UNIVERSITÄT
KOBLENZ · LANDAU
Campus Koblenz, Fachbereich 4: Informatik

Local Area Navigation for Multi-axle Vehicles using Machine Learning Algorithms

Bachelorarbeit

zur Erlangung des Grades eines
Bachelor of Science
im Studiengang Informatik

vorgelegt von

Andreas Barthen

Betreuer:	Prof. Dr. Ulrich Furbach, Institut für Informatik, Universität Koblenz-Landau
Erstgutachter:	Prof. Dr. Ulrich Furbach, Institut für Informatik, Universität Koblenz-Landau
Zweitgutachter:	Dipl.-Inform. Christian Schwarz, Institut für Informatik, Universität Koblenz-Landau

Koblenz, im Juli 2013

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden.	☒	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☒	☐

Kruft, den 4 Juli, 2013 Andreas Barthen

Acknowledgements

A lot of people should be thanked for their patience with me taking my sweet time with this thesis, foremost of all Christian Schwarz for giving me so much time in the first place and my family for supporting me along the way. Also all the people working with me in other projects and groups who had to live with me taking so much time off of the project for my thesis.

Abstract

This thesis describes the implementation of a Path-planning algorithm for multi-axle vehicles using machine learning algorithms. For that purpose, a general overview over Genetic Algorithms is given and alternative machine learning algorithms are briefly explained. The software developed for this purpose is based on the EZSystem Simulation Software developed by the AG Echtzeitysteme at the University Koblenz-Landau and a path correction algorithm developed by Christian Schwarz, which is also detailed in this paper. This also includes a description of the vehicle used in these simulations. Genetic Algorithms as a solution for path-planning in complex scenarios are then evaluated based on the results of the developed simulation software and compared to alternative, non-machine learning solutions, which are also shortly presented.

Zusammenfassung

Diese Arbeit beschreibt die Implementation eines Pfadplanungs-Algorithmus für Seriengespannfahrzeuge mithilfe von Maschinellen Lernalgorithmen. Zu diesem Zwecke wird ein allgemeiner Überblick über genetische Algorithmen gegeben, alternative Ansätze werden ebenfalls kurz erklärt. Die Software die zu diesem Zwecke entwickelt wurde basiert auf der EZSystem Simulationssoftware der AG Echtzeitsysteme der Universität Koblenz-Landau, sowie auf der von Christian Schwarz entwickelten Pfadkorrektursoftware, die ebenfalls hier beschrieben wird. Diese enthält auch eine Beschreibung des, zu Simulationszwecken, verwendeten Fahrzeugs. Genetische Algorithmen als Lösung von Pfadplanungsproblemen in komplexen Szenarien werden dann, basierend auf der entwickelten Simulationssoftware, evaluiert und diese Ergebnisse werden dann mit alternativen, nicht-maschinellen Lernalgorithmen, verglichen. Diese werden ebenfalls kurz erläutert.

Contents

List of Figures

List of Tables

1

Introduction

Navigation systems are a common occurrence and present in almost any vehicle and any smartphone, that is especially true for professional vehicles like delivery trucks. However, the directions provided by a navigation system end as soon as the road does, so when the truck enters the transit station the driver is on his own. With many trucks in the same area, one might want to control the routes they take to make sure that every truck gets to its position safely and quickly. In the next step one might even want to automate the entire process by having the trucks drive autonomously within a defined area, in this case the transit station. Such an autonomous system would lower fuel consumption, safe time and extend the range of the vehicles used [35]. It would also make operation safer, especially in dangerous environments such as mines where, in fact, fully autonomous trucks are already being used in everyday work [36]. What is required for such autonomy is not navigation but path-planning, which is a much more complex problem, especially with multi-axle vehicles like trucks with one, or maybe more, trailers. For a completely autonomous transit system one would also require a central system scheduling the routes of all trucks to make sure everything arrives on time and no trucks crash into each other. This component however will not be considered here as we want to concentrate on the path-planning task, with a focus on trucks with several trailers and the possibility to drive such complex vehicles backwards. Finding paths for such vehicles is possible with common algorithms, which will also be presented in the first half of this thesis, these are however not ideal and optimizing them can take a great amount of time. The greater the complexity, in our case the more trailers and more freedom to move, are added, the harder it becomes to find an optimal path using only incremental algorithms.

In order to find a more efficient solution to this problem a machine-learning based path-planning software has been developed, which is now described in this paper. The software is based on an existing simulation and path-correction software developed by the AG Echtzeitsysteme at the University of Koblenz-Landau and employs a genetic algorithm to generate an optimized path for general-n-trailers.
The software should be able to plan paths for a given map and a given vehicle without its performance suffering too much from a higher trailer count, which is why we chose to approach the problem with a genetic algorithm, a proven [9] solution to complex path-planning tasks. We also want to be able to quickly change our map and vehicle to adjust to new planning tasks and be able to configure parameters within the algorithm to optimize our performance and make comparisons between different setups.
To this end we will first cover a lot of basics, that is, common alternative algorithms working on the same problem, the simulation software used, as well as some facts about the vehicle used in our simulations. The idea and development of the planning software is then described in detail in the second half of this paper and finally the results are evaluated in the last chapter. For the purpose of this work we limit ourselves to only one trailer, though the software can theoretically handle arbitrarily many, and we assume the vehicle does not change directions, driving in forward direction only. The consequences of lifting either of these restrictions will also be considered at the end of this work.

2

Previous Knowledge

Before explaining the developed path planner, various given basics have to be covered. In the following chapter genetic algorithms and their various parts will explained in-depth, for implementations and details about which of these possible functions have been chosen for the developed software, see 4. The second part will explain the vehicle and its limitations assumed in the simulation as well as the representation used, which is based on [12].

2.1 About Genetic Algorithms

A genetic algorithm (GA) is a search heuristic based on the "'survival of the fittest"' idea of evolutionary theory. It was developed by Holland in the 1970s [13] and has since then proved itself in solving complex search problems. The idea is that the best possible solutions of an initial (usually randomly obtained) generation survive into the next generation, similar to natural selection where the strong individuals survive and get a higher chance of passing on their genes, while weaker ones die out. This process is then repeated until an optimal solution is obtained. Any given GA consists of almost the same steps, however, depending on which functions are implemented for these steps, and a number of general parameters, the results can drastically vary. This makes it easy to apply the GA to a large number of problems, since the basic setup is always the same and it is easy to exchange only parts of the algorithm for another one to try and get the best results. In the following the various steps of the GA are explained and several possible algorithms are given. Chapter 4 explains which of these were used and why they were chosen over the other possibilities.
The genetic algorithm can be split in two parts, the first part is only executed once at the

start of the program, it initializes the first generation and evaluates it. Since no previous generation is available at this point the initial generation has to be obtained from outside the algorithm itself. See 2.1.1.
The next steps have to be repeated for every generation, so they should be placed within a loop, which runs until a given termination requiremenet is met, for example until the maximum number of generations is reached or until a specified rating is achieved by at least one member of the current generation. The first step in the loop is the selection, where we take a certain number of members from the old generation for the new one. Next, the generation count is raised since now we compute on the new generation instead of the old one. This new generation is then filled with new members by the crossover function and afterwards mutation is applied. In the last step we evaluate the current generation so that selection can be applied again in the next iteration of the loop. Below is a pseudo code for a sample GA.

Algorithm 1: Pseudocode for a genetic algorithm

$genCount \leftarrow 0$;
$initialization(P_n)$;
$evaluation(P_n)$;
while $genCount$!= $maxGenCount$ **do**
  $genCount \leftarrow genCount + 1$;
  $selection(P_{n-1})$;
  $crossover(P_n)$;
  $mutation(P_n)$;
  $evaluation(P_n)$;
end

2.1.1 Initialization

In order for the GA to do anything it first needs an initial generation which has to be obtained elsewhere. Unlike the following generations the initial population is usually generated randomly, but it may also be obtained from some previous computation done by another algorithm. However, this can also lead to a situation where the GA has no chance of finding the optimal solution and only finds a local optimum since the initial generation was too biased and did not contain enough members of the optimmal solution.
How a random population can be obtained depends heavily on the representation of the given problem within the GA, the genome. The choice of genome is one of the most important when developing a GA and also one of the hardest since there are countless

possible ways to encode a problem and no real way to tell how effective a representation may be without actually implementing it. In most cases a bit string representation is chosen where every bit represents a certain possible choice within the search space, for example whether a certain vertex is part of the solution or not. For certain tasks other representations may be better, but in most cases binary encoding is sufficient and makes the implementations of the other functions easier.

2.1.2 Evaluation

The second step in the algorithm is the evaluation, which, like the representation of the genome, is a function that is not standardized and is, as such, hard to choose and a big factor for the performance of the GA. The evaluation function assigns a fitness rating to every genome of the given population. Every genome describes some, not necessarily possible, solution to the given problem and the fitness rating describes how good a solution that is. The rating is higher for good solutions and lower for bad ones, the exact scaling is not standardized and has to be adjusted to the given problem. For example, in some cases it might be best to remove impossible solutions from the population, in others it may be better to keep them since they could still contain valuable parts for another solution. The scaling also depends on whether there is a way to know an optimal solution, in which case there would have to be a maximum fitness rating to assign to such, in other cases no maximum may be known.
The evaluation function depends on the problem to be solved and will usually require representations of the genome other than its binary form. Functions to obtain one representation from the other have to be implemented and some way to describe the usefulness of a solution has to be found.
Once the evaluation function is done, every genome in the current population should have a rating assigned to it, the consequences of this rating are defined by the other functions, mainly the selection.

2.1.3 Selection

The selection function takes N individuals from the previous generation into the current one according to their fitness rating. The exact process of selecting and consideration of the rating depends on the algorithm employed. Since there are more algorithms, and even more variants of these algorithms, than can reasonably be explained here, we will instead concentrate on the possible functions considered or tested for our own GA. The simplest way to select N individuals would be to take the N best ones, this is usually not done since we want to preserve a certain randomness for two reasons. One, individuals with a low fitness rating should still have a chance to be selected since they may still contain parts of a good solution which could be extracted from the bad part by either crossover or mutation. Second, we may want good solutions to be able to be selected several times to give them a higher chance to reproduce through crossover later. The

most commonly known method which fulfills both these requirements is the fitness proportionate selection [14], also known as roulette-wheel selection. It simulates a biased roulette wheel by assigning each genome a segment of the wheel, proportional in size to the genome's fitness rating, and then spinning the wheel N times. This algorithm takes the fitness rating into account while still giving bad solutions a chance to be selected, albeit a low one, and also allows for good individuals to be chosen several times. If the scaling of the fitness function is bad however this could mean that either too many bad solutions survive, because their chance was not small enough, or it could starve out the algorithm because the gap between good and bad solutions is too big and the same good-but-not-optimal solution, also known as a super-individual, is selected almost every time. Stochastic universal sampling (SUS) is an optimized version [14] of the roulette-wheel algorithm which aims to remove bias, offer minimum spread and have a low computational complexity. SUS works by selecting a random value r and then choosing individuals at evenly spaced intervals instead of generating a new random value every time. This gives weaker members a better chance to be selected instead of being dominated by the better solutions [15]. A different approach to selection is the tournament selection, where several individuals are chosen at random and put in a tournament with the winner being selected for the next generation. The tournament size, that is, the number of members chosen each round, is not fixed and can be adjusted for different needs, more competitors give weaker members a worse chance of winning, a tournament size of 1 would be the same as random selection. A tournament can either be won by the individual with the highest rating or one chosen at random with each member having a probability proportional to their rating.

2.1.4 Crossover

The crossover function chooses two individuals from the population at random and recombines their elements in a certain way to produce two new individuals. One or both of these can be kept for the new population, depending on the implementation. Crossover is considered to be the most important search operator [14]. Its results depend on the crossover operator chosen, whether one or both children are kept and the crossover rate, that is, the percentage of the new population that is acquired by crossover instead of selection. The original crossover operator proposed by Holland is the *one-point crossover* which chooses one position in the genome at random and switches the bits to the right of this point between the two chosen genomes. As such, one child ends up with the left part of parent one and the right part of parent two and the other child vice versa. This simple operator usually leads to inferior results [16–18], so *multi-point crossover* operators have been introduced. These work in the same manner as the *one-point crossover* operator but choose x points and switch x times between values from the first and second parent. Depending on the implementation used, the crossover points may also be fixed instead

of randomly chosen. Empirical studies have shown an *eight-point crossover* to be optimal [17, 19, 20], however, the *two-point crossover* is the most common [17]. The maximum number of crossover points is reached when the number points is equal to the number of bits in a genome, in which case we call it a *uniform crossover*, which decides via fair coin toss for each bit whether it is taken from the first or second parent. This is usually done by generating a bit mask of the length of the genome instead of generating a new random Boolean for each bit separately. Other operators, which are not further explained here, include the *segment crossover* [17], *shuffle crossover* [17] and *punctuated crossover* [21].

2.1.5 Mutation

The mutation operator is a simple function meant to re-introduce certain possibilities into the population that have died out and can not be gained by crossover anymore since no member of the population has the necessary combination of bits. This is achieved by choosing single bits of any given genome at random with a very low probability and then inverting them, regardless of whether this individual has been attained by selection or mutation.

2.1.6 Parameters

Once all the parts of the GA have been decided on there are only a number of parameters left to set. While there are some common practices around, most of these have to be decided by trying them out and comparing results. The most important parameters are population size, number of generations, crossover rate, mutation rate, generation gap and scaling. Population size simply determines how many individuals there are in any generation, too small a number will prevent the GA from covering the search space, too large a number will slow down computation. Population size can be anything from several hundred to several thousand and has to be tested out. Likewise, the number of generations determines how many times the algorithm runs and applies selection/crossover/-mutation/evaluation, and has to be tried out. If an optimal solution is usually found after 40 or 50 iterations there is no need to continue further. This number is not necessarily set, it is a common way to determine the run time of the algorithm, but alternative termination conditions can be set instead, for example for one individual to reach a certain fitness. The crossover rate determines the percentage of the new generation acquired by crossover instead of selection and is usually set very high. 0.6, 0.75 and 0.95 are common values [18, 20, 22]. Similarly, the mutation rate determines the chances of any given bit to be mutated, this rate is usually very low, 1‰, 5‰ or 1% are common [18, 21, 22]. The generation gap determines how many members of the generation are replaced in the next generation, which is usually set to 1.0, meaning the entire population is replaced. Scaling is applied to the fitness rating to prevent super-individuals from scewing the results of the selection early. The exact way of scaling depends on the evaluation function and may not be necessary at all, but a sigma-truncation, which is a truncation of a gaussian

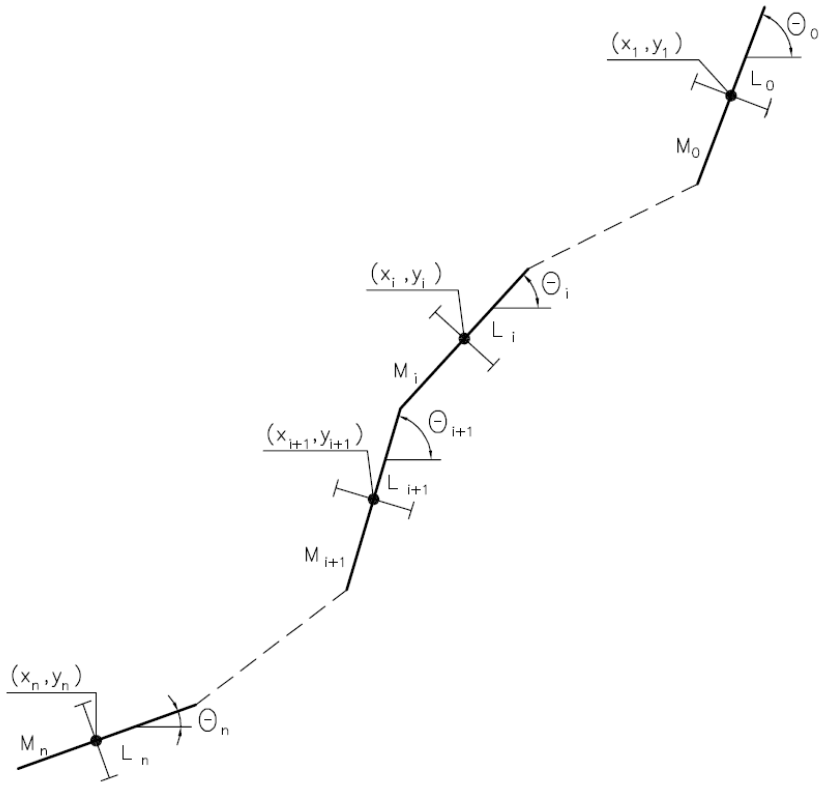

Figure 2.1: A vehicle in the single track model with identifiers [12]

distribution meant to control the bias towards highly fit members, is common [24].

2.2 About the Vehicle

Our simulation software uses the control method for stable driving developed by Christian Schwarz to determine the actual path from the planned one in order to evaluate its fitness. To this end we use the same representation of the vehicle, which will be explained in the following section, for a complete definition and prove of stability see [12]. This section also contains a short overview of nonholonomic systems, of which our vehicle is an example.
The dimensions and number of trailers of the vehicle can be freely defined in the simulation software and since no connection to an actual model is implemented yet, no further restrictions are given. The control method itself has been tested in simulation as well as on a 1:16 general-2-trailer model. In order to describe the vehicle more easily and still have enough detail to achieve a correct representation, the single track vehicle model is used [25]. The vehicle is first split along its flexible coupling, so any axles with rigid couplings are replaced by one virtual axis between them. The steering of the vehicle is considered to be a flexible coupling, so the tractive unit is split into 2 elements, which is why we call it a general-(n-1)-trailer if the model has n elements. Now that the vehicle is a chain of rigid parts, all wheels are replaced by points, meaning they are assumed to

touch the ground in exactly one spot, and then all pairs of wheels are replaced by a single wheel in between them. 2.1 shows a general-n-trailer in its single-track model with identifiers, which are used as followed:

- (x_i, y_i) identifies the position of a vehicle part by the center of its axis
- θ_i identifies the orientation of the element
- L_i identifies the distance from the axis to the front of the vehicle part, the front coupling point
- M_i identifies the distance from the axis to the back of the vehicle part, the rear coupling point

The representation of the entire vehicle can be shortened by giving the positions and angles of all but the first element in relative terms. According to [26] the following rules hold:

- $x_{i+1} = x_i - L_{i+1}cos\theta_{i+1} - M_i cos\theta_i$
- $y_{i+1} = y_i - L_{i+1}sin\theta_{i+1} - M_i sin\theta_i$
- $\alpha_{L_1} := \theta_0 - \theta_1$ is the angle of steering lock of the vehicle
- $\Delta\theta_{i(i+1)} := \theta_{i+1} - \theta_i$ is the i-th angle of the vehicle

This leads to the complete representation of the vehicle's configuration:

$$\vec{g} = \begin{pmatrix} x_1 \\ y_1 \\ \theta_1 \\ \Delta\theta_{12} \\ \vdots \\ \Delta_{i(i+1)} \\ \vdots \\ \theta_{(n-2)(n-1)} \\ \alpha_{L_1} \end{pmatrix}$$

2.2.1 Nonholonomic systems

Nonholonomic systems are a special class of kinematic systems where the degrees of freedom available are greater than the number of degrees the object can move in, which means the object cannot move along all paths that would theoretically be possible. To be more precise, this means that if a configuration space has m dimensions and the vehicle has k constraints such that $0 < k < m$, the system is nonholonomic. It can also be defined as a system with kinematic differential constraints that are non integrable and cannot all

be expressed as a holonomic constraint of the form $f(q,t) = 0$. [27,28]. A car is already a nonholonomic system as it cannot move in every direction of the plane since only one of the 2 axles are controllable and even that is limited. A general-n-trailer has even more constraints and limitations as to how it can move in the same space and is thus also a nonholonomic system. Path-planning for such a system is more complicated due to these additional constraints, called the Pfaffian constraint, which have to be considered in order to gain a proper solution.

2.3 About the control method for stable driving

The EZauto software consists of many different parts for a variety of purposes, only one of which is implemented in our simulation software however: The control method for stable driving [12]. This control method works similar to how humans would drive: By aiming for a certain point on the path we want to take, drive for a short distance and then adjust our steering and aim for a new point [29,30]. It works both for normal and reverse driving, only the reference point θ of the vehicle has to be adjusted. It is either the center of the rear axis of the tractive unit for normal driving or the center of the (last) trailer's axis for reverse.
The desired path in our case is already given by either the random function or the GA so now we have to try to follow this path. To this end, we choose a point on the desired path which is a "'certain distance"' away from our current position as our meeting point. Then we have to determine a circle which contains both the current position and the meeting point, this is the path we have to follow. The circle must be chosen such that the direction at the beginning, where our current position is on the circle, is the same as the direction of our θ, since we can't change direction without moving.
This task now presents us with several steps: Determining the meeting point, from that the radius of the correction circle and finally the steering angle. While this method works regardless of driving direction, the calculation of the steering angle changes.

2.3.1 The Meeting Point

The steps necessary for determining the meeting point depend on whether our desired path is a line or a circle, which are assumed to be the only possibilities. As shown in [12] any other curve can be represented by a mixture of such lines and circles. 2.3 shows the process for a linear desired path, described by the function:

$$g : R \to R^2, \lambda \mapsto g(\lambda) := \vec{g_0} + \lambda \cdot \vec{h}$$

Here $\vec{h}$ is the direction and it is assumed that $\|\vec{h}\| = 1$, so the vector is normalized since its length does not matter. Now we try to determine the ideal point I, which is the point on the path our vehicle would ideally be at if there was nothing to correct. We can see I in 2.3 and can determine it as:

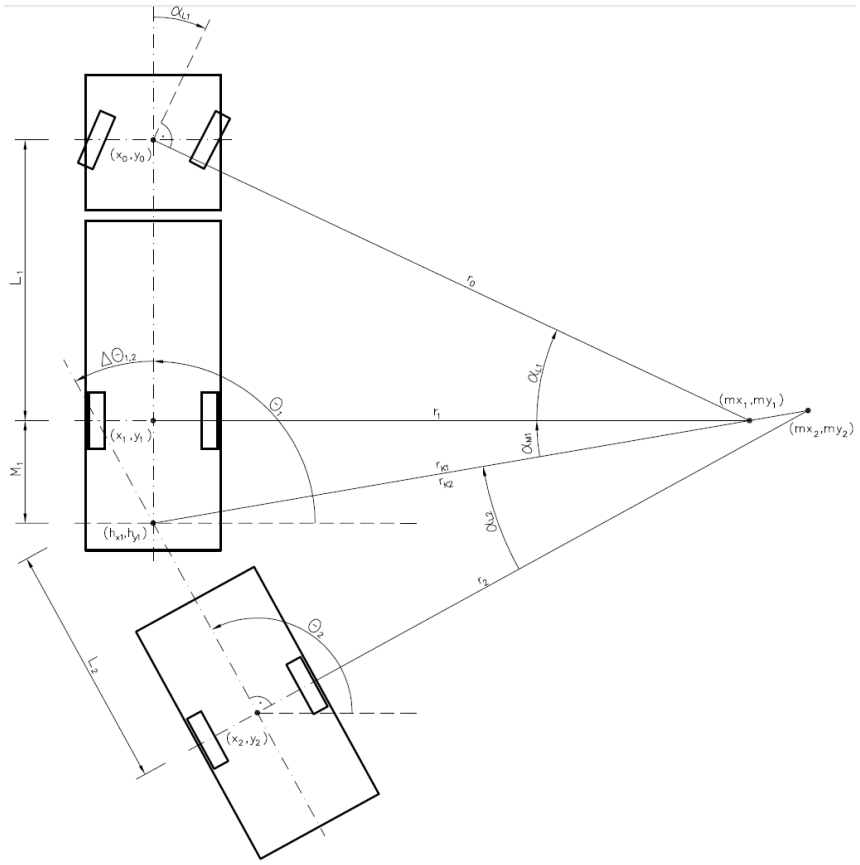

Figure 2.2: General-2-trailer identifiers [12]

$$\vec{i} := \langle \vec{p} - \vec{g_0}, \vec{h} \rangle \cdot \vec{h} + \vec{g_0}$$

Now we can determine the meeting point T, denoted by the vector $\vec{t}$, by moving our I along the desired path by $c_v > 0$. The parameter c_v has to be determined experimentally since it depends on the kinematic of the vehicle, and...

$$\vec{t} := \vec{i} + c_v \cdot \vec{h} = (\langle \vec{p} - \vec{g_0}, \vec{h} \langle + c_v) \cdot \vec{h} + \vec{g_0}$$

If our desired path is a circle, illustrated in 2.4, it is given as:

$$k : R \to R^2, \varphi \mapsto k(\varphi) := \vec{m_s} + r_s \cdot \begin{pmatrix} cos\varphi \\ sin\varphi \end{pmatrix}$$

where $|r_s|$ is the radius of the circle and its sign determines the direction: a negative sign means clockwise. Now we have to determine our ideal point I again, which in this case is the intersection between the center of the circle M_s and our current position P, it is denoted by $\vec{i}$:

$$\vec{i} := \frac{\vec{p} - \vec{m_s}}{\|\vec{p} - \vec{m_s}\|}$$

Analogous to the meeting point for lines, T is now determined by moving I, this time by rotating it around $\vec{m_s}$ by an angle with the arc length c_v, so the line between the ideal point and the meeting point has a length of c_v, just as before:

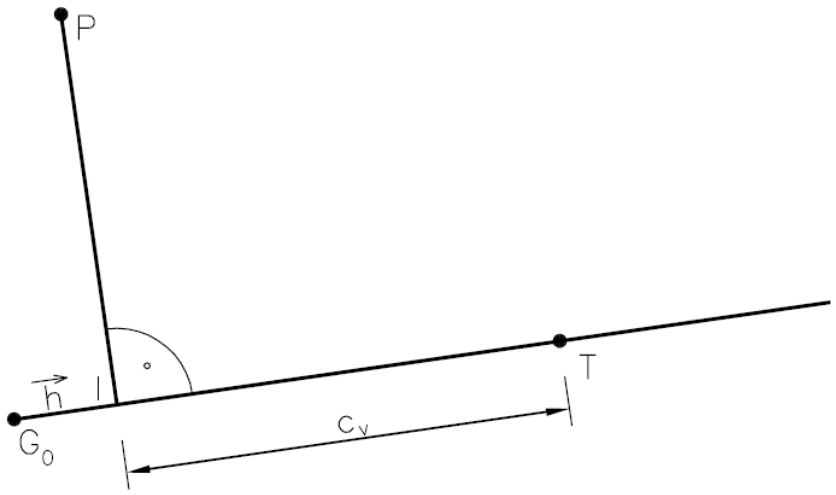

Figure 2.3: Determining the meeting point for a linear path part [12]

$$\vec{t} = \begin{pmatrix} cos\frac{c_v}{r_s} & -sin\frac{c_v}{r_s} \\ sin\frac{c_v}{r_s} & cos\frac{c_v}{r_s} \end{pmatrix} \cdot (\vec{i} - \vec{m_s}) + \vec{m_s} = \frac{r_s}{\|\vec{p} - \vec{m_s}\|} \begin{pmatrix} cos\frac{c_v}{r_s} & -sin\frac{c_v}{r_s} \\ sin\frac{c_v}{r_s} & cos\frac{c_v}{r_s} \end{pmatrix} \cdot (\vec{p} - \vec{m_s}) + \vec{m_s}$$

2.3.2 The Radius

We now have to determine the circle we have to drive to get from our current position P to our desired position T determined in the last step. This circle has to be tangential to the line denoted by our current position T and our current direction θ, since otherwise we would be unable to follow it. Consequentially our center point lies on a line which is orthogonally to θ and runs through P. Also, since our desired meeting point T has to be on the same circle, the line from T to P is a chord of the circle, so our center point lies on a perpendicular bisector of this line. See 2.5 for an illustration. The vector $m\vec{_{KK}}$ denoting this point M_{KK} is obtained by:

$$m := \begin{pmatrix} -sin\theta \\ cos\theta \end{pmatrix} \cdot \frac{\|\vec{p} - \vec{t}\|^2}{2 \cdot \left\langle \begin{pmatrix} -\theta_2 \\ \theta_1 \end{pmatrix}, \vec{p} - \vec{t} \right\rangle} + \vec{p}$$

Accordingly, the radius is:

$$r_{KK} = \frac{\|\vec{p} - \vec{t}\|^2}{2 \cdot \left\langle \begin{pmatrix} -sin\theta \\ cos\theta \end{pmatrix}, \vec{p} - \vec{t} \right\rangle}$$

2.3.3 Calculating the steering angle

The process of determining the steering angle depends on the reference point of the vehicle, which, as mentioned in 2.3.1, depends on the driving direction. In forward driving

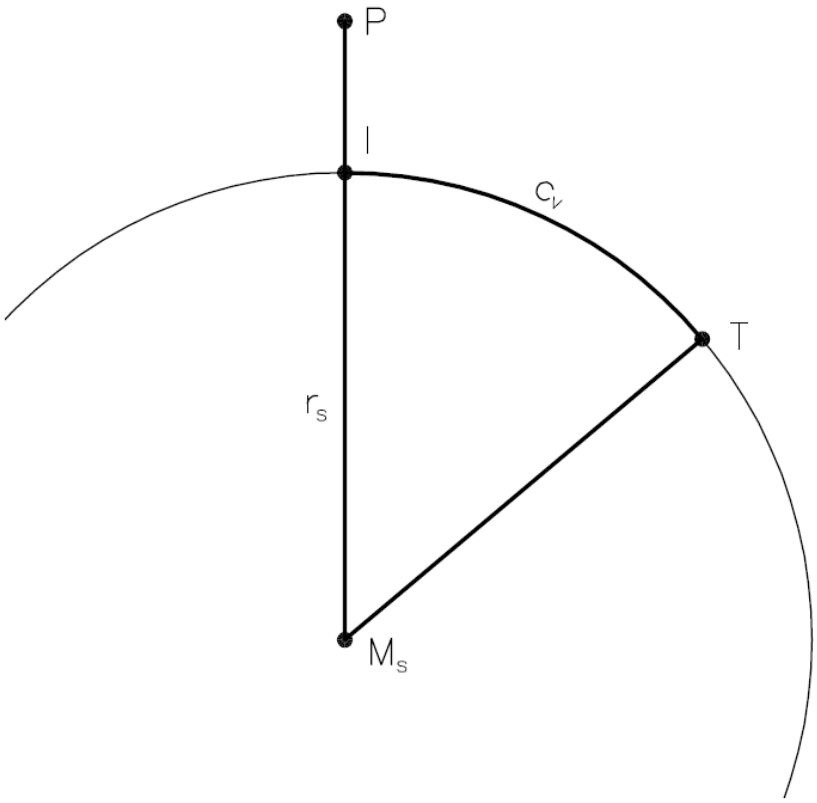

Figure 2.4: Determining the meeting point for a circular path part [12]

our reference point is the rear axis of the traction unit, (x_1, y_1) (see fig. 2.2), our radius is r_1 and our steering angle is:

$$\alpha_{L_1,target} = arctan\frac{L_2}{r_{KK}}$$

If we are driving in reverse then our reference point is the center of the axle of the (last) trailer, our radius is r_2 and our required direction θ is:

$$\Delta\theta_{12,target} = -arctan\frac{M_1}{r_{KK}\sqrt{\frac{L_2^2 - M_1^2}{r_{KK}^2} + 1}} - arctan\frac{L_2}{r_{KK}}$$

Now we have to achieve this $\Delta\theta_{12,target}$ by adjusting our steering angle, to this end we observe the behaviour of θ when adjusting the steering angle in forward driving and try to obtain a conclusion about its expected behaviour in reverse driving. From [12] we know that θ converges towards a $\Delta\theta_{12,stable}$ depending on the given steering angle. We can now determine a steering angle $\alpha_{L_1,stable}$ which will lead us to our desired θ_{12}:

$$\alpha_{L_1,stable} = f(\Delta\theta_{12}) := -arctan\frac{L_1 \cdot sin\Delta\theta_{12}}{L_2 + M_1 \cdot cos\Delta\theta_{12}}$$

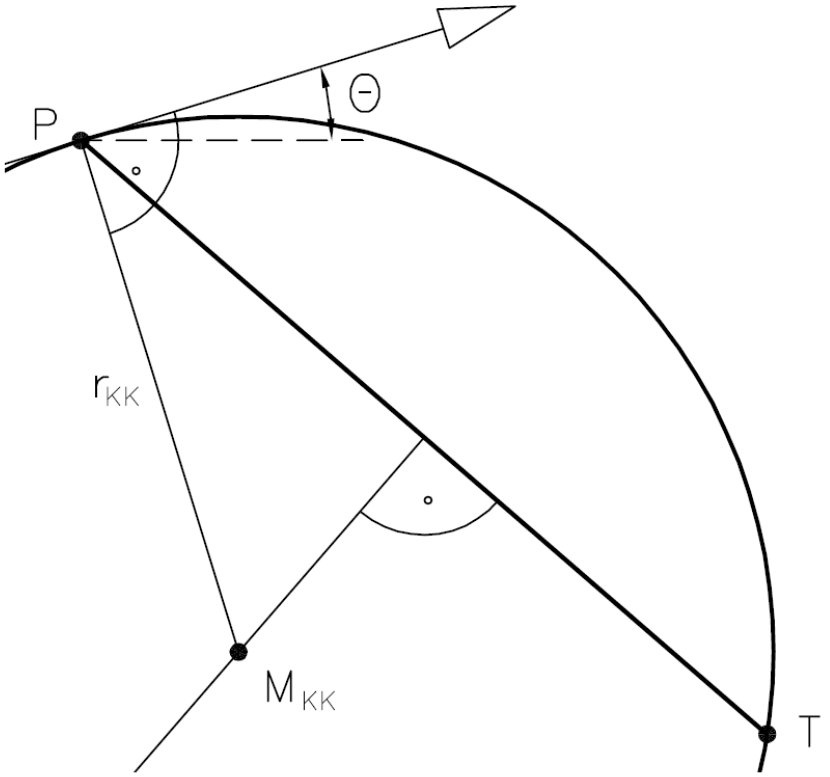

Figure 2.5: Determining the radius of the correction circle [12]

This function f is strictly monotonic decreasing in $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$ as long as $L_1, L_2, M_1 > 0$. This means that our steering angle is $\alpha_{L_1,stable} = f(\Delta\theta_{12})$ as long as our vehicle is in a stable state and if we want to adjust our angle by ϵ we have to set our steering to $\alpha_{L_1,stable} = f(\Delta\theta_{12} + \epsilon)$

3

Related Work

Path planning is a fundamental problem, so it is unsurprising that there are already many possible solutions. Several well-known algorithms have been used to tackle this problem or combined in order to achieve better results. However, it is also a very complex problem and has been shown to be PSPACE-hard [37], with its complexity growing exponentially with the configuration space, in our case the number of trailers [1]. In the following chapter we will take a closer look at some of the solutions proposed and also consider their complexity and their problems with this particular task. This chapter is split in two parts, the first one concentrating on incremental algorithms and probabilistic metaheuristics, the second one instead focusing on other machine learning approaches. The focus in either case is the usefulness of the algorithm as a path-planning solution for a general-n-trailer.

3.1 Common Path-Planning Algorithms

In order to use the algorithms presented here we have to construct a configuration space (C-space), which is achieved by representing any possible configuration, that is direction and position of all elements of the vehicle, as a single point in this C-space. The size of this C-space is one of the biggest factors in the algorithms performance, it is determined by two factors: The number of degrees of freedom our system has, in this case the number of trailers, as well as the magnitude of our grid with which we overlay the map with in order to allow orientation. This magnitude depends on both the size of our map as well as the chosen resolution.

3.1.1 Rapidly-exploring Random Tree

Rapidly-exploring Random Trees (RRT) [38] are an efficient method for path-planning in high dimensional spaces, including cases such as ours, where nonholonomic constraints are given. [31, 32] It works by randomly choosing a configuration (or, if a collision detection is given, a free configuration) from the configuration space C and then looking for a vertex in the tree that is close to the chosen configuration. Then it moves a distance q in direction of that target configuration, taking movement constraints into account. The now reached configuration $q_(new)$, which is close to the initially randomly chosen target configuration, is then added to the RRT and the process is repeated. This algorithm can be started simultaneously from several points of the map, for example the start and target position, and then try to meet the other tree to gain a continuous path through the map. While RRTs can produce paths through complex environments quickly, even while taking different constraints into account, it is not a sufficient path-planning algorithm when used alone. The resulting paths are suboptimal and contain a large number of needles turns and sharp corners. An optimization algorithm is required to flatten the path and make it a more feasible solution. It is, however, a good way to quickly find a path at all, using a different algorithm for optimization later can still result in an overall better performance than comparable algorithms.

3.1.2 Path Transformation

Path transformation is a simple way to find a path in a grid-based environment (see 3.2.1) and combines two other transformations: distance- and obstacle transformation [33]. Distance transformation works by assigning every cell of the grid a value which represents the distance from this cell to the target cell, see fig. 3.1 for an example. How this distance is computed depends on the chosen movement model, basically whether we choose that our vehicle can move diagonally for the same cost as horizontally/vertically, at higher cost, or not at all. The last case means moving diagonally has twice the cost of horizontally/vertically since it is seen as a combination of two such moves. If a map has been updated with such values a path can easily be found simply by always choosing a

cell with a lower value as the next step. The costs of the entire path is also clear from the beginning since it is stated in the first cell. Obstacle transformation works similarly by assigning each cell a value stating the distance between this cell and the closest obstacle as can be seen in fig. 3.2. Path transformation now combines both these maps under a certain weighting which states whether the length or the safety of the path is more important. Avoiding cells with low obstacle value is safer while avoiding cells with high distance value is shorter, so by adjusting this weighting a balance between safety and distance can be found.
While path transformation is a simple and often used way to find a path, it is not well suited to high dimensional problems with nonholonomic constraints such as our case here. Its efficiency also heavily depends on the chosen resolution of the map.

3.2 Algorithms for Graph-Based Path-Planning

In order to plan a path using incremental algorithms we first have to convert our path planning problem into a graph searching problem. To this end we have to construct a suitable graph from a given C-space. There are several approaches to this task, for example the visibility graph approach [2, 3] and the retraction method [4, 5], but for the purpose of this paper it is enough to know that such a conversion into a graph searching problem is possible.

3.2.1 A* Algorithm

The A* algorithm is one of the most widespread solutions for graph traversal problems. [8] In order to utilize it for motion planning, the area has to be overlaid with a configuration space and then transformed into a graph as described above. This leads to several problems for our path planning task. The complexity of the problem depends on the size of the graph, which in turn depends on the size of the configuration space. A finer grid leads to a larger number of vertices in the graph and, as the computation time of the A* algorithm grows exponentially with this number, results in slower performance. However, a coarse grid would ignore possible paths since obstacles would appear greater than they are. To a certain point this can be useful to prevent collisions, but it also means a loss of information and consequently fewer options to choose from, possibly missing better paths. There may also be additional constraints the path has to satisfy, which further slow down computation. So while A*, or similar graph searching algorithms, like Dijkstra's graph search [6], can be used, their performance for this kind of complex problem prevents them from being feasible.

3.2.2 Hill Climbing

Another common algorithm used to tackle graph searching problems is hill climbing. It is popular due to it's simplicity and ability to find a local optimum in a short amount

of time, also it can return a result an any point, even when it is not yet finished. This may be relevant in real-time system, where it is more important to have a solution at a given time than to have an optimized solution later. This may also hold true for motion planning task, when the speed of planning is more important than the quality of the resulting path. In our given case however the quality of the path is more important than the speed of computation. In simple cases, Hill Climbing produces paths of similar quality when compared to simulated annealing or genetic algorithms [8], however, as the search space grows more complex, the algorithm fails to generate good solutions. This is due to its very local searching behaviour which becomes more of an obstacle the larger the search space gets. Various optimization techniques try to mitigate this problem, such as stochastic hill climbing, which does not examine all neighbours but chooses randomly and then evaluates whether to move there or to examine another one, or random-restart hill climbing, which tries to counter the locality issue by choosing its start point at random and then repeat the entire search several times. Even with these optimizations in place, hill climbing still cannot compete with other algorithms in solving problems of the complexity we consider here. Stochastic hill climbing produces paths similar to standard hill climbing at equal or even greater cost [8] and also fails to produce good results once the search space becomes too large. Random-restart hill climbing could potentially produce good results, but in a large search space this would very much rely on luck. Since the algorithm would have to run many times to have at least some chance of finding a globally good path the performance would suffer greatly. Also, unlike A*, hill climbing can never guarantee that the optimal path has been found, this however is also true for our machine learning algorithms.

3.2.3 Simulated Annealing

Simulated Annealing is a probabilistic metaheuristic which searches for an optimum solution in a way similar to hill climbing, that is, by always considering the neighbours of the current position and then using a given function to determine whether or not to move to that neighbour. Unlike hill climbing however, simulated annealing changes its behaviour over time, according to a global parameter T (Temperature). T can be defined freely, but always ends with $T = 0$. Every state is assigned an energy e which is smaller the better the state is. The algorithm favours moves that go towards lower energy states the smaller T is, so it starts out by ignoring local minima and moves loosely towards areas that contain good solutions overall. Then, with lower T, it starts preferring moves that go "'downhill"', meaning towards lower energy states, more so that it starts moving towards the local minimum in the given region once T gets close to 0. Due to this, simulated annealing circumvents the hill climbing problem of solely moving towards local solutions while ignoring the global ones. The algorithm shows results similar to those of a genetic algorithms, but is significantly outperformed as far as number evaluation is concerned. [8]

3.3 Alternative Machine Learning Algorithms for Path-Planning

As we can see from the algorithms presented in 3.2 an iterative approach causes a number of problems when it comes to complex tasks. Since we want to develop a solution for a general-n-trailer, with theoretically infinite degrees of freedom, an algorithm that stops producing feasible results for non-trivial problems is not an option. Of the five algorithms presented, simulated annealing is the only one that produces useful results, but also requires a very large number of computations [8] for more complex problems. In order to avoid these issues, a number of machine learning algorithms have been proposed to solve the motion planning problem. In the following section we will consider three of these approaches, including a second implementation of the genetic algorithm (GA). This GA is important since the comparisons to hill climbing and simulated annealing referred to earlier have been made using that implementation and not the one developed in this paper. [8]

3.3.1 Reinforced Learning

Reinforced Learning is one category of machine learning algorithms with different capabilities depending on the method used. Before focusing further on any of these specific algorithms, a general overview of the principles of reinforced learning will be given. The general idea of reinforced learning is to assign a certain reward to any action. This reward can be positive or negative (punishment) and its value depends on the action taken. The task of the algorithm is to choose a path through this graph such that the accumulated reward at the end is maximal. Since always choosing the greater reward at any point obviously does not mean achieving the greatest accumulated reward, a balance between exploration and exploitation has to be found. Exploitation means picking the best choice available in the current state and exploration means picking sub optimal choices in the hopes that actions further down this part of the decision tree are overall better. To this end an ϵ-greedy algorithm with an ϵ value of 5% is usually used, where a greater ϵ value means more exploration and a smaller value infers more exploitation. For path planning tasks a Markov process is usually assumed, which means that only current inputs, in our case the current position of the truck, are considered and past ones are ignored. This means that old inputs do not have to be stored, so no memory is required for that, and computation becomes much simpler since fewer variables are considered. Reinforced learning algorithms can be split into three groups depending on their model of the environment and their bootstrapping capability. Bootstrapping means that estimates are updated based on other estimates, which speeds up the algorithm. These three groups are dynamic programming methods (DP), monte carlo methods (MC) and temporal difference methods (TD). MC and TD do not need an exact model of the environment, where MC uses no bootstrapping but TD does. DP also uses bootstrapping but needs an exact model of the environment, which makes it less feasible for motion-planning purposes.

[9]. Specific reinforced learning implementations can be categorized according to these three methods. Both TD and MC methods can be split further depending on whether they are on-policy or off-policy, where on-policy uses a first-visit method, where the value of a state-action pair is determined be the average return after the first instance of a given state. Because there is no guarantee that all state-action pairs will be visited, on-policy is ϵ-greedy so it approaches an optimal policy while still maintaining exploration. In our case this simply means that we have an ϵ chance of selection a random action rather than the best action. Off-policy separates the evaluation policy into behaviour and estimation policy. It follows the behaviour policy while it evaluates and improves the estimation policy. Due to this separation, ϵ-greedy is not needed to evaluate all state-action pairs, however, since the learning rate for greedy and non-greedy actions are different the algorithm is slower for some states. In the following we will now have a closer look at two such algorithms.

3.3.1.1 Q-Learning

Q-Learning is an off-policy, ϵ-greedy TD method which learns an action-value function independent from the given policy to make an assumption about the optimal policy. Compared to other algorithms it is exploration heavy and will probably make more random decisions than exploitative decisions, due to this it is also more likely to find the optimal policy in finite of time. Q-Learning requires the actions and states to be defined. In our case, states are sensor data, or our current position, and actions are basic movement actions, here left/right in a certain angle and forward driving. Since reinforced learning takes a large number of samples to learn from, the first step should be done in a simulation, similar to the one developed for the evaluation of our genetic algorithm. Simulations have the downside of being inaccurate since a model can never represent the real world perfectly, but can in turn be used much easier and much more quickly [9]. Also data from a simulation can be stored and evaluated easily and the process can simply be automated. Once an optimal policy has been found using the simulation, it has to be further refined using the actual vehicle. Depending on the accuracy of the simulation this will only require a small number of episodes. This phase may still be the most time consuming however since it cannot be sped up like the simulation. Depending on the vehicle and space required for the task, in our case a parking space and a truck, this task will still be the most challenging.

3.3.1.2 Extended Q-Learning

Classical Q-Learning needs to make $m - 1$ comparisons to maximize the Q-value in a given state, so assuming n states, the overall time-complexity of Q-Learning is $\mathcal{O}(n(m - 1))$. This can be optimized by using extended Q-Learning [11]. We introduce an additional variable, L_x, for each state. L_x is a lock bit and is 1 if the value Q_x of the state S_x is locked and won't be updated further, or 0 otherwise. At the beginning we have

$L_n = 0$ for all S_n except the goal state. There are a number of properties that can be used to quickly set L_n throughout the algorithm's run time:

Property 1: If $L_n = 1$ and $d_{pG} > d_{nG}$ then $Q_p = \gamma \times Q_n$ and set $L_p = 1$
Property 2: If $L_n = 0$ and $d_{pG} > d_{nG}$ then $Q_p = Max(Q_p, \gamma \times Q_n)$
Property 3: If $L_n = 1$ and $d_{nG} > d_{pG}$ then $Q_n = \gamma \times Q_p$ and set $L_n = 1$
Property 4: If $L_n = 0$ and $d_{nG} > d_{pG}$ then $Q_n = Max(Q_n, \gamma \times Q_p)$

Using these properties, both L_i and Q_i can be updated accordingly. Now, in order to determine if a state is optimal, we only need to check whether or not it is locked. So for n states we only need n comparisons, so we save $\mathcal{O}(n(m-1)-n = nm-2n = n(m-2))$ [11] comparisons. For space complexity we previously had $n \times m$ for classical Q-Learning whereas we only need 3 values for any state in the extended Q-Learning algorithm: Q-Value, L_x and the best action at that state. This means a space-complexity of only $3 \times n$ and a saving of $\mathcal{O}(nm - 3n = n(m-3))$. [11]

3.3.2 Genetic Algorithm

In this section we will cover the genetic algorithm (GA) proposed by Andreas C. Nearchou [8] as a comparison to the iterative algorithms covered in 3.2. For a general description of the GA refer to 2.1 and for an in-depth implementation of the GA developed for this paper refer to 4. The genome in this case is represented as a bit string of the length of the path where each bit represents a vertex in the graph and marks whether or not it is part of the solution. The first bit marks the first vertex, the Nth bit the last vertex. The initial generation is obtained by randomly choosing a path length and then flipping a fair coin to decide whether any vertex on the graph is taken or not. This includes impossible paths that would collide with an obstacle, these are then filtered out by the fitness function. In addition, paths get a better rating depending the total weight of their vertices and sigma-truncation is applied 2.1.6. Binary tournament selection, uniform crossover and bit-mutation are then applied. An inversion function is used in addition to these usual GA functions, all of which are explained in depth in 2.1.
The GA is then applied to 13, 23, 33 and 50 vertices graphs with 4 different combinations of minimal and maximal edge-cost and vertex-profit functions. The evaluation of the GAs results in comparison to the other 3 algorithms, hill climbing (3.2.2), stochastic hill climbing (3.2.2) and simulated annealing (3.2.3), are given in their respective sections.

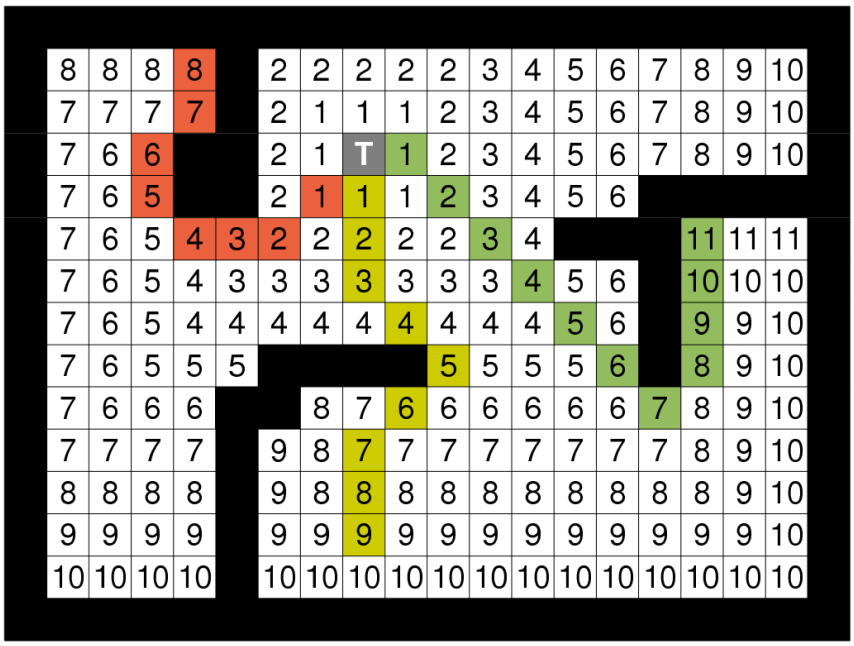

Figure 3.1: A map filled with distance transformation values

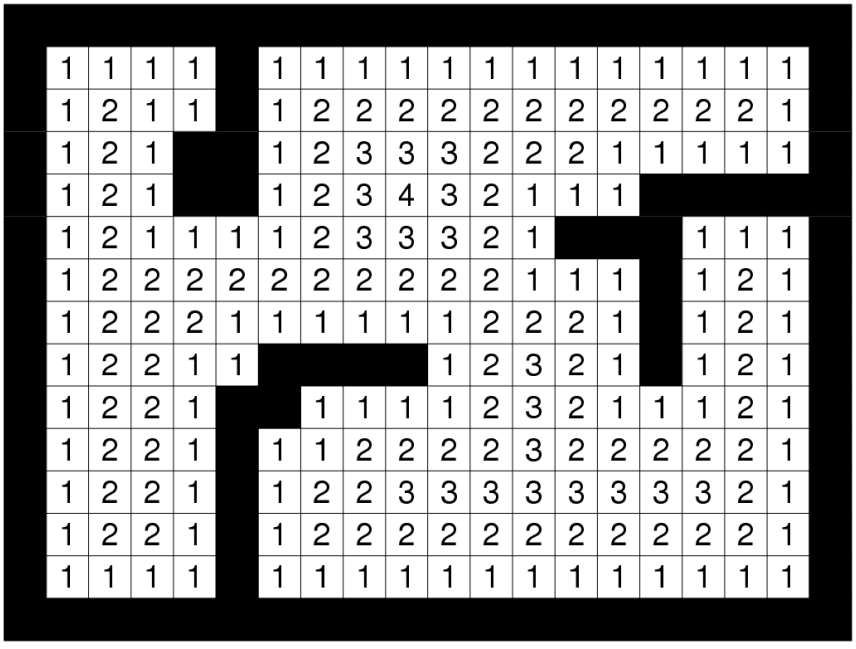

Figure 3.2: A map filled with obstacle transformation values

4

Algorithm Details

Since a general overview of genetic algorithms has already been given in 2.1 this chapter will only cover the choices made for the particular program developed for this paper and justify these choices. Note that some alternative algorithms have been considered and sometimes even implemented, see 6, but due to the time constraints of this thesis not all possibilities could be covered, so in many cases there may be a better solution possible. Some of these will also be considered, but not implemented, in 7.

The general idea is to represent our paths in such a way that they are easy to use in bit-wise crossover and mutation operations. To this end, all paths are stiched together from simple path-primitves: Lines and curves. Several of these, with different lengths and angles, are then put together to create any path we require. These primitives allow for simple generating, saving and modifying of paths, it also gives us the fixed structure we want for our GA operations.
The initial generation to feed the GA has to be obtained randomly, our chosen path representation is also well suited to this task since it can simply choose random path primitives.
The representation of the vehicle should be as close as possible to the one described in 2.2 since the existing components of the EZSystems framework, which we need for evaluation purposes, are also based on this model. For efficient testing we have to be able to modify, save and load the vehicle as well as its configuration.

4.1 Genome Representation

For ease of use, a simple path representation with a fixed genome length has been chosen. As detailed in 5.4, every path consists of exactly 20 parts, each subject to length and/or angle restrictions respectively. Each part is represented by an eight bit string, so the entire path consists of exactly 160 bits. This fixed length makes the application of further operations like crossover and mutation easier and also allows for a simple conversion between the path and genome representation of any given population member.
Consequentially every 160 bit genome can be split into 20 8 bit genome parts, which can be further be assumed to consists of 4 parts: The first bit determines whether this genome part represents a line or a curve and the next 3 bits determine the length of a given path part. The last 4 bits are only considered in case of curves, for linear path parts they are ignored and set to 0. The first 3 of these 4 bits contain the angle of the curve and the last one determines whether this is a right or a left curve.
This genome format can also be thought of as a graph where, starting from left to right, every bit represents one choice in the graph, the first being curve or line, and the next being the length etc.

4.1.1 Genome Conversion

For every member of the population we require three pieces of information: The path representation, needed for our evaluation, the genome representation, needed for our genetic operations, and the fitness rating. While the assignment of the rating is done by fitness function, see 5.6, we also need to be able to convert between the two representations of the path. While a conversion from path to genome representation would certainly be possible, it is not implemented since we never need it. The only part of the algorithm where we do not already have the genome before we need the corresponding path is the initial generation in which we require random paths. Since the paths are random it does not actually matter whether we first generate a genome and then obtain the path or generate a path and obtain the genome from there. We do, however, always need a function that can obtain a path from a given genome since crossover/mutation and reordering will operate on genomes and also return genomes, so since we need that function regardless of what choice we make in the initial generation it is easier to generate the genome first and then obtain the path from there.
The conversion from genome to path part is done simply by following the conventions of the genome, that is, creating a LinePathPart if the first bit is 0 and a CirclePathPart otherwise. The length in either case is $0.5 + length * 0.5$, where length is the number represented by the second, third and fourth bit.
In case of a CirclePathPart we also need the angle, which is $10 + angle * 5$, where angle is the number represented by the fifth, sixth and seventh bit. The last bit sets the direction to either right (0) or left (1).

This conversion is done in blocks of 8 along all 160 bits of the given genome and returns a full LinkedList of 20 path parts.

4.2 Selection

Three selection methods have been implemented and tested (see 6) and can be selected from the program, however, by default it is set to use the tournament selection. The tournament size is set to 2, so 2 members will be selected at random and tested against each other. The selection size is 0.4 by default, so 40% of the next generation will consist of member of the previous generation (disregarding mutation, which is applied to all members), the remaining 60% will be obtained later using crossover between these 40%. Of the two members chosen the one with the higher rating will win. Since members are chosen at random without being removed from the population it is possible for the same member to be chosen several times. This implementation is not weighted, so all members have the same chance of being chosen, but bad members obviously have a low chance of winning, though it is not impossible so long as they are pitted against an even lower member. This keeps the algorithm from starving since it keeps non-ideal members within the population. The selection process can be adjusted by changing the tournament size, a greater number of participants per tournament lowers the chances of winning for members with a low rating.

4.3 Crossover

As with selection, several crossover methods have been implemented and can be selected in the program: Two-fixed-point crossover, single-bit crossover and eight-bit crossover. The later one is chosen by default, it generates a 20 Bit long mask genome randomly and then selects two parent genomes from the current population, also at random. Each Bit in the mask genome represents 8 Bit in the member genome, the value of the bit determines whether the 8 bit block for the child genome is taken from the first or second parent. A second child is also produced in every iteration which simply always gets the other parents block that was not selected for the first child. This eight-bit crossover produces very diverse children, more so than a one- or two-fixed-point crossover function where the switching between the two parents happens much less frequently, but it still preserves the path parts by only working in blocks of eight. Compared to the single bit crossover, which basically works the same but generates a 160 Bit mask and then selects single bits from the parents instead of using blocks of 8, the result is much less random and also significantly faster.
The two-fixed points crossover works similar to the eight-bit crossover, but instead of generating a mask genome to decide whether to take from the first or second parent it always selects the same bit range from the same parent, in our case bits 0 to 4 from parent 1, 5 to 15 from parent 2 and 16 to 19 from parent 1. These fixed points can of course be

moved around and their number can be adjusted, however if you further increased the number while still trying to preserve the path structure you would quickly end up with something very similar to the eight-bit crossover.
As mentioned before, 40% of the new population are obtained by selection, which leaves 60% for crossover, so our crossover rate is set to 0.6. Only half that many operations have to be computed however since every crossover generates two children.

4.4 Evaluation

Evaluation is applied as detailed in 5.6. It iterates over the entire current population, obtains a path from every genome, obtains a simulation for every path and then assigns a rating to the path/genome according to the fitness function.
When compared to the other genetic operations the evaluation probably holds the greatest potential for optimization as there is no fixed number of existing algorithms to choose from. The fitness function entirely depends on the problem to be worked on as well as the given representations of that problem, so it has to be created from scratch for every new task. The current implementation is rather simple and only checks how many pixels of the path are within a wall or outside the map and how far the destination reached is away from the target. Many additional variables to be considered or better ways of judging the currently considered values are possible, some of these will be discussed in 7.

4.5 Mutation

Mutation is applied on the entire new population after crossover and is supposed to keep the algorithm from starving out by possibly re-introducing bit combinations that have been lost in previous selections. Unlike with crossover and selection there is only one algorithms for this available so the only choice to be made here is the mutation rate, which determines the probability with which any bit is chosen, this is set to 0.001 by default. The mutation operator does not care about the genome's structure, it simply generates a random number for every bit of every genome in the entire population and if the number is one (0.1% chance) it flips the current bit.

4.6 Reordering

Unlike the operators discussed so far, the reordering operator is optional and not usually part of a GA. It works by randomly choosing an individual with a certain probability, for example 0.1, and then reversing the order of bits between two points within this genome, these points are also chosen at random. In certain cases this inversion has been shown to significantly improve the performance of the GA by preventing what is called *deception* [8]. This can happen when certain bits in a genome are important but very

far apart, which is called loose linkage. In such a case the crossover operator is likely to separate these building blocks even though they need to be together, something that were less likely to happen if the blocks were closer together. Whether the reordering operator is appropriate depends on the given problem and also the crossover operator employed. [8]

5

Implementation

The program accompanying this work is written in C# and uses the OpenTK Library for OpenGL graphics. Several C# features, such as functional and object-oriented programming, have been employed, but as this program is not meant as a framework or teaching tool, no special importance has been placed on the usage of specific programming models.
Originally the TAO OpenGL framework was used for the Graphics implementation, but this was later dropped due to the fact that most of TAO's libraries are severely outdated and produced several problems during map drawing. OpenTK has been chosen as an alternative due its simplicity, stability and high flexibility.
Visual Studio 2010 was used for development, other than OpenTK, which has to be installed separately, only standard libraries were used.

5.1 OpenTK

OpenTK is a free OpenGL library for .Net/Mono languages and is available across all Windows, Linux, Mac and other Unix-based Systems. It supports multi-monitor setups, a wide range of input devices as well as most GUI options. It also comes with integrated math toolkits for vectors, matrices etc. as well as type-safe bindings, automatic extension loading, error checking and inline documentation. OpenTK itself is also written in C#, but is compatible with all .Net/Mono languages [34].

5.2 Map

The map has to be given as a simple image (.png) file in which empty areas are white and walls are black. While a different representation of the environment may be better suited for collision detection, this simple map format makes it easier to create new maps for any given area. The map is assumed to be 2 dimensional, so height differences are not considered, and only about 40x20 m in size. The default image size is 1024x512 pixels, other sizes are possible but may lead to problems with the map overlay so any map should be re-sized to this. On start-up, a boolean array of the same size as the map is created and filled with values obtained from the given image, where every white pixel is true and every black pixel is false. This array is used for collision detection while the map itself is only used as a texture background for the GUI.

5.3 Vehicle

The vehicle is saved in a format similar to the one described in 2.2 however, all vehicle parts are saved with absolute positions instead of using values relative to the previous trailer. While this requires more space it makes collision detection and drawing of the vehicle a lot easier. For every element of the vehicle the values M and L are saved, which determine the length of the front and rear of the trailer respectively, measured from the axle. In accordance with that representation, the tractive unit is assumed to be 2 parts with a rigid coupling in between. A tractive unit without any trailers is generated randomly every time the program is started, so one can immediately start planning a path without having to configure a vehicle first every time. If a specific vehicle is required, adjustments can be made in the vehicle tab of the GUI. In this tab, M and L can be adjusted for every part of the vehicle, more trailers can be added and vehicles can be saved for later use. While the program can theoretically handle an infinite number of trailers it is currently meant to be used with only one.
For a complete representation of the vehicle more information is required than just the size of every given part, that is, the configuration of the vehicle, found in the configuration tab. This configuration contains the starting point of the vehicle, as well as the angles between all its elements. Effectively, only the angles can be adjusted, the position of the steering vehicle is assumed to be the same as the starting position of the path, set in the main tab, and all other vehicle parts' positions are obtained from that starting position and the elements' angles. The configuration, too, can be saved and loaded.

5.4 Path Representation

To make the representation within the GA easier, certain limitations are enforced on the paths generated by this program, however, most of these are not really constraining since a general-n-trailer is limited in its possible configuration space to begin with. In order to

get a fixed genome length we assume every path to be made up of exactly 20 parts. Paths of length 0 are not possible since any given path part has a minimum length of 0.5 meters, similarly, paths longer than 80 meters are not possible but should not be necessary either due to the limitation of the map-size.
A path is represented as a LinkedList of path primitives, which can be either a curve or a line. A curve, or CirclePathPart, has an angle, a start angle, an end angle, a radius, a center and a direction, that is, whether it is a left or a right curve. A LinePathPart has a start, end and direction, as well as a speed and a boolean value determining whether the vehicle is driving in reverse or not, however neither of these are used in the current implementation of the program and are simply set to default values. The values of any given path part are either set randomly (5.5), obtained from a newly generated genome (4) or set by the previous path part, for example, the direction of a LinePathPart can only be the direction of the previous PathPart since no rough edges are permitted. Due to the selected information encoded in the genome the maximum value of any of these variables as well as the step size between these values are limited. Any given path part can be between 0.5 and 4 meters long and, in case it is a curve, have an angle between 10° and 45° in steps of 5°. A more detailed path can easily be achieved by extending the length of the genome (4.1), but this would also slow down computation.

5.5 Generation 0

The initial generation, also known as generation 0, is obtained by generating random paths and converting these into the genome format, see 4.1 for further details on the exact representation. A function generatePath() in the PathPrimitives class is used to randomly generate a path of a fixed length, by default 20 path parts, as well as calculating the corresponding genome. This function generatePath() uses the function getRandomPathPart() of the same class, which generates a path part within the limitations given in 5.4. After enough paths have been obtained, all of them are evaluated.

5.6 Evaluation

The evaluation is done using a simple weighted fitness function which considers both the distance from the end of the path to the original goal as well as the number of collisions, leaving the map is also counted towards the collision value. The weight of these two factors can be adjusted but is by default set to 2 to 3 for the goal distance. More factors, like the length of the path, the number of turns or the minimal distance to obstacles could be considered in this function as well, but are currently not implemented. It should be noted that the path evaluated here is not the path obtained from the GA directly, but the generated path after it has gone through the simulation class, which uses the path optimization from AG Echtzeitysteme to transform it into a path drivable by the given vehicle.

5.7 Generation 1+

Further generations are evaluated in the same way as the initial one, but are obtained by genetic crossing and not randomly. The exact process used in this implementation of the genetic algorithm is described in 4.

5.8 GUI

The full GUI can be seen in fig. 5.1. On the left we can see a sample map which can be replaced either in the code or by clicking the "'Load Map"' button on the right and providing a fitting image, see 5.2. Black areas on the map are obstacles while white areas are available. The blue dot marks our target destination, the red one our starting point. The lines at the red dot are our current vehicle, the tiny red dot within our vehicle is a coupling, in this case a rigid one since we only have a tractive unit by default. On the right of the map we have buttons for loading a new map, starting the path finding algorithm (Start), starting an entire set of iterations for evaluation purposes (Evaluation), drawing a single random path (Start Path) and obtaining a simulation for a given path (Drive). Next to the "'Drive"' button is a small text box which shows the current generation the algorithm is computing. The current path's genome is shown in the box below those five buttons, every line consists of 8 numbers and represents exactly one path part, as such there are 20 lines. This box is usually read-only, but it can be modified by unchecking the "'Genome Read Only"' box next to it. This way, a previously saved path's genome can be put in and a simulation can be obtained by pressing "'Drive"'. The "'Show Current End"' button shows the configuration of the vehicle at the end of the current path, this could be used in the fitness function but is not yet implemented. The "'Debug Population"' check box causes the algorithm to stop after each generation and output all computed paths along with their rating.
The GUI tab on the right allows us to save, load and modify our general-n-trailer. Each vehicle part is defined by its M and L values, see 2.2.
The configuration tab allows the modification of the current vehicle's position and steering angles. Just like the vehicle, this configuration can be saved, loaded and reset, it should be noted that it is only possible to load a configuration when it fits the current vehicle's number of trailers. In the same tab, the start and end coordinates can bet set, the colours correspond to the points on the map. By selecting "'End Conf"' on the right a target configuration can be set, however, this is not needed at the moment since only the distance to the destination is considered in the fitness function, not the entire configuration.

Figure 5.1: The full view of the GUI with a sample map on the left and the Main window on the right

6

Evaluation

6.1 No Trailer, Generation Count of 50

In the following chapter we will compare the results, that is the path rating, and the required computation time for several different versions of the genetic algorithm. Since there are many values and functions within the GA that can be adjusted and influence its performance, only a few can be considered here. The conclusion drawn in the next chapter 7 will be based on the evaluation here.
We will go from a the simplest case (no trailer) to a more complicated case (1 trailer) and run each scenario twice, once with 50 and once with 100 generations.

The following table shows the rating achieved (higher is better) as well as the time needed for computation (lower is better) for several different setups of the algorithm. For each setup, 7 runs and an average are given. Only the choice in GA functions used is changed between different runs, the remaining settings are consistent throughout the table and are as follows.

The population size is 1000 ,the crossover rate is 60%, a mutation rate of 0.1% and the algorithm works without reordering operation throughout all our tests. It starts at 175,350 and the destination is 740,100 (both are the program's default values). For this first test, the vehicle has no trailer and our algorithm runs for 50 generations. The vehicles has an M value of 24,55, an L value of 42,5, and a starting angle of 45, these values have been randomly generated and were then kept for all test runs as well.
Computation was done on a Core2Quad 6600@2.4GHz with 4GB of System Memory.

In these first four tests we want to compare two selection (Tournament and RouletteWheel) and two crossover (EightBit and TwoPoint) strategies. Based on given knowledge we expect the EightBit crossover to outperform the TwoPoint crossover and the Tournament selection to be work better than the RouletteWheel selection [14].

Run	8Bit, Tournament	8Bit, Roulette	TwoPoint, Tournament	TwoPoint, Roulette
1	152 in 1:50	222 in 1:44	647 in 1:53	49 in 1:57
2	90 in 1:47	134 in 1:51	111 in 154 6.1	127 in 2:01 6.2
3	127 in 1:48	106 in 1:54	240 in 1:48	51 in 1:58
4	240 in :144	61 in 1:46	122 in 1:48	130 in 2:01
5	152 in 1:49	222 in 1:57	274 in 1:49	156 in 1:54
6	163 in 1:49	201 in 1:56	56 in 1:57	78 in 1:51
7	90 in 2:06	76 in 1:53	59 in 1:47	230 in 1:56
Best	240 in 1:44	222 in 1:44	647 in 1:53	156 in 1:54
Worst	90 in 2:06	61 in 1:46	56 in 1:57	49 in 1:57
Average	144 in 1:51	146 in 1:51	198 in 1:50	117 in 1:56

Table 6.1: No Trailer, Generation Count of 50

As can be seen from these results, the time is rather consistent at just below 2 minutes. This of course depends heavily on the hardware used, but surprisingly very little on the operations chosen within the GA. The results are not quite as consistent and have to be handled carefully due to the small sample size. Row 3 (TowPointCrossover, TournamentSelection) seems to deliver the best results, but this is largely due to a single very good path, which can be attributed to "'luck"' in the randomly generated path. If we disregard this single path we get an average of 143, which is almost the same as in the first two rows. The last row (TwoPointCrossover, RouletteWheelSelection) however produces sub-optimal paths rather consistently. It is also noticeable that the difference between the best and worst path is much smaller in the first row when compared to the other three, which is why this setting has been chosen as the default one as it produces an acceptable quality of paths consistently. These results may of course change with a (much) larger, see 6.5.

The overall quality of the paths is rather low, which we will try to mitigate by raising the generation count in the next test, the real cause for this however is probably our primitive fitness function, see 7 for further discussion. Right now, the algorithm depends heavily on its initial random paths which leads to very different and overall sub-optimal results. Two interesting paths are given in fig. 6.1 and fig. 6.2. Here, red is the path planned by the GA and blue is the actual, drivable, path obtained via simulation. In fig 6.1 we see an example of a good path which avoids all obstacles and gets very close to the target position. Whether or not it also hits the target configuration is not considered by the current

fitness function. The path got a rating of 111, which is rather low compared to the ratings of some of the other, objectively worse, paths, which also points to the fitness function as the algorithms current weakness. 6.2 shows a different problem with the current evaluation as it shows a path with a good rating (127) which also seems fine for the most part, but is completely impossible in reality as it passes through a wall. The current collision detection gives a worse rating the "'longer"' the path stays within the black area, which means that crossing a thin wall like this does not lower the rating by much when compared to the positive rating it receives from the distance evaluation, even though the path is obviously still unusable. The current evaluation emphasizes distance to the target over the collision detection in order to obtain more acceptable paths quickly, but further optimization is required to make sure this does not come at the cost of accepting impossible paths.

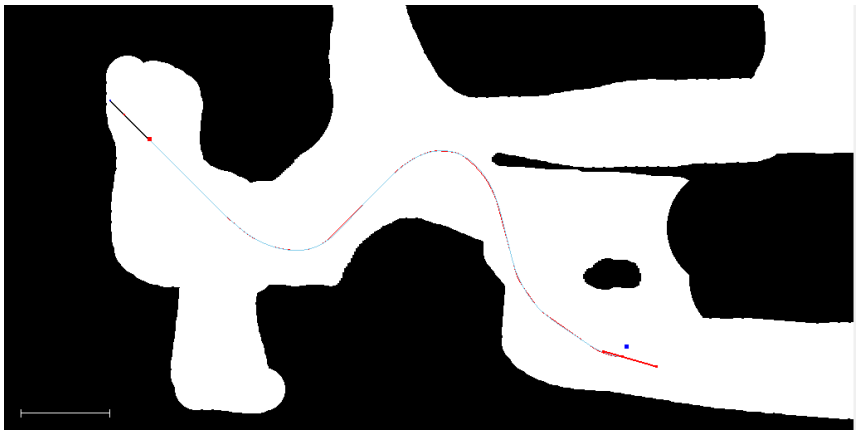

Figure 6.1: Example of a good result computed by the GA

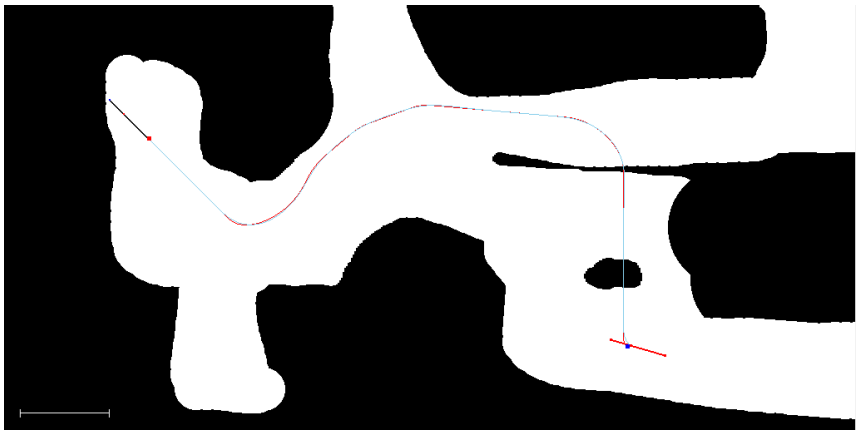

Figure 6.2: A bad path with a good rating produced by the GA

6.2 No Trailer, Generation Count of 100

The settings in the following table are the same as in the previous ones, but the generation count has been doubled to 100 in hopes of getting better paths with more generations. This could, however, also lead to starvation of the algorithm.
Due to time constraints, all evaluations after this points were done on an i5-M460@2,53 GHz instead of the Q6600 stated above. For comparison, the computation of the above table took an average of around 1:25, the same as in 6.3, so about 25 seconds faster. Since, as we will see, the addition of a trailer does not impede the speed of the algorithm, we can simply make comparisons using 6.3.

Run	8Bit, Tournament	8Bit, Roulette	TwoPoint, Tournament	TwoPoint, Roulette
1 291 in 2:44	146 in 2:33	73 in 2:38	93 in 2:37	
2	138 in 2:36	143 in 2:35	175 in 2:38	56 in 2:38
3	235 in 2:35	194 in 2:41	123 in 2:37	48 in 2:39
4	109 in 2:35	65 in 2:42	105 in 2:47	78 in 2:36
5	191 in 2:39	101 in 2:38	539 in 2:36	69 in 2:41
6	259 in 2:48	148 in 2:39	211 in 2:36	180 in 2:36
7	67 in 2:43	156 in 2:35	136 in 2:37	166 in 2:38
Best	291 in 2:44	194 in 2:41	539 in 2:63	180 in 2:36
Worst	67 in 2:43	65 in 2:42	73 in 2:36	48 in 2:39
Average	184 in 2:40	136 in 2:37	194 in 2:38	98 in 1:37

Table 6.2: No Trailer, Generation Count of 100

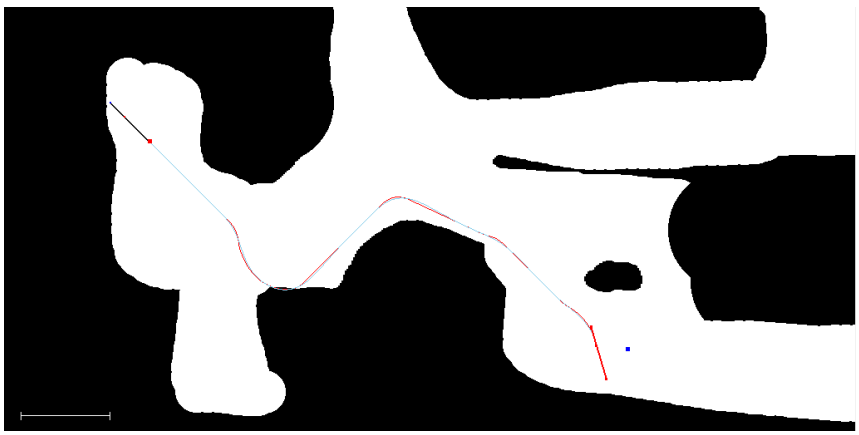

Figure 6.3: Another bad path with a good rating produced by the GA

Only the first row (EightBitCrossover, TournamentSelection) shows a significant improvement with the doubled number of generations, the other are very close to their previous values. The third row, again, takes its good average mostly from a single very good path, if we ignore it as an anomaly we get a result of only 137, which is also comparable to its previous average. While the overall quality of paths has not improved by a lot, it is now more obvious than before that the setting chosen in the first row presents the best case scenario for this algorithm. Computation time is a little less than twice that of a 50 generation computation done on the same hardware (see 6.3), this is not surprising since computation done on each generation is the same so the time required for each iteration can be assumed to be constant, thus doubling the generation count should also double the computation time. The slight offset probably comes from the initial generation, which takes longer than any other and is not affected by the increase in the generation count.

6.3 One Trailer, Generation Count of 50

The settings in the following table are the same as in the previous ones, except that the vehicle now has one trailer and the generation count has been lowered to 50 again.

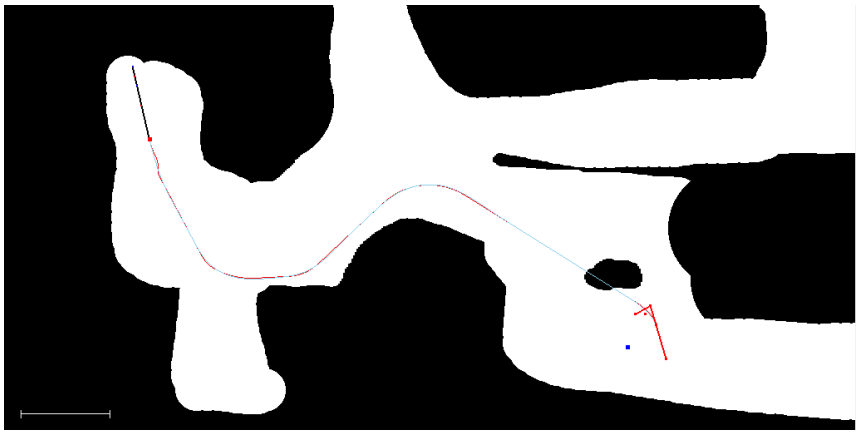

Figure 6.4: A (problem) path for a vehicle with one trailer.

Run	8Bit, Tournament	8Bit, Roulette	TwoPoint, Tournament	TwoPoint, Roulette
1	436 in 1:24	434 in 1:24	104 in 1:24	120 in 1:25
2	144 in 1:25	211 in 1:19	78 in 1:24	44 in 1:23
3	117 in 1:24	113 in 1:22	68 in 1:21	62 in 1:24
4	58 in 1:24	151 in 1:22	102 in 1:21	42 in 1:25
5	215 in 1:24	87 in 1:24	75 in 1:22	119 in 1:21
6	153 in 1:21	170 in 1:22	107 in 1:20	78 in 1:22
7	66 in 1:22	179 in 1:20	138 in 1:24	152 in 1:23
Best	436 in 1:24	434 in 1:24	138 in 1:24	152 in 1:23
Worst	58 in 1:24	87 in 1:24	68 in 1:21	42 in 1:25
Average	169 in 1:23	192 in 1:21	96 in 1:22	88 in 1:23

Table 6.3: One Trailer, Generation Count of 50

The first thing noticeable about this table is that the overall quality of paths has clearly decreased when compared to 6.3. The addition of a trailer seems to influence the algorithm more than the generation count, however, time does not seem to be significantly affected, which is precisely what we want from a GA: The ability to compute very complex (in our case this means more trailers) problems in acceptable time. Since we changed hardware since the original 50 generation/no trailer run we cannot draw a clear conclusion here, but when comparing 6.3 and 6.4 later we will be able do so. It is also notable that the last two rows, which employ the TwoPointCrossover, are now significantly worse than the first 2, which was not that noticeable before. Also, our second row is now better than our first, however, this may also be due to our small sample size. We will see how this changes for a generation count of 100 in the following section.

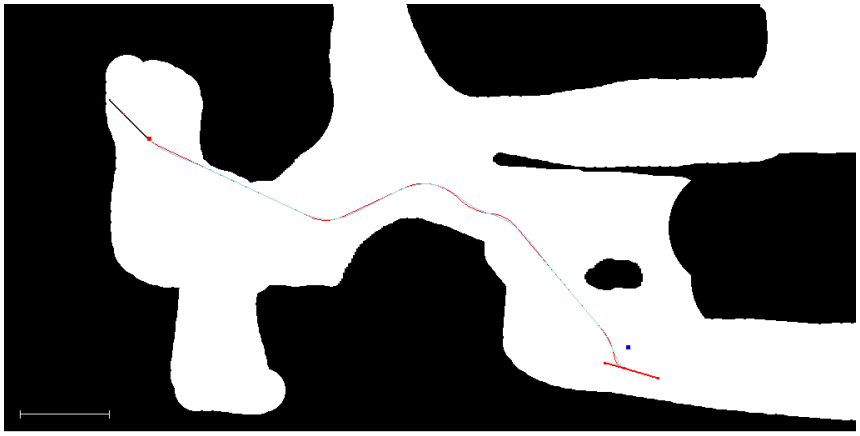

Figure 6.5: Another example of a good result computed by the GA

6.4 One Trailer, Generation Count of 100

The settings in the following table are the same as in the previous ones, but the generation count has been doubled to 100 again. The vehicle still has one trailer.

Run	8Bit, Tournament	8Bit, Roulette	TwoPoint, Tournament	TwoPoint, Roulette
1	200 in 2:40	93 in 2:43	73 in 2:44	74 in 2:38
2	77 in 2:36	59 in 2:46	62 in 2:35	60 in 2:45
3	126 in 2:36	100 in 2:40	45 in 2:46	129 in 2:44
4	58 in 2:41	131 in 2:40	135 in 2:50	101 in 2:43
5	96 in 2:40	261 in 2:40	118 in 2:44	61 in 2:45
6	59 in 2:43	128 in 2:41	71 in 2:45	84 in 2:38
7	97 in 2:37	133 in 2:41	47 in 2:42	64 in 2:37
Best	240 in 1:44	222 in 1:44	647 in 1:53	156 in 1:54
Worst	90 in 2:06	61 in 1:46	56 in 1:57	49 in 1:57
Average	144 in 1:51	146 in 1:51	198 in 1:50	117 in 1:56

Table 6.4: One Trailer, Generation Count of 100

We can now clearly see that the addition of one trailer had almost no impact on our required computation time, in row 3 the time is up by 5 seconds, in 2 and 4 it is up by 4 seconds and in row 1 it is even down by 1 second. The overall quality of paths has further diminished, so using a generation count of 100 does not seem to be a good idea as the algorithm starves out instead of producing better results. Row 2 still produced the best results, but is also not very consistent, the TwoPointCrossover-based results are still far below the EightBitCrossover ones. We will further compare row 1 and 2 with a larger sample size in the next section.

We will conclude here that the addition of trailers has far more impact on the quality of the path, which is bad, than on the computation time, which is good. We can also clearly see that the EightBitCrossover operation outperforms the TwoPointCrossover, which is what we expected. The difference between using Tournament or RouletteWheel Selection is not nearly as clear, we will need a much larger sample size to draw a conclusion here.
There also seems to be a limit as to how much we can raise the generation count before we lower the quality instead of raising it. We currently conclude that 50 generations are better than 100, though whether or not there is another, better, value than that will require further testing.

6.5 Further Testing

The first two rows, that is EightBit Crossover and Tournament or RouletteWheelSelection have been selected for further testing with a larger sample size. The generation count has been set to 50 again and all other settings are the same as above, computation was done on the same Core i5 system. Due to the large sample size (100 runs without trailer, 500 runs with 1 trailer) the tables will not be given here, but instead we will discuss the results.
Without a trailer we get an average rating of 184 for the first row (TournamentSelection) and 161 for the second (RouletteWheelSelection), which confirms our previous assumption that the tournament selection is better than the roulette wheel selection, even though our small scale tests seemed to contradict that. We get an average time of 1:19 each (1:19:070707 and 1:19:212121), so we can safely assume that the choice in selection function has no influence on our computation time. The best paths had a rating of 1383 and 584, the worst 39 and 56 respectively. This means an average deviance of 109 for the first row and only 72 for the second, which, again, contradicts our small scale testing.
Our second test case, that is a vehicle with one trailer, received an even larger sample size of 500 runs. The setup is the same as above, so the first row has results for tournament selection, the second for roulette wheel selection. The averages are 176 and 163 respectively, which means that our tournament selection got slightly worse, but not as much as our small scale test would have us believe, while the roulette wheel selection remains almost unaffected, but still worse overall. The average deviances are 98 and 88, so much closer than in our previous test with no trailer, the same goes for our best and worst paths, which are 2358, 2716 and 39, 43. The average times are 1:21 in both cases (1:21:833667 and 1:21:733466) which, again, shows that the addition of a trailer makes almost no difference and that is even less of a difference between the two selection methods.
As we expected in the beginning, TournamentSelection seems to outperform RouletteWheel Selection, albeit not by much.

7

Conclusion and Future Work

We have shown the usability of machine learning algorithms, with the genetic algorithm as an example, as a tool for finding or optimizing paths in high dimensional spaces under consideration of nonholonomic constraints. We have also introduced alternative algorithms and, as far as possible without actually implementing every possible algorithm under the same circumstances, made some comparisons to our own results. As already mentioned in 4 many more alternative implementations and further optimizations of the genetic algorithm are possible, some of which will be summarized here.

The accuracy of the GA could be further optimized by adjusting the collision detection to avoid getting close to obstacles, not just avoiding the obstacles themselves. The overall quality of the path could be raised by considering more factors, such as length, number of turns or minimum distance to any obstacle along the path.
The current implementation is tested to provide the best performance with the given operators, however, since there are many different possible GA operators and many ways to weight and combine these, there are possibly combinations that would provide better results that have not been considered here. While there are certain operators that are more suited to a given task than other, many of these preferences can only be determined by extensive testing of all possibilities, which would go beyond the scope of this thesis.
The current version of the program makes two assumptions about the vehicle that may be lifted with slight alterations and further testing: The vehicle is always driving forward, never changing direction and the vehicle only has a maximum of one trailer. The software is already capable of handling an infinite number of trailers, however, since this has not been tested, it is currently not possible to define more than one trailer in the GUI. This assumption is made mostly due to the fact that general-n-trailers with infinite trailers

are purely theoretical. As shown in [8], GAs are not slowed down by higher complexity as much as alternative solutions, so while the performance of our proposed algorithm would certainly suffer from having to consider more trailers, it would not slow down as much as iterative algorithms and would further outperform classic solutions. Our own testing in 6 confirms this assumption.
Changing directions during the drive is partially implemented insofar that both the genome and the path part classes contain the information necessary, in the current version these are all set to default values however. The program has not been tested with such paths and the current random path generator does not allow for the path to contain corners since it always uses the direction of the end of the previous path part when generating the next one. This limitation would have to be loosened since a change in direction would obviously break this restriction.
The algorithm may also be optimized at its very basis, that is, the genome representation. The current representation limits both the maximum and minimum length of the path as well as the granularity of the choices made. By using different or simply more detailed representations the algorithm may be able to come up with better paths, gain better performance or both.

While many such optimizations are possible and are certainly already being worked on by various research groups around the world, the purpose of this paper was not to develop a best possible algorithm, but simply to show that such machine learning algorithms for path-planning tasks exist and are actually useful. While the given algorithm may be far from optimal, it is a working solution for high dimensional problem which, depending on the chosen complexity, can outperform classic path planning algorithms.

Appendix

Table A.1: Eight Bit, No Trailer, Generation Count of 100

Run	Tournament	Time	Roulette	Time
01	62	1:18	154	1:21
02	132	1:20	133	1:20
03	83	1:20	236	1:20
04	90	1:21	138	1:20
05	91	1:18	194	1:23
06	200	1:21	138	1:18
07	86	1:24	220	1:18
08	39	1:18	159	1:19
09	122	1:20	90,	1:17
10	81	1:17	498	1:18
11	249	1:21	109	1:18
12	71	1:19	127	1:18
13	308	1:18	113	1:19
14	105	1:24	584	1:21
15	100	1:23	264	1:17
16	184	1:23	323	1:20
17	235	1:22	122	1:19
18	205	1:18	173	1:18
19	288	1:23	108	1:20

20	328	1:20	77	1:21
21	94	1:23	80	1:20
22	96	1:20	219	1:17
23	239	1:22	85	1:17
24	129	1:16	92	1:19
25	68	1:18	172	1:20
26	163	1:19	91	1:20
27	124	1:21	218	1:18
28	113	1:19	78	1:17
29	507	1:22	84	1:21
30	175	1:18	190	1:18
31	171	1:19	91	1:17
32	63	1:18	142	1:16
33	71	1:17	82	1:21
34	115	1:23	260	1:23
35	72	1:20	89	1:19
36	1383	1:18	169	1:17
37	302	1:16	237	1:17
38	197	1:22	172	1:20
39	297	1:18	102	1:19
40	132	1:19	75	1:20
41	127	1:18	120	1:19
42	60	1:19	83	1:18
43	120	1:18	102	1:18
44	84	1:17	207	1:20
45	97	1:15	119	1:21
46	66	1:18	65	1:19
47	340	1:19	239	1:19
48	202	1:18	171	1:18
49	71	1:19	107	1:20
50	451	1:17	386	1:17
51	88	1:18	134	1:18
52	58	1:17	211	1:18
53	69	1:20	232	1:22
54	115	1:21	78	1:19
55	175	1:21	124	1:21
56	54	1:19	114	1:19
57	156	1:17	98	1:20
58	154	1:20	69	1:20

59	175	1:20	118	1:15
60	98	1:20	101	1:19
61	350	1:18	126	1:22
62	131	1:22	257	1:18
63	757	1:18	114	1:21
64	479	1:15	102	1:17
65	111	1:21	58	1:18
66	456	1:20	94	1:18
67	331	1:19	143	1:21
68	75,	1:18	242	1:18
69	164	1:19	134	1:20
70	54	1:19	118	1:19
71	39	1:24	120	1:18
72	360	1:19	79	1:19
73	222	1:17	269	1:19
74	86	1:16	83	1:23
75	114	1:21	168	1:21
76	127	1:18	521	1:18
77	60	1:20	217	1:19
78	98	1:17	271	1:18
79	171	1:16	104	1:21
80	188	1:18	268	1:17
81	125	1:22	167	1:17
82	965	1:22	85	1:19
83	216	1:21	391	1:17
84	160	1:18	214	1:21
85	95	1:18	75	1:20
86	123	1:18	389	1:19
87	228	1:16	113	1:21
88	77	1:18	148	1:18
89	80	1:16	117	1:24
90	56	1:16	106	1:22
91	246	1:17	119	1:21
92	92	1:15	153	1:19
93	228	1:19	108	1:20
94	90	1:18	70	1:21
95	160	1:16	87	1:20
96	109	1:18	241	1:22
97	170	1:17	217	1:17

98	208	1:20	56	1:20
99	138	1:21	75	1:18
Average	184	1:19	161	1:19
Maximum	1383	1:24	584	1:24
Minimum	39	1:15	56	1:15

Table A.2: Eight Bit, One Trailer, Generation Count of 100

Run	Tournament	Time	Roulette	Time
001	70	1:20	79	1:27
002	72	1:22	74	1:20
003	144	1:22	87	1:21
004	207	1:23	148	1:22
005	373	1:21	106	1:22
006	324	1:21	101	1:23
007	66	1:21	202	1:21
008	71	1:22	128	1:21
009	239	1:25	915	1:21
010	630	1:21	328	1:21
011	125	1:20	193	1:19
012	198	1:21	56	1:23
013	122	1:23	138	1:20
014	163	1:24	150	1:22
015	463	1:21	91	1:23
016	123	1:20	90	1:24
017	485	1:22	272	1:19
018	75	1:23	54	1:23
019	137	1:21	109	1:18
020	64	1:22	397	1:21
021	680	1:22	80	1:21
022	61	1:21	58	1:21
023	226	1:21	99	1:22
024	106	1:21	130	1:22
025	642	1:23	57	1:20
026	525	1:23	168	1:21
027	110	1:22	84	1:25
028	72	1:21	102	1:22
029	207	1:22	154	1:18

030	141	1:21	109	1:20
031	203	1:21	93	1:20
032	106	1:21	105	1:21
033	58	1:20	328	1:22
034	223	1:19	101	1:22
035	43	1:21	157	1:20
036	161	1:20	91	1:22
037	177	1:21	340	1:24
038	63	1:21	68	1:21
039	83	1:18	171	1:22
040	287	1:25	73	1:23
041	102	1:24	132	1:23
042	177	1:21	85	1:20
043	200	1:24	564	1:21
044	231	1:24	71	1:23
045	111	1:22	77	1:20
046	77	1:24	109	1:21
047	2358	1:22	96	1:26
048	130	1:22	235	1:22
049	159	1:20	60	1:22
050	220	1:24	122	1:23
051	153	1:22	181	1:20
052	99	1:24	156	1:23
053	237	1:19	82	1:23
054	185	1:21	96	1:24
055	89	1:21	164	1:21
056	78	1:22	159	1:22
057	168	1:20	112	1:23
058	324	1:24	149	1:24
059	74	1:22	207	1:21
060	167	1:22	204	1:21
061	760	1:20	90	1:22
062	210	1:20	182	1:22
063	186	1:23	112	1:19
064	234	1:23	112	1:22
065	61	1:19	87	1:21
066	436	1:22	2716	1:23
067	582	1:19	107	1:23
068	187	1:23	65	1:20

069	78	1:23	235	1:20
070	73	1:20	176	1:18
071	113	1:23	169	1:21
072	103	1:24	96	1:20
073	143	1:21	77	1:23
074	209	1:20	142	1:19
075	143	1:22	159	1:23
076	109	1:23	149	1:22
077	101	1:20	90	1:24
078	98	1:22	106	1:20
079	95	1:22	130	1:22
080	103	1:23	325	1:22
081	91	1:18	100	1:23
082	181	1:24	324	1:22
083	241	1:24	86	1:21
084	296	1:21	63	1:22
085	57	1:23	89	1:19
086	264	1:20	73	1:21
087	59	1:19	539	1:21
088	123	1:21	157	1:21
089	126	1:20	241	1:20
090	106	1:21	89	1:23
091	130	1:20	134	1:22
092	185	1:21	91	1:21
093	158	1:24	72	1:21
094	64	1:21	207	1:26
095	82	1:20	236	1:20
096	366	1:19	165	1:23
097	234	1:18	169	1:21
098	122	1:20	297	1:20
099	160	1:22	147	1:24
100	78	1:19	56	1:22
101	102	1:20	273	1:24
102	107	1:20	151	1:22
103	167	1:22	226	1:23
104	66	1:19	130	1:20
105	249	1:22	97	1:22
106	597	1:22	203	1:20
107	164	1:27	71	1:21

108	83	1:24	121	1:20
109	398	1:20	135	1:19
110	111	1:24	141	1:23
111	131	1:17	69	1:19
112	107	1:20	617	1:18
113	174	1:20	79	1:21
114	115	1:21	235	1:22
115	113	1:23	59	1:23
116	90	1:20	340	1:19
117	203	1:23	165	1:22
118	144	1:22	99	1:23
119	70	1:23	139	1:20
120	75	1:20	143	1:21
121	75	1:21	103	1:23
122	217	1:20	469	1:25
123	386	1:23	358	1:22
124	80	1:25	164	1:22
125	572	1:29	48	1:22
126	96	1:21	62	1:21
127	85	1:25	79	1:25
128	91	1:20	93	1:19
129	74	1:19	303	1:21
130	523	1:22	148	1:21
131	104	1:23	124	1:24
132	140	1:21	106	1:22
133	156	1:18	132	1:22
134	82	1:21	165	1:22
135	59	1:21	175	1:20
136	153	1:20	300	1:19
137	61	1:22	87	1:19
138	110	1:24	73	1:22
139	369	1:22	160	1:24
140	111	1:19	43	1:24
141	48	1:21	123	1:19
142	146	1:22	157	1:20
143	131	1:22	82	1:20
144	62	1:20	75	1:21
145	282	1:20	289	1:25
146	155	1:23	93	1:19

147	81	1:21	99	1:21
148	58	1:19	83	1:22
149	70	1:23	91	1:24
150	219	1:22	141	1:22
151	243	1:18	62	1:21
152	112	1:22	100	1:24
153	1058	1:21	108	1:20
154	330	1:22	67	1:23
155	101	1:25	162	1:23
156	162	1:20	114	1:20
157	98	1:21	70	1:21
158	1043	1:18	67	1:21
159	117	1:19	81	1:19
160	205	1:21	131	1:20
161	300	1:21	412	1:22
162	72	2:82	86	1:21
163	106	1:22	119	1:20
164	175	1:19	227	1:23
165	118	1:21	169	1:22
166	118	1:18	188	1:19
167	110	1:24	71	1:19
168	534	1:19	64	1:21
169	65	1:20	161	1:20
170	121	1:20	351	1:22
171	98	1:19	178	1:20
172	95	1:21	83	1:19
173	248	1:19	111	1:21
174	48	1:23	72	1:22
175	153	1:23	96	1:22
176	105	1:20	132	1:24
177	167	1:18	133	1:22
178	80	1:23	195	1:21
179	127	1:23	92	1:20
180	300	1:22	90	1:22
181	75	1:23	386	1:21
182	312	1:22	621	1:22
183	518	1:22	546	1:22
184	110	1:27	77	1:22
185	64	1:22	143	1:21

186	110	1:22	96	1:22
187	150	1:20	202	1:21
188	169	1:22	145	1:21
189	122	1:21	73	1:18
190	336	1:21	59	1:20
191	79	1:21	45	1:22
192	306	1:21	96	1:19
193	68	1:25	135	1:22
194	87	1:22	115	1:22
195	118	1:24	58	1:22
196	722	1:20	188	1:21
197	120	1:21	134	1:20
198	146	1:25	74	1:23
199	103	1:24	75	1:20
200	58	1:20	128	1:19
201	51	1:23	97	1:23
202	82	1:23	155	1:20
203	78	1:26	108	1:21
204	281	1:21	96	1:20
205	95	1:21	121	1:21
206	263	1:20	417	1:21
207	118	1:21	250	1:23
208	80	1:20	160	1:24
209	147	1:22	87	1:22
210	57	1:20	450	1:24
211	62	1:20	282	1:22
212	97	1:22	162	1:22
213	77	1:23	124	1:22
214	926	1:21	119	1:25
215	238	1:21	163	1:19
216	157	1:25	84	1:23
217	62	1:20	106	1:22
218	502	1:22	194	1:22
219	77	1:23	90	1:22
220	62	1:25	67	1:20
221	256	1:18	202	1:20
222	121	1:20	255	1:25
223	70	1:25	139	1:22
224	124	1:25	159	1:22

225	377	1:21	89	1:24
226	77	1:23	66	1:23
227	424	1:23	491	1:21
228	147	1:23	119	1:21
229	101	1:23	115	1:20
230	280	1:21	173	1:21
231	79	1:24	55	1:19
232	75	1:20	180	1:20
233	299	1:21	191	1:22
234	153	1:22	199	1:19
235	142	1:21	211	1:20
236	356	1:24	77	1:23
237	100	1:24	73	1:22
238	200	1:21	59	1:21
239	365	1:26	69	1:28
240	63	1:28	78	1:20
241	200	1:22	151	1:19
242	177	1:23	85	1:21
243	590	1:20	179	1:25
244	106	1:24	106	1:19
245	105	1:21	154	1:19
246	182	1:23	69	1:23
247	189	1:21	80	1:24
248	108	1:24	124	1:21
249	173	1:20	238	1:20
250	189	1:22	80	1:22
251	143	1:24	118	1:23
252	110	1:22	157	1:23
253	268	1:25	134	1:22
254	119	1:22	201	1:22
255	102	1:23	154	1:20
256	247	1:23	95	1:23
257	92	1:21	66	1:22
258	319	1:22	274	1:23
259	949	1:23	138	1:24
260	56	1:27	439	1:22
261	66	1:24	63	1:22
262	276	1:23	105	1:21
263	373	1:22	74	1:19

264	351	1:20	71	1:22
265	104	1:23	188	1:21
266	82	1:20	132	1:20
267	226	1:21	271	1:22
268	70	1:21	73	1:20
269	56	1:19	71	1:21
270	211	1:22	106	1:24
271	509	1:24	275	1:23
272	119	1:19	141	1:19
273	407	1:23	175	1:22
274	161	1:20	667	1:26
275	181	1:19	151	1:19
276	142	1:20	69	1:22
277	164	1:21	137	1:26
278	157	1:23	70	1:22
279	108	1:26	234	1:24
280	70	1:22	211	1:23
281	435	1:23	73	1:24
282	58	1:22	74	1:22
283	105	1:24	93	1:22
284	128	1:19	56	1:20
285	161	1:21	411	1:19
286	148	1:22	143	1:20
287	87	1:26	96	1:23
288	88	1:21	153	1:22
289	94	1:24	69	1:20
290	149	1:21	44	1:21
291	80	1:24	168	1:19
292	103	1:22	81	1:20
293	216	1:21	100	1:21
294	65	1:22	73	1:20
295	60	1:24	158	1:19
296	120	1:21	215	1:23
297	118	1:26	147	1:21
298	187	1:22	91	1:21
299	174	1:21	43	1:23
300	187	1:26	86	1:17
301	123	1:24	563	1:20
302	156	1:22	100	1:22

303	266	1:20	93	1:22
304	156	1:20	75	1:24
305	147	1:20	161	1:21
306	215	1:20	141	1:20
307	210	1:20	77	1:22
308	89	1:22	184	1:20
309	168	1:21	101	1:19
310	187	1:22	287	1:21
311	52	1:22	86	1:20
312	156	1:23	213	1:21
313	157	1:23	83	1:21
314	60	1:23	97	1:22
315	170	1:23	108	1:20
316	378	1:22	51	1:20
317	293	1:25	72	1:21
318	53	1:21	210	1:25
319	91	1:24	168	1:20
320	194	1:24	88	1:22
321	77	1:25	100	1:22
322	363	1:24	115	1:24
323	118	1:23	170	1:22
324	77	1:20	341	1:22
325	56	1:23	111	1:23
326	64	1:23	85	1:22
327	137	1:25	128	1:22
328	95	1:25	290	1:20
329	459	1:21	111	1:20
330	67	1:24	196	1:22
331	151	1:22	215	1:21
332	74	1:20	196	1:20
333	62	1:23	64	1:24
334	74	1:24	63	1:22
335	198	1:20	125	1:19
336	375	1:24	79	1:22
337	67	1:23	127	1:21
338	133	1:21	114	1:20
339	79	1:21	194	1:22
340	432	1:23	216	1:21
341	113	1:20	81	1:23

342	150	1:21	88	1:22
343	65	1:20	223	1:23
344	83	1:23	100	1:20
345	88	1:26	344	1:20
346	100	1:24	112	1:22
347	93	1:26	234	1:21
348	64	1:22	99	1:21
349	199	1:22	121	1:23
350	225	1:21	94	1:21
351	503	1:22	128	1:21
352	45	1:22	146	1:22
353	81	1:23	259	1:22
354	292	1:22	185	1:24
355	382	1:20	87	1:21
356	170	1:22	108	1:22
357	408	1:20	453	1:24
358	404	1:22	93	1:20
359	139	1:21	64	1:21
360	68	1:17	112	1:22
361	85	1:23	216	1:28
362	286	1:24	54	1:24
363	94	1:24	98	1:23
364	126	1:22	74	1:20
365	56	1:20	177	1:20
366	74	1:20	71	1:20
367	153	1:24	89	1:20
368	74	1:23	440	1:24
369	308	1:24	90	1:21
370	114	1:23	106	1:22
371	132	1:21	105	1:20
372	70	1:20	146	1:21
373	219	1:24	336	1:23
374	208	1:21	93	1:23
375	85	1:22	71	1:23
376	188	1:22	120	1:21
377	73	1:22	161	1:22
378	280	1:24	179	1:24
379	491	1:20	84	1:20
380	82	1:19	72	1:22

381	107	1:24	67	1:21
382	99	1:19	95	1:22
383	155	1:26	150	1:24
384	56	1:26	282	1:23
385	90	1:21	85	1:22
386	84	1:25	462	1:20
387	122	1:20	144	1:21
388	67	1:24	55	1:20
389	123	1:20	63	1:20
390	265	1:21	257	1:22
391	162	1:21	363	1:21
392	156	1:20	129	1:23
393	97	1:21	73	1:25
394	62	2:83	204	2:82
395	76	1:21	115	1:20
396	98	1:20	145	1:24
397	107	1:18	134	1:22
398	164	1:24	84	1:26
399	116	1:23	47	1:23
400	140	1:23	95	1:20
401	223	1:21	75	1:23
402	124	1:22	66	1:23
403	133	1:20	92	1:23
404	150	1:21	65	1:21
405	56	1:21	129	1:21
406	89	1:18	44	1:20
407	183	1:22	219	1:22
408	160	1:21	91	1:26
409	79	1:22	130	1:22
410	440	1:21	83	1:23
411	153	1:19	111	1:24
412	179	1:24	1025	1:20
413	88	1:22	92	1:21
414	97	1:21	156	1:21
415	78	1:22	472	1:22
416	142	1:23	207	1:22
417	74	1:24	482	1:21
418	167	1:19	150	1:24
419	166	1:21	77	1:22

420	131	1:22	104	1:26
421	197	1:27	71	1:28
422	223	1:22	127	1:20
423	119	1:22	97	1:21
424	186	1:21	140	1:19
425	80	1:22	220	1:24
426	70	1:22	157	1:23
427	139	1:26	118	1:19
428	133	1:20	74	1:26
429	81	1:27	108	1:22
430	305	1:25	325	1:20
431	51	1:23	112	1:21
432	118	1:23	134	1:21
433	134	1:21	83	1:22
434	39	1:18	127	1:26
435	142	1:21	109	1:23
436	111	1:23	204	1:21
437	93	1:23	137	1:20
438	109	1:21	103	1:20
439	287	1:21	1176	1:19
440	94	1:21	104	1:27
441	96	1:26	157	1:23
442	77	1:22	99	1:20
443	45	1:21	60	1:22
444	92	1:17	193	1:25
445	74	1:21	87	1:24
446	472	1:20	136	1:21
447	122	1:20	131	1:23
448	109	1:21	1614	1:23
449	138	1:21	124	1:20
450	88	1:23	61	1:19
451	163	1:22	145	1:23
452	188	1:23	82	1:21
453	175	1:21	119	1:24
454	291	1:25	100	1:22
455	198	1:23	129	1:23
456	351	1:23	151	1:24
457	141	1:23	135	1:27
458	135	1:23	82	1:22

459	314	1:21	119	1:21
460	164	1:22	111	1:22
461	57	1:19	71	1:20
462	101	1:21	145	1:21
463	250	1:22	192	1:23
464	107	1:23	245	1:21
465	212	1:24	846	1:25
466	133	1:23	130	1:22
467	171	1:24	471	1:23
468	81	1:22	98	1:21
469	62	1:24	185	1:25
470	108	1:22	227	1:24
471	103	1:26	138	1:21
472	73	1:20	159	1:22
473	238	1:19	183	1:22
474	148	1:21	100	1:23
475	88	1:21	86	1:21
476	124	1:18	73	1:23
477	132	1:21	103	1:22
478	107	1:20	170	1:28
479	46	1:21	114	1:23
480	121	1:20	292	1:24
481	156	1:21	136	1:22
482	121	1:18	66	1:25
483	273	1:22	140	1:21
484	229	1:21	125	1:24
485	95	1:21	123	1:21
486	237	1:22	132	1:21
487	104	1:24	68	1:24
488	315	1:22	107	1:23
489	108	1:22	101	1:21
490	54	1:24	82	1:22
491	114	1:22	98	1:22
492	202	1:23	64	1:24
493	119	1:24	432	1:20
494	121	1:23	91	1:24
495	189	1:20	90	1:24
496	193	1:20	97	1:21
497	348	1:19	299	1:26

498	92	1:24	93	1:22
499	83	1:21	408	1:22
Average	176	1:21	163	1:21
Maximum	2358	1:29	2716	1:28
Minimum	39	1:17	43	1:17

B

Bibliography

[1] J.T. Schwartz and M. Sharir, "'A survey of motion planning and related geometric algorithms"', Artificial Intelligence 37, 157–169 1988.

[2] N.J. Nilsson, "A mobile automaton: an application of artificial intelligence techniques", Proc. of the 1st Int. Joint Conf. on Artificial Intelligence, pp. 509–520, Washington D.C. 1969

[3] H. Mitchell, "An algorithmic approach to some problems in terrain navigation", Artificial Intelligence 37, 171–201 1988

[4] C. O'Dunlaing and C.K. Yap, "A retraction method for planning the motion of a disc", Journal of Algorithms 6, 104–111 1982

[5] C. O'Dunlaing, M. Sharir and C.K. Yap, "Retraction: a new approach to motion planning", Proc. of the 15th ACM Symp. on the Theory of Computing, pp. 207–220. Boston 1983

[6] E. W. Dijkstra, "A note on two problems in connection with graphs", Numerische Math. 1, 269–271 1959

[7] P.E. Hart, N.J. Nilsson and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths", IEEE Trans.on Systems, Man, and Cybernetics SMC-4, No. 2, 100–107 July, 1968

[8] Andreas C. Nearchou, "'Path planning of a mobile robot using genetic heuristics"', Robotica (1998) Volume 16, pp. 575-588, Cambridge University Press, Patras, 1997

[9] Lavi M. Zamstein "'Koolio: Path planning using reinforcement learning on a real robot platform"' University of Florida 2006

[10] Smart, W. and Kaelbling, L. "'Effective Reinforcement Learning for Mobile Robots"', Proceedings of the 2002 IEEE International Conference on Robotics Automatons, Washington, DC, 2002

[11] Indrani Goswami, Pradipta Kumar Des, Amit Konar, R. Janarthana, "'Extended Q-Learning Algorithm for Path-Planning of a Mobile Robot"', Simulated Evolution and Learning, Lecture Notes in Computer Science Volume 6457, pp 379-383 2010

[12] Christian Schwarz, "'Entwicklung eines Regelungsverfahrens zur Pfadverfolgung für ein Modellfahrzeug mit einachsigem Anhänger"', Diplomarbeit, Koblenz, 2009

[13] J.H. Holland, "'Adaptation in Natural and Artificial Systems"', University of Michigan Press, Ann Arbor, 1975

[14] Thomas Bäck, "'Evolutionary Algorithms in Theory and Practice"', 106-120, Oxford University Press, 1996

[15] J.E. Baker "'Adaptive selection methods for genetic algorithms"', Proceedings of the 1st International Conference on Genetic Algorithms, 101-111 1985

[16] Richard A. Caruana, Larry J. Eshelman, J. David Schaffer, "'Representation and hidden bias II: eliminating defining length bias in genetic search via shuffle crossover"' Proceeding IJCAI'89 Proceedings of the 11th international joint conference on Artificial intelligence - Volume 1, 750-755, 1989

[17] Richard A. Caruana, Larry J. Eshelman, J. David Schaffer, "'Biases in the crossover landscape"', Proceedings of the third international conference on Genetic algorithms, 10-19 1989

[18] Richard A. Caruana, Larry J. Eshelman, J. David Schaffer, Rajarshi Das, "'A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization"', Proceedings of the 3rd International Conference on Genetic Algorithms, 51-60 1989

[19] John .J. Grefenstette, "'A User's Guide to GENESIS"', Navy Center for Applied Research in Artificial Intelligence, Washington, D.C., 1987

[20] Kenneth Alan De Jong, "'An analysis of the behavior of a class of genetic adaptive systems"', Doctoral Dissertation, University of Michigan Ann Arbor, MI, USA, 1975

[21] J. David Schaffer, Amy Morishima, "'An adaptive crossover distribution mechanism for genetic algorithms"', Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application, 36-40, 1987

[22] John .J. Grefenstette, "'Optimization of Control Parameters for Genetic Algorithms"' IEEE Transactions on Systems, Man and Cybernetics, Volume 16, Issue 1, 122-128 1986

[23] D.E. Goldberg, C.L. Bridges "'An analysis of a reordering operator on a GA-hard problem"', Biological Cybernetics, Volume 62, Issue 5, 397-405 1990

[24] D.E. Goldberg, "'Genetic Algorithms in Search, Optimization, and Machine Learning"', Addison-Wesley, Reading, Mass., 1989

[25] M. Gerdts, "'The single track model"', Universität Bayreuth, 2003

[26] C. Altafini, "'Some properties of the general n-trailer"', International Journal of Control 74, 2001

[27] Matthew T. Mason "'Nonholonomic constraint"', Mechanis of Manipulation, 2013

[28] Steven M. LaValle, "'Planning Algorithms"', Cambridge University Press, 2006

[29] C. Pradalier, K. Usher, "'Robust trajectory tracking for a reversing tractor trailer"', In: Journal of Field Robotics 25, Nr. 6, 378-399, 2008

[30] A. Preißer, "'Heuristische Ansätze zur Regelung eines Fahrzeugs mit Anhänger"', Diplomarbeit, Universität Koblenz-Landau, 2001

[31] Wikipedia contributors, "'Rapidly exploring random tree"', Wikipedia, The Free Encyclopedia, 2013

[32] Steve LaValle, Rapidly-exploring Random Trees (RRTs), The RRT Page

[33] Stephan Wirt "'Autonome gründliche Exploration unbekannter Innenräume mit dem mobilen Roboter „Robbie“"', Studienarbeit im Fach Computervisualistik, 2007

[34] The Open Toolkit library, http://www.opentk.com/project/license, 2006-2009

[35] Dieter Zöbel "'Autonomous Driving in Goods Transport"'

[36] Cat Minestar System, http://www.catminestarsystem.com/capability_sets/command/packages/autonomous

[37] J. Hopcroft, J.T. Schwartz, M. Sharir, "'On the complexity of motion planning for multiple independent objects; PSPACE-hardness of the warehouseman's problem"', International Journal of Robotics Research, 3(4):76-88, 1984.

[38] James J. Kuffner, Jr., Steven M. LaValle, "'RRT-Connect: An Ecient Approach to Single-Query Path Planning"', In Proc. 2000 IEEE Int'l Conf. on Robotics and Automation (ICRA 2000)