Bachelor Thesis

Parkingbehaviour of RC-Cars
Comparison of A* and RRT

by

Tom Bullmann

29th September 2015

Supervisor:

Prof. Dr. Raul Rojas

Prof. Dr. Daniel Göhring

Free University of Berlin

Department of Mathematics and Computer Science

Institute of Computer Science

Contents

Free University of Berlin
Department of Mathematics and Computer Science
Institute of Computer Science

1 Introduction

1.1 Motivation

Driver assistence systems are common usage in newer vehicles and even autonomous cars are advancing further and further each year. Both are intended to help make car travel more secure. Be it warning the driver of getting to close to a car in front of him, giving a signal when crossing the side lane or parking automatically to avoid bumping into a parked car. Autonomous trucks are even driving on the highway in Nevada, USA.

One very essential yet basic feature of autonomous cars is determining the path to take to get from where they are to the target location, preferably without touching any obstacle in the way. This can be done using any one of numerous algorithms. I am going to compare two of those to determine an efficient and stable way to calculate paths without coming into contact with other obstacles.

Using a path finding algorithm for a basic maneuver as parking could be construed as overdoing, as parking is a relatively simple procedure. Yet it does bring advantages compared to a static procedure that executes the parking once the car gets into the right starting position. Probably the most important advantage is that the behaviour of the car is more flexible. All it needs to park is find an appropriate parking spot with enough space to maneuver in without touching any surroundings.

That way it does not matter what kind of parking space is beeing used. Wether it is parallel parking, horizontal or diagonal. As soon as the spot is found, the appropriate path will be calculated and the car parks without incident.

1.2 Carolo-Cup

The Carolo-Cup is an annual tournament held by the TU Braunschweig. Its purpose is a competition of autonomous modell cars in the scale of 1:10 built only by students. In prepa-

ration for the tournament the students learn a lot about constructing and programming an intelligent car able to cope with different situations. To determine which team wins the tournament, three dynamic disciplines and a presentation have to be participated in.

Parallel parking The first dynamic disclipline is parallel parking. In the parking lane are three different sized randomly located spots to park in. The smaller the space that is being parked in and the quicker the parking is done, the more points the team gets. If the car touches any obstacle while parking, drives over the outer lane or parks with more than 5 degrees discrepency to the parallel line, a penalty will be given.

Obstacle free course After parking, the car has to drive as many rounds as possible in 2 minutes. Although there are markings at crossroads, the car does not need to stop. But if the car crosses over the right line with more than one wheel, does not use the braking lights in the correct manner or gets of the track and needs to be driven back by remote control, because it is not able to do it itself, the team gets a penalty. As an added difficulty up to two lane markings might be missing on any given part of the track.

Course with obstacles The last discipline is driving the course while stopping at crossings, evading static and at least one dynamic obsactle in the form of white paper boxes. When passing an obstacle the car needs to blink, otherwise there will be a penalty. The same goes for not stopping at the stopping line, stopping to far away at a crossing, when touching obstacles or taking to long to pass obstacles.

1.3 Thesis structure

Up until this point the paper describes the intend of this thesis. The second chapter introduces a few related works and describes the hardware and software of the used modell car. Chapter 3 explains the basis of the two algorithms that are being compared. After the basis, pseudocode and advantages and disadvantages of both algorithms are explained as well as changes in the implementations that differ from the pseudocode.
The main part of this thesis comes in chapter 4 and 5. Where firstly the experiment is being described with all its circumstances, the results shown as graphics and an explanation

of the graphics. And in chapter 5 the results of the experiment are being discussed. Finally the last chapter concludes the thesis with a brief summarization and a perspective of future additions.

2 Prerequisites

2.1 Related work

The most important related articles are the introduction of A* in 1968 by Hart et al.[1] and RRT in 1998 by LaValle and Kuffner Jr.[2].
To my knowledge there are no works that directly compare A* and RRT. But there has been a direct comparison of Dijkstra's algorithm and RRTs by Knispel and Matousek[3]. In which they found that Dijkstra's Grid is better in a maze type map but RRT is faster when used in an open type map.
But there are several papers on improving RRT and A* for better efficiency and also on comparing A* or RRT with similiar algorithms. As Santoso et al. have shown that A* has a shorter running time compared to Ant or Dijkstra.[4]
Kuwata et al. showed that RRT is a general purpose planning tool when some improvements are made to increase the performance.[5]
As sampling based algorithms have been used a lot in practice and work very well, there are several works that provide improvements on the created solution like Karaman and Frazzoli have show by introducing RRG and RRT*[6] or PRM*.[7] Another work on improving the solution of RRT is from Salzman and Halperin who introduced the LBT-RRT which is asymptotically near optimal.[8]

2.2 Target environment

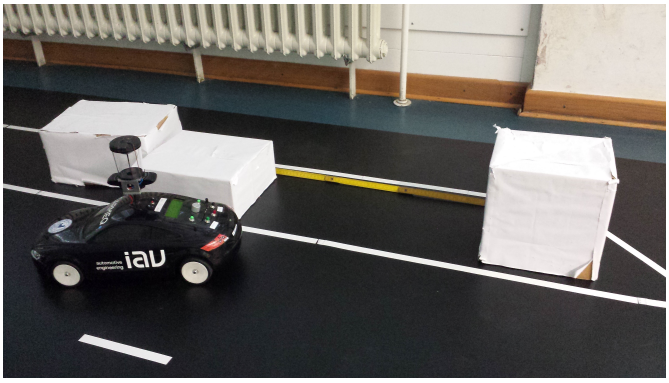
This thesis is being done while being part of the Berlin United Racing Team, which is a group of students that develop a modell car for the usage in the Carolo Cup. At the time of this thesis, there are two modell cars available within the group, but only one of those can actually be used, as the newer built car is currently not able to provide enough power for its built in components.

The next sections will describe the used hard- and software of the car.

2.2.1 Hardware

The hardware has already been described by Boldt and Junker[9], therefore I will only name some basics.
The chassis of the car is the Xray T3 Spec. 2012, which uses as main processing unit an ODROID-X2 with 1.7 GHz ARM Cortex-A9 Quad Core CPU and 2 GB RAM with multiple USB 2.0 slots, a 100 MBit/s Ethernet slot and UART. Through the USB-slots we added a USB-WLAN-Adapter which is the main way we communicate with the car. Through the serial UART-port the main CPU is connected with the team internally developed OttoBord V2. The OttoBoard consists of a STM32 F4 microcontroller with a 168 MHz ARM Cortex-M4F processor. With the OttoBoard we control a few hardware components.
The movement of the car comes from an electric motor that is able to go up to 4 m/s in combination with the transmission. A steering servo is used to steer the front axis. On the OttoBoard we have a gyroscope, an accelerometer and a magnetometer. Remote control with the modell car can be established through a 40 MHz receiver. There are also turning and breaking lights simulated with LEDs, as well as one LED that clearly signals the active usage of the remote control, as required for the Carolo Cup.
At the back side of the car are some controls to set the current driving strategy and to activate the car and the motor or to calibrate some hardware components. The probably most obvious about our car is the omnidirectional camera which is placed i the center of the car and rises up. There is a usual web cam used with a parabolic mirror for the omnidirectional view. A complete view of the car can be seen in figure 2.1

Figure 2.1: Complete view of the car



2.2.2 Software

The ODROID uses an ARM-Linux, which comes with a lot of standard libraries and offers a lot of easy of use out of the box. The software that is developed by the students is written in C++ and cross compiled with GCC for ARM, to be transfered from the computer which are used to develop to the car.

The microcontroller uses software written in C. The software developed for the Carolo CUp is being developed with the help of the BerlinUnited Framework, which has been created as a collective work from the Humbold-University Berlin and the Free University of Berlin for the Robo-Cup Football robots.

The framework uses a blackboard system in which the source code is divided into modules and representations. Representations provide data for other modules to use. While the modules are used for processing the data. According to the requirements of the modules and representations an execution chain is being created which will be executed in a sequential manner. The execution depends on the current frame rate, for 30 frames a second one execution cycle is 33 milliseconds and in that time all necessary calculations have to be finished.

3 Algorithmic basis

3.1 A*-Algorithm

The A* algortihm is an heuristic path searching algorithm, which allows for an optimal result from any given starting node s to a target node t. A* is an informed search algorithm and requires that all edges in the graph have positive costs.

3.1.1 Explanation

A* can be seen as an extension of the Dijkstra algorithm in that it uses an evaluation function $f(n) = g(n) + h(n)$ where

- $f(n)$ is the approximated cost for the optimal solution through n
- $g(n)$ is the cost from the starting node s to this node n and
- $h(n)$ is the estimated cost of the optimal path from n to the target node t.

The A* algorithm uses two lists in which the already expanded nodes(closed) and those yet to explore(open) are stored. At the start the node from which to start will be added to the open list. After that a loop is started that ends either if all nodes have been visited and no path is found or when a path has been found. Whitin this loop the node with the lowest $f(n)$ from the open list is taken. That node is checked if it is the target node, and if so the algorithm ends, or it will be added to the closed list and then be expanded.

When a node is expanded, all successor nodes of that node will be checked. Should a successor node already be in the closed list, the next successor will be checked. Otherwise $g(n)$ of the node that is currently being expanded will be added to the cost from that node to the successor node. If the successor node is already in the open list and the just calculated $g(n)$ of the successor is greater or equal than that already calculated, the next successor will be checked. If that is not the case, the parent of this successor is being set to the node

Free University of Berlin
Department of Mathematics and Computer Science
Institute of Computer Science

currently being expanded and the $g(n)$ and $f(n)$ will be set with the just calculated $g(n)$ and the heuristic estimate $h(n)$. Should that successor already be in the open list, its values will be updated, otherwise it will be added.

The heuristic $h(n)$ has to fullfil one of the following attributes:

- admissible
  If $0 \leq h(n) \leq h_t(n)$, where $h_t(n)$ are the unkown real costs of the optimal path from n to t, the heuristic is an addmissible one. That means, $h(n)$ never overestimates the cost from n to t.

- monotone
  When $h(n_i) - c(n_i, n_j) \leq h(n_j)$ applies and a heuristic is admissible, it is also monotone. Where n_j is the successor node of n_i and $c(n_i, n_j)$ is the cost from n_i to n_j and $h(n_i)$ and $h(n_j)$ are the heuristic estimates of the nodes n_i and n_j.
  In short, the heuristic estimate either remains equally well or gets better throughout the search.

3.1.2 Pseudocode

Input:

- G : graph

Output:

- path from starting node s to target node t

```
1: procedure ASTAR(G)
2:     init OPEN
3:     init CLOSED
4:     OPEN.add(startNode, f_value)
5:     while OPEN.isNotEmpty() do
6:         currentNode = OPEN.removeMin()
7:         if currentNode == targetNode then
```

```
        return PathFound
    else
        CLOSED.add(currentNode)
        expand(currentNode)
    end if
  end while
  return PathNotFound
end procedure


function EXPAND(currentNode)
  for each successor of currentNode do
    if CLOSED.contains(successor) then
      continue
    end if
    tentative_g = g(currentNode) + c(currentNode, successor)
    if OPEN.contains(successor) & tentative_g ≥ g(successor) then
      continue
    end if
    successor.predecessor = currentNode
    g(successor) = tentative_g
    f = tentative_g + h(successor)
    if OPEN.contains(successor) then
      OPEN.decreaseKey(successor, f)
    else
      OPEN.enqueue(successor, f)
    end if
  end for
end function
```


Free University of Berlin
Department of Mathematics and Computer Science
Institute of Computer Science

3.1.3 Characteristics

Let c_{op} be the costs of the optimal path. Then the following statements are valid:

- A* expands all nodes n where $f(n) < c_{op}$
- A* expands some nodes n where $f(n) = c_{op}$
- A* expands no nodes n where $f(n) > c_{op}$

The effective branching level of A* is b:

- let N be the number of nodes explored by A*
- let d be the depth of the optimal path from s to t
- let b be the branching level, which a uniform tree of the depth d needs to store all nodes N

  Then $\sum_{i=0}^{d} b^i = N$ applies.

Hint:

- The closer $h(n)$ is to the real costs from s to t, the smaller b gets.
- If $h(n) = 0$ then A* is reduced to a simple breadth-first-search.

Advantages

- Completeness

  Should a way from start to goal exist, A* will find a solution.
- Optimality

  The A* algorithm is optimal, so it will find an optimal solution if at least one exists.
- Optimally efficient

  No other algorithm exists, that can guarantee to expand less nodes than A* does. Because if such an algorithm were to not expand all nodes with $f(n) < c_{op}$, that algorithm would risk to miss the optimal solution.

Disadvantages

- Space complexity

  The space complexity for A* is $O(b^d)$, where b denotes the level of branching and d is the depth of the resulting optimal path. b is highly dependent on the used heuristical function, the more accurate the heuristical functions result is, the lower is b. So if the heuristical function were to return the exact amount of cost from the currently visited node to the goal node, b would be 1.

- Efficiency versus accuracy

  When using A* in time sensitive applications like real time environments a trade-off between a more accurately generated graph or a faster calculation time has to be made. As the graph gets bigger the algorithm has to generate more nodes and calculate the costs of successor nodes more often, including a possibly very costly heuristical approximation.

  If the given time constraints are too sparse and A* is not able to finish, there will be no result. As A* either returns an optimal path or nothing at all.

3.1.4 Implementation specific details

My implementation of the A* algorithm uses two sets for storing the nodes to avoid duplicate entries as there is no complete graph to traverse but the nodes have to be generated on the fly which could mean duplicate nodes can be created multiple times. The reason I do not use a priority queue as is usually suggested is, that the extraction of the entry with the lowest $f(n)$ happens only once per iteration while testing for existing entries happens multiple times.

As mentioned there is no complete graph to traverse in this context, which leads to the necessity to create the graph on the fly. This happens at the beginning of the expand-function, a predetermined number of nodes is being created for nodes in front and back of the currently expanding node taking into account a maximum step size and the turning angles the car is capable of.

When the algorithm has finished and a path has been found, the last node that was in the vicinity of the target node is saved in an extra variable to be able to extract the path by

traversing the parent nodes until the starting node has been reached.
Aside from that the implementation is analogous to the pseudocode.

3.2 RRT-Algorithm

The Rapidly exploring Random Tree algorithm is an algorithm used for path and motion planning in holonomic, nonholonomic and kinodynamic environments, which is perfect for our nonholonomic system. RRT is a Monte Carlo algorithm (probalistic algorithm), due to the fact that it adds randomly selected points inside a configuration space and stops after a finite time.

3.2.1 Explanation

The Rapidly exploring Random Tree algorithm is an algorithm that iteratively expands an existing tree structure. During every iteration step the algorithm picks a new randomly selected point within the configuration-space.

In order to calculate a new input configuration the nearest neighbour has to be selected. From this nearest neighbour a certain distance towards the randomly generated point is moved. Usually Dubins Curves are used to create the path from the nearest neighbour to the random point and that path is then being shortened to the maximum distance per step. If that path crosses an obstacle, the path is reduced as much as necessary to avoid the obstacle. With those informations a new point is created to add to the tree of already known points. While the predetermined number of iterations is not reached, the algorithm adds new points to its tree in every iteration.

3.2.2 Pseudocode

Inputs:

- $tinit$: initial configuration
- K : number of vertices
- Δd : incremental distance

Output:

- T : tree

1: **procedure** RRT($tinit, K, \Delta d$)
2: $T.init(tinit)$
3: **for** $k = 1$ *to* K **do**
4: $x_{rand} \leftarrow RAND\_CONF()$
5: $x_{near} \leftarrow NEAREST\_NEIGHBOUR(x_{rand}, T)$
6: $u \leftarrow SELECT\_INPUT(x_{rand}, x_{new})$
7: $x_{new} \leftarrow NEW\_STATE(x_{near}, u, \Delta d)$
8: $T.addVertex(x_{new})$
9: $T.addEdge(x_{near}, x_{new})$
10: **end for**
11: **return** T
12: **end procedure**

3.2.3 Characteristics

Advantages

- Space complexity

  RRT only adds as much points to the tree as it has iterations. That leads to a space complexity of $O(K)$ where K is the number of iterations.

- Iterative use

  If necessary RRT can be used iteratively . Which means it can be run for a fixed or dynamically determined amount of time and then use a feasible path. That way a part of the complete path can be determined, instantly used and then appended in the next calculation circle.

- Probalistic approach

  Due to the probalistic nature of RRT the number of run iterations can be predetermined by calculating after how many iterations a certain probability of finding a path to the goal is reached.

- Incompleteness

  After RRT has finished its K iterations there will be a path that can be used, even if the path is neither optimal nor complete, but there will be a usable path towards the goal.

- Non-greedy

  As RRT is even capable to find a path if a greedy algorithm fails to do so, for example when running straight inside an u-shaped obstacle.

Disadvantages

- Completeness

  RRT is only probalistically complete, which means, the probability of RRT for solving a given solvable problem reaches 1 if the running time goes to infinity.

- Optimality

  As RRT generates points randomly and adds them to the nearest neighbour, the found path, if any, unfortunately is not guaranteed to be optimal. In fact, the probability for finding an optimal path with an unimproved RRT goes towards 0.

- Optimally efficient

  RRT is not optimally efficient. One reason is that the chance of even finding an optimal path is almost zero.

  Also when RRT is used without any bias in generating the random points, the number of nodes explored until finding a path varies heavily. As such adding any heuristic function will have a big influence on decreasing the number of nodes explored until a path is found. So even if RRT were to find an optimal path, the explored nodes until that happens varies within multiple attempts of the same algorithm anyway.

- Step size

  Depending on the problem that is to be solved, determining the best step size is a problem. In a parking context for example. A too big step size can lead to unmaneuverability inside the parking space. On the other hand a too small step size can drastically increase the number of iterations necessary.

- Collision detection

  The process of detecting a collision can be quite complicated. Taking into account the direction the object is heading towards, the objects dimensions, velocity and the dimension the object is moving in can complicate the necessary calculations for checking if the new point is inside or passes through an obstacle.

3.2.4 Implementation specific details

As the intended use case for this implementation is different parking situations, the configuration space that is to be searched is quite small. To not create an unnecessary overhead while searching the path to park, I used a simple list instead of a kd-tree, because it is to assume that the number of required iterations to find a path is not that big.

In my implementation of RRT I made a few adjustments to improve on the performance. To influence the randomly generated points (in line 4) towards the target node, another randomly generated number, from a new configuration space surrounding the randomly generated point and the target node, is being added to the randomly generating point.(as shown in the figure on the right) That way a heavy bias towards the target node is added and the chance of finding a path in as less as possible iterations is increased immensely.

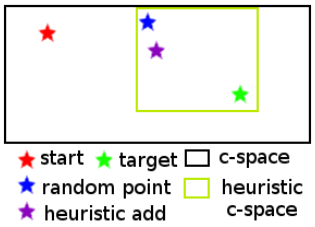


Figure 3.1: RRT with heuristic

For a faster retracing of the found path, if any path is found, the parent for each added node is being added (after line 7). This allows for a faster iteration of the path after finding the nearest neighbour.

The last adjustment made is a check if the newly created point that has been added is the target node or in a predetermined small vicinity of it. Should that check return a true statement, the algorithm ends. As the algorithm always adds newly generated and calculated points to the nearest neighbour that is already known, the chance for improving the path is very small. On the other hand it improves greatly on the runtime of the algorithm, as long as the path is found before the maximum iterations are done.

For evaluation purposes the implementation includes a flag to decide wether the original RRT or the slightly improved RRT with a heuristic addition is being used.

4 Experiment

4.1 Goal

The purpose of this experiment is to determine which variation of the implemented algorithms is the best to be used for parking the car.
For that purpose I tested three different variations of A* and nine variations each of RRT without improvements and RRT with an additional heuristic estimate.

I suspect that RRT with an heuristic estimate will be the fastest algorithm, as the used heuristic is heavily biased and the tree grows very likely towards the target node. With that heavy bias on the generation of random points RRT should be more efficient than A*, which needs to build its graph that is to be traversed on the fly, and it is very likely to be much faster than RRT without improvements. As RRT without improvements always runs a certain number of iterations without any guarantee that a result will be returned, due to the randomly selected points being uniformly distributed within the complete configuration space.

This experiment is done within the given context of parking a modell car of the scale of 1 to 10 and with 100 tries for each setting to ensure that a mean can be calculated.

4.2 Setup

This experiment consists of three different settings in which three algorithms with different step sizes and maximum number of iterations for RRT will be tested. Following the general circumstances of the three settings and what is to be tested will be explained.

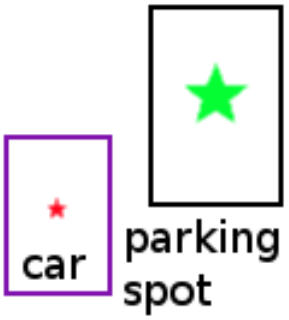
4.2.1 Setting

The parking space is 56 centimeter long and 30 centimeter wide and the car is 42 centimeter long and 19 centimeter wide in all three positions of the experiment.

Position 1

In position one the car has to find a way into the parking spot while standing slightly behind the parking spot, as seen in figure 4.1. The car is positioned 6 centimeter left of the parking space and the back of the car is 19 centimeter behind the parking space.

Figure 4.1: Position 1



Position 2

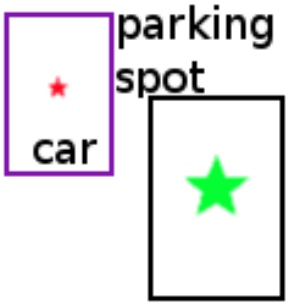
For the second position the car has to find a way into the parking spot that is right beside it, as seen in figure 4.2. The distance between the car and the parking space is 6 centimeter and the car has 7 centimeter vertically to each edge of the parking space.

Figure 4.2: Position 2

Position 3

For the last type of test the car is also 6 centimeter besides the parking spot and has 17 centimeter from the back of the car to the end of the parking spot, as in figure 4.3.

Figure 4.3: Position 3



4.2.2 Method

The process of testing is the same in all three settings. The car needs to find a way from its starting point to the target point. The starting point is marked by the red star in figures 4.1 to 4.3 whereas the target point is actually a target area of a 5 centimeter radius around a predefined target point inside the parking spot. This is due to A* having to create the successor nodes on the fly with a fixed step size, so that there is no chance of accidentally missing the target because the step size is too big and the starting point is unfortunate. As this tests are done with the modell car, there is one restriction that applies to the movement of the car. Which is the restriction of moving either in a straight line ahead or in a circular way within plus or minus 20 degrees, right and left respectively. This restriction has been included in the implementation of the algorithms to create new nodes, that way the calculated path from any of the used algorithms already contains points that can actually just be used without further manipulations.

Free University of Berlin
Department of Mathematics and Computer Science
Institute of Computer Science

4.3 Data and explanation

In this section the data from the experiments will be shown in different graphics, followed by an explanation of the seen content. The time that is displayed on the following graphics is shown in seconds, in three instances a multiplier is on top of the axis, which is 10^{-3}. A vertical line inside the diagram represents the given time constraints of 33 ms that apply in practical usage. The legends of the used graphics differentiate between the three algorithms by naming A*, RRT and the improved RRT A*, RRT and hRRT respectively.

The bar diagrams display how much runs in percent of each algorithm, without dividing into step size and maximum used iterations, have been able to find a path, have not been able to find a path and how much completed in less than 33 ms, which is relevant for usage within the given framework.

The box diagrams show how close the closest point to the target node was after finishing the maximum number of iterations. The red line within the box is the mean of all tries that did not find a solution before reaching the maximum iterations. Within the blue box are almost 50% of the data, while 25% each are within the grey dotted lines and the red pluses are some singular occurences outside the norm.

Position 1

(a) step size 50mm (b) step size 75mm (c) step size 100mm

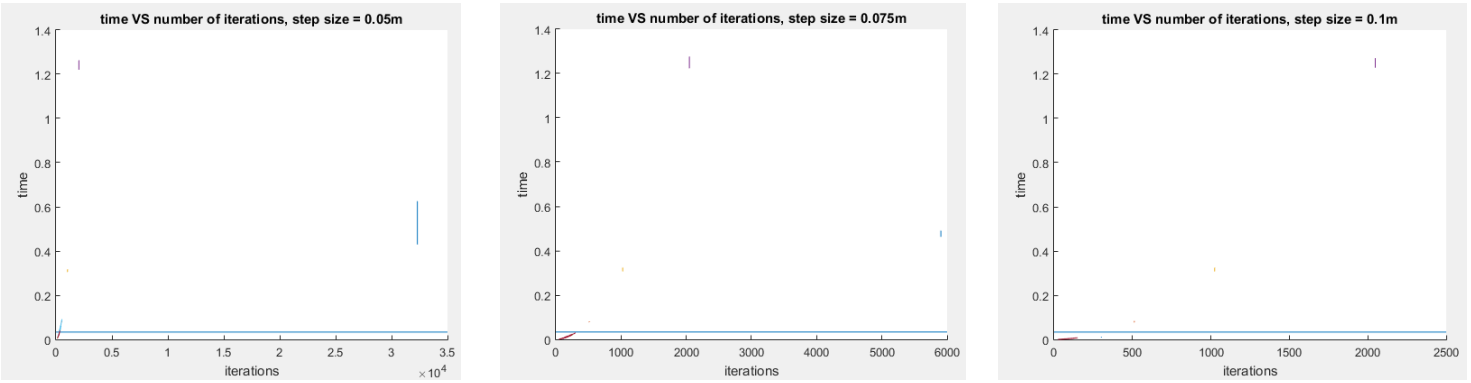


Figure 4.4: Position 1: Time in relation with iterations used during the algorithm, complete data

The figures 4.4a, 4.4b and 4.4c are the complete data regarding time of all tests run in this position with their respected step size.

Figure 4.4a shows that A* is run between 0.4 seconds and a little more than 0.6 seconds while using more than 32 thousand iterations to find the path. RRT needs more than 1.2 seconds to run 2048 iterations and about 0.3 seconds to run 1024 iterations. RRT with 512 iterations and the improved RRTs are explained in figure 4.5a.

In figure 4.4b A* runs approximately 0.5 seconds and uses just under 6,000 iterations. RRT runs for more than 1.2 seconds in 2048 iterations, 0.3 seconds with 1024 iterations and 0.1 second for 512 iterations. The improved RRT versions are explained in figure 4.5b.

In contrast to the two step sizes before, 100 millimeter step size leads to A* finishing in just under 25 milliseconds. Whereas RRT again needs 1.3 seconds,0.3 seconds and 0.1 second for 2048, 1024 and 512 iterations respectively. The imoroved RRTs will be shown in figure 4.5c.

Free University of Berlin
Department of Mathematics and Computer Science
Institute of Computer Science

(a) step size 50mm (b) step size 75mm (c) step size 100mm

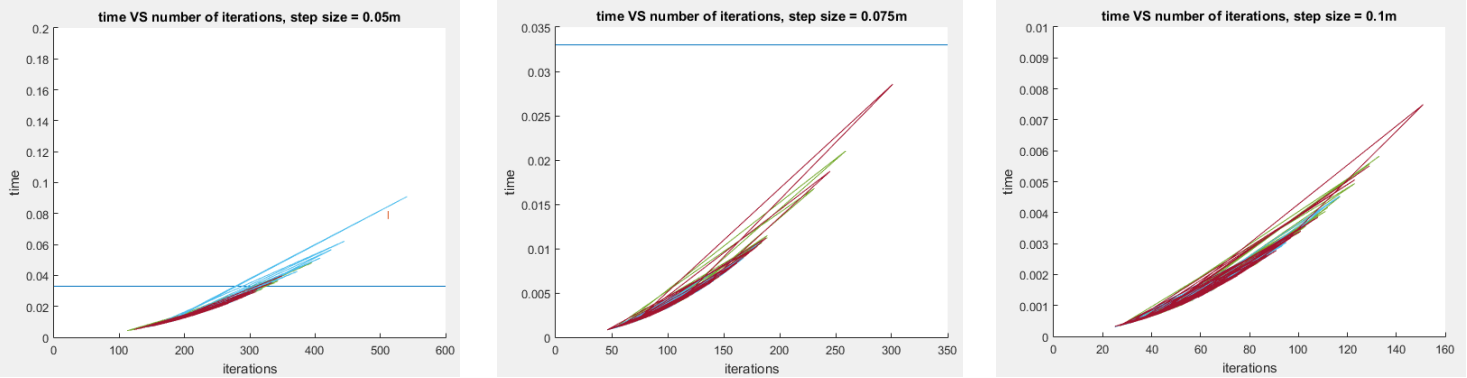


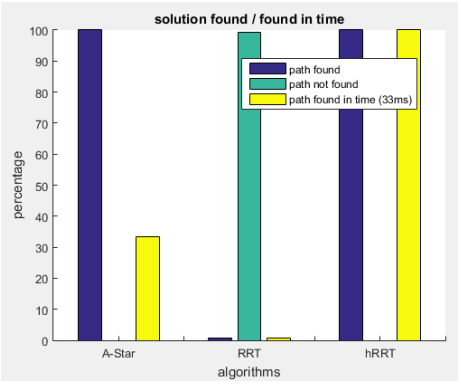
Figure 4.5: Position 1: Time in relation with iterations used during the algorithm, relevant data for real time usage

The above graphics display all relevant data from the complete time comparison that are within or very near the time constraint of 33 ms.

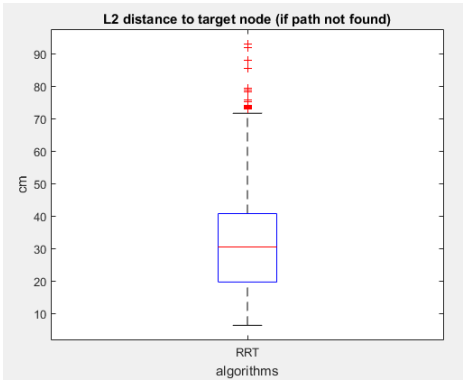
The entry with the lowest time in figure 4.8a has around 5 ms with 110 iterations and from about 280 to 325 iterations used all algorithms exceed the time limit. Noteworthy is the fact, that RRT with 512 iterations finishes in 80 milliseconds and is close to the improved RRTs when comparing the time only.

Figure 4.8b has no entries exceeding the time limit. The smallest time needed to conclude the path finding is as small as 1 ms for just under 50 iterations. The highest number of used iterations is 300 and the time is under 30 milliseconds.

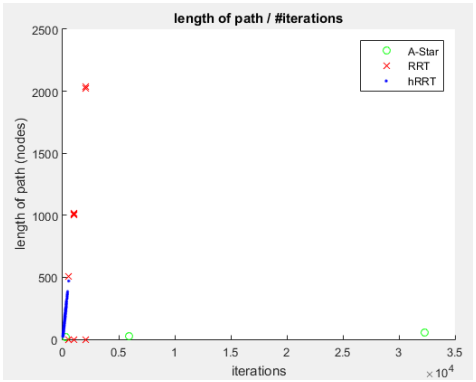
For 100 mm step size the time is even better than with 75 millimeter step size. Ranging from under half a millisecond with 25 iterations to complete and find a path to 150 iterations in 7.5 milliseconds, as seen in figure 4.8c.



(a) solutions found, not found and found in time in percent



(b) L2 distance of nearest node to target node if no path found



(c) path length in nodes with related iterations used to find

Figure 4.6: Position 1: path found, L2 and path length

As can be seen in figure 4.6a, A* always finds a path, but only in about 33% of the time within the given time frame. Which translates to one of the three step sizes found a way in time and the other two did not, as explained above and seen in figures 4.4a, 4.4b and 4.4c. RRT has had almost no success in finding a path, but in the instance it did, it was within the time limit. The improved RRT always found a way in time.

In figure 4.6b most of the closest points near the target are 20 to 40 centimeter away and on average around 30 centimeter, some are getting as close as 6 or 7 centimeter, which is pretty close and alsmost in the target area.

In figure 4.6c are the number of nodes that the path consists of and how much iterations it took to get them for all three algorithms. For A* the number of nodes in the final path is always very small, but the number of iterations it took to get the path varies strongly with the step size used.

RRT without imrovements has both no path length, as no path has been found in that instance, and a path length which is near the maximum number of iterations. While RRT with a heuristic addition has varying path lengths that begin around the same size as A* has found and go up to almost 500. But in contrast to the normal RRT the number of iterations that have been used is never higher than 512.

Position 2

(a) step size 50mm (b) step size 75mm (c) step size 100mm

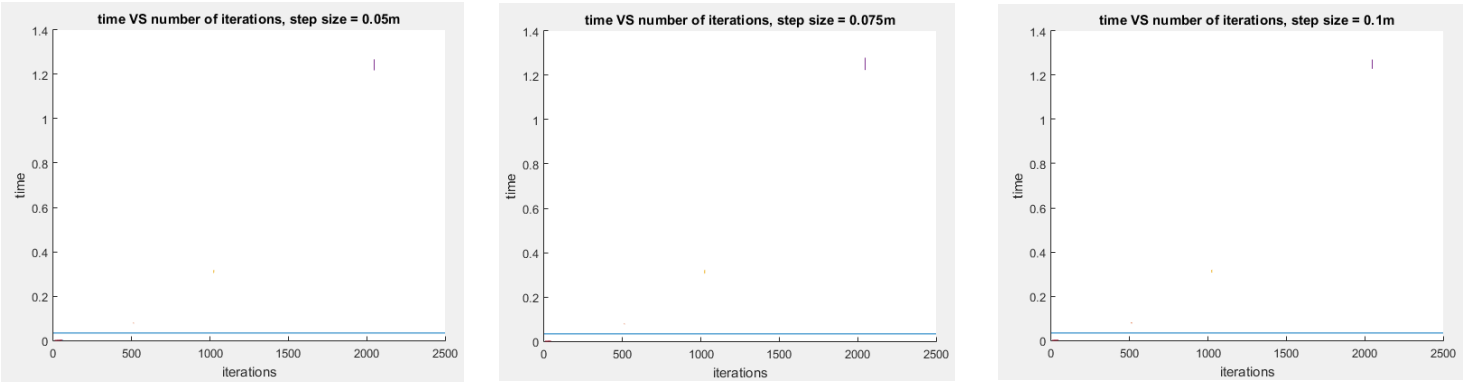


Figure 4.7: Position 2: Time in relation with iterations used during the algorithm, complete data

In figure 4.7a, 4.7b and 4.7c a complete display of time taken for the different algorithms and their respected step size is being shown.
Figure 4.7a, 4.7b and 4.7c all show RRT with 2048 iterations runing under 1.3 secods, RRT with 1024 iterations needing just above 0.3 seconds and with 512 iterations 0.1 seconds are necessary. The improved RRT versions are all under the time limit and will be shown and explained in figure 4.8a, 4.8b and 4.8c.

(a) step size 50mm (b) step size 75mm (c) step size 100mm

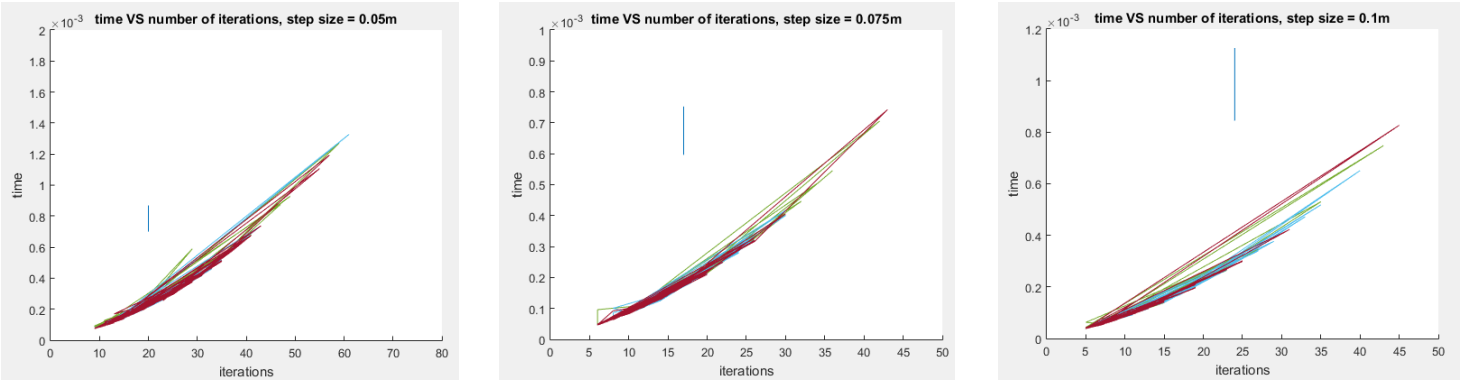


Figure 4.8: Position 2: Time in relation with iterations used during the algorithm, relevant data for real time usage

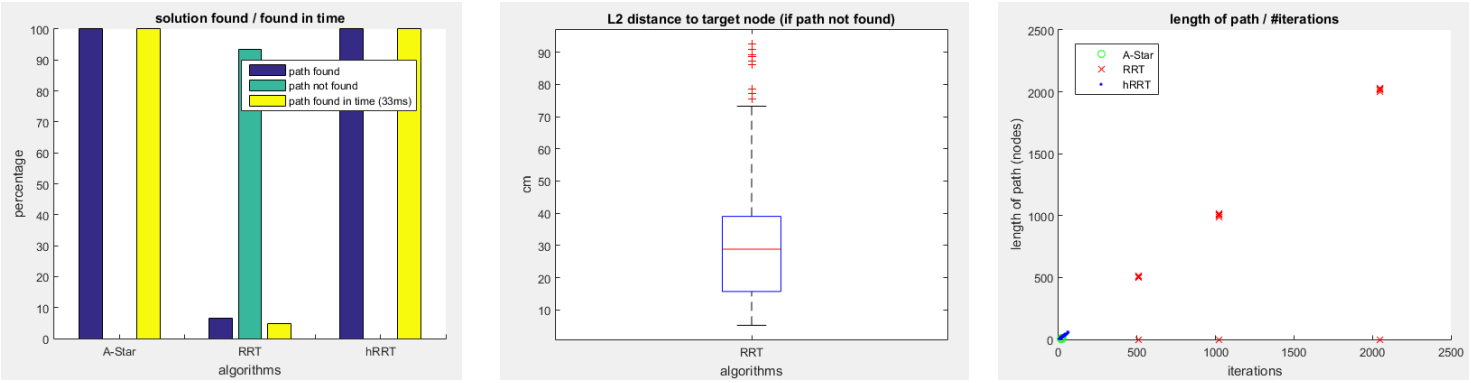
The above figures show all data for the tested algrithms that ended within the given time

constraint of 33 ms.

The entry with the lowest time in figure 4.8a has around 0.1 ms with 10 iterations. In this test A* finished within 33 ms for the step size of 50 millimeter and can compete with the improved RRTs.

For 75 millimeter step size the times get better compared to 50 millimeter step size, as shown in figure 4.8b. With only 6 iterations in 0.05 milliseconds the improved RRT has found a solution. The highest time necessary is 0.8 milliseconds for A* and some variants of the improved RRT with 17 and 43 iterations respectively.

In this instance A* needs a bit more time than with the smaller step sizes in this test. For 25 iterations 0.8 milliseconds to 1.1 milliseconds are necessary to find a path with A*. While the improved RRT is rouhgly the same as with 75 millimeter step size. Starting with 0.03 milliseconds for 5 iterations and ending with 0.8 milliseconds for 45 iterations, as seen in figure 4.8c.



(a) solutions found, not found and found in time in percent

(b) L2 distance of nearest node to target node if no path found

(c) path length in nodes with related iterations used to find

Figure 4.9: Position 2: path found, L2 and path length

In figure 4.9a A* finds a path every time in the given time constraint. The success rate to find a path of RRT has improved to nearly 10%, but not all of those finished in time. The improved RRT has always found a way in time.

The majority of the closest points is within 15 to 40 centimeter with an average of 29 centimeter, as seen in figure 4.9b. And again some came as close as 6 centimeter to the target, which is almost the target area.

Free University of Berlin
Department of Mathematics and Computer Science
Institute of Computer Science

As figure 4.9c shows, A* has found a very short path in very little iterations, and there are no differences between the step sizes. The same goes for the improved RRT where the length of the path varies from as little as A* has found to not even 100 nodes in the path. But RRT without improvements had a path length of around the same size as its maximum number of iterations if a path was found at all.

Position 3

(a) step size 50mm (b) step size 75mm (c) step size 100mm

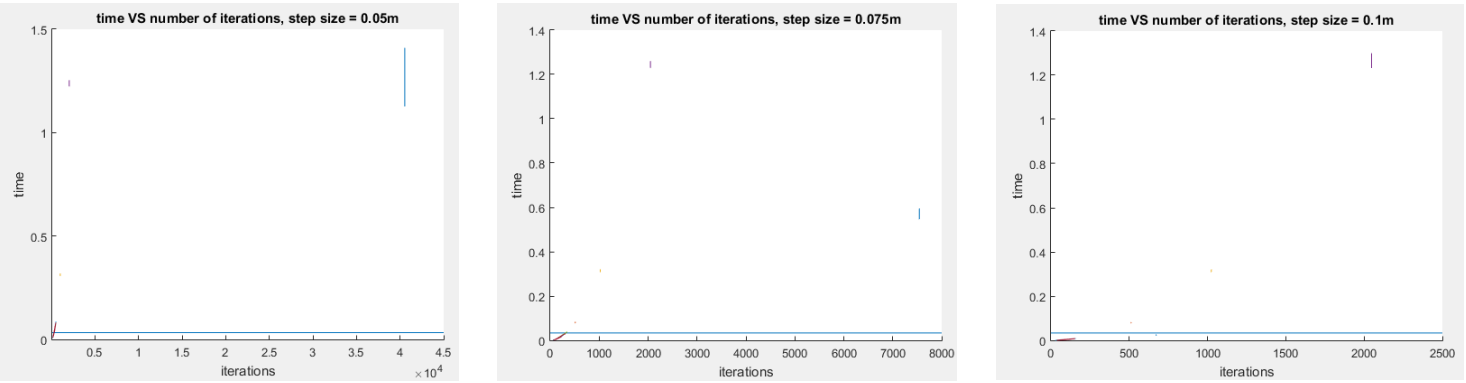


Figure 4.10: Position 3: Time in relation with iterations used during the algorithm, complete data

The figures 4.10a, 4.10b and 4.10c are the complete data set of all tests run in this position with their respected step size.

What figure 4.10a clearly distinguishes from the other two is the immens number of iterations that are shown on the x-axis. The reason for this is, that A* needed over 42 thousand iterations to find a solution. As seen very clearly, the time A* took in these tests varies from 1.2 seconds to 1.4 seconds, whereas RRT with 2048 iterations runs about 1.3 secods and RRT with 1024 iterations runs around 0.3 seconds. The improved RRT versions are all in the vicinity of the time limit and will be shown and explained in figure 4.11a.

In figure 4.10b A* runs for around 0.6 seconds and uses 7,500 iterations. RRT with 2048 iterations runs for more than 1.2 seconds, RRT with 1024 iterations uses 0.3 seconds and with 512 iterations RRT needs not quite 0.1 second to finish. The improved RRT versions are explained in figure 4.11b.

Contrary to the two step sizes before, with the step size of 100 millimeter A* finishes in

just under 25 milliseconds, while RRT with 2048 iterations runs 1.3 seconds, with 1024 iterations 0.3 seconds and with 512 iterations just under 0.1 second. The imoroved RRTs will be shown in figure 4.11c.

(a) step size 50mm (b) step size 75mm (c) step size 100mm

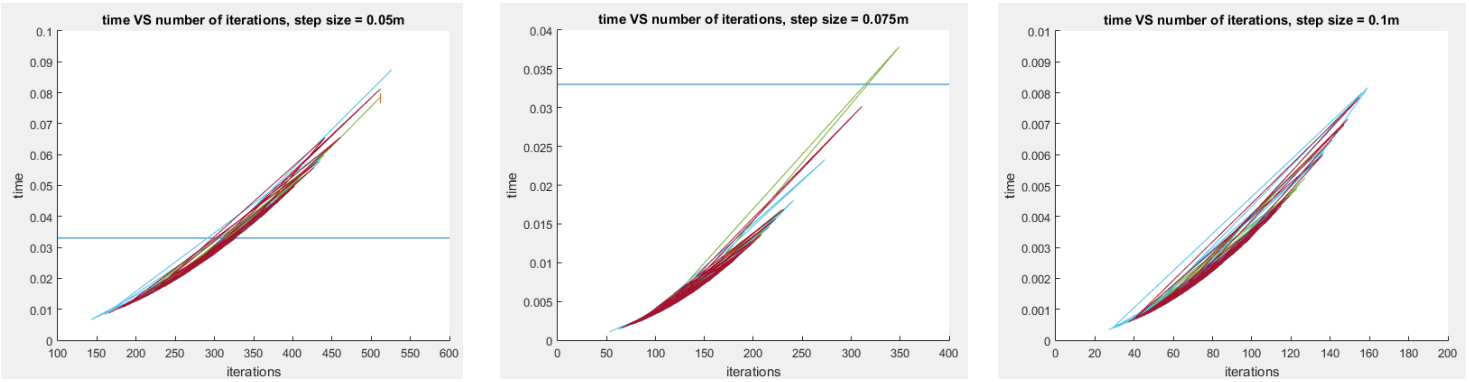
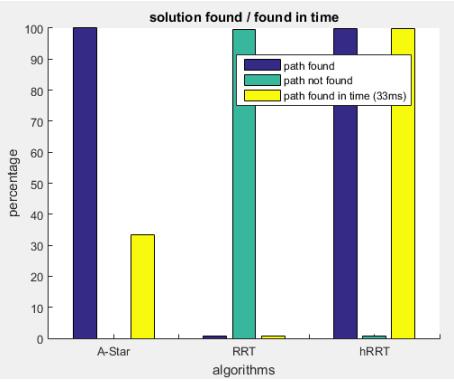


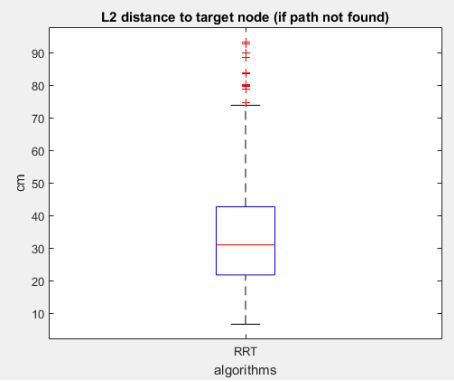
Figure 4.11: Position 3: Time in relation with iterations used during the algorithm, relevant data for real time usage

The above figures show all relevant entries from the complete data set that are within or very near the time constraint of 33 ms.
The entry with the lowest time in figure 4.11a is around 7 ms with 140 iterations and from about 280 to 325 iterations used all algorithms exceed the time limit.
In figure 4.11b on the other hand are almost all entries are within the time limit, only to variations exceed the time limit when using more than 320 iterations. The smallest time needed to conclude the path finding is as small as 1 ms for just above 50 iterations.
Unlike the other two step sizes, for 100 mm step size there are no entries that exceed the time limit in figure 4.11c. With not even 30 iterations a path is been found in less than half a millisecond. In just above 8 ms with less than 160 iterations all algorithms have found a path.

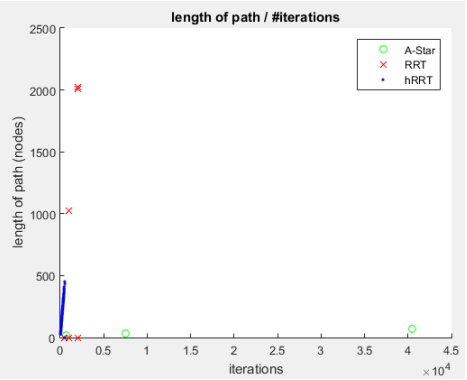
Free University of Berlin
Department of Mathematics and Computer Science
Institute of Computer Science



(a) solutions found, not found and found in time in percent



(b) L2 distance of nearest node to target node if no path found



(c) path length in nodes with related iterations used to find

Figure 4.12: Position 3: path found, L2 and path length

Figure 4.12a clearly states that A* again has always found a path to the target but only for 33% of runs in the given time constraint. Which is explained above and seen in figures 4.10a, 4.10b and 4.10c. RRT has almost never found a path, but when it did, it did so in the time constraint. And the improved RRT almost always found a way in time, but there was an instance where it did not.

The L2 distance in figure 4.12c is mainly between 22 and 45 centimeter with an average of 31 centimeter. And there are again some that come as close as 6 centimeter, which is almost the target area.

RRT without imrovements has both no path length, and a path length near the maximum number of iterations. Whereas RRT with improvements has varying path lengths that begin around the same size as the size of A* and rise up to almost 500. But takes less iterations than the normal RRT and never more than about 500.

5 Comparison

This chapter compares the algorithms on a theoretical basis and the empirical results to evaluate the used algorithms in the context of parking a modell car.

5.1 Theoretical comparison

The following table contains the results of the theoretical analysis of the two algorithms A* and RRT.

	Pros	Cons
A*	completeness	space complexity
	optimality	efficiency versus accuracy
	optimally efficient	
RRT	space complexity	completeness
	iterative use	optimality
	probabilistic approach	optimally efficient
	incompleteness	step size
	non-greedy	collision detection

- In a theoretical scenario the advantages of A* outweigh the disadvantages
- A* always finds an optimal path
- As A* is optimally efficient, theoretically there is no other algorithm that could be more efficient, not even RRT, even if it would be complete and optimal, which it is not
- The disadvantage of the space complexity can be ignored, due to a very small configuration space in which it will be used
- Considering those points, it is to assume that A* would be the best choice for the parking scenario

Free University of Berlin
Department of Mathematics and Computer Science
Institute of Computer Science

5.2 Data evaluation

In the given context of trying to park the car a step size of 100 millimeter seems to be the best of the examined choices, no matter the algorithm.

The data of three different algorithms will be compared, A*, RRT and an improved version of RRT.

For A* the data suggests that

- A* always finds the optimal path, as the result is the shortest found, that is barely reached with the improved RRT
- For position 1 and 3 A* finds the way in 1/3 of the tests and in position 2 every time
- A* seems to be one of if not the fastest algorithm, as the time until finishing is seldomly more than that of RRT with 2048 iterations and that inspite of having expanded tens of thousands of nodes in the same time(for example in figure 4.10c can be seen that A* needs around 24 milliseconds with 672 iterations whereas RRT needs 77 miliseconds for 512 iterations,not even considering that RRT did not even find a path)

For RRT the data shows that

- RRT almost never found a path to the target
- RRT needs more time until it finishes and usually does not come up with a result
- on average the distance from the closest node to the target is 30 centimeter, if no path is found

The improved RRT suggests that

- finding a path succeeds in more than 99%
- the improved RRT almost never finds an optimal path, yet it did sometimes find a path as short as that of A*, but always below 500 nodes as a path
- aside from the step size of 50 millimeter the improved RRT always finds a solution in time

5.3 Summary

At the beginning I suspected that RRT would be the better algorithm to approach the parking maneuver. But theoretically A* seems to be the better choice, aside from the space complexity. Empirically it has been shown that A* does indeed qualify in comparison to RRT.

The only downside to A* is, that the time until finishing the algorithm varies very much depending on the right step size. Although the step size can be determined before the use in the tournament, it does pose a certain risk when circumstances are not the same or differ slightly. So A* might not be the best choice for a generall purpose path finding algorithm without further adjustments.

Due to the rigidness of RRT without any improvements, I also added a variation of RRT that uses a heuristic function and stops as soon as the target has been found. This variation of RRT has shown to be the best algorithm used in these tests. Contrary to A* the improved RRT almost always finished before the 33 milliseconds were over, especially with a step size of 100 millimeter it always finished in time.

Concluding the comparison it has to be said, that the improved RRT is the best choice for the given context of parking a modell car. Due to it as good as always finding a way and that in the short amount of time that the environment determines, unlike A* which might always find an optimal path but not quite in time.

6 Conclusion

After executing the empirical tests and evaluating the theoretical basis of the algorithms and the test results I came to the conclusion, that both A* and RRT are suitable to be used for parking the car. Of course they should have some custom optimizations to improve on the execution time of the algorithms as much as possible and to ensure a wide variety of possibile use cases. Without further improvements the best choice of the algorithms so far would be the improved RRT, which is pretty fast and also adaptable for other purposes if necessary.
The purpose of this thesis to be able to park with the modell car has basically been fulfilled, but there is room for improvements.

6.1 Perspective

One thing would be to consider obstacles in the path planning, which would require an obstacle detection so that the boxes simulating the obstacles can be detected with the camera and not only be guesses due to the fact that there have to be obstacles determining the size of the parking spot.
If obstacles can be detected, it would be a good idea to track there whereabouts, so that multiple calculations can be done should the path planning not be done in 33 milliseconds. Which also is a big improvement for RRT, as the calculation can be limited to a certain amount of time, as the MIT DARPA vehicle is doing. When the calculation is being limited to a certain amount of milliseconds, it is possible to not rely on a complete and optimal path, as multiple small parts to the target can be used to drive and plan in real time by using incomplete paths.
Up until now it is also not possible to calculate the path in a 3 dimensional manner.

Bibliography

[1] Peter E. Hart, Nils J. Nilsson and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. In: *IEEE TRANSACTIONS OF SYSTEMS SIENCE AND CYBERNETICS* SSC-4, No 2 (July 1968).

[2] Steven M. LaValle and James J. Kuffner Jr. Rapidly-Exploring Random Trees: A New Tool for Path Plannig. Department of Computer Science, Iowa State University.

[3] Lukas Knispel and Radomil Matousek. A Performance Comparison of Rapidly - exploring Random Tree and Dijkstra's Algorithm for Holonomic Robot Path Planning. Institute of Automation and Computer Science, Faculty of Mechanical Engineerig, Brno University of Technology, 2013.

[4] Leo Willyanto Santoso, Alexander Setiawan and Andre K. Prajogo. Performance Analysis of Dijkstra, A* and Ant Algorithm for Finding Optimal Path Case Study: Surabaya City Map. Informatics Department, Faculty of Industrial Engineering, Petra Christian University.

[5] Yoshiaki Kuwata et al. Motion Planning for Urban Driving using RRT.

[6] Sertac Karaman and Emilio Frazzoli. Incremental Sampling-based Algorithms for Optimal Motion Planning.

[7] Sertac Karaman and Emilio Frazzoli. Sampling-Algorithms for Optimal Motion Planning.

[8] Oren Salzman and Dan Halperin. Asymptotically near-optimal RRT for fast, high-quality, motion planning. Blavatnik School of Computer Science, Tel-Aviv University, Israel, 4th Mar. 2015.

[9] Jan Frederik Boldt and Severin Junker. Evaluation von Methoden zur Umfelderkennung mit Hilfe omnidirektionaler Kameras am Beispiel eines Modellfahrzeugs. Institut für Informatik, Freie Universität Berlin, Dec. 2012.

[10] First autonomous truck granted license for road use in Nevada, Usa. 8th May 2015. URL: http://media.daimler.com/dcmedia/0-921-899449-1-1810863-1-0-0-0-0-1-0-1549054-0-1-0-0-0-0-0.html?TS=1439480182531 (visited on 13/08/2015).

[11] Carolo-Cup Homepage. 28th Oct. 2012. URL: https://wiki.ifr.ing.tu-bs.de/carolocup/carolo-cup (visited on 16/08/2015).

[12] Sanjay Jena and Nils Liebelt. Heuristische Algorithmen am Beispiel des A*-Algorithmus / 8 Puzzle. Campus Gummersbach, FH Köln.

[13] Prof. Dr. X. Jiang. Künstliche Intelligenz. Institut für Informatik, Universität Münster, 2005.

[14] Howie Choset. Robotic Motion Planning: RRT's.

[15] Fatma Akin and Aneliya Maneva. Suchalgorithmen zur Bahnplannung. Universität Bremen, 2007.

[16] Prof. Dr. R. Rojas. Die Weltformel: Wie man ein Auto einparkt. Institut für Informatik, Freie Universität Berlin.

List of Figures

Ich erkläre hiermit, dass diese Arbeit von niemand Anderem als meiner Person verfasst wurde. Alle verwandten Hilfsmittel wie Berichte, Bücher, Internet- seiten oder Ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kentlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Tom Bullmann
Berlin, 29th September 2015