Application of Hybrid A* to an Autonomous Mobile Robot for Path Planning in Unstructured Outdoor Environments

Janko Petereit, Thomas Emter, Christian W. Frey
Fraunhofer Institute of Optronics, System Technologies and Image Exploitation IOSB, Karlsruhe, Germany

Thomas Kopfstedt, Andreas Beutel
Diehl BGT Defence GmbH & Co. KG, Überlingen, Germany

Abstract

Efficient path planning is one of the main prerequisites for robust navigation of autonomous robots. Especially driving in complex environments containing both streets and unstructured regions is a challenging problem. In this paper we present the application of the Hybrid A* algorithm to a nonholonomic mobile outdoor robot in order to plan near optimal paths in mostly unknown and potentially intricate environments. The implemented algorithm is capable of generating paths with a rate of at least 10 Hz to guarantee real-time behavior.

1 Introduction

Mobile robots can support humans in various ways. They are able to act in hazardous or inaccessible environments and can relieve humans by taking over monotonous tasks. Current applications of mobile robots range from service robots in the industrial environment through reconnaissance robots in disaster areas to robot systems for inspection and exploration tasks in the deep sea.

For executing complex tasks in an autonomous way, a mobile robot requires three core capabilities: It has to use its sensors to locate itself in the world and build a map of its environment and it needs to plan a path for its future motion according to its particular goal. Depending on the mobile platform kinematics and the intended use of the robot, this path planning has to take into account many constraints which causes path planning to be a rather complex and computationally expensive optimization problem. It is especially challenging in unstructured outdoor environments where no previously built map is available. In this case the robot possesses only local knowledge of its environment because of limited sensor coverage. Due to this partial observability, the planned path will generally be globally suboptimal. Furthermore, the planner may get stuck in a local optimum in the presence of extended obstacles.

Another important constraint for planning feasible paths is given by the vehicle kinematics. In this paper we consider a regular nonholonomic car-like robot. Consequently, the steering angles are limited and thus turning on the spot is not possible. This limitation has to be explicitly taken into account during the path planning to guarantee that the underlying path following controller can indeed generate valid trajectories for the robot (cf. [2]).

In order to be employed in real-world applications, the path has to be computed several times per second to exploit the steadily updated information which the robot gains during traveling towards the goal. Thus, for a robust and efficient path planning, the algorithm has to meet real-time requirements, which means that in our case planning a path has to finish within 100 ms.

The paper will be organized as follows: Section 2 will give a short overview of the basic Hybrid A* algorithm to allow for a detailed presentation of the extensions we have introduced to it in section 3. Section 4 will show some of the results obtained from simulations and finally section 5 will conclude the paper.

1.1 Problem formulation

The path planning problem can be stated as follows: Given a map m, find the cost-optimal path from the vehicle's current location x_s to a goal region G subject to the constraints given by the vehicle kinematics and the terrain. The discrete grid map m consists of three layers: The first one m_o contains whether a cell is blocked by an obstacle or clear, the second one m_s contains information about the surface quality and the third layer m_h contains a height value for each cell, resulting in a 2.5D height map.

The vehicle's current pose (i.e., the starting pose for the path planner) is given by the two-dimensional vector x_s and its heading θ_s. The height of the vehicle can safely be ignored and is assumed to be the same as the height of the corresponding cell $m_\mathrm{h}(\tilde{x}_\mathrm{s})$. Throughout this paper the tilde denotes discrete coordinates (i.e., cell indices). They can be obtained from continuous coordinates using the formula

$$\tilde{x} = \left\lfloor \frac{x - o_m}{\zeta} \right\rfloor , \tag{1}$$

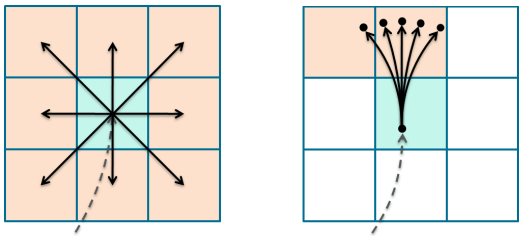
where o_m is the map origin and ζ is the cell size.

To allow for an approximate passing of waypoints, the goal region G is defined by the set of all locations x within a given radius r around the commanded waypoint w:

$$G = \{x \mid \|x - w\| < r\} \,. \tag{2}$$

2 Hybrid A* algorithm

The application of the well-known regular A* algorithm [3] is limited to discrete state spaces (see Figure 1a). In the case of a simple four- or eight-connected grid it would allow the vehicle to turn on the spot, which is not possible in the presence of nonholonomic constraints. The Hybrid A* algorithm is an extension to the regular A* algorithm to overcome its shortcomings. It has been successfully applied in various areas such as obstacle avoidance flight maneuvers [7] or path planning for autonomous mobile robots during the Urban Challenge [5].



(a) regular A* **(b)** Hybrid A*

Figure 1: Possible neighbors for a cell.

Unlike the regular A*, the Hybrid A* algorithm is capable of taking into account the continuous nature of the search space representing the real world. This is achieved by using a set of precomputed motion primitives to determine reachable states and, thus, to construct the search tree online. The continuous position which a motion primitive arrives at is stored alongside the discrete position in the corresponding cell.

2.1 Search space representation

The continuous state of a vehicle moving in a plane is given by (x, θ) with x being the vehicle position and θ being its heading. This results in a three-dimensional search space, which has to be discretized in order to be searchable by the Hybrid A* algorithm. The translational discretization is already given by the map representation.

Except the start heading θ_s, which is rounded to its nearest discrete value $\tilde{\theta}_\mathrm{s}$, the heading of the vehicle does not need to be rounded during the planning procedure as there exists only a limited set of admissible motion primitives all ending up in a heading which fits into the discretization scheme. Thus, $\tilde{\theta} = \theta$ follows. Although knowledge of the discrete position $\tilde{x}$ would be sufficient for the graph search, the continuous position x for reaching a cell is stored for each cell as well. It can then be used as the root for the node expansion (i.e., neighborhood search). This hybrid approach guarantees that a found path is by all means traversable. If only the cell centers had been considered, this could not be ensured.

In addition, similar to the regular A*, a cost function computes how expensive driving from the current position to an adjacent position would be. This cost is added to the so far accumulated costs and this sum is denoted as g. Furthermore, each node contains some book keeping data for the A* algorithm: a back pointer n_p to its parent for the final path reconstruction and the priority queue key f (containing the sum of real costs g and the estimated costs h to the goal). Thus, each node n of the search graph is completely defined by

$$n = (\tilde{x}, \tilde{\theta}, x, g, f, n_\mathrm{p}) \,. \tag{3}$$

2.2 Motion primitive construction

The smallest entities of the path are the *motion primitives*. They are defined by arcs (see Figure 1b) which have to satisfy the following three conditions:

- The driven distance must suffice to leave the current cell (i.e., chord length $l > \sqrt{2} \cdot \zeta$).
- The curvature is limited by the maximum steering angle α_max.
- The heading change δ has to be a multiple of the discretization step size of the heading dimension of the search space.

Using the single-track model, the steering angle α can be computed as a function of the chord length l and the heading change δ by

$$\alpha(l, \delta) = \operatorname{atan}\left(\frac{2b}{l} \sin\frac{\delta}{2}\right) \,, \tag{4}$$

with b being the wheelbase. The set of valid motion primitives $\mu(\theta, \delta)$ is given by all heading changes δ, for which $\alpha(l, \delta) \leq \alpha_\mathrm{max}$ holds true.

2.3 The Hybrid A* algorithm in detail

The Hybrid A* algorithm shares its major concepts with the regular version of the A* algorithm. It also utilizes two sets which keep track of the states during the search. The open set O contains the neighboring nodes of nodes already expanded during the search, the closed set C contains all nodes which have been conclusively processed.

The following listing gives a short overview of the general algorithmic scheme being the basis for the extensions we introduced to the Hybrid A*.

Algorithm 1 Standard version of Hybrid A*

procedure PLANPATH(m, μ, $x_\mathrm{s}, \theta_\mathrm{s}, G$)
  $n_\mathrm{s} \leftarrow (\tilde{x}_\mathrm{s}, \tilde{\theta}_\mathrm{s}, x_\mathrm{s}, 0, h(x_\mathrm{s}, G), \text{-})$
  $O \leftarrow \{n_\mathrm{s}\}$
  $C \leftarrow \emptyset$
  while $O \neq \emptyset$ **do**
    $n \leftarrow$ node with minimum f value in O
    $O \leftarrow O \setminus \{n\}$
    $C \leftarrow C \cup \{n\}$
    if $n_x \in G$ **then**
      return reconstructed path starting at n
    else
      UPDATENEIGHBORS(m, μ, O, C, n)
    end if
  end while
  return no path found
end procedure

procedure UPDATENEIGHBORS(m, μ, O, C, n)
  for all δ **do**
    $n' \leftarrow$ succeeding state of n using $\mu(n_\theta, \delta)$
    if $n' \notin C$ **then**
      if $m_\mathrm{o}(n'_{\tilde{x}}) =$ obstacle **then**
        $C \leftarrow C \cup \{n'\}$
      else if $\exists n \in O : n_{\tilde{x}} = n'_{\tilde{x}}$ **then**
        compute new costs g'
        if $g' < g$ value of existing node in O **then**
          replace existing node in O with n'
        end if
      else
        $O \leftarrow O \cup \{n'\}$
      end if
    end if
  end for
end procedure

3 Extensions to the Hybrid A*

In order to be useful for real-world applications, there are several challenges that cannot be accomplished by solely applying the Hybrid A* search. In this section we will describe three limitations and their respective overcoming of the standard Hybrid A* algorithm that we were facing in the context of our intended application scenario.

3.1 Planning via several waypoints

In a more complex mission scenario it is not practical to restrict the path planning to a search for a path to only one given goal region. Instead, the operator often wants the autonomous robot to pass a list of waypoints one after another. This raises the following problem: It cannot be determined a priori with which heading a given waypoint has to be arrived at in order to guarantee the reachability of the next waypoint starting at this heading.

Therefore, when planning a path via several waypoints, it is not feasible to just plan between pairs of waypoints and concatenate the paths as the vehicle's heading on arriving at a waypoint's goal region may lead to a dead-end if the vehicle is in an adverse pose in front of an obstacle. Fig. 2 shows this situation: The planner finds the shortest path to the goal region G_1 of the first waypoint arriving in the state (x_1, θ_1) but then fails to find a path to G_2 because of the limited turning radius of the vehicle (dashed line).

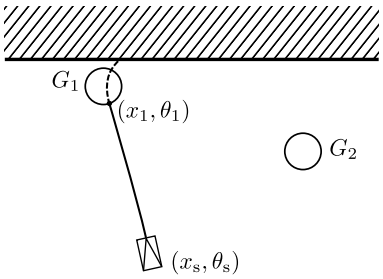


Figure 2: Planning using only the next waypoint.

This limitation can be overcome by planning not only to the next waypoint but to the next two waypoints. The path starting at $(x_\mathrm{s}, \theta_\mathrm{s})$ and going to G_2 has to fulfill the additional constraint of passing through G_1. Fig. 3 shows an exemplary path consisting of two segments, the first one ranging from $(x_\mathrm{s}, \theta_\mathrm{s})$ to (x_1, θ_1) (solid line) and the second one from (x_1, θ_1) to (x_2, θ_2) (dashed line).

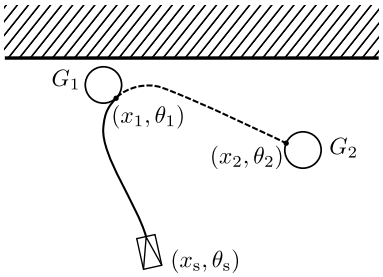


Figure 3: Planning using the next two waypoints.

In order to incorporate this constraint directly into the core of the planning algorithm, we modified algorithm 1 to explicitly take into account the passing of G_1. For this purpose, each node n gets an additional flag s which stores whether this node was added to the open set O during the planning of the first segment from $(x_\mathrm{s}, \theta_\mathrm{s})$ to G_1 ($s = 1$) or during the planning of the second segment from G_1 to G_2 ($s = 2$). So any node is now completely defined by

$$n = (\tilde{x}, \tilde{\theta}, x, g, f, n_\mathrm{p}, s) \; . \tag{5}$$

The planning procedure starts with a node, which is associated with the first segment ($s = 1$). When an expanded node reaches the first goal region G_1, the new successors are generated with their flag s set to 2. Only if a node with $s = 2$ gets expanded in the final goal region G_2, a path

is considered to be found. The following listing shows the extended versions of the Hybrid A*.

Algorithm 2 Extended version of Hybrid A*

procedure PLANPATH(m, μ, $x_\mathrm{s}, \theta_\mathrm{s}, G_1, G_2$)
  $n_\mathrm{s} \leftarrow (\tilde{x}_\mathrm{s}, \tilde{\theta}_\mathrm{s}, x_\mathrm{s}, 0, h(x_\mathrm{s}, G_1, G_2), \text{-}, 1)$
  $O \leftarrow \{n_\mathrm{s}\}$
  $C \leftarrow \emptyset$
  while $O \neq \emptyset$ **do**
    $n \leftarrow$ node with minimum f value in O
    $O \leftarrow O \setminus \{n\}$
    $C \leftarrow C \cup \{n\}$
    if $n_s = 1$ **then**
      if $n_x \in G_1$ **then**
        UPDATENEIGHBORS(m, μ, O, C, n, 2)
      else
        UPDATENEIGHBORS(m, μ, O, C, n, 1)
      end if
    else if $n_s = 2$ **then**
      if $n_x \in G_2$ **then**
        return reconstructed path starting at n
      else
        UPDATENEIGHBORS(m, μ, O, C, n, 2)
      end if
    end if
  end while
  return no path found
end procedure

procedure UPDATENEIGHBORS(m, μ, O, C, n, s)
  for all δ **do**
    $n' \leftarrow$ succeeding state of n using $\mu(n_\theta, \delta)$
    $n'_s \leftarrow s$
    if $n' \notin C$ **then**
      if $m_\mathrm{o}(n'_{\tilde{x}}) =$ obstacle **then**
        $C \leftarrow C \cup \{n'\}$
      else if $\exists n \in O : n_{\tilde{x}} = n'_{\tilde{x}} \wedge n_s = n'_s$ **then**
        compute new costs g'
        if $g' < g$ value of existing node in O **then**
          replace existing node in O with n'
        end if
      else
        $O \leftarrow O \cup \{n'\}$
      end if
    end if
  end for
end procedure

Using our approach, there is no particular event of switching all over from planning of the first segment to planning of the second segment. Instead, the open set simultaneously contains nodes of both segments once the first goal region has been reached. So during each node expansion the planning may continue with a node from either segment – whichever seems more promising according to the previously computed priority queue key f.

In order to guarantee the proper order of node expansions, the f value must be computed using a heuristic h that ranges over both segments. In the standard version of the Hybrid A* (i.e., when planning only to the next waypoint) a simple heuristic reflecting the Euclidean distance

$$h(x, w_1, r_1) = \max(0, \|x - w_1\| - r_1) \tag{6}$$

would be sufficient. However, in the extended version of the Hybrid A* the heuristic becomes slightly more sophisticated. Two cases have to be distinguished:

- Case 1: The path leading to the currently considered position x has not yet reached G_1 (i.e., $s = 1$):

$$h(x, w_1, r_1, w_2, r_2) = \max(0, \\ \|x - w_1\| + \|w_1 - w_2\| - 2r_1 - r_2) \tag{7}$$

- Case 2: The path leading to the currently considered position x has already passed G_1 (i.e., $s = 2$):

$$h(x, w_2, r_2) = \max(0, \|x - w_2\| - r_2) \tag{8}$$

Although this heuristic is admissible (subtracting the waypoint radii r_1 resp. r_2 ensures that it never overestimates the real costs), it is generally not consistent due to the increase in costs when changing from the first to the second segment. This results in expanding more nodes than theoretically needed depending on the size of the waypoint radius r_1 and the waypoints' relative positions. However, in a practical implementation this can be neglected as computing the exact shortest path from x to G_2 touching G_1 would be very expensive. Furthermore, the proposed heuristic does not take into account the heading of the vehicle, which would further reduce the number of expanded cells. Although this could be implemented (e.g. by using Dubins curves [4]) this would result in an unmanageable computational complexity.
The extension of our approach to more than two waypoints is straightforward but in practice there is generally no need for this as planning over two waypoints is sufficient to cope with situations like the one depicted in Fig. 2 and 3. This is due to the fact that planning is performed in an iterative manner: As soon as the vehicle reaches w_1 the planner continues to plan to w_3 via w_2 etc.

3.2 Incorporation of terrain characteristics

The costs g for traveling to a cell incorporate several efficiency measures like driven distance, needed energy for climbing hills, needed steering energy and surface quality. As one motion primitive may span multiple cells of the discrete map, it will not suffice to consider only the transition from the start cell of the motion primitive to its end cell. Instead, all intermediate cells have to be taken into account as well. This is achieved by rasterizing each motion primitive off-line in advance using techniques borrowed from computer graphics like Bresenham's algorithm [1]. This results in a list of cells for each motion primitive, which can be used for further cost computations. Pivtoraiko et al.

share this idea in the context of their state lattice planner and call the list of swept over cells a *swath* [6].
We use this swath to compute two portions of the costs g. The first one incorporates the terrain with respect to its surface quality m_s. This includes, for example, information about whether the current motion primitive leads through off-road regions or lies on a street. This surface assessment is tightly coupled with the costs due to the driven distance

$$g_\text{dist}(x, x') = l_\mu \cdot (1 + c_\text{off-road}) \ , \tag{9}$$

where l_μ is the arc length of the motion primitive μ used to drive from x to x'. The penalty $c_\text{off-road}$ for driving off-road can be defined in different ways. However two approaches turned out to be reasonable choices.
The first one computes an *off-road ratio* η which is defined by

$$\eta = \frac{\text{number of off-road cells in swath}}{\text{total number of cells in swath}} \ . \tag{10}$$

Using this off-road ratio, the penalty $c_\text{off-road}$ is given by

$$c_\text{off-road} = \eta \cdot \lambda_\text{off-road} \ , \tag{11}$$

where $\lambda_\text{off-road}$ is a fixed non-negative penalty factor.
The second approach penalizes driving partly off-road independently of the amount of off-road cells in a swath:

$$c_\text{off-road} = \begin{cases} \lambda_\text{off-road} & \text{if at least one off-road cell in swath} \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

Again, $\lambda_\text{off-road}$ is a fixed non-negative penalty factor.
The second portion of the costs g which we compute using the swath, is an estimate of the roughness/slope of the terrain, which we denote g_slope. For this purpose, the height of each cell in the swath is looked up in the height map m_h. After that, the sum Σ_slope of the absolute height differences of all adjacent cells in the swath can be computed. Finally, the costs due to slopes is computed by

$$g_\text{slope} = \lambda_\text{slope} \cdot \Sigma_\text{slope} \ , \tag{13}$$

where again λ_slope is the parametrizable weight of this cost portion.

3.3 Heuristic for waypoints outside the local map

As our planning concept is based on only local knowledge of the environment, the path planner must be able to cope with waypoints that lie outside the local map. In this case, the planner can only plan to the border of the map but it does not know a priori to which point on that border it should plan to. To guide the planning in a proper direction, the standard heuristic function (6) is not sufficient. Instead, a more complex heuristic has to be used as the chosen heuristic has a significant impact on *where* the local map border will be reached first. If the heuristic underestimated the real costs outside the map by too far, the planned path would try to leave the map on the most economical way, which could make the vehicle stay on streets – even if they do not lead to the waypoint.
As our planner possesses no knowledge of the environment outside the local map, the heuristic assumes that there are – in contrary to the inside – no streets outside the map. Thus, the distance v from the map border crossing B to the waypoint will get more weight during the cost estimation than the distance u inside the map which is assumed to be drivable on a street in the best case (see Fig. 4).

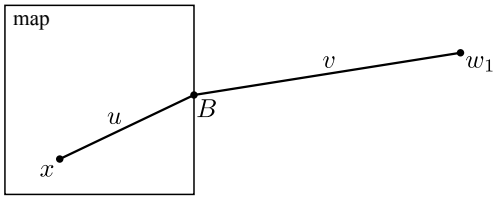


Figure 4: Heuristic for waypoint outside local map.

This poses the following optimization problem, which is numerically solved for each node using the Newton-Raphson mehod:

$$h(x, w_1) = \min_B (u + \lambda_\text{off-road} \cdot v) \tag{14}$$

subject to the following constraints:

$$u = \|x - B\| \ , \tag{15}$$

$$v = \|w_1 - B\| \ , \tag{16}$$

$$B \text{ on map border.} \tag{17}$$

Note that the solution of this optimization problem is independent of the waypoint radius r_1. It is merely an offset to the heuristic, which can safely be omitted, as the planning stops at the map border and, thus, the waypoint outside the map is not reached by the planner during the current iteration.

3.4 Wall following

Because the path planner possesses only partial knowledge about the environment, as it only has a local map, it is possible that the planner gets stuck if an obstacle is U-shaped or has very large dimensions lateral to the direction to the next waypoint. While performing free planning, a large obstacle causes the vehicle to travel along the obstacle for some time, as the heuristics allows for a certain detour. If the obstacle is too large and the detour becomes too long, the vehicle will turn around and travel along the obstacle in the reverse direction. If the obstacle is also large in this direction, the vehicle will turn around again, leading to an infinite loop. Thus approximately reaching a pose in which the vehicle has been before is used to detect this particular situation.
Afterward, the obstacle is explored by wall following. This is performed using the Hybrid A* algorithm as well: Automatically generated waypoints are placed along a virtual

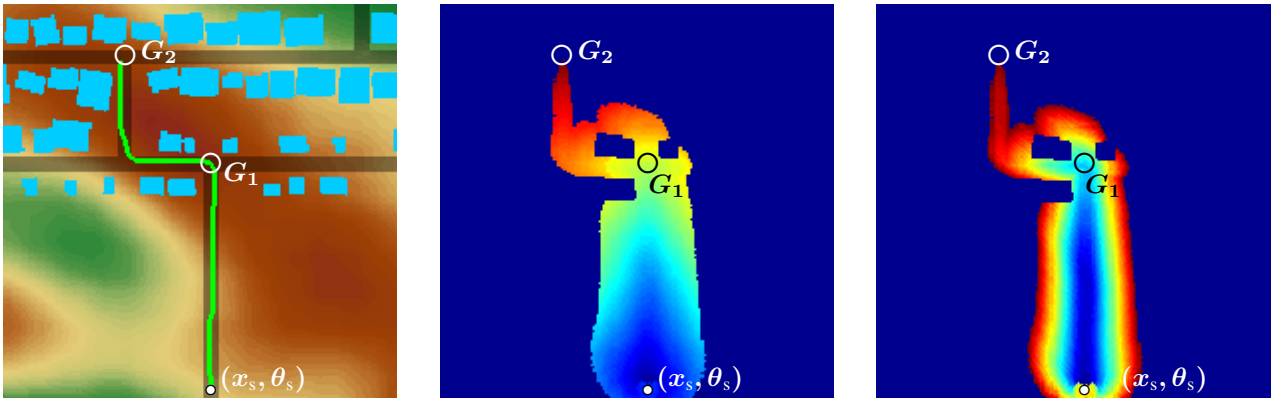


Figure 5: Planned path in an urban environment (left), real costs $\min_\theta g$ (center), predicted costs $\min_\theta f$ (right).

street surrounding the detected obstacle. So the robot follows this street around the obstacle until the behavioral planner decides to cease wall following in order to reach a goal on the other side.
The availability of this wall following capability is very important to our planning concept as the coverage of the local map may be quite limited compared to the size of existing obstacles like walls, fences, etc.

4 Results

The planner algorithms have been evaluated both on simulated as well as real-world data. Due to its efficient implementation in C++ it is capable of generating paths with a rate higher than 10 Hz on a Notebook equipped with a moderately fast Intel Core 2 Duo P8400 CPU with 2.26 GHz, which totally fulfills the real-time requirement.
Fig. 5 shows a planning result. In the left figure one can see the generated path through an urban environment. Thanks to the incorporation of the off-road penalty into the path costs the vehicle stays on the road instead of cutting through off-road regions. Furthermore, due to the higher costs associated with off-road areas, the Hybrid A* algorithm explores primarily the road network, which can be seen from Figure 5 (center) that shows the real costs g and the predicted costs f (right).
Especially from the right figure, it can be seen clearly that the cell expansion during the search for a path to G_2 is guided through the goal region G_1 of the first waypoint.

5 Conclusions

In this paper we presented the application of Hybrid A* to path planning for autonomous mobile robots in unstructured outdoor environments. It respects kinematic constraints of the vehicle and takes surface conditions into account. Thanks to the extensions we made to the original algorithm, it is capable of explicitly planning *via* one waypoint to a second waypoint. Thus, the generated paths are guaranteed to be drivable by the vehicle. The use of thoroughly chosen cost functions and heuristics allow for the incorporation of various constraints like driving preferably on-road. Finally, we pointed out how planning to a waypoint lying outside the local map can be performed using an appropriate heuristic function and taking into account obstacles exceeding the map dimensions.

References

[1] Bresenham, J.: *A linear algorithm for incremental digital display of circular arcs*, Communications of the ACM 20(2), pp. 100–106, 1977

[2] Kopfstedt, T.; Restle, M.-O.; Grimm, W.: *Terrain Optimized Nonholonomic Following of Vehicle Tracks*, 7th IFAC Symposium on Intelligent Autonomous Vehicles, Lecce, Italy, 2010

[3] Hart, P.; Nilsson, N.; Raphael, B.: *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*, IEEE Transactions on Systems Science and Cybernetics 4(2), pp. 100-107, 1968

[4] LaValle, S. M.: *Planning Algorithms*, Cambridge University Press, 2006

[5] Montemerlo, M.; Becker, J.; Bhat, S. et al.: *Junior: The Stanford Entry in the Urban Challenge*, Journal of Field Robotics 25(9), pp. 569–597, 2008

[6] Pivtoraiko, M.; Knepper, R. A.; Kelly, A. *Differentially Constrained Mobile Robot Motion Planning in State Lattices*, Journal of Field Robotics 26(3), pp. 308–333, 2009

[7] Richards, N. D.; Sharma, M.; Ward, D. G.: *Hybrid A*/Automaton Approach to On-Line Path Planning with Obstacle Avoidance*, AIAA 1st Intelligent Systems Technical Conference, 2004