Path planning for off-road vehicles with a simulator-in-the-loop

Thomas Hellström, thomash@cs.umu.se
Ola Ringdahl, ringdahl@cs.umu.se

UMINF 08.07
ISSN-0348-0542
Department of Comuting Science
Umeå University
SE-901 87 Umeå, Sweden

May 6, 2008

Abstract

This paper describes the development of a real-time path planner for off-road vehicles using a simulator. The work was triggered by a need for an obstacle-avoidance and path-planning system in our work with autonomous forest machines. The general idea with the presented system is to extend a standard path-tracking algorithm with a simulator that, in real-time, tries to predict collisions in a window forward in time. This simulation is based on current sensor data giving information about the environment around the vehicle. If a collision is predicted, the vehicle is stopped and a path-search phase is initiated. Variants of the original path are generated and simulated until a feasible path is found. The real vehicle then continues, now tracking the replanned path. In simulated tests, this way of using a simulator to predict and avoid collisions works well. The system is able to safely navigate around obstacles on and close to the path in a way that is hard or impossible to achieve with standard obstacle-avoidance algorithms that do not take the shape of the vehicle into account. Another scenario, also envisioned in forest environment, is off-line path planning of a longer route, based on map information. An approximate path given by a straight line from start to goal is then modified in the same way as described above.

1 Introduction

Many path-planning algorithms make simplifying assumptions regarding the robot's geometry or kinematic behavior. Typically the robot is approximated by either a point or a circle, and is assumed to be able to move in a predictable

way given by simple kinematics equations. This usually works fine for regular indoor robots but is less suitable for large outdoor robots and autonomous vehicles. In our research on autonomous forest machines [8, 9, 14, 10], the principal differences between avoiding obstacles while following a path with a small indoor robot and a 10-meter-long vehicle with articulated steering have become evident. First, the former can indeed often be treated as circular, while the latter has a much more complicated shape. The articulated joint between the two vehicle parts serves as steering mechanism but also causes the physical shape of the vehicle to change depending on the steering angle ϕ. As a consequence of this, the front and rear part of the vehicle move considerably to the sides when changing steering angle, even when standing still. For our specific vehicle Valmet 830, ϕ may change from -43 to $+43$ degrees. If not taken into account, this may cause the vehicle to hit obstacles on the side when trying to avoid obstacles in front. Second, the kinematics for most indoor robots is well known and mostly accurate for flat surfaces. For an off-road vehicle, ground slip may be significant and cause both random and partly predictable changes in motion as described further on.

For these reasons, a system, SIMLOPP, has been developed and is described in this paper. The general idea with SIMLOPP (SIMulator-in-the-LOop for Path Planning) is to extend a standard path-tracking algorithm with a simulator that, in real-time, tries to predict collisions in a window forward in time. This simulation is based on current sensor data giving information about the environment around the vehicle. If a collision is predicted, the vehicle is stopped and a path-search phase is initiated. Variants of the original path are generated and simulated until a feasible path is found. The real vehicle may then continue, now tracking the replanned path.

The paper is organized as follows: Section 2 contains a brief overview of related work. In Section 3, the developed system is described in detail. Results from tests in a simulator environment are presented in Section 4, and in Section 5 conclusions and future work are discussed. Appendix A describes the kinematic equations used in the simulator to model a vehicle with articulated steering.

2 Related work

This work was initiated by a need for an obstacle-avoidance and path-planning system in our work with autonomous forest machines. In our previous work [14] [8] we implemented the well known algorithm VFH+ [16] which works well for regular indoor robots. For a forest machine it performs reasonably well at avoiding obstacles in open areas, but has difficulties with narrow passages between obstacles close to both sides of the vehicle. This is due to the fact that the algorithm does not distinguish between obstacles in front and on the sides of the vehicle. Furthermore, all obstacles are enlarged with the length of the front part of the vehicle (which is larger than the width). Due to this, an obstacle close to the side of the vehicle may be regarded as being partly in the way of the vehicle. Ulrich and Borenstein, the authors of VFH+, have developed

an enhanced obstacle-avoidance algorithm with look-ahead verification, called VHF* [17]. VFH+ calculates a number of suitable directions to travel, assigns a cost value to each of them, and selects the direction with the lowest cost as the new direction of travel. In contrast, VFH* analyzes the further consequences of going in each of these directions before making a final decision. This is done by computing where the vehicle would be in the next time step if it should go in each target direction. VFH+ is then applied for the vehicle's new poses and the cost of going in each new direction is calculated. By repeating this process, a search tree is created, and the A* search algorithm is applied to select the best direction to move. This helps to avoid local trap situations that the original VFH+ algorithm would not detect. Still, the improved algorithm has the same strategy for prediction of collisions. It treats the vehicle as a point and expands the obstacles to the size of the vehicle. Furthermore, the algorithm does not take into account that the rear part of a large vehicle may hit an obstacle even after the front part has passed it when the steering angle is changed.

Thrun et.al. [15] use an online path planner that calculates possible trajectories to avoid obstacles. This is done by adding a lateral offset to a reference path, and also a rate at which the vehicle will attempt to adjust to this new trajectory. The planner is implemented as a search algorithm that minimizes a linear combination of continuous cost functions. The vehicle model includes several kinematic and dynamic constraints, such as maximum steering angle, maximum steering rate and maximum deceleration. The cost functions penalizes running over obstacles and leaving the specified corridor the vehicle is allowed to be within.

Noguchi and Terao [13] have developed a simulator for an agricultural robot using a neural network. This simulator applies a genetic algorithm to plan an optimal path given the required initial and final states of the vehicle. To check if the vehicle collides with an obstacle the distance between the vehicle position and an enlarged obstacle is computed. The genetic algorithm generates steering angles and changes in steering angles. These are then sent to the neural network which simulates each generated path.

Capozzi [4] uses evolution-based path planners to find optimal and collision-free paths through environments containing both static and dynamic obstacles and target locations. In this work, a genetic algorithm is used with a simulator to guide air vehicles to their destination without colliding with obstacles. To detect collision with obstacles, *minimally enclosing rectangles* are used. Bounding rectangles are first fitted around the vehicle and each obstacle. The overlap between the vehicle rectangle and each obstacle rectangle is then computed. Capozzi concludes that the computation time mainly is spent at collision detection, and that faster methods should be found. The author also suggests that the method to generate a new path by moving the original one a distance to the side can be generalized to a sequence of maneuvers. Such maneuvers could be, for example, *turn left slowly* and *turn right quickly*.

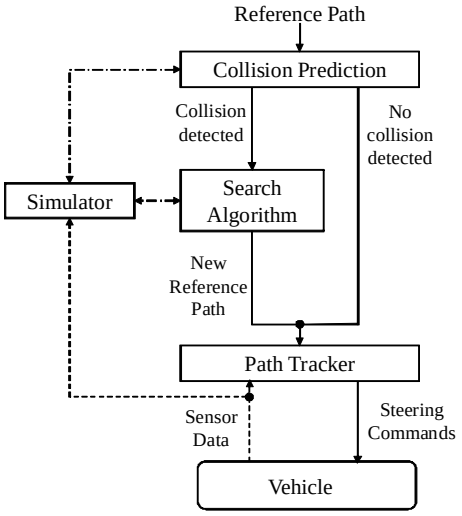

Figure 1: System description of the simulator-driven path planner. The collision prediction module uses a simulator to track the path about 5 meters ahead of the current position while computing the nearest distance to any obstacles. If no collision is detected, the path tracker guides the real vehicle along this path. If a collision is detected, the system tries to find a new path. This is solved as a search problem in which candidate paths are evaluated by the simulator to predict the outcome of path tracking along the candidate path. When a new feasible path is found, it replaces the reference path and is used by the path tracker to guide the real vehicle around the obstacles.

3 System Description

In this section, the simulator-driven path-planning system is described in detail. Pseudo-code for the top-level routine is given in Algorithm 1. The main loop performs regular path tracking (Line 10) with the wanted path p. Every meter, the path is evaluated in a simulated path-tracking 5 meters forward to predict the risk for collision (Line 3). To track the path, the simulator uses the path-tracking algorithm Follow the Past [9]. Other path-tracking algorithms may be used as well, provided they are based on the perpendicular distance to the path, and do not aim at the path further ahead (i.e., the Pure Pursuit algorithm [6] does not work well in this context). The simulation is performed by the *SimCheck* function described in Algorithm 3. If no collision is predicted, path tracking along p continues until the goal is reached. If a collision is predicted, a new path is constructed (Line 5) by modifying p. This is done by the *PathSearch* function described in Algorithm 2. If a feasible path can be constructed, it replaces the reference path and is used by the path tracker to guide the real vehicle around the obstacles. A graphical overview of the system is given in Figure 1 and the main components are described in detail below.

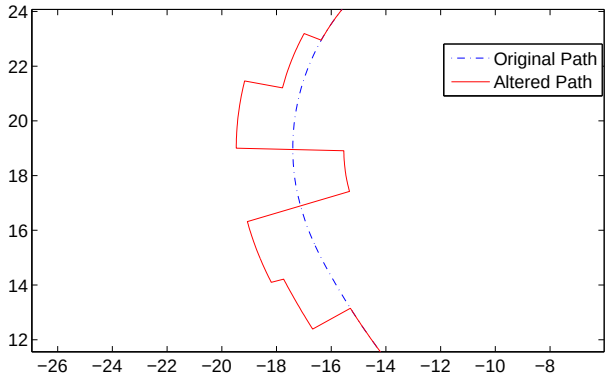

Figure 2: 10 meters of the original path in front of the vehicle are divided into 5 segments. By offsetting each segment perpendicular to the original path the search function tries to find a combination of offsets that allow the vehicle to avoid the obstacles.

3.1 Path searching

The *PathSearch* function in Algorithm 2 attempts to find a candidate path, similar to the original path that a simulated vehicle can track for 10 meters without hitting any obstacle or straying too far away from the original path. This path evaluation is done by simulated path tracking along the candidate path and is performed by the *SimCheck* function (see Algorithm 3) that returns a fitness value for the candidate path. The fitness concept is further described in Section 3.2.

The search problem formulated in Algorithm 2 aims at finding a vector x_{opt} such that the candidate path generated by the function $p(x_{opt})$ has a high enough fitness value.

The $p(x)$ function works as follows. 10 meters of the original path in front of the vehicle is extracted and equally divided into five segments. The elements of the x vector represent lateral offsets for the segments, such that they are shifted perpendicular to the original path, thus creating a modified candidate path, as illustrated in Figure 2. The search algorithm tries to find an acceptable combination of these five offsets. Initially, a sequence of offsets to the steering commands from the path-tracking algorithm were used instead of modifying the reference path, as suggested by Capozzi [4]. This is hard to combine with path tracking that tries to keep the vehicle on the path. An offset steering command causes the vehicle to deviate from the path. The path tracker then commands the vehicle to turn back towards the path to compensate, thereby counteracting the offset.

Figure 3 illustrates how an offset segment makes it possible for the vehicle to avoid obstacles. In this paper we have implemented and evaluated three different search algorithms that are described in more detail in Section 3.3.

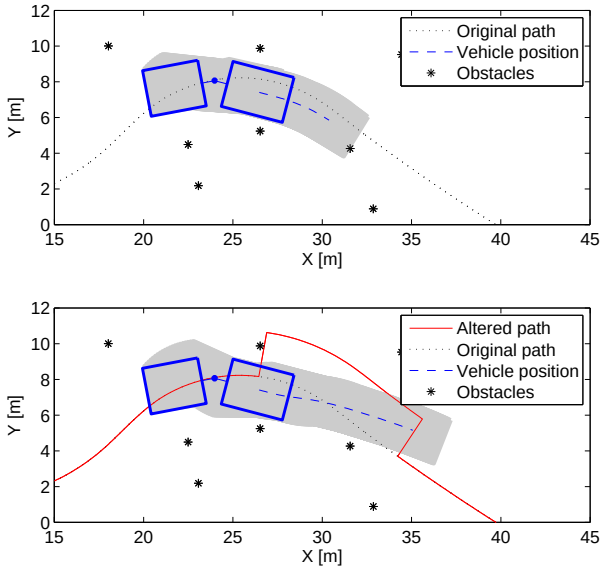

Figure 3: If the simulation predicts a collision with an obstacle, as in the upper figure, the path is replanned such that the obstacles are avoided, as shown in the lower figure. The figures show the original position of the vehicle and the area it would cover while tracking the path (the gray area). In the lower figure, the altered replanned path can be seen as well.

3.2 Evaluating paths

As described above, paths are evaluated at two places in the system: In the main loop to decide if the vehicle can go on tracking the path, and in the *Path-Search* function, used to generate alternative paths. In both cases, the function *SimCheck* performs the evaluation as described in Algorithm 3. *SimCheck* simulates path tracking for a fixed distance D ahead on the path. For each meter, performance is computed such that the performance for the path as a whole can be returned as the minimum of all momentary performance values. The kinematics of a forest machine with an articulated joint is more complex than for many other vehicles and robots. This is, as described above, part of the motivation for the development of the simulator-based path planner, and is also a necessary component of the real-time simulator. Appendix A describes the kinematics for the simulated articulated forest machine in more detail.

A momentary performance is computed by the *fitness function* comprising two parts. One part is related to the distance between the vehicle and the obstacles, and the other is related to the distance between the vehicle and the original reference path p_0. In many off-road environments it is not safe to stray too far away from the reference path because of obstacles that can not be detected by the vehicle's sensors, e.g mire ground or deep trenches. Therefore, in addition to avoiding obstacles, the vehicle must also stay reasonably close to the reference path. In our particular implementation, the fitness takes values between 0 and 10, where 10 is best, and 0 means either that the vehicle almost collides with an obstacle (distance smaller than 0.1 meter) or strays too far from the path (distance larger than 2.5 meters). The fitness function is illustrated graphically in Figure 4. In the experiments reported in this paper the minimum accepted fitness value is 5, which is the value of the constant *minfitness* referred to at many places in the listed algorithms. Fitness 5 corresponds to a minimum distance of 0.5 meters between any part of the vehicle and any obstacle, and a maximum distance of 2.5 meters between the vehicle and the path.

It is important to distinguish between the momentary fitness for a single point, computed by the fitness function, and the fitness for a whole path, which is computed as the minimum of the fitness values for all vehicle positions when simulating path-tracking along the path. This path fitness is computed by the function *SimCheck* in Algorithm 3.

3.2.1 Distance between vehicle and obstacles

At each simulated time step the shortest distance between the vehicle and all obstacles is calculated (Algorithm 3, line 3). This is a key operation in the system and has to be both fast and accurate. The sensor data provides an updated view of the environment around the vehicle. This is used by the simulator together with a geometrical model of the vehicle to detect collisions with obstacles. The vehicle is modeled by eight lines, four for the front part and four for the rear part, connecting the vehicle's corners. The distance between the vehicle and an obstacle is defined as the shortest distance between the obstacle, defined as a

Algorithm 1 Simulator-driven path planner

Given an initial reference path p_0

1: $p = p_0$
2: **while** not at goal **do**
3: $pathfitness = SimCheck(p, 5, p_0)$ {simulate 5m along p (Alg. 3)}
4: **if** *pathfitness* < *minfitness* **then** {path not feasible}
5: $p = PathSearch(p_0)$ {try to generate a modified path (Alg. 2)}
6: **if** p = NULL **then**
7: EXIT
8: **end if**
9: **end if**
10: use p for 1 meter of path tracking with the target vehicle
11: **end while**

minfitness is the minimum accepted fitness value, set to 5 in our experiments

Algorithm 2 The function *PathSearch* finds and returns a 10-meter-long modified feasible path

$function\ PathSearch(p_0)$
Given a path p_0, search for a vector x_{opt} such that

$$SimCheck(p(x_{opt}), 10, p_0) \geq minfitness$$
$$and$$
$$-maxdev < x_{opt}(i) < maxdev, \forall i$$

where

$p(x_{opt})$: A new path generated by offsetting the segments of p_0 perpendicular to the left or right by amounts given by the vector x_{opt}

maxdev: The maximal allowed distance between the original path and the vehicle, set to 2.5 meters in our experiments.

if search successful **then**
return $p(x_{opt})$
else
return NULL
end if

Algorithm 3 The function *SimCheck* simulates path tracking D meters along p_1, and computes the path fitness as the smallest of all fitness values along the path:

$function\ SimCheck(p_1, D, p_0)$

1: *d=0; collision=FALSE; pathfitness=inf*
2: **while** $d < d$ and *pathfitness* > 0 **do**
3: d_1 = shortest distance between the vehicle and the obstacles within a 5.3 meter large radius around the vehicle's articulation link
4: d_2 = shortest distance between the vehicle and the original reference path p_0
5: *pathfitness = min(fitness(*d_1, d_2*)pathfitness)*
6: simulate 1 meter path tracking along p_1
7: $d = d +$ simulated covered distance
8: **end while**
9: return *pathfitness*

fitness is a function that computes how desirable it is to have the vehicle d_1 meters away from the nearest obstacle and d_2 meters away from the original path (refer to Figure 4).

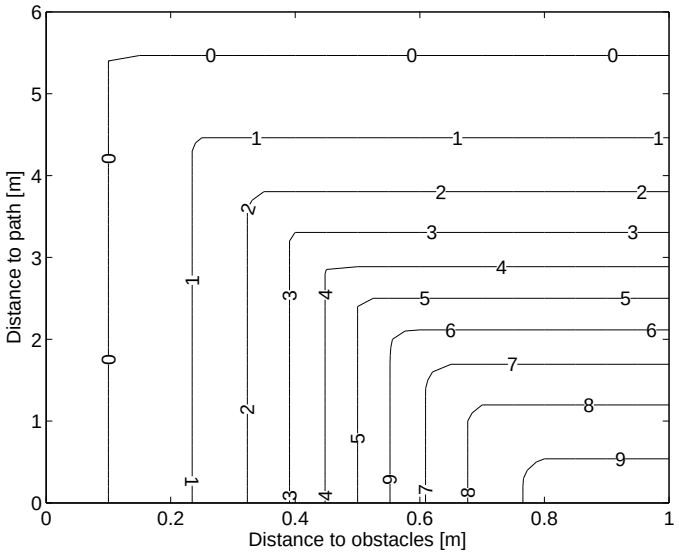

Figure 4: Level curve for the fitness function used for path search. The fitness value is a function of the vehicle's shortest distance to any obstacle and its distance to the original reference path. If the fitness value is 5 or above, the path is regarded as traversable at this particular point. An entire path is regarded traversable if all points the vehicle passes through in path-tracking are traversable.

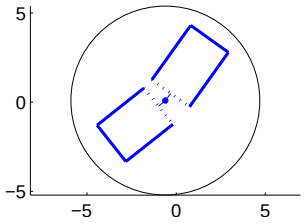

Figure 5: To speed up the computation of the smallest distance between obstacles and the vehicle, only the continuous lines in the figure are used. The dotted lines can never be closest to an obstacle due to the the vehicle's geometry, unless a collision was detected in a previous time step. Furthermore, only obstacles within the bounding circle drawn around the vehicle are considered in the computation.

point, and each of these lines. To speed up the computation, two things are done: first, only obstacles within a radius of 5.3 meters around the vehicle's articulation joint are considered. The radius is calculated as the the length of the front part of the vehicle plus a safety margin of one meter; second, line-obstacle distances are only computed for six of the eight lines, as illustrated in Figure 5. This is based on the assumption that the two lines closest to the articulation link can never be closest to an obstacle, unless a collision was detected in a previous time step, due to the the vehicle's geometry.

3.3 Search algorithms

Three different algorithms have been evaluated for the informed search task described in Section 3.1; DIRECT [11] which is a numerical optimization algorithm using no derivatives, a genetic algorithm (GA) [5] and, as a benchmark, a random search algorithm. Each algorithm generates a *population* P consisting of N *individuals* $I_i, 1 \leq i \leq N$, each one representing a candidate path identified by the offset values $x_i(1), ..., x_i(5)$. The principal difference between the algorithms is the way in which new individuals are generated/selected based on the previous population (i.e. the p function in Algorithm 2). Random search generates random individuals. This method is used as a benchmark for the other two algorithms.

The genetic algorithm is implemented using the Genetic Algorithm Toolbox [5] for Matlab. The basic steps in the genetic algorithm are described in Algorithm 4. It is inspired by nature's way of improving a population over several generations by combining genes from successful parents. To increase the chances of finding a global maximum, the mutation operation is introduced. It selects one or more chromosomes $x_i(j)$ in an individual i and randomly changes them. This random process is often seen as a (theoretical) assurance that the probability of finding any given individual is never zero [7]. To avoid testing

the same solution more than once in each generation, an individual is not considered if an equal one has already been generated in the same generation. Two individuals I_1 and I_2 are considered equal if $\max_j |x_1(j) - x_2(j)| \leq 0.01$, i.e. if all corresponding chromosomes in the two individuals differ by less than 1 centimeter.

We use a Matlab implementation gblSolve [3] of the global optimization algorithm DIRECT [11]. The first step in the DIRECT algorithm is to transform the search space to a unit hypercube. The fitness function is then sampled at the center-point of this cube. The algorithm identifies a set of potentially optimal rectangles in each iteration. All potentially optimal rectangles are further divided into smaller rectangles, whose center-points are sampled. Conventional optimization algorithms like DIRECT aim at minimizing an object function, as opposed to genetic algorithms, which aim at maximization. The search condition $SimCheck(p(x_{opt}), 10, p_0) \geq minfitness$ in Algorithm 2 is therefore changed to -$SimCheck(p(x_{opt}), 10, p_0) \leq -minfitness$ when using the DIRECT algorithm. To speed up execution, DIRECT has been modified to return immediately if an object function value $< -minfitness$ is found.

4 Results

To evaluate the developed system and the three different search algorithms an additional vehicle simulator is used. The simulator tracks each of the 5 paths in Figure 6 100 times with the 3 algorithms[1]. To test the algorithms' ability to plan collision-free paths in difficult situations, the obstacles are placed such that the original paths cause collision and need careful modifications to be feasible. However, in most real situations, the vehicle has much more space to maneuver.

As described in Section 3, the path-tracking simulator is called every meter and the search module is called if a collision is predicted (or rather if the predicted path fitness is too low). Depending on the path and the obstacles in the environment each search either finds a solution or fails within the allowed maximum number of iterations. Figure 7 shows the result of replanning a path with DIRECT. Note that the vehicle drives along a smooth trajectory, even if the replanned path is not smooth. Tables 1 to 5 summarize the results for the three search algorithms for the five paths. GA denotes the genetic algorithm, DIR denotes the DIRECT algorithm, and Rand denotes the random search. The results presented in the tables are:

[1]DIRECT is a deterministic algorithm, i.e the result for each of the 100 runs will be exactly the same. However, for a uniformed presentation of results, 100 runs are presented also for the DIRECT algorithm.

Algorithm 4 Genetic algorithm for path search

1: $t = 0$ {generation}
2: Create an initial *population* P_0, consisting of N *individuals* $I_1, \ldots, I_N$, each one consisting of 5 *chromosomes* $x(1), \ldots, x(5)$
3: **repeat** {max 3 times}
4: Compute path fitness pf for each individual (performed by the function *SimCheck* in Algorithm 3)
5: Replace all individuals with $pf = 0$ with new randomly generated ones
6: **until** $nFit > 6$ or $maxFit \geq 5$ {obtain a large enough population for breeding}
7: **while** $maxFit < minfitness$ and $t < 25$ **do**
8: $t = t + 1$
9: Use roulette wheel selection [2] to select the best 90% individuals from the last population P_{t-1}. These individuals are denoted *parents.*
10: Generate new individuals, denoted *children*, consisting of a combination of chromosomes from two parents each. In our case, this is done by intermediate recombination [12].
11: Mutate the genes in each individual with probability 0.2. This means that in average, one chromosome per individual will be mutated, i.e get a random value within the allowed range.
12: Evaluate the new individuals according to the fitness function
13: Let P_t consist of the N most successful individuals from the parents and children.
14: x_{opt} = best individual
15: **end while**

where

$$maxFit = max(pf(I_1), ..., pf(I_N))$$
$$nFit = |\{I_i | pf(I_i) > 0\}|.$$

Algorithm	GA	DIR	Rand
Successful searches [%]	96	100	44
Average search time [s]	12	17	24
Average no. of fitness evals	109	152	625
Average time to halt [s]	90	-	78
No. of search operations	581	900	170

Table 1: Results for path 1

Successful searches:	Number of solved search problems
Average search time:	Computed for all successful search problems
Average no. of fitness evals:	Computed for all successful search problems
Average time to halt:	Computed for all failed search problems (maximum allowed iterations was reached):
No. of search operations:	The number of times the algorithm was called (total for 100 runs).

In general, the genetic and DIRECT algorithms have roughly the same performance while the random search usually does not solve as many situations and also takes longer. This is to be expected, as random search just tries random solutions, while the other two algorithms try to solve the search problem in a systematic way. In almost all test cases, the DIRECT and genetic algorithms manage to safely navigate the vehicle along the whole path, but the performance for path 2 is not as good as for the other paths. This path contains particularly difficult situations, making it hard to find a traversable path. The reasons for this may be that a path correction in the beginning of the path leads the vehicle into an impossible situation further on. This is confirmed by the good results for the random algorithm for the same path, and also by the relatively higher performance for the genetic algorithm which contains a larger element of randomness than does the DIRECT algorithm. The problem is related to the limited 10 meter prediction window. Traversable paths may be possible to generate if the size of this window is increased, of course at the price of longer simulation times.

The time it takes to solve a search problem depends on the necessary number of calls to the fitness function. It also depends on how far along the path the simulator has to run for each candidate path to find out if the vehicle will collide with an obstacle or stray too far from the reference path, i.e if the fitness becomes zero. In our tests, the average time to find a solution varies between 2 and 24 seconds. Another important performance measure is the time it takes before giving up on a problem if no solution is found. This time depends mainly on the number of individuals evaluated, but as explained above also on how far the simulator has to run for each individual. In our tests, this time varies between 33 and 98 seconds.

Figure 8 shows the improvement of fitness over several generations in the genetic algorithm for ten different problems. The final individuals in this example all have acceptable fitness values (at least 5), but depending on the difficulty of

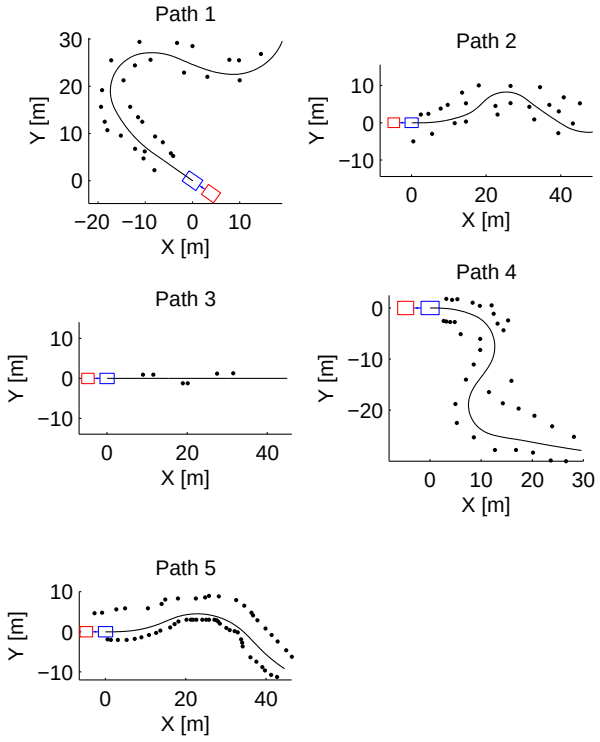

Figure 6: The 5 paths used to evaluate the algorithms. The vehicles are shown in their starting positions.

Algorithm	**GA**	**DIR**	**Rand**
Successful searches [%]	80	67	81
Average search time [s]	4	2	3
Average no. of fitness evals	186	111	194
Average time to halt [s]	98	33	64
No. of search operations	500	300	406

Table 2: Results for path 2

Algorithm	**GA**	**DIR**	**Rand**
Successful searches [%]	100	100	99
Average search time [s]	4	2	2
Average no. of fitness evals	52	19	24
Average time to halt [s]	-	-	66
No. of search operations	354	500	370

Table 3: Results for path 3

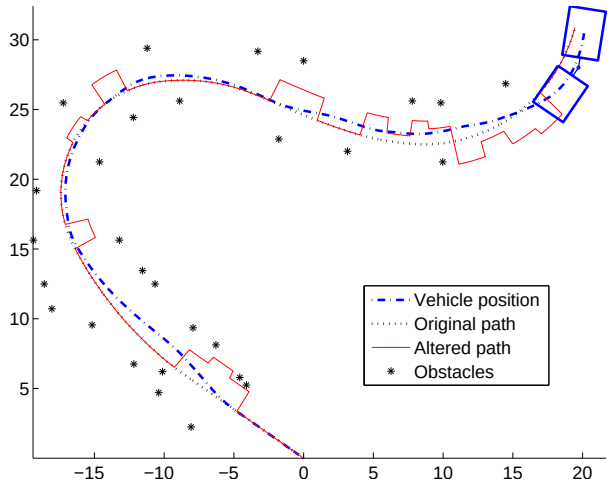

Figure 7: The result of replanning a path. The figure shows the original path as well as the replanned one, and the vehicle's position as it tracks the new path. The vehicle is shown in its final position.

Algorithm	**GA**	**DIR**	**Rand**
Successful searches [%]	99	100	87
Average search time [s]	6	8	9
Average no. of fitness evals	45	47	184
Average time to halt [s]	64	-	85
No. of search operations	421	300	362

Table 4: Results for path 4

Algorithm	**GA**	**DIR**	**Rand**
Successful searches [%]	100	100	98
Average search time [s]	4	4	8
Average no. of fitness evals	25	29	81
Average time to halt [s]	-	-	95
No. of search operations	400	400	416

Table 5: Results for path 5

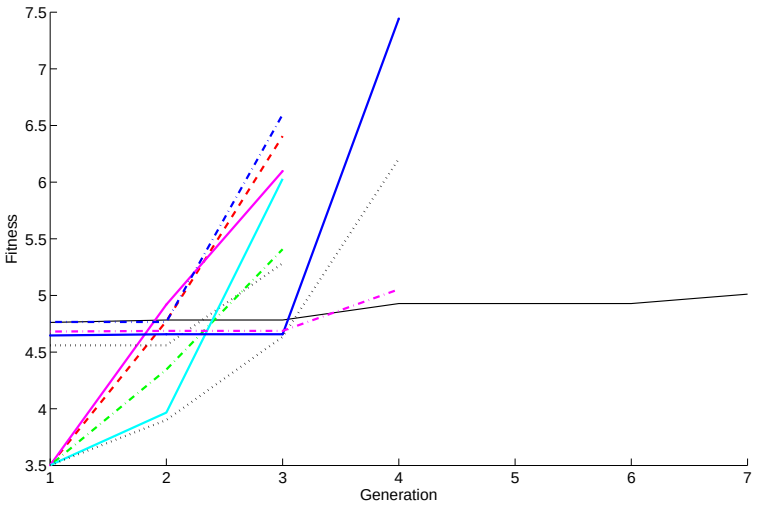

Figure 8: Maximum path fitness in a population as a function of generation, for ten different problems solved by the genetic algorithm. When the fitness value becomes at least 5, the path is considered traversable and search is terminated.

the situation, it takes between 3 and 7 generations to get there.

In addition to avoiding obstacles on a predefined path, the developed SIMLOPP system can be used to plan a totally new path, where only start and goal points are initially given together with an accurate map that includes all obstacles in the environment. The planning is done off-line in advance, with an additional simulator replacing the real vehicle. The simulated vehicle moves along a path generated by repeated calls to the path-search mechanism that generates feasible 10-meter-long path segments when necessary. In this scenario the vehicle should be given more freedom to deviate from the original path, since it only reflects the general preferred direction of motion and not a reasonable path alternative. In our tests, the vehicle was allowed to move up to 10 meters away from the line between start and goal. Figure 9 shows the results, where a 200-meter straight path between start and goal points is specified, and the environment contains a random set of obstacles. The system manages to generate a collision-free path from start to goal. Of course, this way of performing off-line path planning can also be combined with the on-line use of the system. In this way, obstacles that were not present on the map can be taken into account as soon as they are seen by the vehicle's sensors.

5 Conclusions and Future Work

The proposed SIMLOPP system can be used in various scenarios for autonomous vehicles and robots. Referring to the forest machine application for which the

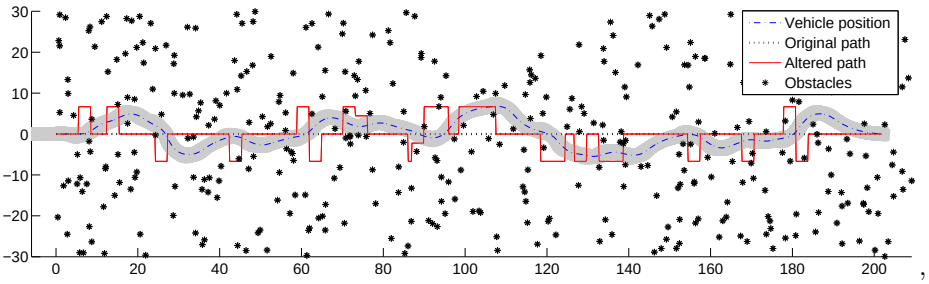

Figure 9: A totally new path can also be planned, given only the start and goal points. In this example, a random set of obstacles is generated. The original path is a straight line between the start and goal points. The vehicle is allowed to drive at most 10 meter away from this path and no closer than 0.5 meter to any obstacle.

system is developed, the most obvious use is for improved obstacle avoidance, where a reference path is first demonstrated by a human driver [9]. To autonomously track this learnt path, the vehicle would simulate 5 meters ahead, and stop if a collision is predicted. Alternative modifications of the reference path will then be simulated until a feasible modified track has been generated. If no such path can be found, the vehicle would have to take other actions such as calling for human assistance. Of course the entire operation would take place in real-time and obstacles would be provided by the vehicle's sensors. This scenario is illustrated in Figure 7. Our tests show that using a simulator to predict and avoid collisions works well. The system is able to safely navigate a large forest vehicle around obstacles on and close to the path. The system is superior to VFH+ at handling hard situations with narrow passages, and it also takes the whole vehicle into consideration, not only the front part. Another scenario, also envisioned in a forest environment, is off-line path planning of a longer route, based on map information. An approximate path given by a straight line from start to goal is then modified in the same way as described above. An example of this scenario can be seen in Figure 9. Of corse, the two scenarios may also be combined such that a rough plan is first generated off-line. While tracking this path, replanning may occur if new obstacles appear in the visible neighborhood of the vehicle.

The presented work uses a simulator instead of the physical vehicle for the evaluation of the developed system. The next step is to move the system to a real forest machine to validate the results. The simulator used for path evaluation in the path-search routine can be replaced by a physics-based simulator. We have initiated development of such a simulator for forest machines, based on the Colosseum3D framework [1]. This 3D-simulator includes realistic vehicle kinematics and dynamics and can improve the quality of the path simulations, for example by simulating ground slip. However, the current 2D-simulator runs over 100 times faster than real time, which may be hard to match with a more

advanced physics-based simulator.

The efficiency of the implemented algorithms and the simulator could probably be increased if the implementation was moved from Matlab to a program language more suited for real-time applications, e.g Java or C/C++. This also means that the average search times reported in this paper would decrease or allow the algorithms to evaluate additional possible solutions, or predict longer distances ahead, in the same amount of time. This would also mean that the vehicles most often do not really have to stop to do perform path planning. A new feasible path may be generated in a fraction of a second such that the vehicle can move on without interruption.

Appendix A

Kinematic equations

A simplified mechanical layout is illustrated in Figure 10. The center of each wheel axle rotates along a circle with radius r at an angular velocity $\omega = v/r$, where v is the speed of the vehicle and r depends on the geometry of the vehicle including the steering angle ϕ. The front and rear parts of the vehicle rotate along circles with different radii. In the following, only the front part is considered. For the front axis, the radius r is given by

$$r = \frac{a + \frac{b}{\cos\phi}}{\tan\phi}. \tag{1}$$

To avoid modeling slip, r is calculated for a virtual axle located in between the real axles of the front part. Given a vehicle position (x, y) and heading θ, measured at the middle of the front axle at time t, the position (x', y') and heading θ' at time $t + \Delta$t is given by

$$\begin{aligned} x' &= \cos(\omega\Delta t) r \sin(\theta) - \sin(\omega\Delta t) r \cos(\theta) - r\sin(\theta) + x \\ y' &= \sin(\omega\Delta t) r \sin(\theta) - \cos(\omega\Delta t) r \cos(\theta) + r\cos(\theta) + y \\ \theta' &= \omega\Delta t + \theta + c\Delta\phi \end{aligned} \tag{2}$$

where the constant $c \in [0, 1]$ describes how the change in steering angle ϕ affects the vehicle's heading θ. c is largely a function of slip, which in turn depends on the weight distribution for the two vehicle parts: the front part moves more if the rear part is heavily loaded. Other important factors influencing slip are tires, surface conditions and minor obstacles which may influence the turning motion. In this work $c = 0.2$ was used, corresponding to a relatively heavy front part that moves less than the rear part of the vehicle.

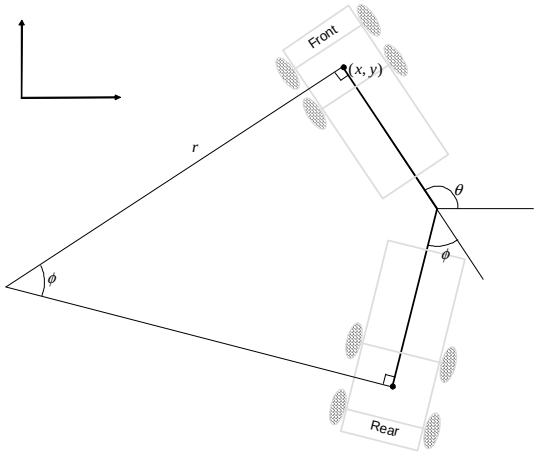

Figure 10: Definition of steering angle ϕ, position (x, y), heading θ, and turning radius r for an articulated forest machine. The global coordinate system for the pose (x, y, θ) can be seen in the upper left part of the figure.

References

[1] Anders Backman. Colosseum3d - authoring framework for virtual environments. In *Proceedings of 11th EGVE Workshop*, 2005.

[2] J. E Baker. Reducing bias and inefficiency in the selection algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms and their Application [ICGA2]*, pages 14–21, Hillsdale, New Jersey, USA, 1987. Lawrence Erlbaum Associates.

[3] Mattias Björkman and Kenneth Holmström. Global optimization using the direct algorithm in matlab. *Advanced Modeling and Optimization*, 1(2):17–37, 1999.

[4] Brian J. Capozzi. *Evolution-Based Path Planning and Management for Autonomous Vehicles*. PhD thesis, University of Washington, 2001.

[5] A. J. Chipperfield and P. J. Fleming. The matlab genetic algorithm toolbox. *IEE Colloqium on Applied Control Techniques Using MATLAB, Digest*, (1995/014), 1995.

[6] R. Craig Coulter. Implementation of the pure pursuit path tracking algorithm. Technical Report CMU-RI-TR-92-01, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, January 1992.

[7] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley Publishing Company, January 1989.

[8] Thomas Hellström, Thomas Johansson, and Ola Ringdahl. *Development of an Autonomous Forest Machine for Path Tracking*, volume 25 of *Springer Tracts in Advanced Robotics*, pages 603 – 614. Springer, field and service robotics: results of the 5th international conference edition, 2006.

[9] Thomas Hellström and Ola Ringdahl. Follow the past - a path tracking algorithm for autonomous vehicles. *Int. J. Vehicle Autonomous Systems*, 4(2-4):216–224, 2006.

[10] Thomas Johansson and Thomas Hellström. A software infrastructure for sensors, actuators, and communication. In *In Proceedings of The Third Swedish Workshop on Autonomous Robotics (SWAR05)*, 2005.

[11] D. R. Jones, C. D. Perttunen, and B. E. Stuckman. Lipschitzian optimization without the lipschitz constant. *Journal of Optimization Theory and Applications*, 79(1):157–181, October 1993.

[12] H. Mühlenbein and D. Schlierkamp-Voosen. Predictive models for the breeder genetic algorithm. *Predictive Models for the Breeder Genetic Algorithm*, 1(1):25–49, 1993.

[13] Noboru Noguchi and Hideo Terao. Path planning of an agricultural mobile robot by neural network and genetic algorithm. *Computers and Electronics in Agriculture*, 18:187–204, 1997.

[14] Ola Ringdahl. *Techniques and Algorithms for Autonomous Vehicles in Forest Environment*. Licentiate thesis, Department of Computing Science, Umeå University, 2007.

[15] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney. Stanley, the robot that won the darpa grand challenge. *Journal of Field Robotic*, 23(9):661–692, 2006.

[16] I. Ulrich and J. Borenstein. VFH+: Reliable obstacle avoidance for fast mobile robots. *IEEE Int. Conf. on Robotics and Automation*, pages 1572–1577, May 1998.

[17] Iwan Ulrich and Johann Borenstein. VFH*: Local obstacle avoidance with look-ahead verification. In *IEEE International Conference on Robotics and Automation*, pages 2505–2511, April 2000.