A Robust Modified Hybrid A*-based Closed-loop Local Trajectory Planner for Complex Dynamic Environments

Chinnawut Nantabut[a] and Dirk Abel[b]
Institute of Automatic Control, RWTH Aachen University, Campus Boulevard 30, Aachen 52074, Germany

Keywords: Collision Avoidance, Autonomous Driving, Path Planning, Trajectory Planning, Object Detection, L-Shape Fitting, Hybrid A*, Weighted A*, Stanley Controller, PID Controller, Bicycle Reciprocal Collision Avoidance.

Abstract: In the research field of autonomous driving, planning safe and effective trajectories is a key issue, which also requires reliable detection of objects in the environment. This publication introduces a new approach to compute safe trajectories for automated road vehicles quickly and robustly, also considering reliable object detection for static and dynamic objects. For this purpose, the Hybrid A* algorithm modified with Weighted A* is used to accelerate the planning of a collision-free path because the weight w can make the heuristic term h become more important and make the tree much more narrow in the direction of the goal. Afterwards, PID- as well as Stanley controllers are utilized to realize reliable trajectories. This combined algorithm is extended with the L-Shape fitting algorithm to detect objects in the environment. The entire approach is evaluated for unstructured and semistructured environments using simulations of an automated vehicle with a realistic interaction of dynamic obstacles in the presence of model and sensor uncertainties, guarantees a real-time capability of 1 s, and results in collision-free vehicle movement. The whole algorithm, which yields very promising results, will be transferred to a C++ framework and tested with flexible test vehicles in real environments in the future.

1 INTRODUCTION

Many companies and research institutions are engaged in the realization of autonomous driving, which is very challenging because the autonomous system must perform all functions from planning and implementing driving functions to system monitoring and emergency management. In order to avoid any collision in any case, it is essential that the system has an exact scene understanding of the static environment and the complex behavior of dynamic obstacles. The resulting trajectories, which the system calculates based on the current traffic situation, must be adjusted permanently in real time by means of a closed control loop. Normally, path or trajectory planning approaches are validated only for structured (infrastructure is previously known), unstructured environment (obstacles are detected online) or semistructured (the combination of both cases). In our case, we consider both types of environments where obstacles have to be detected and the replanning is oftentimes activated when needed. Furthermore, we take into account the interactions of all traffic participants for more realistic test scenarios. Recently, researchers are mainly developing methods to tackle the following problems: scene understanding, trajectory planning and trajectory control. The implemented methods then perform well either in complex dynamic environments, which must be permanently observed, or in structured environments, whose layout may already be known in advance. In most scientific work, dynamic obstacles are considered challenging due to their poorly predictable behavior, but the movement must still be considered at all times. Most works in literature focus only on the planning of collision-free paths that can simply be driven using a feedforward controller which does not continuously prove if the desired behavior is actually reached or not. If dealing with collision avoidance, many papers describe non-convex optimization problems to deal with unstructured and cluttered environments. But the calculated path is probably local minimal or it is quite time-consuming due to the non-convex search space. In order to be more specific and to identify development needs, an overview of relevant scientific work is given next.

[a] https://orcid.org/0000-0002-5767-6023
[b] https://orcid.org/0000-0003-0286-3654

Nantabut, C. and Abel, D.
A Robust Modified Hybrid A*-based Closed-loop Local Trajectory Planner for Complex Dynamic Environments.
DOI: 10.5220/0010997800003191
In *Proceedings of the 8th International Conference on Vehicle Technology and Intelligent Transport Systems (VEHITS 2022)*, pages 26-35
ISBN: 978-989-758-573-9; ISSN: 2184-495X

The path planning method can be categorized into graph-based and sample-based methods (Paden et al., 2016), (González et al., 2016). Other approaches use neural networks or stochastic methods, e.g., evolutional strategies and can be combined with conventional methods to optimize the results (Patle et al., 2019). For the graph-based approaches, Dijkstra or A* algorithms are applicable for the problems that are defined by a grid-based formulation. For example, the continuous search space is partitioned into a discrete one defined by a grid. The search space is not infinitely large anymore and, therefore, facilitates the searching algorithm for finding a suitable path. Unfortunately, there is no guarantee of finding an optimal path if not enough calculation time is provided. A*, however, does not perform well in dynamically changing environments because it recalculates the path always from start point to end point in each time step. That means the method is quite computationally intensive, especially when the dimension of the search space is high. D* has been developed then, which is able to correct the calculated path considering new obstacles on the way. Both algorithms, still, do not consider the realistic motion behavior of vehicles where the path to be followed is curvy. To address this challenge, Hybrid A* first demonstrated its successful usage on the Stanford Racing Teams robot in the DARPA Urban Challenge in 2007 (Dolgov et al., 2008). They tested the implementation on the simulated environments such as labyrinths, parking lots and U-turns. A real-time Hybrid A* was successfully implemented and evaluated for the KTH Research concept vehicle using Dubins curve for forward driving (Dolgov et al., 2010). The cost function used for smoothing a path in Hybrid A* was extended by considering the lane network of the parking lot for semistructure application by combining the topological graph search and the free-space path planning to reduce the number of expanded nodes (Kurzer, 2016). The three papers above, however, updated the environments using SLAM as an input for the exact path planning, which comes with high computational costs. Hybrid A* was applied to solve the path planning problem in structured environments containing streets and in unstructured environments for nonholonomous vehicles (Petereit et al., 2012). The outstanding feature of this approach is the smooth visit of a series of given waypoints. The obstacles were, however, predefined and the experimental part only observed one specific example. To increase the performance of the algorithm, the visibility graph was used for the given polygonal obstacle's vertices to find the optimal waypoints of the holonomic path up to the goal region, which helps dramatically reduce the number of expanding nodes. Here, the obstacle configuration and the available map were pre-defined as well (Sedighi et al., 2019). Hybrid A* was then applied for unstructured environments, whereas state lattice verification technique has switched to structured applications (Guirguis et al., 2019). The method was successfully tested in simulations for lane following tasks, but not for a cluttered environment. the Hybrid A* was optimized for wheeled robots in consideration of a collision risk with dynamic obstacles (Nemec et al., 2019). The overall calculation time was limited to 1 s so that the real-time capability of the path planning was guaranteed. However, the obstacles are also predefined and it is actually not clear if the weight helps accelerate finding the path.

In the papers mentioned above, the obstacle recognition part normally utilizes the occupancy grid mapping. However, this uses the full information of the obstacle configuration is not necessary when we can assume almost all of the obstacles are rectangular as in the parking lot or street scenarios. Therefore, Hybrid A* as a local planner in combination with model-based, i.e. using the geometry information, object detection seems promising but has not received much attention in any research study until now.

To acquire an accurate detection of the environment, multiple sensors, such as LiDAR, radars, and cameras, are used in this paper combined with the well-known L-Shape approach to detect other vehicles in the vicinity. In our approach, unstructured environments are considered, for which the Hybrid A* approach has proven to be very promising. It calculates a path using discrete steering angle values and a kinematic model of the ego-vehicle in a discrete search space containing position and orientation. Furthermore, Weighted A*, an algorithm to accelerate the search with a suboptimal solution, is combined, which seems suitable for unstructured and semistructured environments. The interaction of dynamic obstacles, here vehicles, is implemented using the bicycle reciprocal collision avoidance method (Alonso-Mora et al., 2012). The trajectory control is then realized by applying a PID controller for velocity control along the path and a Stanley controller for keeping the vehicle on the track laterally.

The paper furthermore focuses on the detection and the decision-making modules. Detected objects can be fed into the decision-making module directly where path and trajectory planning take place. The decision maker performs a calculation of the path parameterized by time to realize the trajectory planning. One hundred scenarios for each unstructured and semistructured environment are randomly generated to validate the robustness of the approach.

The approach introduced in this paper calculates collision-free trajectories in all situations while keeping the calculation time under 1 s for real-time applications. However, this assumption is valid in the simulation when using the search algorithm because collisions can still occur in reality, especially in the high-speed networks as on highways. The work is therefore applicable for the case of low-speed range (< 10 m/s) as in parking lots.

2 METHOD

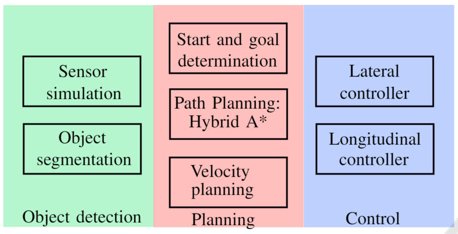

Figure 1: A workflow of the trajectory planning demonstrated in this work is presented.

At first, the position, the orientation, the size as well as the velocity of an ego vehicle is previously known. A reference path is given for the vehicle to follow and no obstacles are present. The vehicle tracks on this given path as long as no obstacles are detected. If that is not the case, a collision avoidance framework presented in this paper is activated and a collision-free path and a desired velocity are then calculated. Figure 1 summarizes the entire work consisting of three main modules. Obstacles are defined previously, detected and then segmented in the simulation by the sensor of an autonomous vehicle at every time step. During the object detection step, LiDAR data is continuously processed using an L-Shape fitting approach to generate suitable rectangular bounding boxes for all obstacles in sight. The bounding boxes are then fed to the trajectory planning consisting of a planning and velocity planning part. In the planning module, the start and goal points are reasonably determined for finding a new path using Hybrid A* in combination of Weighted A* algorithm, in dependence on the type of environments. They have to be far enough all of the obstacle points for the sake of the collision avoidance. The path planning algorithm is activated in case that the obstacles stay in the way. The velocity planner then uses the distance to the next front object to find a desired or reference velocity. The controllers hereinafter are activated to keep the vehicle on track.

2.1 Object Detection: Sensor Simulation

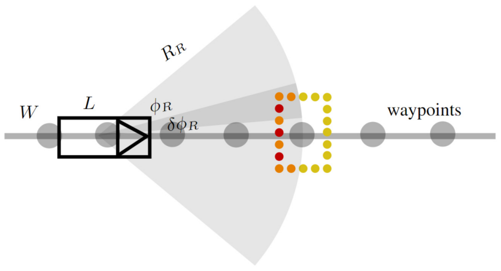

Figure 2: The points detected by the LiDAR sensor attached on a vehicle of length L and width W are simulated.

Figure 2 illustrates how to simulate a LiDAR sensor used in the experiments in the paper. A previously defined path for the ego vehicle is given by a set of gray circular waypoints. The detection range of the sensor is provided by a gray circular arc on the black ego vehicle. A static obstacle depicted in yellow points is defined beforehand in the simulation. The orange points are the ones that lying within the sensor range and the red points are the real ones that will be preprocessed in the object segmentation module.

The waypoints defining the path can be determined offline by using the previously known information with a fixed goal point by, for example, a conventional A* algorithm. Otherwise, the path can be just defined by a center line of the lane in case that the vehicle just follows a lane in case of a route planning. In this paper, however, the waypoints are given and the ego vehicle tries to track the resulting path while keep itself away from collision with the surrounding obstacles detected online.

Supposingly, the yellow rectangular obstacle's points representing both static such as vehicles standing still in a parking lot or the driving ones are predefined pointwise. The orange points within the sensor range of radius R_R and angle φ_R are considered. Only one red point of each circular segment of angle $\delta\varphi_R$ is regarded for the calculation in the upcoming part to optimize the calculation time. All of these detected points are then submitted to the preprocessor in order to calculate the possible rectangular configuration of the obstacles defined by 4 vertices for each at that moment. If at least one of the obstacle's points is standing within the range around the predefined path, the path planning and the control module of the framework are then activated so that the ego vehicle can avoid serious collisions.

2.2 Object Detection: Object Segmentation

L-Shape fitting is a model-based vehicle detection and tracking used under the assumption that rectangle obstacles are detected from the side by the laser scanner in the form of L (Zhang et al., 2017) . The procedure generates a sample of rotated boxes for each segment of the point cloud and then compares them to determine which one best suits all of the points using a minimal distance criterion. These points are the red ones that we obtained from the procedure that we just explained earlier.

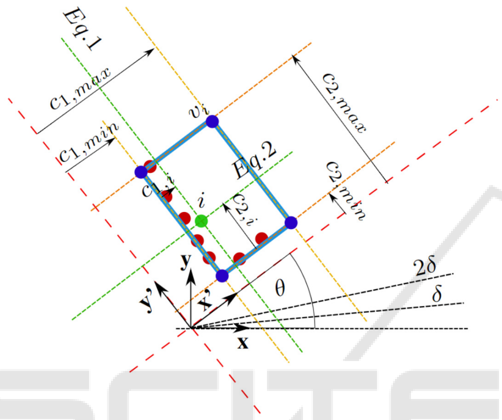

Figure 3: An L-Shape fitting algorithm is utilized for the detection of a rectangular obstacle.

Figure 3 demonstrates how the L-Shape fitting algorithm works. One of the clusters of the preprocessed point cloud, in red, from the previous sensor simulation in Figure 2 is processed in this step, resulting in the blue bounding box with the dark blue vertices. Each green point i from this cluster is considered for comparing the distance to the bounding box.

The coordinate system (x, y) is fixed at the sensor. The rotated coordinate system (x', y') which suits the obstacle best is to be determined. The algorithm deals with the combinatorial least square problem described by searching for the two perpendicular line equations and choose which line each point of the cluster belongs to. Out of three criteria, the closeness maximization is utilized because of the good average time consumption and exactness of detection (Zhang et al., 2017). This criterion finds the configuration of the rotated box the sum of which of the distance of the red points to their closest side of the box is minimal.

Instead of directly solving the least-square problem, a brute force search is carried out to reduce the calculation time by defining a search angle increment $\theta = n\delta$, where $n \in \mathbb{Z}^+ \cup \{0\}$ and $0 \leqslant \theta < \frac{\pi}{2}$. The calculation for each angle can be done parallely to improve the real-time capability and the solution is acceptable when the angular resolution δ is small enough. The corresponding perpendicular line equations for point i (green from Figure 3.) at this angle are

$$x_i \cos(\theta) + y_i \sin(\theta) = c_{1,i} \tag{1}$$

and

$$-x_i \sin(\theta) + y_i \cos(\theta) = c_{2,i}. \tag{2}$$

$c_{1,i}$ and $c_{2,i}$ are the distances of point i to the rotated coordinate system which can be compared to find the boundary of the box. This is defined by the line equations with $c_{1,min} = \min c_{1,i}$ and $c_{1,max} = \max c_{1,i}$ for the first equation and $c_{2,min} = \min c_{2,i}$ and $c_{2,max} = \max c_{2,i}$ for the second equation. The minimal distance of this point to the bounding box $d_{i,min} = \min(\min(c_{1,i} - c_{1,min}, c_{1,max} - c_{1,i}), \min(c_{2,i} - c_{2,min}, c_{2,max} - c_{2,i}))$ is calculated and the sum of this minimal distance of each point $S_\theta = \sum d_{i,min}$ quantifies how well the current box matches the cluster. The box at the angle of θ_{min} with the smallest sum is chosen. The matrix L summarizes all the coefficients of the lines plus a repetition of the first line to determine the four vertices.

$$L = [l_{i,j}]_{5\times3} = \begin{bmatrix} \cos(\theta_{min}) & \sin(\theta_{min}) & c_{1,min} \\ -\sin(\theta_{min}) & \cos(\theta_{min}) & c_{2,min} \\ \cos(\theta_{min}) & \sin(\theta_{min}) & c_{1,max} \\ -\sin(\theta_{min}) & \cos(\theta_{min}) & c_{2,max} \\ \cos(\theta_{min}) & \sin(\theta_{min}) & c_{1,min} \end{bmatrix} \tag{3}$$

The vertex v_i (blue) of each box with $i \in \{1,2,3,4\}$ is calculated by using the element of the matrix L as follows,

$$v_i = \frac{1}{l_{i,1}l_{i+1,2} - l_{i,2}l_{i+1,1}} \begin{pmatrix} l_{i,3}l_{i+1,2} - l_{i,2}l_{i+1,3} \\ l_{i,1}l_{i+1,3} - l_{i,3}l_{i+1,1} \end{pmatrix} \tag{4}$$

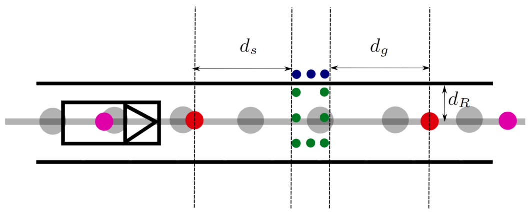

Figure 4: The start and goal configurations for determining a collision-free path by Hybrid A* are differently chosen in dependence of the environment types.

These four vertices of each segment are then sent to the path planning module for the collision avoidance to determine the start and goal configuration using in the path planning algorithm.

Figure 4 explains how to determine start and goal points needed for the path planning algorithm. The points for the unstructured environments are portrayed in pink, while the ones for the semistructured scenarios in red. Obstacles' points that could lead to potential collision danger to the vehicle are depicted in green, while those in the safer zones are illustrated in blue.

In case of unstructured environments, the start point can be easily defined as the current position and the goal point can be the end of the given path. For semistructured cases, however, the vehicle has to stay within the street as long as possible. All obstacle points within range d_R are projected to the original path. Let d_s, the safe distance from the closest projected obstacle point, first equal to d_g, the safe distance from the most distant projected obstacle point. If the start point goes behind the vehicle, we then set the start point to the current position and orientation of this vehicle.

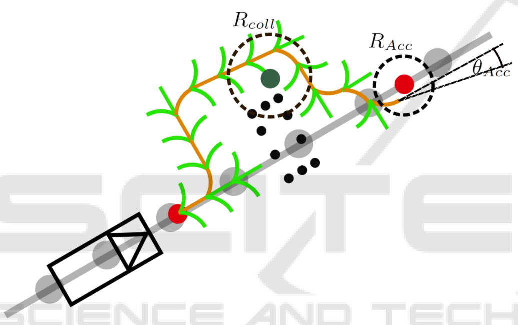

Figure 5: Tree expansion of the Hybrid A* algorithm is applied in order to determine an optimal collision-free path for a nonholonomic vehicle. The tree is not expanded in the vicinity of the obstacles and the expansion process is stopped when the last node is in the goal area with an acceptable orientation deviation.

2.3 Planning: Path Planning with Modified Hybrid A*

Hybrid A* calculates, in comparison to A*, a smooth path comprising a set of small smooth paths from one state to another, depicted in Figure 5 (Dolgov et al., 2008). It uses the red start and goal configuration to find a collision-free path based on the expansion of the green tree starting incrementally from the current node to its neighboring nodes and so on. As a result, an orange optimal path is calculated.

The algorithm compares the approximated length f of the path via each neighboring node and chooses the most promising one with the least f. This term can be expressed as

$$f = g + wh, \tag{5}$$

where g is the exact path length from the start to that neighboring node, h is the approximated path length from the neighboring node to the goal node which can be a heuristic and w is the weight to accelerate the search with the greedy behavior to the goal point which is normally used in Weighted A*.

By expanding the tree, the dark green node cannot be added if it is too close to the obstacle points within the range of R_{coll}. The tree expansion is stopped when the successor node is within the radius of R_{Acc} and the deviation of the orientation is within θ_{Acc}. The optimal path is then traced back from the goal to the start node.

The original Hybrid A* algorithm employs two admissible heuristics to find the most promising successor (Dolgov et al., 2008), (Dolgov et al., 2010). That means the heuristics h are not larger than the real path length from the current to the goal node so that the search algorithm is applicable.

The first heuristic considers the nonholonomity of the vehicle without taking into account the obstacles in the environments. It uses motion primitives which can be seen as a set of short paths that can be reused in every node expansion step by rotating in accordance with the orientation of the current node. The inputs of each motion primitive, i.e. the control input and the vehicle model parameters, stay the same until the next node is reached. Dubins or Reeds-Shepp curves are used to accelerate the tree expansion.

The second heuristic, on the contrary, regards the environments, but not the kinematic model of the vehicle or the smoothness of the path. It uses the "flow field" when multiple start points are given for a certain goal point (Emerson, 2019). This can be applied as well for the Voronoi field. This determines which grid cell of the search space belongs to which obstacle by using the obstacle itself as the start point.

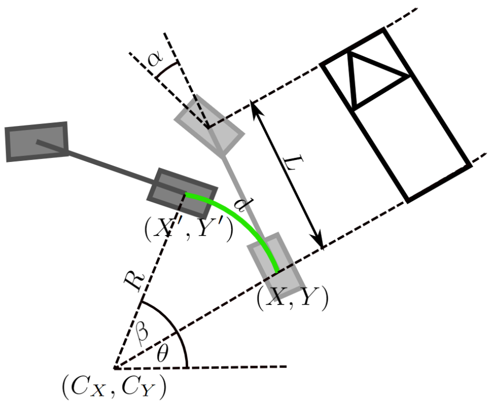

Figure 6: The motion primitive is calculated for the tree expansion by using the single track model for Hybrid A* as explained in Equations (6)-(8) (Udacity, 2015).

Instead of expanding the successors around the current node into the next cells as in A*, Hybrid A* uses the kinematic model to expand the tree for the first heuristic as shown in Figure 6. Hereby, the black vehicle is transformed into a single track model depicted as a "bicycle" in gray. The new state of the vehicle is illustrated as the black bicycle after traversing the green path with a distance of length d.

The algorithm discretizes the control variables, such as the steering angle, and determines the successor nodes with the help of single track model for nonholonomic vehicles. The model assumes that the wheels of each axis are shifted to the longitudinal axis of the vehicle and the center of mass to the ground to reduce the degrees of freedom of the model and facilitate the calculation of the successor nodes depicted in gray.

To determine the motion primitive, the algorithm calculates first the driving angle β given the driving distance d which is the length of the diagonal of the cell plus an offset so that the successor node can locate in the neighboring grid cell, wheel base L and the steering angle α from the set of $\{-\alpha_{max},\dots,0,\dots,+\alpha_{max}\}$, with the vehicle projected to a single track model:

$$\beta = \frac{d}{L}\tan(\alpha). \tag{6}$$

The radius R of the motion is then calculated and used for determining the center of the circular motion

$$R = \frac{d}{\beta}. \tag{7}$$

The angle from the turning center to the rear wheel center θ in return is used for calculating the successor node $(X',Y')^T$.

$$\begin{pmatrix} X' \\ Y' \end{pmatrix} = \begin{pmatrix} C_X - R\sin(\theta) + R\sin(\theta+\beta) \\ C_Y + R\cos(\theta) + R\cos(\theta+\beta) \end{pmatrix} \tag{8}$$

To limit the calculation time, the maximal iteration is set to $N_{I,max}$, each with the maximal steering angle α_{max} and n_α discrete values. The tree can then be expanded by a Dubins curve to link the node which yields the smallest f value and the goal point. Dubins curve use the start and end configuration, calculate the path lengths of 6 possible movements when available, and compare them to find the shortest path. Furthermore, we do not consider the obstacle, since practically, we still do not know how the obstacle configuration looks like when it is located further from the current position. We then expand the tree from the node that has the largest approximated cost f because the path that ends with this node is found with the exactness of path length quantified by f. The approximated remaining path length from the Hybrid A*'s end node to the goal point, or h, can be determined by the simple geometry of Dubins or Reed-Shepps curves.

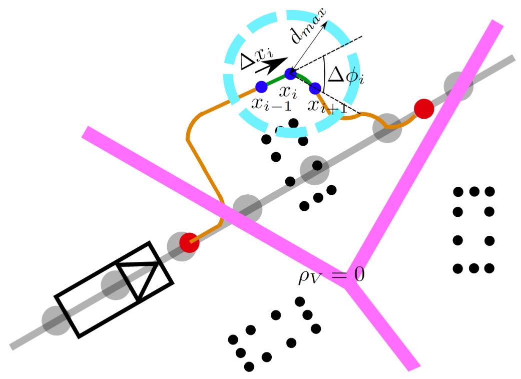

Figure 7: A path calculated by the tree expansion is optimized in view of four criteria.

$$J = w_\rho \sum_{i=1}^{N} \rho_V(x_i,y_i) + w_o \sum_{i=1}^{N} \sigma_o(|x_i - o_i| - d_{max})$$
$$+ w_\kappa \sum_{i=1}^{N-1} \sigma_\kappa\left(\frac{\Delta\phi_i}{|\Delta x_i|} - \kappa_{max}\right) + w_s \sum_{i=1}^{N-1} (\Delta x_{x+1} - \Delta x_i)^2 \tag{9}$$

The cost function J from Equation 9 is illustrated in Figure 7 (Dolgov et al., 2008). It optimizes the calculate path in orange, defined by N way points considers the Voronoi field value σ_v (=0 in Voronoi edges (pink) and =1 next to and in obstacles) the distance to the nearest obstacle $|\, x_i - o_i \,|$ compared to maximal allowed distance (light blue) d_{max} , the actual curvature $\frac{\Delta\phi_i}{|\Delta x_i|}$ compared to maximal allowed curvature κ_{max} and the vector change to the neighboring nodes of node i. The last two terms use the information about the current and both consecutive points (green). The two terms in the middle are quadratic penalties. The points are then updated by conjugate gradient optimization (CG), which can be easily implemented (Shewchuk, 1994). An oversampling of nodes is carried out by placing one more point between two consecutive segments and re-optimized by using the same optimization techniques.

Due to the fact that Hybrid A* rarely yields the path that leads to the exact continuous goal configuration, the end of the tree is set to the predefined goal configuration.

2.4 Planning: Velocity Planning

In Figure 8, the controllers for both longitudinal and lateral direction are illustrated. The desired longitudinal velocity of the ego v_{lon} depends on the current distance d_x from the front axis of the ego vehicle along the path to the nearest next vehicle within the range of d_y. The desired velocity profile can be defined in three sections by introducing two offsets $d_{x,1}$ and $d_{x,2}$. When the distance d_x is less than $d_{x,1}$, the vehicle tries to stop and therefore $v_{lon,desired}$. In the second field, the desired velocity $v_{lon,desired}$ rises linearly with the current distance d_x. After the offset $d_{x,2}$, the desired velocity is kept constant according to the ego's vehicle desired velocity.

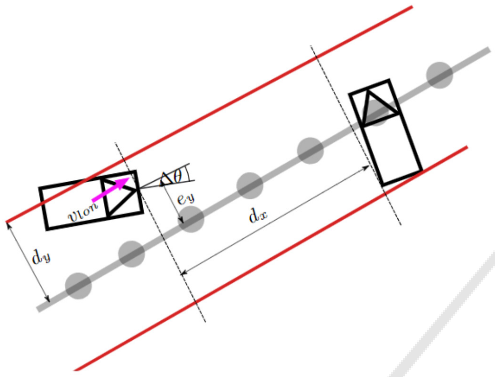

Figure 8: A PID controller is utilized for the longitudinal speed control and a Stanley controller for the lateral control.

2.5 Control: Longitudinal and Lateral Controller

As mentioned earlier, two controllers are employed, which are illustrated in Figure 8. A PID controller is applied for regulating the vehicle's longitudinal velocity v_{lon} to be as close to $v_{lon,desired}$ as possible. The smaller its value is, the less velocity the ego vehicle desires to have. To eliminate the deviation of the lateral distance to the path e_y and the deviation of the orientation $\Delta\theta$, a Stanley controller is utilized (Thrun et al., 2006).

3 VALIDATION

We evaluate the performance of the modified Hybrid A* in a combination of L-Shape fitting object detection in the simulations for both unstructured and semistructured environments in the presence of dynamic obstacles.

3.1 Simulation Setup

The implementation of Hybrid A* is based on the open-source C# implementation (Nordeus, 2015). However, we decided to reimplement it in MATLAB. There are many reasons for this. Firstly, we can carry on an analysis of the algorithm through implementing on our own and can develope it for the future by combining Hybrid A* with more algorithms from the A* family such as Dynamic A*, Anytime Repairing A* or even Anytime Dynamic A*. Secondly, although MATLAB also provides a Navigation Toolbox comprising of Hybrid A* path planning, we cannot modify the weight inflating the heuristic as used in this paper. Lastly, we will transfer the implemented MATLAB code onto a Simulink Real-Time target used in a host computer on our model vehicle "Buggy" developed by our institute (Reiter et al., 2017). We use Dubins curve instead to consider only forward movements. If the Reed-Shepps curves were implemented, which is normally used in Hybrid A*, the behavior of node expansion would still be similar. The maximal iteration $n_{I,max}$ for tree expansion is set to 10. The maximal steering angle α_{max} is set to 45 deg to make the vehicle only drive forward and $n_\alpha = 5$ discrete values. The driving distance d is set to $\sqrt{2}+0.01$ m, which is the diagonal length of each grid cell (cell size is 1 m long) plus some offset to make sure the successor node lies in another grid. The wheel base of all vehicles L is 2 m. In addition, all weights w_i of the cost function J for the optimization are set to one for simplicity in order to equally consider all of the components of the cost function. The weights are needed to be tuned using machine learning methods in the future.

We investigate the use of the modified Hybrid A* in two types of environments as always mentioned. The search space of size 20 m × 20 m is filled with previously known rectangular obstacles. The two experiments handle the unstructured and semistructured environments of grid size 20 m × 20 m for a full trajectory planning which consider both obstacle detection and control steps.

Figure 9 summarizes the interaction of the ego vehicle and the obstacles in the unstructured and semistructured environments. Hereby, the static obstacles x_{obj} (green) are available for all traffic participants. We assume that all dynamic obstacles (red) have full knowledge of the configuration of static obstacles x_{obj}. Meanwhile, the ego vehicle (blue) can only receive partial information of static obstacles due to the equipped sensor and can be described by $sensor\left(x_{obj}\right)$.

Furthermore, to model the vehicle's interaction, these pieces of information of another dynamic obstacles are exchanged among themselves. The bicycle reciprocal collision avoidance method (B-ORCA) is utilized due to the fact that it considers the non-

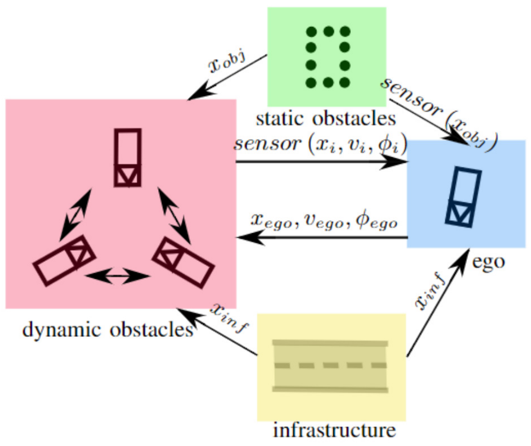

Figure 9: We test the modified Hybrid A* in both environments in the presence of static and dynamic obstacles. The infrastructure is activated in addition for the semistructure scenarios.

holonomic conditions of vehicles based on optimal reciprocal collision avoidance (ORCA) (Alonso-Mora et al., 2012). The vehicles interact properly to the other vehicles which act differently, which is useful in our case since one of the traffic participants, here an ego vehicle, is equipped with the modified Hybrid A* to guide its behaviour. The responsibility factor of each dynamic obstacle to another one or to the ego vehicle is set to 0.5 since each vehicle tries to avoid each other or taking half of the responsibilities. The static obstacles, on the other hand, cannot move itself and thus have no responsibility to avoid other traffic participants. The responsibility factor is then set to 0, while all vehicles take full responsibility (the factor equals 1).

Here as well, the vehicles have the full knowledge of another dynamic obstacles' configuration x_i, v_i, ϕ_i and the ego's position, velocity and orientation, $x_{ego}, v_{ego}, \phi_{ego}$. Meanwhile, due to the equipped sensor, the ego vehicle can only collect partial information described by $sensor\,(x_i, v_i, \phi_i)$.

On top of that, The infrastructure part (yellow) is deactivated for the second case, i.e. the unstructured environment. In this use case, the infrastructure can be constructed as a road defined by two parallel pointwised static obstacles with one side of length zero. Since the infrastructure is previously known, its full information x_{inf} can be sent to all vehicles in the last use case.

For both cases, the robustness of the modified Hybrid A* is evaluated by 100 random simulations each. In the case of semistructured environment, we additionally define road boundaries with a lane width of 2 m each. Static obstacles are placed within the road, while the vehicles try to follow their assigned lane.

3.2 Simulation Results

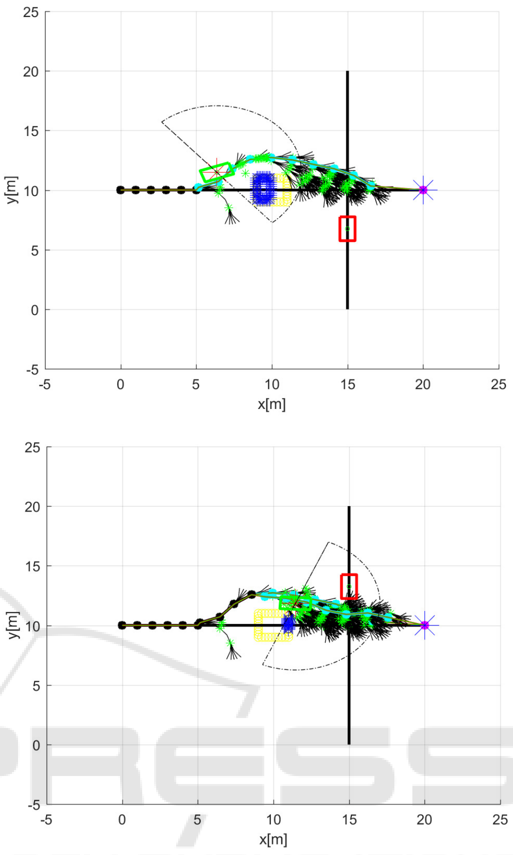

Figure 10: Application of the modified Hybrid A* as a local planer in an unstructured environment: In the first part of the figure, the ego vehicle plans the new path to avoid the collision with the static obstacle. As soon as the dynamic obstacle is detected, a new path is calculated as well, as in the second part of the figure.

In Figure 10, one of the testing scenarios for an unstructured environment is evaluated. The green ego vehicle moves to the right, while the red obstacle vehicle progresses upwards. Their predefined straight paths are marked as black lines. As explained earlier for the unstructured case, the start point in red star falls upon the current position of the ego vehicle, while the goal point in blue star corresponds to the last waypoint of the track. A yellow rectangular static obstacle is placed in the middle of the path. The blue points are result of the L-Shaped fitting algorithm and display a current potential danger on track for the ego vehicle. The black branches of the tree illustrate all of the possible route, while the green leaves demonstrate the potential nodes within the closed list of the Hybrid A* algorithm. At first, the ego vehicle is shifted to the left side of the predefined path to see whether the lateral displacement is later compensated by the Stanley

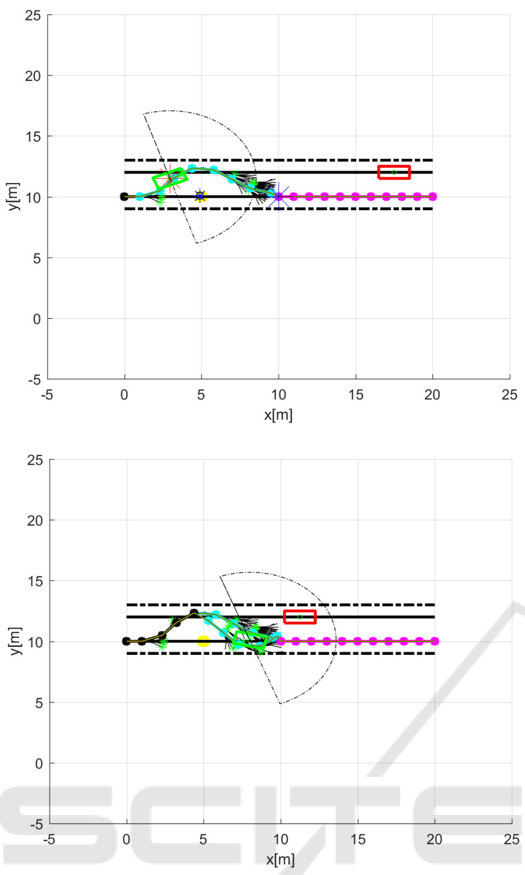

Figure 11: Application of the modified Hybrid A* as a local planer in an semistructured environment. The scenario is divided into two parts as in Figure 10. The road boundaries are, however, previously known and constrain the free space for the tree expansion.

controller. When the static obstacle is located within the sensor range, the detected points are checked if they could be seen dangerous to the vehicle by calculating their distance to the tracked path. When the obstacle hinders the movement of the ego vehicle, the points detected by the sensor are feeded into the object detection path to calculate the current obstacle configuration using L-Shape fitting. A new path is calculated using the modified Hybrid A* so that the vehicle can avoid the currently detected obstacle. The tree is expanded from start to goal points lying in the original path. The new path is then followed using Stanley controller while trying to keep its desired velocity using an underlying PID controller. Once the other vehicle is detected, the ego vehicle slows down as in Figure below. However, another new path is calculated so that the ego vehicle does not have to wait until the dynamic obstacle finally leaves the dangerous zone. Afterwards, the ego vehicle speeds up to its preferred speed and reach the original black path.

Figure 11 illustrates a collision avoidance for the semistructured case by adding boundaries of the road, marked as black dotted lines. Other illustrating elements are the same as in Figure 10. Furthermore, since the search space is limited, the procedure of finding the path until the fixed end point in large blue star is facilitated in comparison to the previous case. The lila waypoints illustrates the path joining the end point of the planning and the end of the street. An obstacle vehicle in this case drives in another lane in opposite direction. It is clear that the tree only expands inside of these lane boundaries. As soon as the ego vehicle detects the moving obstacle, it replans the path and moves to the original lane as we can see from Figure 11.

We test for robustness by randomly placing vehicles and obstacles in both unstructured and semistructured environments. As a result, the ego vehicle can safely avoid the collision with all the obstacles in all of the simulation scenarios within the required time for real-time capability of 1 s. This time limit is the maximal time to be accepted in the real-time capability in the guidance of autonomous vehicles and can be reduced by improving the implementated code or moving to the more efficient platform such as Robot Operating System(ROS). Furthermore, the inertia of the vehicle is omitted which plays an essential role in the dynamics and will be considered in the future work.

4 CONCLUSIONS

We investigate the use of the modified Hybrid A* on the robustness in unstructured and semistructured environments with a more realistic interaction of the traffic participants. The approach, combined with the model-based L-Shape fitting object detection, the Stanley controller for lateral and PID controller for longitudinal control, successfully finds the collision-free path in the simulation with the presence of both static and dynamic obstacles. The vehicle can robustly avoid collision within less than 1 s in the guidance of autonomous vehicles.

In future works, we are going to validate quantitatively, in both simulations and experiments with random configurations, the calculated path and to optimize the weights of the cost function of Hybrid A* for a better result of the path planned in both unstructured and semistructured environments. The algorithm will be tested in both Simulink-based model using xPC Target and be transferred to a C++-based ROS environment in the test vehicle for a real-time capability comparison. A more robust object detection algorithm can also be used for a more precise localisation of the obstacles for a more robust path planning in the real application.

ACKNOWLEDGEMENTS

The first author would like to express attitude towards National Science and Technology Development Agency (NSTDA) for the financial support during the doctoral study in Germany as well as towards the second author for the supervision in the doctoral study.

REFERENCES

Alonso-Mora, J., Breitenmoser, A., Beardsley, P., and Siegwart, R. (2012). Reciprocal collision avoidance for multiple car-like robots. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 360–366.

Dolgov, D., Thrun, S., Montemerlo, M., and Diebel, J. (2008). Practical search techniques in path planning for autonomous driving. In *Proceedings of the First International Symposium on Search Techniques in Artificial Intelligence and Robotics (STAIR-08)*.

Dolgov, D., Thrun, S., Montemerlo, M., and Diebel, J. (2010). Path planning for autonomous vehicles in unknown semi-structured environments. *The International Journal of Robotics Research*, 29(5):485–501.

Emerson, E. (2019). Crowd pathfinding and steering using flow field tiles. *Game AI Pro 360*.

González, D., Pérez, J., Milanés, V., and Nashashibi, F. (2016). A review of motion planning techniques for automated vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 17(4):1135–1145.

Guirguis, S. E., Gergis, M., Elias, C. M., Shehata, O. M., and Abdennadher, S. (2019). Ros-based model predictive trajectory tracking control architecture using lidar-based mapping and hybrid a* planning. *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 2750–2756.

Kurzer, K. (2016). *Path Planning in Unstructured Environments: A Real-time Hybrid A* Implementation for Fast and Deterministic Path Generation for the KTH Research Concept Vehicle*. PhD thesis.

Nemec, D., Gregor, M., Bubeníková, E., Hruboš, M., and Pirník, R. (2019). Improving the hybrid a* method for a non-holonomic wheeled robot. *International Journal of Advanced Robotic Systems*, 16(1):1729881419826857.

Nordeus, E. (2015). Explaining the hybrid a star pathfinding algorithm for selfdriving cars. https://blog.habrador.com/2015/11/explaining-hybrid-star-pathfinding.html.

Paden, B., Cáp, M., Yong, S. Z., Yershov, D. S., and Frazzoli, E. (2016). A survey of motion planning and control techniques for self-driving urban vehicles. *CoRR*, abs/1604.07446.

Patle, B., Babu L, G., Pandey, A., Parhi, D., and Jagadeesh, A. (2019). A review: On path planning strategies for navigation of mobile robot. *Defence Technology*, 15(4):582–606.

Petereit, J., Emter, T., Frey, C. W., Kopfstedt, T., and Beutel, A. (2012). Application of hybrid a* to an autonomous mobile robot for path planning in unstructured outdoor environments. In *ROBOTIK 2012; 7th German Conference on Robotics*, pages 1–6.

Reiter, M., Wehr, M., Sehr, F., Trzuskowsky, A., Taborsky, R., and Abel, D. (2017). The irt-buggy – vehicle platform for research and education. *IFAC-PapersOnLine*, 50(1):12588–12595. 20th IFAC World Congress.

Sedighi, S., Nguyen, D.-V., and Kuhnert, K.-D. (2019). Guided hybrid a-star path planning algorithm for valet parking applications. In *2019 5th International Conference on Control, Automation and Robotics (ICCAR)*, pages 570–575.

Shewchuk, J. R. (1994). An introduction to the conjugate gradient method without the agonizing pain. Technical report, USA.

Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., Lau, K., Oakley, C., Palatucci, M., Pratt, V., Stang, P., Strohband, S., Dupont, C., Jendrossek, L.-E., Koelen, C., and Mahoney, P. (2006). Stanley: The robot that won the darpa grand challenge. *J. Field Robotics*, 23:661–692.

Udacity (2015). Intro to artificial intelligence. https://www.udacity.com/course/intro-to-artificial-intelligence–cs271.

Zhang, X., Xu, W., Dong, C., and Dolan, J. M. (2017). Efficient l-shape fitting for vehicle detection using laser scanners. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 54–59.