Rice University, Houston, TX 77005, USA. `www.kavrakilab.org`
[4] ROS (Robot Operating System). `www.ros.org`

due to inherent characteristics of the system that is being planned for or external factors like e.g. obstacles.

Local planner For systems with controls the local planing component describes it's propagation within the configuration space due to the control input.

Motion validation The validation of motions between different states within the state space is a component is a variation of local planning for geometric motion planning system.

In addition to these basic components the OMPL API provides a variety of components and classes that allow the implementation of more or less sophisticated motion planning systems. Figure 2.6 shows an overview of the OMPL API that contains the mentioned components as well as some classes like e.g. `SimpleSetup`, `Planner`, and `SpaceInformation` which support the setup of a motion planning algorithm.

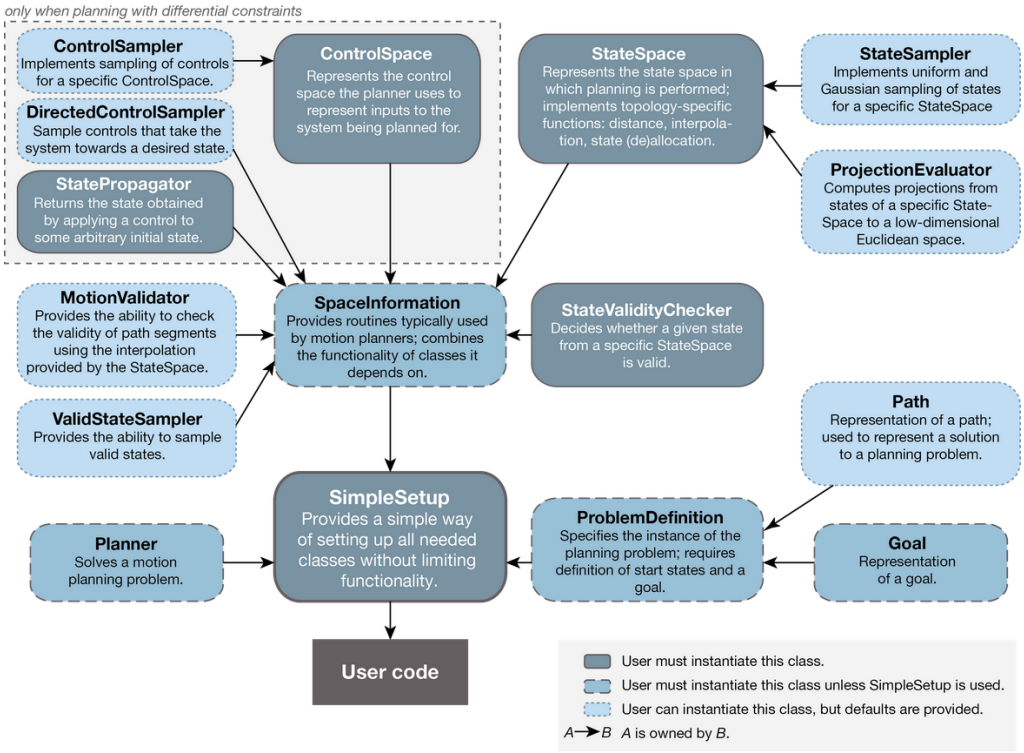


Figure 2.6: Overview of the OMPL API.
Figure courtesy of Kavraki Lab, `http://ompl.kavrakilab.org/api_overview.html`

The planners provided by OMPL mainly cover geometric planning but there a also some available that support control planning for systems with kinematics.

These are the planners available for planning under geometric constraints:

- Kinematic Planning by Interior-Exterior Cell Exploration (KPIECE) [9]
- Bi-directional KPIECE (BKPIECE) [9, 3])
- Lazy Bi-directional KPIECE (LBKPIECE) [9, 3]
- Single-query Bi-directional Probabilistic Roadmap with Lazy collision checking planner (SBL) [12]
- Parallel Single-query Bi-directional Lazy collision checking planner (pSBL) [12]
- Expansive Space Trees (EST) [10]
- Rapidly-exploring Random Trees (RRT) [6, 7]
- RRT Connect (RRTConnect) [8]
- Parallel RRT (pRRT) [8]
- Lazy RRT (LazyRRT) [8]
- Probabilistic RoadMaps (PRM) [2]
- PRM* [4]
- RRT* [4]
- Ball Tree RRT* [4]
- Transition-based RRT (T-RRT) [15]

For planning under differential constraints the following set of planners is provided by OMPL:

- Kinodynamic Planning by Interior-Exterior Cell Exploration (KPIECE) [9]
- Rapidly-exploring Random Trees (RRT) [16]
- Expansive Space Trees (EST) [10]
- Syclop using either RRT or EST as the low-level planner [17]

The variety of available planning algorithms as well as the possibility to implement and integrate further algorithms were the main factors why OMPL was chosen to implement the algorithms and simulations described in Chapter 4.

3. Robotic Arm Project

The purpose of this project was the development and demonstration of an immobile robotic arm that is capable of performing basic pickup and delivery tasks while detecting and avoiding thermal obstacles. The robot which is shown in Figure 3.1 was developed by the Structures and Composites Laboratory (SACL) of the Department of Aeronautics and Astronautics at Stanford University[1].

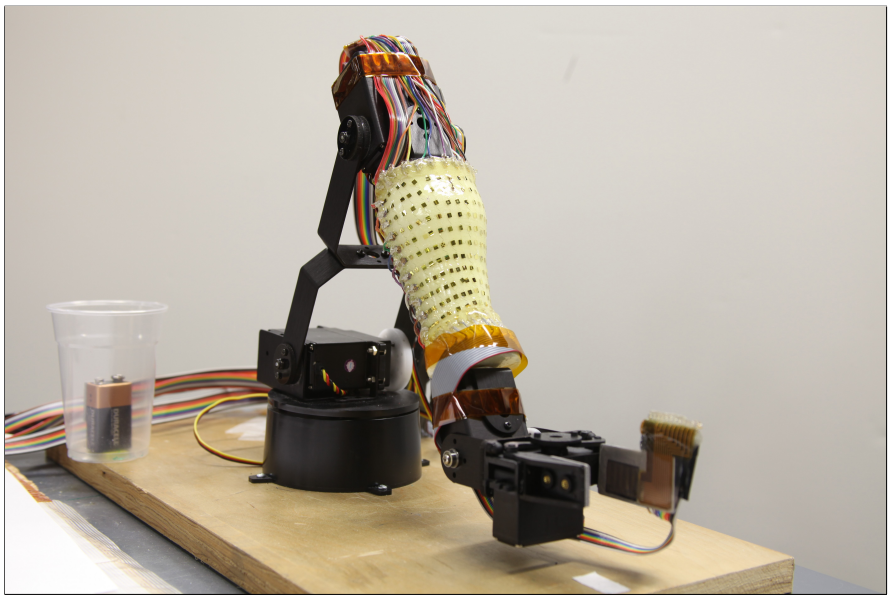


Figure 3.1: The robotic arm developed by the SACL that was used for development and demonstration of motion planning with thermal hazards.

Image courtesy of Joseph A. Starek, Department of Aeronautics and Astronautics, Stanford University.

This project was intended to present a showcase for combined sensing and planning strategies under thermal hazards. Potential fields of application for

[1]Structures and Composites Laboratory, Department of Aeronautics and Astronautics, Stanford University, CA 94305, USA. `structure.stanford.edu`

such a technology could be found in fields like e.g. bomb defusal, search-and-rescue operations or robotic assembly.

3.1 Problem Formulation

The task that was created in order to demonstrate the capabilities of the robotic arm was the pickup of a certain payload at a know position on the base plane of the construction as well as its delivery to a know goal location. In this scenario the payload is a simple 9V battery and the goal location is 8-ounce plastic cup. Thermal obstacles were introduced by a heat gun at random times during the scenario. Based on this configuration there were several tasks that have to be performed:

- Development of a motion plan from each of the predefined locations / configurations to the next one. These locations are the initial configuration of the robot, the pickup location of the payload, and the delivery position. This basically results in three independent motion plans that have to be determined by the planning algorithm. Since the last motion plan returns the robot to its initial configuration the scenario can be performed repetitively.

- During the execution of these motion plans the sensor network has to identify possible thermal obstacles and estimate their position within the configuration space. In this work it was assumed that the position of thermal obstacles is fixed.

- In case of a violation of a motion plan by a detected thermal obstacle the planning algorithm has to identify a feasible plan that avoids a collision. A fast re-routing is aspired in order to minimize the delay of the task and to avoid potentially damaging thermal obstacles early enough.

- In order to simulate a *natural* behavior during the pickup and delivery scenario additional temperature and pressure sensors at the end-effector are used to determine the presence of a payload as well as its temperature. This shall prevent the robot from performing a task without a payload or grabbing a payload that might be *hot*.

3.2 System Design

The robotic arm is fixed on a rotating base and combines three joints between the arms elements as well as an end-effector. The joint angles as well as the slewing range of the rotating based are limited due to constructional constrains mainly caused by the wiring of the actuators and the sensors. The introduction of thermal obstacles is sensed by a skin-like sensor network applied on the *forearm* of the robot (see 3.1). In order to be able to detect the presence as well as the temperature of the payload there are additional combined pressure and temperature sensors applied in the inside of the end-effectors gripper.

3.3 Planning Strategy

As planning strategy for the robotic arm a sampling-based motion planning algorithm was chosen. The configuration space C is therefore reduced to the manipulator joint angle space of $Q \subset \mathbb{R}^4$ representing the 4 degrees of freedom of the manipulators that move the arm. This includes the assumption that the gripper separation as well as the dynamics of the robot can be neglected for the planning task.

In order to prevent the arm from taking configurations that would lead to a collision of its parts with each other, the environment, or obstacles it is represented as a set of oriented bounding boxes (OBB). The corners of these boxes as well as randomly sampled points on their surface are taken into account when collision are checked during the motion planning. This representation is also chosen for obstacles as well as the payload which are either represented as cylinders, truncated frustums, or cuboids. Further collision checks are performed between the robots parts and planes that can be used to represent the environment.

In this setup the configuration space C is different from the world space $W \subset \mathbb{R}^3$ that represents the positions of the robot, the goals of the different motion plans, and the obstacles. This requires the introduction of a practical transformation method that allows to transfer positions into joint angle configurations. Since the robot can be seen as a kinematically–constrained chain of rigid bodies such a system can represented as a set of homogeneous transformation matrices. In this work the Devanit–Hartenberg (DH) parameters [18] are used to define these matrices that represent the transformation from one link to a connected one. This allows transformation from one coordinate system into the other one and therefore provides us with the ability to check if a certain joint angle configuration in C would lead to a collision

in W. The introduction of obstacles into C allows to specify the regions that are covered by obstacles $C_{obs} \subset C$ as well as the obstacle free space $C_{free} = C \setminus C_{obs}$.

The actual planning task within the configuration space C was performed by a sampling based motion planning algorithm. This algorithm is based on the RRT-Connect algorithm which was introduced in Section 2.1.1. The modified version called RRT*-Connect uses bi-directional planning which produces two trees T_a and T_b in C_{free} growing from the initial state q_I and the goal state q_G, respectively. As shown in Figure 3.2 these trees a built-up by expansion of one of them towards a randomly sampled node q_r. The node q_{near} that is nearest to q_r is selected and by *steering* it towards q_r a new node q_{new} is introduced and connected to the tree. Thereupon the other tree is investigated and its node that is closest to q_{near} is incrementally *steered* towards it until either a connection can be established or a obstacle is reached. In the next iteration the behavior of the two trees is swapped and the next sample node is picked. This sampling within the configuration space is performed by application of the Halton sequence [19]. This produces a deterministic, quasi-random sequence of samples for each dimension of C which covers a broad range of sampling regions while ensuring minimum-possible dispersion.

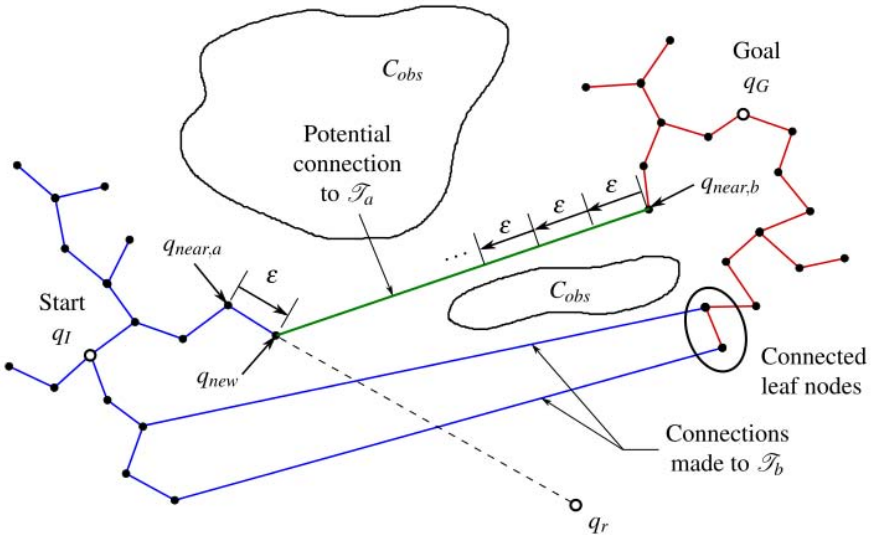


Figure 3.2: Illustration of the RRT*-Connect algorithm. The two trees T_a and T_b try to connect within the obstacle free space C_{free}.

Image courtesy of Joseph A. Starek, Department of Aeronautics and Astronautics, Stanford University.

As soon as a connection between the two trees has been established the RRT*-Connect algorithm has found a valid transition path from the initial to the goal state. Such a path contains motions that are represented by the nodes along it. Since RRT*-Connect also performs a rewiring mechanism similar RRT* (see 2.1.1) such a solution can be improved by constantly introducing new samples which might produce paths from q_I to q_G that are even less cost intensive.

Due to the introduction of new obstacles it is likely possible that a currently established path form the initial state to the goal state is being invalidated. In that case the RRT*-Connect algorithm is either able to determine an alternative path by investigating previous solutions or the algorithm has to try to reconnect its two trees as described earlier. This forces the algorithm to explore new regions of C_{free} until a new feasible path is found.

3.4 Control Logic

The motion of the robotic arm follows a closed-loop strategy the mainly consists of the following three maneuvers: pick up the payload, deliver payload to goal, return arm to initial state. This sequence can be repeated as often as desired although the resulting motion may be determined by multiple external influences. As mentioned earlier the trajectory of the robotic arm can be re-routed due to thermal hazards. This can happen during any of the three mentioned maneuvers and requires the system to identify a feasible trajectory that avoids the collision with an introduced thermal obstacle. In case the algorithm is not able to identify such a path the arm performs a predefined emergency maneuver that returns it to the initial position in a way that is assumed to have a very low risk of collision with a thermal obstacle.

In order to generate a behavior that might seem more *natural* to a human observer the robotic arm was equipped with additional sensors on the end-effector. These are able to detect whether the closed end-effector has actually grabbed a payload as well as determining its temperature. The resulting logic has the effect that the robotic arm performs the main operational loop of delivering a payload to the goal only if has actually been picked up and does not exceed a specific temperature threshold.

The combination of the main operational logic and the introduced checks, re-routing operations, and emergency maneuvers leads to a close-loop decision logic for the robotic arm that is illustrated in Figure 3.3. The main path includes the three mentioned maneuvers as well as the operations to grab and release the payload. The identification of an alternative route that

avoids thermal obstacles can be initiated during all these phases. In case this re-routing does not identify a feasible path within a certain time the emergency maneuver is performed, returning the arm to the initial configuration.

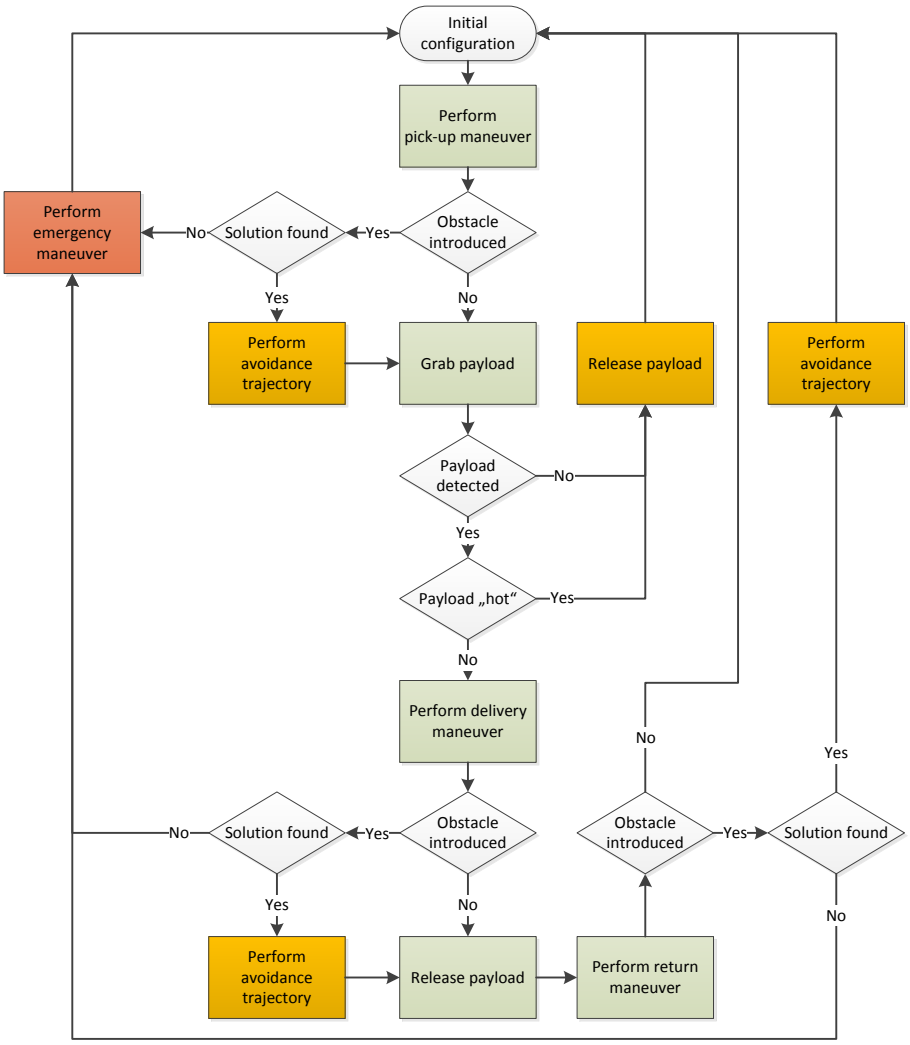


Figure 3.3: The closed-loop control of the robotic arm deals with introduced obstacles and tries to imitate a *natural* behavior when picking up the payload.

3.5 Sensor System

The previously mentioned a skin-like sensor network is applied on the robotic arm in order to be able to detect thermal hazards and estimate their position and orientation within the world space W. The thermal obstacles that might be identified by this sensors could e.g. be blowtorches, vents, or low-temperature open flames, such as lit matches or candles. The sensor network used for the presented robot was developed by Stanfords SACL and is introduced in [20] and [21]. It is based on a expandable sensor grid that is capable of estimating the position as well as the intensity of a thermal source. By using a k-means clustering algorithm the hottest region of the sensor network is determined. Afterwards a heuristic mapping technique is use to estimate the position and orientation of the thermal hazard based on the temperature distribution. This shall emulate the behavior of a human in reaction to a given temperature distribution across the surface of his or her own skin and is referred to as "best-cone" search in [22]. Based on this estimation the temperature obstacle and represented as a truncated frustum that has a certain distance and orientation in relation to the irregular surface (see Figure 3.4).

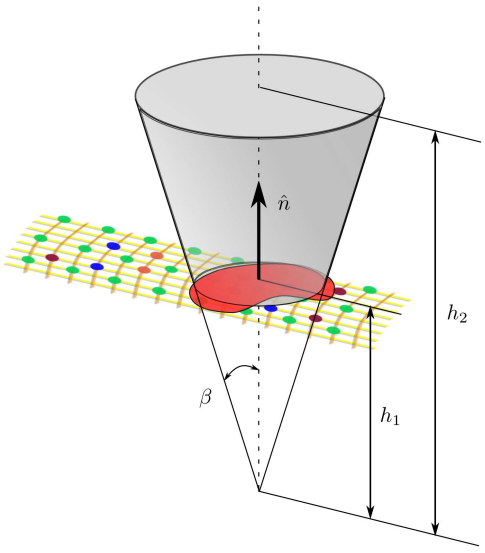
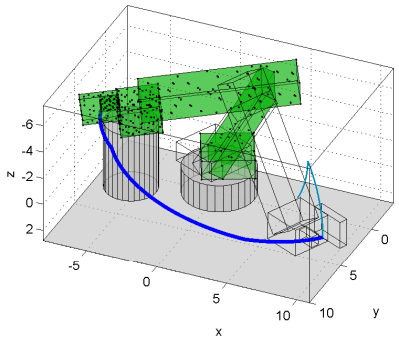


Figure 3.4: Representation of a temperature obstacle in relation to the sensor network on the arm.

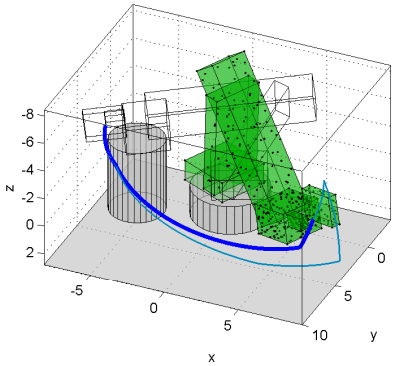
Image courtesy of Joseph A. Starek, Department of Aeronautics and Astronautics, Stanford University.

3.6 Tests

The demonstration of the robotic arm was performed by executing the scenario described earlier, where a 9V battery had to be put into a cup while randomly placed thermal obstacles had to be avoided. The results of the motion planning algorithm where therefore plotted in the three-dimensional world space and the robot was represented as a set of oriented bounding boxes. Figure 3.5 shows a sequence of motion plans that have been identified to be feasible paths for the robotic arm to perform the motions from the initial state to the position of the payload, from there to the goal position, and finally back to the initial configuration.



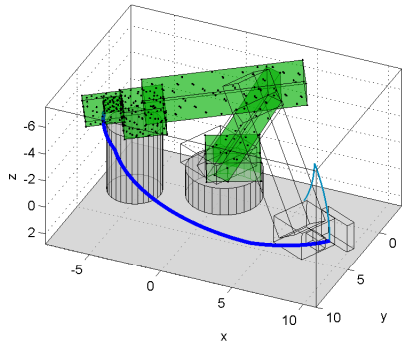
(a) Undisturbed motion towards the goal position above the cup.



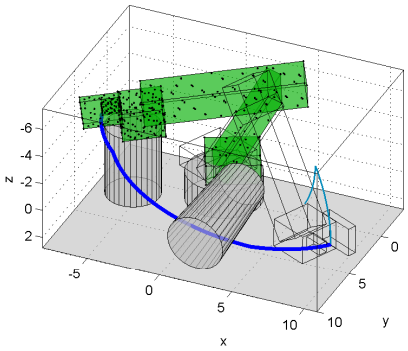
(b) Motion plan towards the initial configuration.

Figure 3.5: The RRT*-Connect algorithm has identified feasible paths for both the motion plan towards the delivery location (a) and the initial configuration (b).

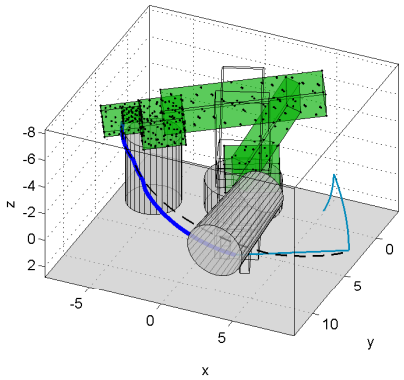
The introduction of an obstacle during the execution of the second motion plan forces the planning algorithm to rewire the trees and identify a new feasible path. Figure 3.6 shows this behavior and the resulting motion plans.



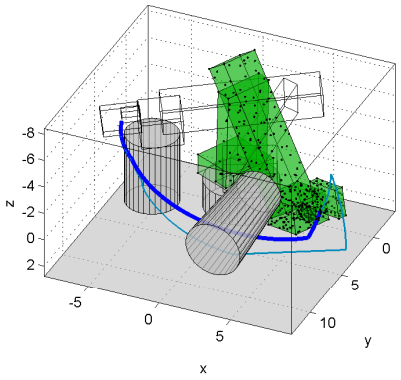
(a) Intended motion path.



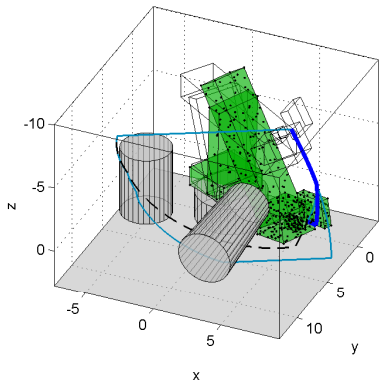
(b) Introduction of an obstacle.



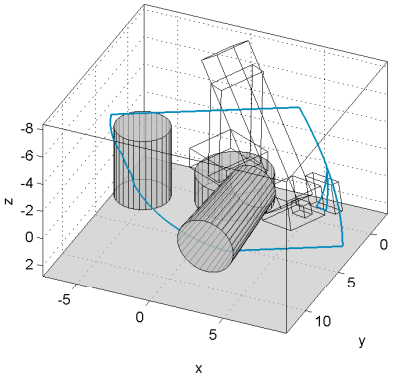
(c) Avoiding path below obstacle.



(d) Intended motion towards the initial configuration.



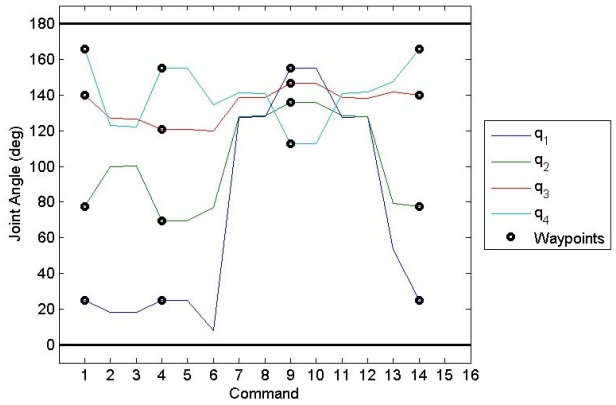
(e) Re-routed path towards the initial configuration.



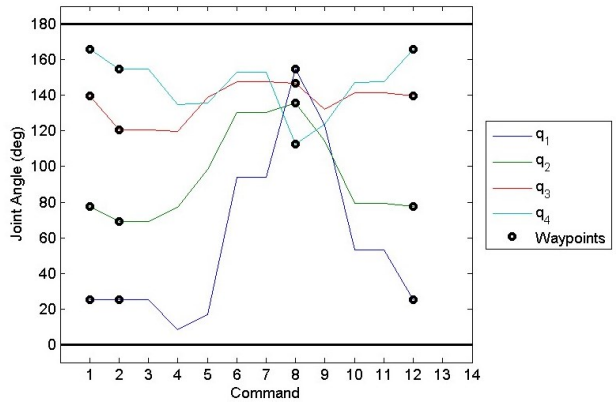
(f) Total motion path avoiding the obstacle.

Figure 3.6: The intended motion (a) is invalidated by an obstacle (b). The rewiring process of the RRT*-Connect algorithm identifies a new valid path to the goal (c). After performing the same for the last motion path (d) and (e) the total motion plan avoiding the obstacle is shown in (f).

The Figures 3.5 and 3.6 nicely illustrate the difference between the motion paths that are generated depending on whether there is an obstacle that has to be avoided or not. These paths are shown in the world frame W the different motion plans in the configuration space C can be seen in Figure 3.7. It shows the sequence of motion commands for the four different actuators of the robotic arm.



(a) Sequence of motion commands without an obstacle.



(a) Sequence of motion commands with an obstacle.

Figure 3.7: The sequence of motions that is planned by the RRT*-Connect algorithm within C is different without (a) and with (b) the introduction of an obstacle.

Image courtesy of Joseph A. Starek, Department of Aeronautics and Astronautics, Stanford University.

In order to test the capabilities of the RRT*-Connect algorithm and the sensor network as well as their robustness, plenty of test runs have been performed with the robotic arm. The scenarios included the introduction of

thermal obstacles in form of a heat gun during randomly chosen phases of the motion plan. In order to avoid invalidation of goal points these positions were mainly chosen in an area similar to the one illustrated in Figure 3.6 as well as an area above the position of the payload, that would prevent the arm from picking it up. Furthermore the control logic was tested by placing heated payloads of similar size as a 9V battery. Given a certain delay and threshold the temperature sensors applied on the inside of the gripper were able to detect whether a payload was too hot for transportation or not.

3.7 Conclusion and Future Work

The development process of the introduced system as well as the simulations and test runs performed on the robotic arm test environment have given multiple findings:

- The concept of the RRT*-Connect algorithm is basically capable of finding feasible paths within the obstacle free configuration space C_{free}. The simulations have shown that a number of a few thousand nodes was sufficient to do so. The continuous incrementation of nodes leads to even more smooth motion plans after multiple runs.

- The re-routing process due to the invalidation of a motion path by a thermal obstacle typically is able to find feasible new paths within an appropriate time of a few seconds. Problematic configurations like the invalidation of a goal node might cause the system to perform the emergency scenario.

- The skin-like sensor network has proven itself to be capable of identifying thermal hazards in form of a heat gun with a high reliability. The intense performance of subsequent test runs has shown that the material the sensor network was put on tends to heat up. This increased the probability of false positive detected temperature obstacle and often corrupted the estimation of their position and orientation in relation to the robotic arm.

Future versions of the developed system might incorporate the previously introduced Open Motion Planning Library (OMPL) (see Section 2.2). This would allow a more modular approach based on already existing components and reduce the implementation effort.

4. Enhancement of RRT*

As described in Section 2.1.1 the RRT* algorithm which is based on rapidly exploring random trees was developed by Karaman and Frazolli (see [4]). It uses the concept of the RRT algorithm to connect randomly sampled nodes to a growing tree and additionally performs a rewiring process to create paths that are less cost-intensive. This rewiring process is performed every time a newly added node would provide one of its neighbors with a lower costs-to-come. Due to the concept of RRT* this rewiring is limited to the first-order neighborhood of a node.

In [4] it is claimed that RRT* is both probabilistically complete and asymptotically optimal. Probabilistic completeness is provided by the fact that the probability that RRT* is not able to return a solution, if one exists, approaches zero as the number of nodes approaches infinity. Asymptotic optimality is claimed based on the theorem that a single rewiring level is sufficient to let the currently best solution converge towards the global optimal solution as the number of nodes approaches infinity.

4.1 The RRT^{N*} Algorithm

Recent investigations by the Autonomous Systems Laboratory (ASL) at Stanfords Department for Aeronautics & Astronautics have raised speculations that the claim of asymptotic optimality of RRT* might not hold and that there might be modifications of the algorithm that either guarantee asymptotic optimality or at least achieve a higher convergence rate than the original implementation. The concept that was investigated in this chapter uses recursive rewiring and is subsequently referred to as RRT^{N*}. This means that nodes found within the local neighborhood of a newly added sample that were successfully rewired are the new origin of a secondary rewiring step. This procedure is repeated until a certain maximum rewiring level is reached or no more nodes could be rewired in order to decrease their costs.

4.1.1 Implementation

As mentioned the RRT^{N*} is basically implemented like RRT* with the addition of recursive rewiring strategy. Algorithm 4.1 illustrates the structure of the modified RRT* as well as the recursively called procedure `RewireNodes`.

Algorithm 4.1: RRT^{N*}

Input: Initial node x_{init}, number of samples n.
Output: Tree that stems from x_{init}.

$V \leftarrow \{x_{init}\}; E \leftarrow 0; X_{rewire} \leftarrow 0;$
for $i \leftarrow 1$ **to** n **do**
  $x_{rand} \leftarrow$`SampleFree`$_i$;
  $x_{nearest} \leftarrow$`Nearest`(G=(V,E),x_{rand});
  $x_{new} \leftarrow$`Steer`($x_{nearest}$,x_{rand});
  if `ObstacleFree`($x_{nearest}$,x_{rand}) **then**
    $X_{near} \leftarrow$`Near`(G=(V,E),x_{new}, $\min\{\gamma_{RRT*}(log(card(V))/card(V))^{1/d}, \eta\}$);
    $V \leftarrow V \cup \{x_{new}\}$; $x_{min} \leftarrow x_{nearest}$;
    $c_{min} \leftarrow$`Cost`($x_{nearest}$)+c(`Line`($x_{nearest}, x_{new}$));
    foreach $x_{near} \in X_{near}$ **do**
      if `CollisionFree`(x_{near},x_{new})$\wedge$ `Cost`(x_{near})+c(`Line`(x_{near}, x_{new}))$< c_{min}$ **then**
        $x_{min} \leftarrow x_{near}$;
        $c_{min} \leftarrow$`Cost`(x_{near})+c(`Line`(x_{near}, x_{new}));
    $E \leftarrow E \cup \{(x_{min}, x_{new})\}$;
    foreach $x_{near} \in X_{near}$ **do**
      if `CollisionFree`(x_{new},x_{near})$\wedge$ `Cost`(x_{new})+c(`Line`(x_{new}, x_{near}))<`Cost`(x_{near}) **then**
        $E \leftarrow (E \setminus \{(\texttt{Parent}(x_{near}), x_{near})\}) \cup \{(x_{new}, x_{near})\}$;
        $X_{rewire} \leftarrow X_{rewire} \cup x_{near}$;
    if $maxRewiringLevel > 1$ **then**
      `RewireNodes`(X_{rewire}, 2, V, E);
return G=(V,E);

In contrast to the original RRT* algorithm, RRT^{N*} keeps track of the nodes that have been rewired during the regular first-order rewiring process by adding them to the set X_{rewire}. If the defined maximum rewiring level $lvl \geq 2$ the procedure `RewireNodes` is called given X_{rewire} as input parameter. This procedure basically repeats the search for near nodes as well as the rewiring step for all nodes in X_{rewire} and again keeps track of those who have successfully been rewired. If the maximum rewiring level is still greater than

Procedure: RewireNodes(X_{rewire}, lvl, V, E)

Input: Set of nodes X_{rewire} that shall be rewired on level lvl as well as vertices V and edges E.

$Y_{rewire} \leftarrow 0$;
foreach $x_{rewire} \in X_{rewire}$ **do**
    $X_{near} \leftarrow$ `Near`(G=(V,E),x_{rewire}, $\min\{\gamma_{RRT*}(log(card(V))/card(V))^{1/d}, \eta\}$);
    foreach $x_{near} \in X_{near}$ **do**
        if `CollisionFree`$(x_{rewire},x_{near})\wedge$ `Cost`$(x_{rewire})+c($`Line`$(x_{rewire}, x_{near}))<$`Cost`(x_{near}) **then**
            $E \leftarrow (E \setminus \{(\texttt{Parent}(x_{near}), x_{near})\}) \cup \{(x_{rewire}, x_{near})\}$;
            $Y_{rewire} \leftarrow Y_{rewire} \cup x_{near}$;
if $maxRewiringLevel > lvl$ **then**
    `RewireNodes`(Y_{rewire}, $lvl + 1$, V, E);

the current level `RewireNodes` is called recursively with lvl incremented by 1. This procedure ends as soon as the maximum rewire level is reached or there were no more nodes found that could be rewired successfully on a certain level.

4.1.2 Simulations

A variety of test scenarios has been performed in order to investigate the assumption that RRT^{N*} might deliver a better performance than the original RRT* algorithm. Similar to [4] the test scenarios mainly included simulations within a configuration space $X \subset \mathbb{R}^d$ where dimension $d \in \mathbb{N}$ and $2 \leq d \leq 8$. The configuration space was further restricted by the unit square so that $X = (0, 1)^d$. The initial state $x_{init} = (0.5)^d$ is placed in the center of X.

Figure 4.1 and Figure 4.2 show rapidly-exploring random trees in $X \subset \mathbb{R}^2$ without obstacles and with 25% obstacle coverage, respectively. The results (a)–(d) show the trees generated by RRT* and the multi-level rewiring derivatives for $i = 2..4$, respectively.

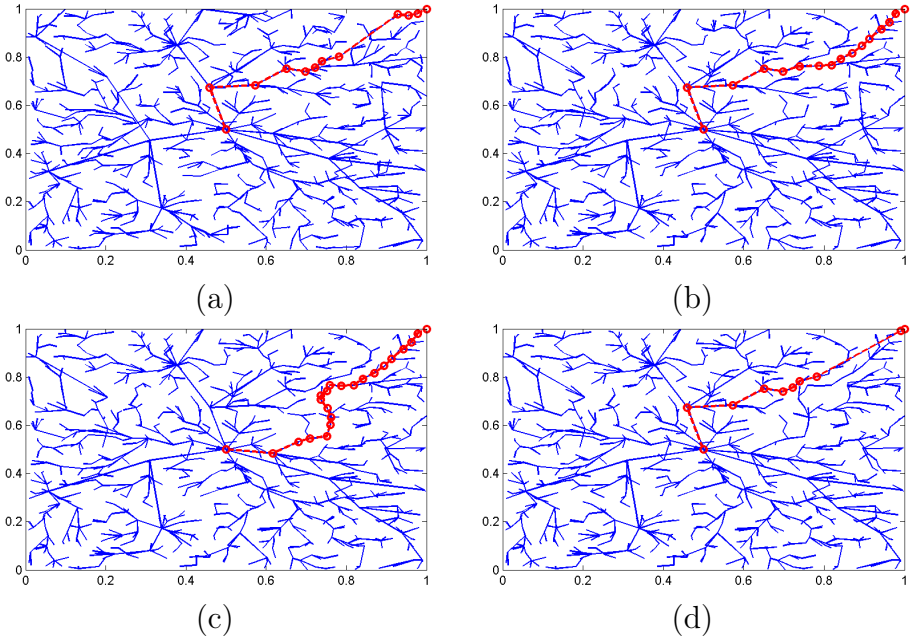


Figure 4.1: Resulting trees for a 2D configuration space without obstacles for RRT* (a), RRT^{2*} (b), RRT^{3*} (c), and RRT^{4*} (d), respectively.

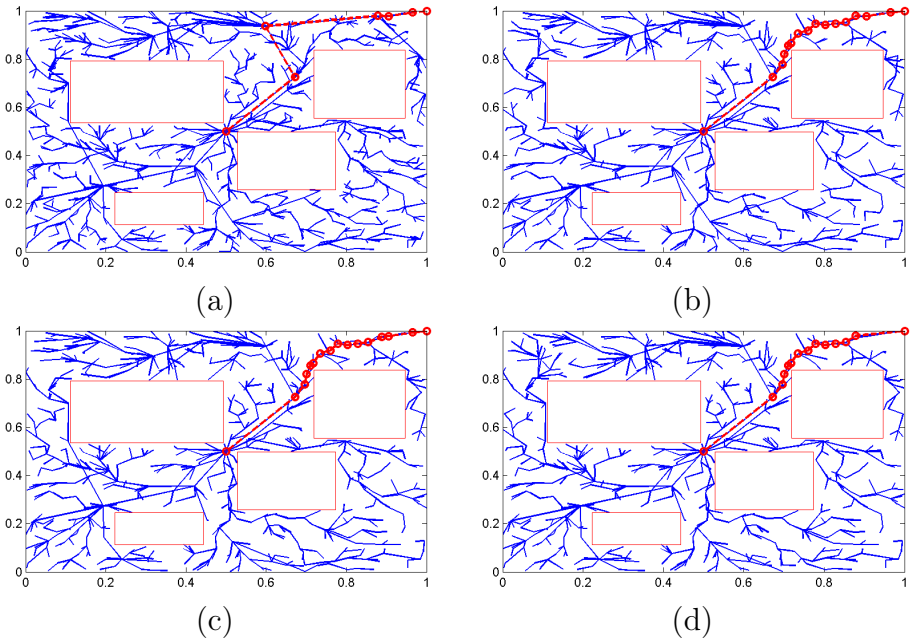


Figure 4.2: Resulting trees for a 2D configuration space cover 25% by obstacles for RRT* (a), RRT^{2*} (b), RRT^{3*} (c), and RRT^{4*} (d), respectively.

4.1.3 Evaluation

The following plots show the performance of the RRT* algorithm in comparison with its multi-rewiring derivatives for an obstacle-free configuration space in multiple dimensions. The results are represented as relative costs over iterations and runtime, respectively. Furthermore the absolute cost difference as well as the performance measured in iterations per second are compared. Since different levels of rewiring were investigated the term RRT^{i*} indicates the application of an i–level rewiring algorithm. RRT^{N*} represents a version of the RRT* algorithm where recursive rewiring is not limited by a specific level. This basically represents an infinite rewiring strategy which terminates when no more neighbors are found that can successfully be rewired.

The results presented in this section were achieved by performing sampling-based motion planning up to a limit of 100.000 iterations. Every version of the algorithm was performed 50 times and the results were averaged. The optimal costs were defined as the direct distance from $x_{init} = (0.5)^d$ to $x_{goal} = (1)^d$. The calculations presented here show the results for 2, 4, 6 and 8 dimensions (see Figures 4.3–4.6) and the sequence of sampled nodes was generated in a deterministic way in order to guarantee the same configuration for all algorithms.

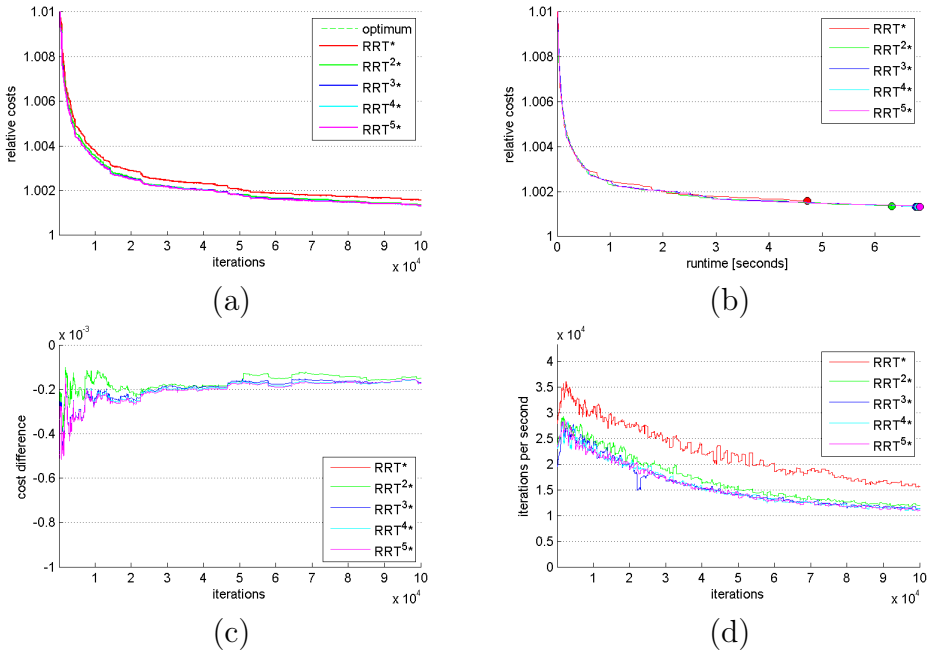


Figure 4.3: 2D: Costs (a), runtime (b), cost difference (c) and performance (d).

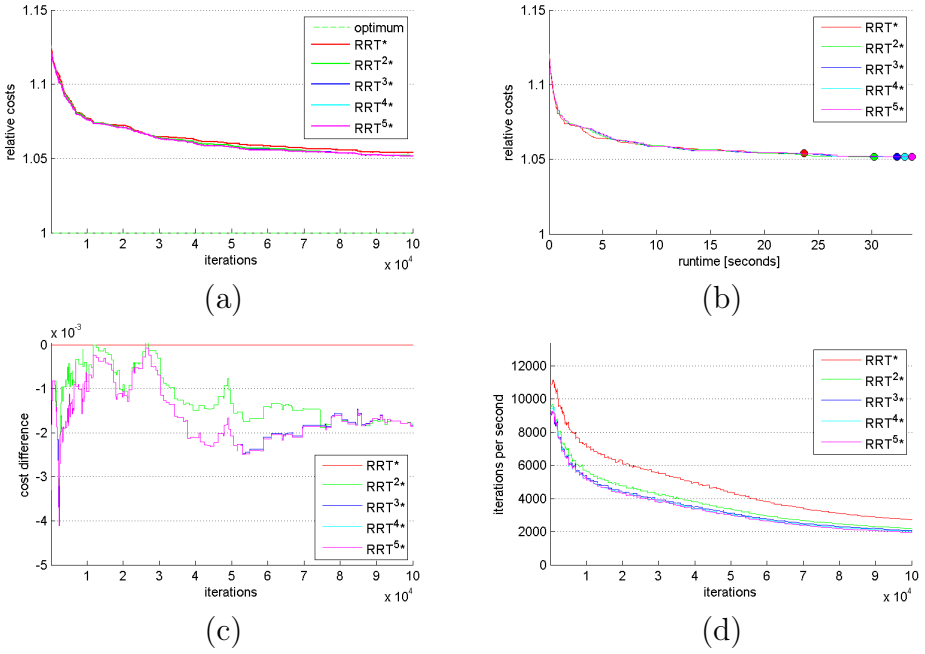


Figure 4.4: 4D: Costs (a), runtime (b), cost difference (c) and performance (d).

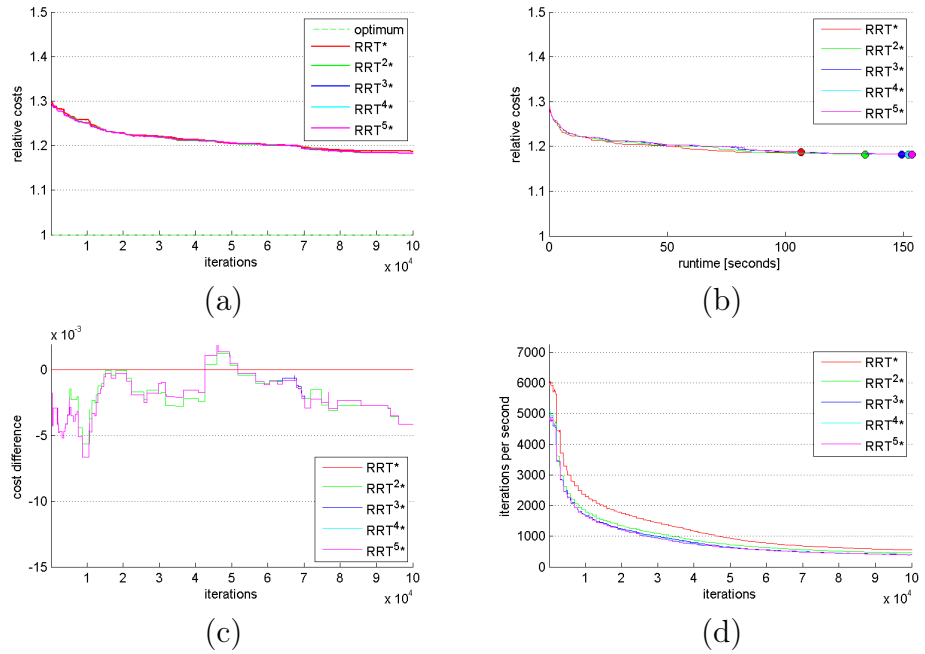


Figure 4.5: 6D: Costs (a), runtime (b), cost difference (c) and performance (d).

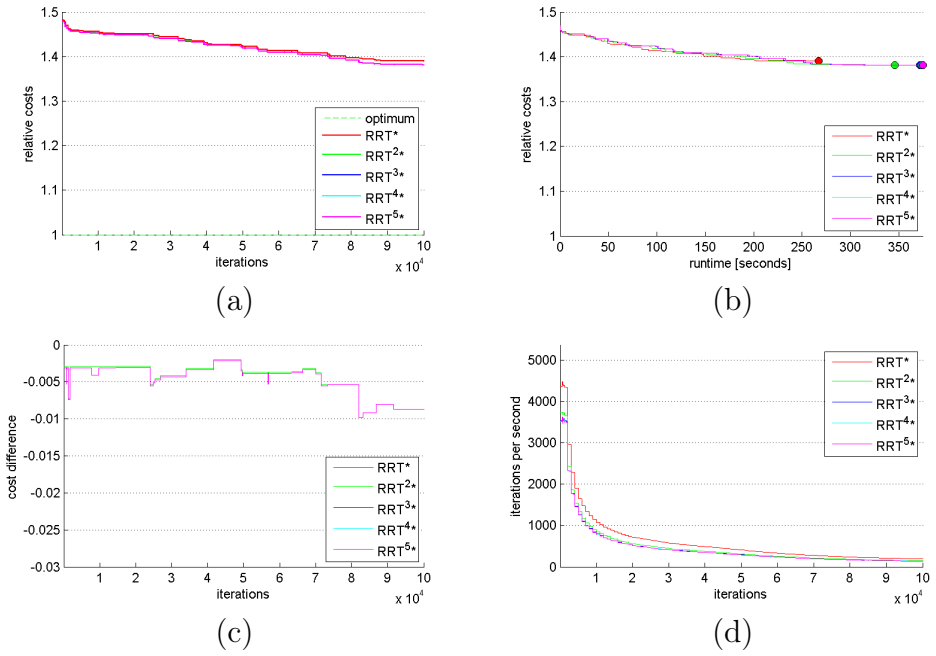


Figure 4.6: 8D: Costs (a), runtime (b), cost difference (c) and performance (d).

The results for the various dimensions basically show similar characteristics in terms of relative costs as well as performance and runtime over costs. While the modified RRT^{N*} derivatives are able to achieve a higher convergence rate towards the optimal costs based on the number of iterations, this advantage basically becomes imperceptible when relative costs are compared to actual runtime. Since the regular RRT* limits the rewiring component to the first level this results in a significantly higher performance in terms of iterations per second. Nonetheless the multi-level rewiring concept shows some improvement in terms of relative costs especially when the number of dimensions gets higher. This might indicate that the original RRT^{N*} algorithm is not asymptotically optimal or that at least the results produced by the versions with multiple rewiring steps converge to the optimal result faster. Unfortunately the current results are not significant enough to allow confirmation whether this assumptions hold or not.

High number of samples

The application of the multi-level rewiring strategy for the RRT* algorithm has also been investigated on a number of 1.000.000 sampled nodes. Due to the exhaustive computation this was limited to 2, 3, and 4 dimensions, in each case with 20 independent runs. Figures 4.7, 4.8, and 4.9 show the evaluation results for these simulations with very high number of samples.

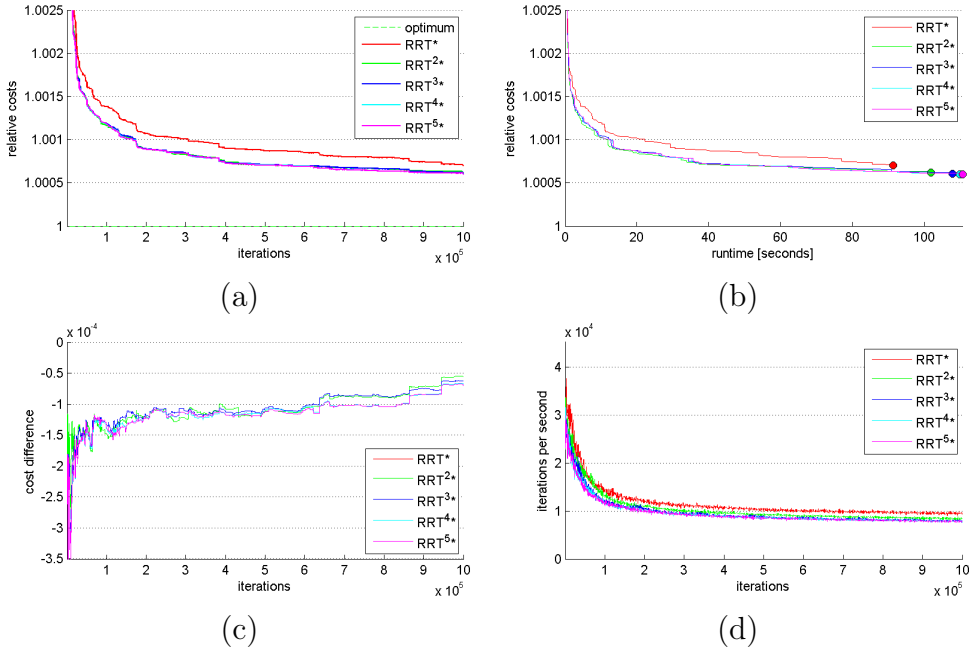


Figure 4.7: 1.000.000 nodes in 2D: Costs (a), runtime (b), cost difference (c) and performance (d).

The results for test runs with up to 1.000.000 iterations seem to confirm the trends that were identified in the previous section. While the convergence rate in terms of node time seems to be higher for multi-level rewiring approaches, the performance in terms of runtime can only be considered to be significantly better for the two-dimensional case where RRT^* seems to converge significantly slower than its derivatives which are able to achieve better result even faster. For the three-dimensional case this advantage seems to become smaller and for four dimensions the runtime performance can basically be considered to be equal. The cost difference shows decreasing and increasing trends for 2D and 3D, respectively, while the 4D case does not allow making assumptions.

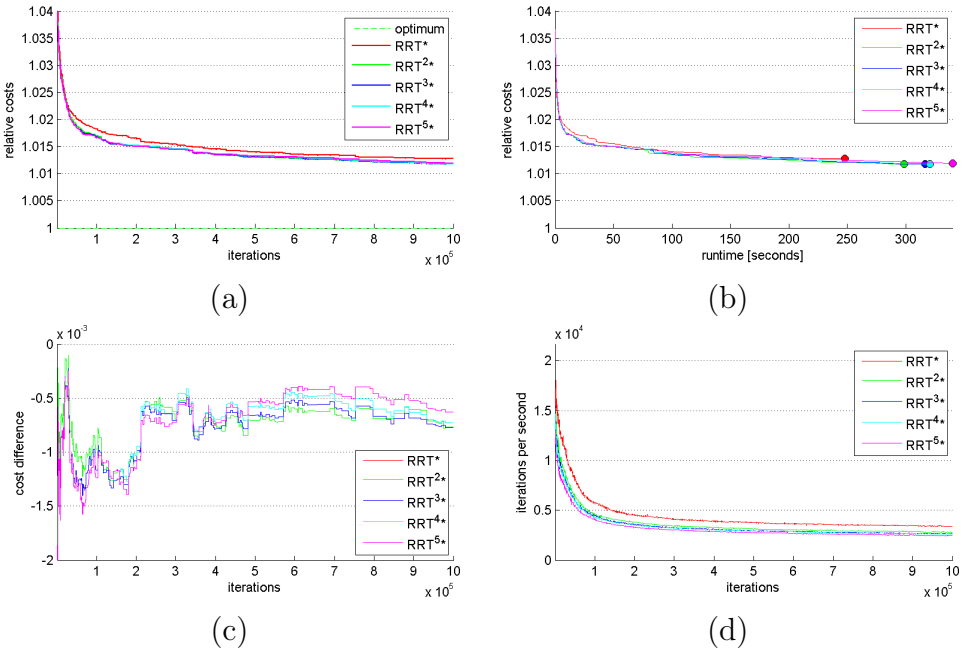


Figure 4.8: 1.000.000 nodes in 3D: Costs (a), runtime (b), cost difference (c) and performance (d).

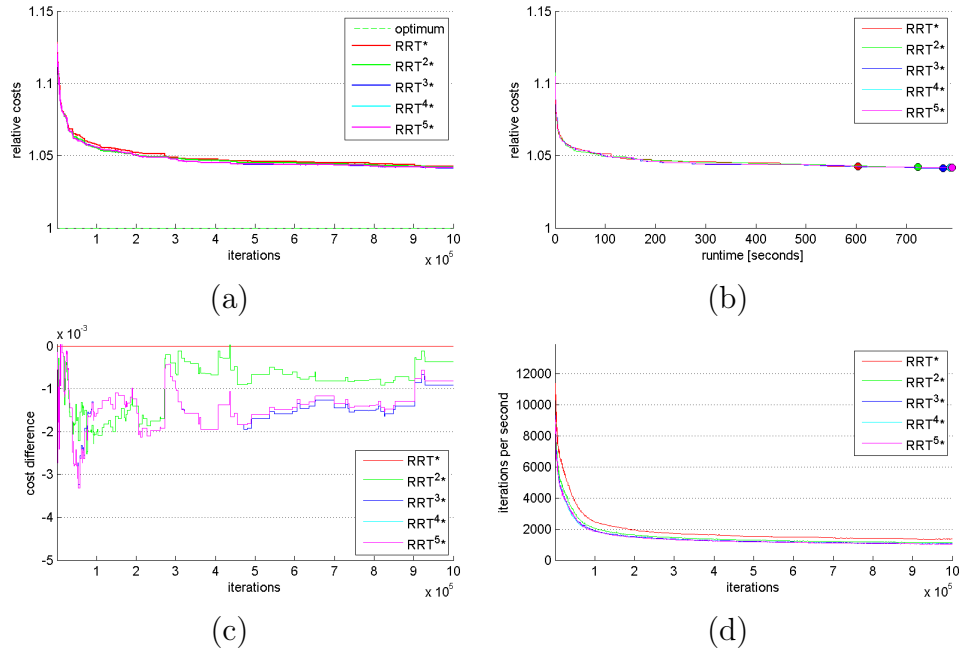


Figure 4.9: 1.000.000 nodes in 4D: Costs (a), runtime (b), cost difference (c) and performance (d).

Simulations with Obstacles

Since sampling-based motion planning algorithms are primarily used in configuration spaces that contain obstacles this section investigates how the multi-rewiring approach influences the performance in such an environment. The following results were investigated for a two-dimensional configuration space that was partially occupied by obstacles similar to the examples shown in Figure 4.2.

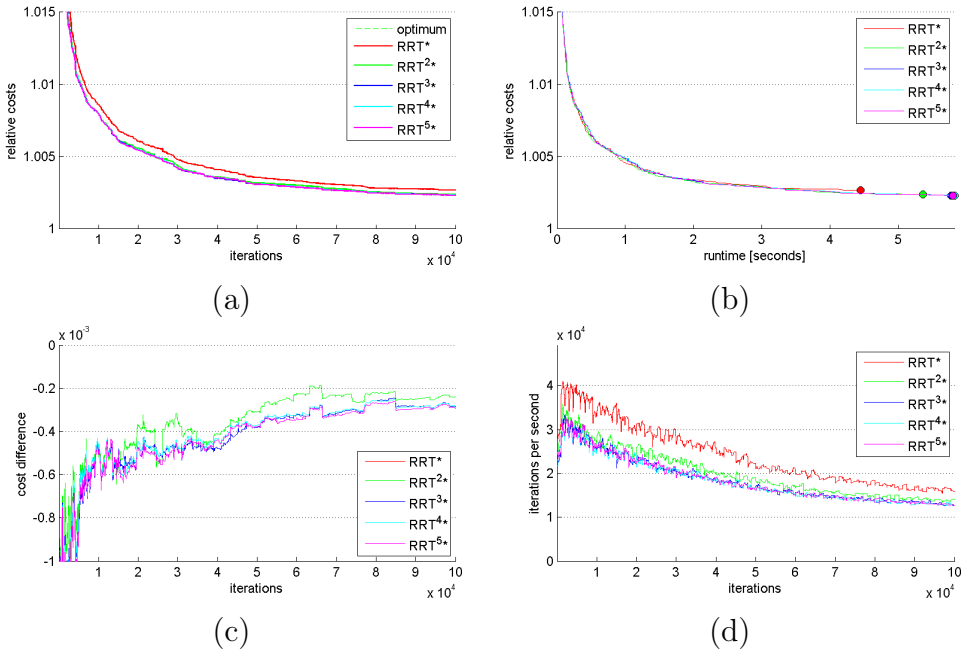


Figure 4.10: 2D with obstacles: Costs (a), runtime (b), cost difference (c) and performance (d).

The results presented in Figure 4.10 basically show the same characteristics as those presented for configuration spaces without obstacles. While the multi-level rewiring approaches tend to need less iterations to achieve better results this advantage cannot be repeated in terms of actual runtime. Future test will have to show whether a higher density of obstacles and configurations spaces that are designed in a more challenging way allow more significant conclusions. Furthermore such configuration spaces containing various constellations of obstacles have to be tested in higher dimensional spaces.

4.2 Conclusion and Future Work

As already mentioned in Section 4.1.3 the numerical analysis of RRT* and especially its multi-rewiring derivatives show some common characteristics throughout all evaluated configuration spaces. On the one hand RRT^{N*} is able to converge towards the optimal solution faster than the original implementation of RRT*. On the other hand the higher computational costs due to the multiple rewiring steps decrease the performance of RRT^{N*}. Although some evaluation results, like e.g. those presented in Figure 4.7 for a two-dimensional obstacle free configuration space, show evidence that RRT^{N*} has advantages in terms of relative costs over runtime this assumption cannot really be confirmed by all performed analyses. Some of the evaluation data even confirms the speculation that a higher number of rewiring steps might even lead to worse results due to "unfortunate" constellation of samples where a rewiring on level i might lead to a better global solution than on level $i + 1$. Nonetheless, basically all analyses showed that in average the quality of results produced by RRT^{N*} in terms of relative costs over iterations/samples surpasses RRT*.

Although RRT^{N*} tends to converge towards the optimal costs faster than RRT* the presented results were not able to distinguish whether or not the claimed asymptotic optimality for both RRT* or RRT^{N*} holds or not. Even the simulations with up to 10^6 samples show some progress in terms of costs for both variations and premature convergence could not be experienced and therefore it seems likely that they are both asymptotically optimal.

Future investigations will have to show if a higher density of obstacles within the configuration space C might have more significant effects on the discrepancy between the investigated algorithms. Furthermore a comparison between RRT^{N*} and another RRT*-based algorithm called RRT$^{\#}$ (RRT sharp) might be of interest. This algorithm introduced in [23] and [24] is also alleged to be asymptotically optimal and furthermore claims to produce a consistent tree after each iteration. According to [23] in this context a consistent tree is given if all nodes that have the potential to be part of the resulting solution path already have minimum cost-to-come values. The potential of a node is determined by a heuristic value that takes into account the costs-to-come value as well as the estimated costs to reach the goal. Although the approach of RRT$^{\#}$ is not directly comparable to the strategy used by RRT^{N*} it also uses some kind of rewiring strategy that is intended to reduce inconsistencies regarding which nodes have the potential to be included in the solution path.

5. Conclusion

The results and experiences achieved during the described research stay at Stanford University have given some valuable insights in the field of motion planning. Especially the mix of practical and theoretical aspects has provided the author with a substantiated understandings of a variety of concepts and algorithms.

The participation in the robotic arm project has shown that the concepts of sampling-based motion planning algorithms can easily be adopted to practical problems. However, the course of the project has shown that the incorporation of hardware components gives rise to a variety of difficulties that are mainly related to the correlation between the physical system and its virtual representation. Furthermore, the intensive software development phase has shown, that a library like OMPL, which was not available at the time the system was implemented first, would have been particularly helpful to implement standard components of the motion planning architecture.

As mentioned before the results achieve by evaluation of the RRT^{N*} were not able to validate whether or not this algorithm significantly improves it's original implementation of RRT*. The extensive simulations show that the convergence rate might be slightly better in basically every tested environment but this advantage becomes irrelevant when the relative costs over runtime are investigated.

In general the experiences in the field of sampling-based motion planning represents a valuable extension to reachability analysis which was the main focus of the authors work in the field of hybrid systems. The capabilities of sampling-based motion planning in high-dimensional configuration spaces could be especially beneficial. Until now the authors research (see [25] and [26]) mainly focused on the performance of numerical techniques like level set methods on a multi-dimensional grid in order to determine the space that can be reached from a certain configuration within a certain time. Sampling-based motion planning might help to overcome the problem of the computationally very intensive reachability analysis.

Bibliography

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

[2] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 4, pp. 566–580, 1996.

[3] R. Bohlin and L. E. Kavraki, "Path planning using lazy PRM," in *Robotics and Automation, 2000. Proceedings. ICRA ́00. IEEE International Conference on*, vol. 1, 2000, pp. 521–528 vol.1.

[4] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *I. J. Robotic Res.*, vol. 30, no. 7, pp. 846–894, 2011.

[5] L. E. Kavraki, M. N. Kolountzakis, and J.-C. Latombe, "Analysis of probabilistic roadmaps for path planning," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 1, pp. 166–171, 1998.

[6] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," Tech. Rep., 1998.

[7] S. M. Lavalle and J. J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," in *Algorithmic and Computational Robotics: New Directions*, 2000, pp. 293–308.

[8] J. J. Kuffner and S. M. LaValle, "RRT-Connect: An efficient approach to single-query path planning," in *ICRA*, 2000, pp. 995–1001.

[9] I. A. Sucan and L. E. Kavraki, "Kinodynamic motion planning by interior-exterior cell exploration," in *WAFR*, 2008, pp. 449–464.

[10] D. Hsu, J.-C. Latombe, and R. Motwani, "Path planning in expansive configuration spaces," 1997.

[11] J. Phillips, N. Bedrossian, and E. Kavraki, "Guided expansive spaces trees: a search strategy for motion- and cost-constrained state spaces," in *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, vol. 4, 2004, pp. 3968–3973 Vol.4.

[12] G. Sánchez and J.-C. Latombe, "A single-query bi-directional probabilistic roadmap planner with lazy collision checking," 2001.

[13] Kavraki Lab, "Open motion planning library: A primer," http://ompl.kavrakilab.org/OMPL_Primer.pdf, 2013, [Online; accessed 18-May-2013].

[14] I. A. Sucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robot. Automat. Mag.*, vol. 19, no. 4, pp. 72–82, 2012.

[15] L. Jaillet, J. Cortés, and T. Siméon, "Sampling-based path planning on configuration-space costmaps," *Robotics, IEEE Transactions on*, vol. 26, no. 4, pp. 635–646, 2010.

[16] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," 1999.

[17] E. Plaku, L. Kavraki, and M. Vardi, "Motion planning with dynamics by a synergistic combination of layers of planning," *Robotics, IEEE Transactions on*, vol. 26, no. 3, pp. 469–482, 2010.

[18] J. Denavit and R. S. Hartenberg, "A kinematic notation for lower-pair mechanisms based on matrices." *Trans. of the ASME. Journal of Applied Mechanics*, vol. 22, pp. 215–221, 1955.

[19] J. H. Halton, "Algorithm 247: Radical-inverse quasi-random point sequence," *Commun. ACM*, vol. 7, no. 12, pp. 701–702, Dec. 1964.

[20] N. Salowitz, Z. Guo, Y.-H. Li, K. Kim, G. Lanzara, and F.-K. Chang, "Bio-inspired stretchable network-based intelligent composites," vol. 47, no. 1, pp. 97–105, 2013.

[21] Z. Guo, K. Kim, G. Lanzara, N. Salowitz, P. Peumans, and F.-K. Chang, "Micro-fabricated, expandable temperature sensor network for macro-scale deployment in composite structures," in *Aerospace Conference, 2011 IEEE*, 2011, pp. 1–6.

[22] F. Mazzini *et al.*, "Tactile mapping of harsh, constrained environments, with an application to oil wells," Ph.D. dissertation, Massachusetts Institute of Technology, 2011.

[23] O. Arslan and P. Tsiotras, "The role of vertex consistency in sampling-based algorithms for optimal motion planning," *CoRR*, vol. abs/1204.6453, 2012.

[24] ——, "Use of relaxation methods in sampling-based algorithms for optimal motion planning," in *2013 IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

[25] W. Pointner, G. Kotsis, P. Langthaler, and M. Naderhirn, "Using formal methods to verify safe deep stall landing of a mav," in *Digital Avionics Systems Conference (DASC), 2011 IEEE/AIAA 30th*, 2011, pp. 5D1–1–5D1–10.

[26] W. Pointner, G. Kotsis, and M. Naderhirn, "General aviation landing assistance using formal methods-based system design," in *Digital Avionics Systems Conference (DASC), 2012 IEEE/AIAA 31st*, 2012, pp. 2C4–1–2C4–9.