S. Solmaz, R. Muminovic, A. Civgin and G. Stettinger, "Development, Analysis, and Real-life Benchmarking of RRT-based Path Planning Algorithms for Automated Valet Parking," 2021 IEEE International Intelligent Transportation Systems Conference (ITSC), 2021, pp. 621-628, doi: 10.1109/ITSC48978.2021.9564413.

Development, Analysis, and Real-life Benchmarking of RRT-based Path Planning Algorithms for Automated Valet Parking[§]

Selim Solmaz[*,†], Rijad Muminovic[‡], Amar Civgin[†], Georg Stettinger[†]

***Abstract*— One of the major application areas of highly automated vehicles is the problem of Automated Valet Parking (AVP). In this work, we analyze solutions and compare performances of RRT (rapidly exploring random tree) based approaches in the context of the AVP problem, which can also be applied in a more general low-speed autonomy context. We present comparison results using both simulation and real-life experiments on a representative parking use case. The results indicate better suitability of RRTx and RRV for utilization in typical AVP scenarios. The main contributions of this work lie in real-life experimental validation and comparisons of RRT approaches for use in low-speed autonomy.**

I. INTRODUCTION

The automotive industry is currently undergoing a paradigm shift regarding its plans due to the advancement of automated driving technologies. This is further amplified by ever-increasing attention from the industry, academia, and the end-users alike. This rapid change gives a unique opportunity for innovative companies to pave the way into the automotive industry. This puts strong pressure on existing OEMs to change their business models, which typically lag in such innovations. Nevertheless, the automated driving technology with varying autonomy levels ranging from SAE level 1-3 is already present today. Here an increasing SAE index indicates greater independence from a human driver, while level 5 corresponds to full autonomy [1].

While we are far from fully autonomous driving in the near future, it is well known that most OEMs are working heavily on SAE level 3 and level 4 operations. The first application of these, starting from automated vehicles with SAE level 2 autonomy, is the automated valet parking (AVP) as well as smart parking and retrieval operations. The main reasons for the AVP to be perhaps one of the most attractive use cases are: 1) the vehicle dynamics are mostly negligible and motion is predominantly kinematic; 2) surrounding environment does not involve rapid variations; 3) parking and vehicle retrieval is usually perceived as a burden for the driver and passengers, and therefore AVP solutions are likely to have more user acceptance than general automated operations.

The recent literature surveys in the field of AVP and smart parking subjects include the compilation of Barriga et al. [2]. They focus on available approaches for the general problem of smart parking and the utilized sensors, without focusing on the path planning problem and algorithms. Banzahf et al. [3] on the other hand provides a more holistic overview of the problem and particularly classify the planning problem into the three stages of global, local, parking abstraction levels. The global planning problem is described typically on a static road network (i.e., graph) and the available approaches include graph-search and random sampling techniques. On the other hand, local planning typically involves reactions to dynamic environment elements. This utilizes graph-search, sampling as well as optimization techniques. Finally, the planning for parking maneuvers is handled using geometric and graph-search approaches. Similar three-staged planning was utilized in the general context of robotic motion planning [4]. Moreover, the surveys by Paden et al. [5], and Gonzalez et al. [6], and the references therein also provide a detailed overview on the path planning for automated parking applications.

The algorithms utilized in literature for path planning can be roughly categorized into geometric, deterministic sampling, random sampling, and optimization-based techniques.

The geometric path planning approaches include the work of Hsu et al. [7] which describes a complete solution (including sensors, free spot detection,...etc), utilizing a simple geometric approach making use of two circles for (parallel) parking maneuver. Also, Vorobieva et al. [8], [9] introduced a geometric approach for final slot entrance (e.g., circular arcs and clothoids) with applications in a real vehicle. Moreover, Qin et al. [10] suggested Bezier curves approximate circular arcs path in the form of cubic curves for parallel, quadratic curves for vertical parking maneuvers.

In sampling-based techniques, Chen et al. [11] introduced a sampling approach and benchmarked it against heuristic search with Hybrid A* and RRT in simulation. Similarly, Schwesinger et al. [12] described a complete parking solution in the scope of the V-Charge project (incl. perception, road modeling, local planner, and obstacle avoidance logic). The parking planner utilized three stages including: 1) Reeds-Shepp planner; 2) deterministic three motion planner, 3) Hybrid A* to improve flexibility.

In another strand of sampling-based parking algorithm research, RRT-type approaches are preferred. To the best of our knowledge, Han et al. [13] made the first RRT implementation for autonomous parking with the motivation to solve all parking cases with a single ”unified” algorithm. The simulation studies they performed provided promising results. Afterward, Klemm et al. [14] introduced the RRT*-connect as a bi-directional RRT method. This was to relieve a typical shortcoming of parking planners in narrow passages and maze-like structures. It showed significant performance

* Corresponding Author: selim.solmaz@v2c2.at

† Virtual Vehicle Research GmbH, Electrics Electronics and Software Department, Control Systems Group, Inffeldgasse 21a, 8010, Graz, Austria.

‡ Former master student at Virtual Vehicle Research GmbH.

§ This work was supported by EU-ECSEL Joint Undertaking Program.

gains for autonomous parking applications with real-world results. Kiss et al. [15] described a global planner implementation of RRT in a similar fashion to bidirectional RRT Connect. They also utilized sub-optimal continuous-curvature local planners (based on clothoids). These planners offer fewer segments than an optimal solution for more human-like driving behavior. The authors conducted extensive comparisons in simulation with RRT connect. Similarly, Zeng et al. [16] utilized bi-directional RRT for parking, allowing extensive maneuvers in simulation. Finally, Vlasak et al. [17] utilized the RRT* for parking with the motivation to have better performance in unexpected environments and when more than a simple maneuver is required.

In optimization-based techniques, a recent paper by Chen et al. [18] provides a good overview of related work and describes a "real-time" MPC method, demonstrated in simulation for automated parking.

In summary, we can list the main motivations to use RRT-type sampling-based approaches for the automated parking path planning problem as:

- To provide an algorithm for all parking use-cases [13]
- To avoid pitfalls (also for simple RRT methods) due to narrow passages or maze-like arranged obstacles [14]
- Operation in unexpected environments, or if more than a simple maneuver is required [17]

Based on the above motivations, it was decided within the scope of the project PRYSTINE[1] to benchmark available RRT methods with representative forward and backward parking use-cases. In the current paper, we report the results of this analysis, which is presented as follows. We first give the motivation for the automated parking problem along with an overview of the main approaches in the literature. We then link this to the RRT algorithm and its derivatives for the described problem. Finally, simulation and real-life experimental setup and their implementation details are described with concluding comparisons of the results.

II. Path Planing for Parking

The goal of path planning is to find feasible trajectories that connect a starting point A with destination point B in a given static or dynamic environment. With such a constructed path, subsequent blocks can perform path tracking and control the vehicle that such a trajectory is followed and the vehicle is moved to its destination. This is especially useful for the task of automated vehicle parking and retrieval.

The trajectory planner performs its function on a given static environment and ego vehicle state, with a given start and desired goal state for the ego vehicle. It assumes the availability of an occupancy map in the global frame inside which it performs the planning algorithm. A list of way-points is then produced that describe the two-dimensional x and y coordinates and orientation θ of the ego vehicle as it navigates through the environment. On top of these three parameters, a velocity profile for the vehicle is calculated as a reference for tracking along these way-points.

[1] http:\www.prystine.eu.

In more detail, the path planning assumes the availability of the following parameters:

- All physical dimensions of the ego-vehicle and its surrounding environment, including the minimal turning radius.
- Environment occupancy map represented by a bitmap, in Birds-eye-view perspective, with a relatively precise resolution (more than 5 pixels per meter), and all obstacles represented on this map. This map is a 2D matrix, and it's flat on the ground since planning is only done for x,y coordinates.
- Start and goal pose (i.e., position and orientation) given in the map's reference coordinate frame.
- Other data needed for transforming the given local coordinates on the bitmap to the corresponding global coordinates.

The output of the path planning module includes discretized vehicle states given over a feasible trajectory. That is, a full trajectory of control inputs is not generated. Low-level control tasks convert these generated way-points into trajectories and then further into specific vehicle control commands. The given feasible path has no sudden changes in the generated vehicle state values, and the two adjacent states are assumed to be achievable in short time intervals. Moreover, the feasible path satisfies the maximal turning radius condition and the kinematic model used for planning. In the case when the angle increments along the adjacent states are above a certain threshold, indicating a sharp turn, such maneuvers can be conducted by stopping the vehicle and turning the wheels.

III. A brief overview of the used RRT algorithms

In this section, we first introduce the generic algorithm structure of the RRT algorithms. Then we describe the specific RRT algorithms considered in the analysis by focusing on the differences of each method. The term RRT stems from "Rapidly-exploring Random Tree", which is an algorithm designed to efficiently search non-convex, high-dimensional spaces by randomly building a space-filling tree. The tree is constructed incrementally from samples drawn randomly from the search space and is inherently biased to grow towards large unexplored areas of the problem.

The basic algorithmic structure of the RRT-based algorithms is shown in Figure 1 [19]. Various RRT algorithm types found in the literature differ in ways of implementing the basic RRT steps. These steps are: initialization, new random state generation, finding new configurations, and adding vertices and edges, as described in the basic RRT algorithm steps in Figure 1.

A. RRT:

Standard RRT is often used when feasible and quick solutions are needed for traveling between two points over a planar space avoiding obstacles. It was developed originally in the context of mobile robotics [20]. Points are randomly generated and connected to the closest available node. Each

Input: Initial configuration q_{init}, number of vertices in the tree K, incremental distance Δq

Output: RRT graph G

$G.init(q_{init})$

for $k = 1$ ***to*** K ***do***

$q_{rand} \leftarrow$ randomConfiguration()

$q_{near} \leftarrow$ nearestVertex(G, q_{rand})

$q_{new} \leftarrow$ newConfiguration$(q_{near}, q_{rand}, \Delta q)$

G. addVertex(q_{new})

G. addEdge(q_{near}, q_{new})

return G

Fig. 1. RRT Algorithm Structure.

time a vertex is created, a check must be made that the vertex lies outside an obstacle.

RRT is especially good in high-dimensional spaces [20]. In each iteration, this algorithm constructs a new valid state configuration, finds the nearest state from it, and tries to connect it to the sampled one. If the connection is possible – without collisions, feasible, and satisfies limits on the connection length – it is added to the tree. This iteration is repeated until the goal state is added to the tree.

One of the most significant problems in path planning with RRT is the problem of narrow passages in the environment. The probability of randomly sampling desired states is much lower than in open spaces, and the probability of successfully connecting the sampled state to the tree is even lower. For example, if the vehicle enters a narrow passage with the wrong orientation, it will be hard to connect it to the goal state. This is due to the fact that the maneuver needed to turn the vehicle around would be impossible inside this narrow passage.

B. RRT Connect:

This method somewhat simplifies the problems of connecting the final state to the tree if it is inside a narrow passage or separated from the start state by a narrow passage. The main idea is to use two trees that are constructed in parallel, one stemming from the goal state and the other from the start state. In each iteration, a node is added to one tree, after which an attempt is made to connect it to the other tree. If the connection is successful, the path is found, otherwise this process is repeated for the other tree, where the kinematics is appropriately mirrored to allow for connecting the states "in reverse".

C. RRT:*

RRT* or RRT Star is an optimal algorithm based on random sampling. The main idea is to perform rewiring during the tree construction to find shorter paths from the starting node to other nodes. After it finds a valid solution, it continues adding new nodes to the tree to try and further shorten the path. Some additional problems this approach brings are:

- Rewiring requires the ability to connect two states with different orientations, with a change in vehicle travel direction when connecting them.
- The algorithm's execution can last much longer while it is trying to improve the paths.
- It is necessary to track the distance of all the nodes to the root node; this is something that becomes progressively harder in later stages.

RRT* has properties of an optimal algorithm, but its convergence rate is slow. Based on the experience gained in this study, frequent rewiring was observed, which leads to the slow propagation of this search algorithm. There are some recent improvements in RRT algorithms, such as the Informed RRT* [21], and RRT*-connect [14]. However, these approaches were not included in this analysis due to the complex metrics they require, conflicting with the real-time performance requirements.

D. RRTx:

RRTx is a more recent and advanced method used in path planning, which has better convergence properties compared to RRT* [22]. Its better organization of the created graph and some additional information it keeps allows performing re-planning when some changes occur in the environment. Therefore, this method is often used in dynamic scenarios. Since this is an optimal algorithm, it will also perform best in terms of path length and reverse-path length metrics. One of the improvements done to this algorithm is an idea from the Quick-RRT* algorithm [23], using the triangular inequality to shorten generated paths.

E. RRV:

Rapidly exploring random vines [24] is an algorithm, which is made to handle planning inside and around narrow passages. It does this by sampling new states around the ones that are already in the RRT tree, after which it performs PCA (Principal Component Analysis) to reason about the surroundings of the node. With this information, proceeding with planning is much easier. This sampling is illustrated in Figure 2.

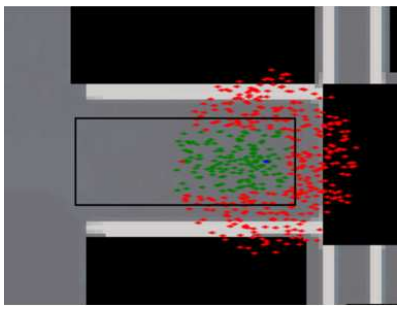

Fig. 2. Sampling around the vehicle in tight spaces. Red and green colors indicate the random samples with and without collision probability, respectively.

In the scope of the current study, the state is represented by an angle, which is not a linear variable, but a value in the

interval $(-\pi, \pi]$, where the value of π is at a distance of 0 from the value of $-\pi$. This fact can invalidate the assumptions for using the PCA analysis directly on state variables. For this reason, some adjustments to this algorithm, where projections from the PCA eigenvectors are used to develop control primitives that follow them.

IV. Basic RRT algorithm components

Some changes were made in the components of the original RRT algorithm and its variations to be able to use them in path planning for vehicles. Specifically:

- Collision checking needs to be done for a complex object, not a material point, so a different collision checking approach is used, described in the "Collision checking" subsection.
- Finding the closest neighbor in SE(2) space is not a simple task, especially with all the motion conditions. This part is described in the "Distance metric" subsection.
- Creating a valid path. In this approach, we use the Reeds-Shepp curves, as discussed in the "Kinematically valid and optimal paths" subsection.
- Connecting the final state to the tree.

In the following subsections, these changes are described in more detail.

A. Vehicle models:

Using a sufficiently complex dynamical model for vehicles would require a wide variety of parameters and would further complicate the planning process. Because of these reasons, we ignore the vehicle dynamics, which is in line with the occurring slow velocities used in parking [5]. The bicycle model is widespread in its use, and the flaws of this model compared to more complex models are well studied [25]. The main assumption is that the vehicle can be represented with two wheels in the middle of the wheel axis, as illustrated in Figure 3. The (x, y, ψ) tuple represents the vehicle configuration, the turn angle δ is the control angle of front wheels, and L represents the distance between axle centers. The turn radius is denoted by R. The following relation holds between the turn radius and the turn angle

$$R = \frac{L}{tan\delta} \tag{1}$$

The differential model of the vehicle with a given velocity of the rear axle center v is given by

$$\begin{aligned} \dot{x} &= vcos\psi \\ \dot{y} &= vsin\psi \\ \dot{\psi} &= \frac{v}{L}tan\delta \end{aligned} \tag{2}$$

B. Occupancy map:

Representing the environment using a bitmap for vehicle movement planning would require computationally intensive collision checks even when using only 16 pixels to represent the vehicle. Instead, we use a different approach. The given environment map is mapped to an occupancy map with pixels that represents rectangles of dimension $0.25m \times 0.25m$, such that if at least one pixel in the original environment map is occupied, the pixel it maps to will be considered occupied as well.

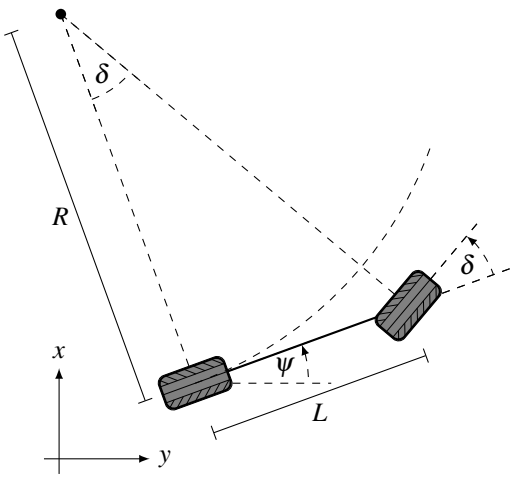

Fig. 3. Kinematic vehicle model.

C. Collision checking:

To further speed up collision checks and avoid sampling the whole vehicle rectangle, we instead model the car with several circles, which is an approach often used in the literature [26]. We model the vehicle with five circles of equal radii R. Instead of checking whether the area under these five circles is occupied in the occupancy map, we instead dilate the occupancy map with a circular kernel of radius r. This allows us to check just five pixels in the dilated map, which is a significant improvement in run-time. This effectively increases the number of occupied states, but in our tests, it proved to be negligible since the algorithm still managed to find valid paths to given goal states.

D. Space sampling:

The space sampling is performed inside the whole available space for $x \in [0, max\_width]$ and $y \in [0, max\_height]$ in meters, and angle $\psi \in [-\pi, \pi]$. From these intervals, all three values are sampled independently and uniformly. Also, for every K-th sample, the final state is "randomly" sampled to try and connect it to the tree.

E. Kinematically valid and optimal paths:

For connecting two free states, we use the Reeds-Shepp curves [27], assuming that the vehicle moves with an arbitrary velocity and with a part-constant, bounded steering angle. Reeds-Shepp curves are an extension of the Dubins curves [28], which solve the problem of the shortest path between two states (position and orientation), subject to a bounded curvature. Reeds-Shepp curves extend this solution enabling backward driving, which is essential especially in parking use cases and obviously yields shorter paths than the corresponding Dubins curves. Note that due to the limited steering dynamics, part-constant, bounded steering angle profiles can be executed exactly by stopping the vehicle before turning the steering wheel. However, the assumption

of slowly-varying part-constant steering angle profiles gives a sufficient approximation of valid vehicle paths at low vehicle speeds.

F. Distance metric:

Given two states A and B, with coordinates (x_s, y_s, ψ_s) where $s \in \{A, B\}$, we need to find the distance between these two states. Using an Euclidian measure would not be a good approximation since the system is non-holonomic. The Reeds-Shepp curve distance metric would be a satisfactory measure but since it is computationally complex to calculate it would be expensive. Instead, we use a mapping from the 3D space to a 4D space given by the relation

$$f(x_s, y_s, \psi_s) = (x_s, y_s, \alpha sin\psi_s, \alpha cos\psi_s) \quad (3)$$

where the Euclidean distance $d_1(A,B)$ approximates the following measure

$$d_1(A,B) \approx \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2 + \alpha d_\psi(\psi_A, \psi_B)^2}. \quad (4)$$

Here d_ψ is the arc distance between two angles.

G. Nearest neighbor search:

It is costly to exhaustively search through all the points and calculate the distance from them. This is due to the fact that the tree is expanded with new nodes during the execution of the algorithms. For efficient nearest neighbor search, we instead use the k-d tree [29] spatial structure, coupled with the Euclidean metric we described above. This enables fast queries of k nearest neighbors for each new point.

V. Test scenarios and key performance indicators

The scenario is shown in Figure 4, and its main purpose is to model a typical parking situation in a static environment setting. We note that for the purpose of the current analysis, the dynamic environment is neglected. The static environment in this context contains 10 parking spots as seen in Figure 4, each one with dimensions 3×5 meters and a distance of 8.2 meters across the two rows. Parking is tested on parking spots 1, 4, 8, and 10. The start and the destination coordinates of the vehicle (where the reference point is the CG of the vehicle) and the goal parking spots (where the respective reference point is the geometric center of the parking spot) are given in Table I.

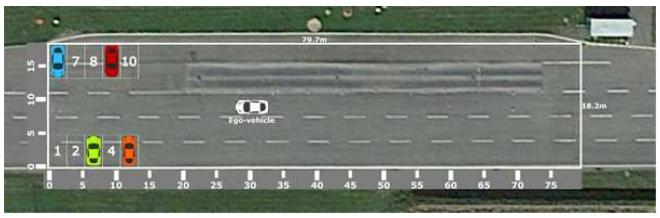

Fig. 4. Parking scenario description.

TABLE I
Vehicle starting and goal positions for the test scenario.

Vehicle state	Coordinates (x, y) in [m]
Starting state	(30, 8.5)
Parking spot 1	(1.5, 2.5)
Parking spot 4	(10.5, 2.5)
Parking spot 8	(7.5, 15.7)
Parking spot 10	(13.5, 15.7)

A. KPIs for path planning for parking

The KPIs are mainly chosen to analyze the trade-off between quality of results and computation time. In this respect, the KPIs are selected and grouped mainly to measure path quality and computation time. Specifically:

Path-Quality KPIs:

- Success rate
- Length of the planned path (in meters),
- Length of the planned path in reverse (in meters).

Computation-time KPIs:

- Execution time, for the path planning step (in seconds),
- Number of nodes in the tree,
- Number of collision checks performed.

Since the RRT algorithm is probabilistic, these results are reported with their mean and standard deviation value across 50 random runs of the algorithm. We limit the execution to no more than 30 seconds and report the mean and standard deviation calculated across successful runs only. Finally, we also report the success rate for each algorithm in the given scenario.

VI. Verification and Test results

A. Simulation verification

Here we describe the testing and evaluation of the introduced RRT-based trajectory planning algorithms utilizing a simulation environment. The environment simulation is based on the Carla Simulator [30] and the path planning algorithms were implemented as a module based on the Autoware.AI software stack [31].

RRT-based path planning for parking was evaluated using the described scenario, with separate simulations at parking spots 1, 4, 8, and 10. For the sake of brevity, only the results for the simulations for parking at spots 1 and 4 are reported here. The start and goal positions are defined in Table I. An example path planning result for RRTx is shown in Figure 5 for parking at the spot-10. The parking simulation results for various RRT-algorithms for parking at the spot-1 are summarised in Table II and the corresponding results for the spot-4 are summarised in Table III.

Note that in summary, the RRT-Connect, RRV, and RRTx planning algorithms had a success rate of 100% in finding a parking path in each of the 50 test runs. Furthermore, all of these algorithms additionally provide the output path within a relatively short execution time and within a reasonable deviation in path length. RRT and RRT*, however, were not able to find a valid path within the limited execution time of 30 seconds (see the KPIs in Section V-A). Therefore, they

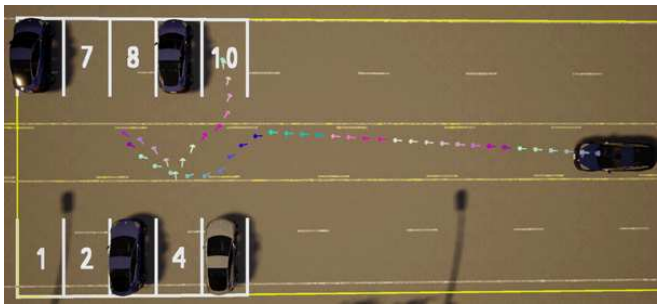

Fig. 5. Example simulation of a planned path for a single RRTx run.

TABLE II

SIMULATION-BASED KPI RESULTS FOR PARKING AT SPOT-1.

Algorithm	Success rate	Execution time (s)	Number of nodes (#)	Path length (m)	Reverse path length (m)	Collision checks (x1000)
RRT	5/50	3.61 ± 4.0	5148.00 ±5368.1	56.21 ±5.6	18.02 ± 14.1	276.66 ±254.7
RRT Connect	50/50	0.66 ± 0.4	5.68 ±2.2	63.66 ±14.3	28.05 ± 17.8	16.25 ± 8.8
RRT*	2/50	4.28 ± 0.0	5001.00 ±0.0	53.17 ±1.1	5.46 ± 0.8	286.57 ±4.0
RRV	50/50	2.05 ± 2.8	43.66 ±16.1	54.05 ±3.2	9.68 ± 9.8	209.48 ±288.2
RRTx	50/50	0.87 ± 0.4	26.78 ±15.1	56.30 ±4.7	12.77 ± 7.7	46.34 ± 22.8

show a dramatically low success rate. This can be attributed partially to the algorithm, but also that they don't start from the end state, which makes it harder to reach. RRV, RRTx and RRT-Connect, on the other hand, have the end state already in the starting trees.

Based on these initial simulation results, in general, RRT-based planners appear to be pretty appropriate algorithms for parking purposes. Furthermore, all the provided output paths are kinematically feasible, which makes this approach even more attractive for utilization in path planning for parking. This is because the vehicle dynamics at low speeds are not dominant, and the motion is predominantly constrained by the vehicle kinematics.

B. Real-world validation tests

In order to validate the developed RRT-based trajectory-planning algorithms, an automated driving demonstrator vehicle seen in Figure 6 was utilized. The vehicle is a stock Ford Fusion MY2018 test vehicle that is equipped with several additional sensors and computational hardware as well as custom software components. The basis hardware is the Dataspeed ADASKit [32]. It enables the basic steering, braking, and acceleration controls of the vehicle while also providing access to all the onboard sensor signals available on the CAN system of the car. Additionally, the vehicle has a dual-antenna Novatel Propak6 RTK-dGPS system, a Continental ARS408 long-range radar, a Mobileye 630 intelligent camera, a Blackfly S GIGE RGB camera, and an Ouster OS1-64 LiDAR sensor. The mounting positions of the perception sensors are depicted in Figure 6 where the radar is marked in green, cameras are in orange, and the LiDAR is marked in blue. For the implementation, the Autoware.AI software stack [33] based on the ROS1 middleware [34] is utilized to integrate the existing perception and control modules from Autoware with the developed path planning algorithms. The integrated software was running on an Ubuntu x86 PC mounted in the trunk of the vehicle.

TABLE III

SIMULATION-BASED KPI RESULTS FOR PARKING AT SPOT-4.

Algorithm	Success rate	Execution time (s)	Number of nodes (#)	Path length (m)	Reverse path length (m)	Collision checks (x1000)
RRT	27/50	1.28 ± 3.7	1576.63 ± 4436.1	44.46 ± 6.4	19.90 ± 13.3	84.99 ± 212.9
RRT Connect	50/50	0.01 ± 0.0	2.68 ± 1.0	48.66 ± 13.8	24.25 ± 12.5	1.53 ± 0.5
RRT*	20/50	4.20 ± 0.1	5001.00 ± 0.0	42.24 ± 4.6	6.55 ± 6.2	283.69 ± 7.4
RRV	50/50	0.06 ± 0.0	24.72 ± 11.1	46.82 ± 10.7	28.44 ± 19.1	7.07 ± 2.4
RRTx	50/50	0.17 ± 0.1	20.96 ± 11.6	41.47 ± 5.9	16.30 ± 8.4	18.76 ± 12.6

Fig. 6. Virtual Vehicle's automated driving demonstrator vehicle that was utilized during the real-world validation tests.

To evaluate path planning results for parking, a static occupancy map of the testing environment was created. Then the real-world scenarios were executed with the same scenario setup used in the simulation study. However, it must be noted that the dimensions in the simulation and the real-life validation tests are not exactly the same due to physical constraints and also possible measurement errors. It was also of interest to see how small differences in the environment and initial conditions can affect the planning results. Similar to the simulation study, the evaluation was done for parking spots 1, 4, 8, and 10. However, the results for parking spots 1 and 4 are reported in comparison to the simulation results above. An example path planning algorithm test video is available on Youtube[2], where a snapshot from it is seen Figure 7. Please note that the small tracking offset of the vehicle that is visible in the snapshot is caused by the low-level tracking controller based on the pure-pursuit control logic. The design of a new and optimal tracking controller is not in the scope of the current study. Therefore, the tracking module of the Autoware.AI software stack was utilized as a part of the implementation.

The real-life parking test results for the selected set of different RRT algorithms for parking at the spot-1 are

[2]https://www.youtube.com/watch?v=OV1AdaRoUd0&ab_channel=VIRTUALVEHICLE

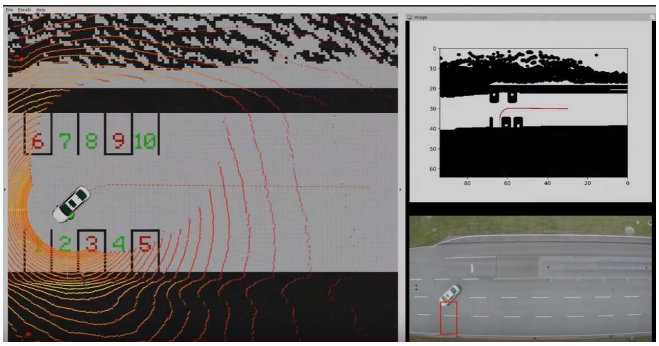

Fig. 7. A snapshot of the demonstration video available on Youtube, which is accessible from this direct link. The main visualization on the left shows the localization and the tracked path during the maneuver. On the top right, the dilated occupancy map (see Section IV.C) together with the planned path is shown. The bottom right is the birds-eye view of the vehicle during the experiment.

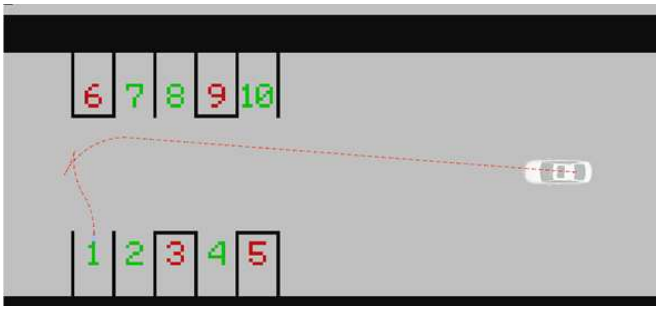

Fig. 8. Example visualisation of a planned path for a single RRT* test run. Green colored numbers indicate empty parking spots and red colored ones indicate occupied spots.

summarised in Table IV and the corresponding results for the spot-4 are summarised in Table V.

TABLE IV

REAL-LIFE VALIDATION KPI RESULTS FOR PARKING AT SPOT-1.

Algorithm	Success rate	Execution time (s)	Number of nodes (#)	Path length (m)	Reverse path length (m)	Collision checks (x1000)
RRT	37/50	0.12 ± 0.3	200.38 $\pm$465.0	71.62 $\pm$13.5	17.23 ± 18.8	6.19 ± 11.8
RRT Connect	48/50	0.85 ± 1.7	953.08 $\pm$1794.3	63.56 $\pm$8.9	6.89 ± 4.9	56.04 $\pm$77.9
RRT*	49/50	3.32 ± 0.1	2001.00 $\pm$0.0	59.48 $\pm$6.3	5.46 ± 4.3	338.76 $\pm$12.2
RRV	50/50	0.11 ± 0.1	40.60 $\pm$22.6	58.94 $\pm$9.7	24.64 ± 25.0	7.05 ± 3.4
RRTx	50/50	0.27 ± 0.2	39.24 $\pm$26.3	64.55 $\pm$10.4	13.83 ± 12.4	17.14 $\pm$13.7

Comparisons of specific KPIs between the simulation and validation tests for spots 1 and 4 are given in Table I. These indicate similar performance values in simulation and experiments without significant deviations. There is, however, an observable difference in the success rates of RRT and RRT* in the validation tests, which is higher in comparison to the simulation studies. These can be explained by small discrepancies in the environment, vehicle, and sensor models. The results show how basic RRT implementations are susceptible to hard constraints imposed by the environment, which makes a big impact on performance. However, this is not the case for RRT-Connect, RRV, and RRTx based

TABLE V

REAL-LIFE VALIDATION KPI RESULTS FOR PARKING AT SPOT-4.

Algorithm	Success rate	Execution time (s)	Number of nodes (#)	Path length (m)	Reverse path length (m)	Collision checks (x1000)
RRT	26/50	1.75 ± 1.9	2776.00 $\pm$2775.8	53.69 $\pm$16.4	19.99 ± 21.2	66.89 $\pm$61.5
RRT Connect	37/50	2.15 ± 2.0	2264.57 $\pm$2162.6	52.95 $\pm$13.6	9.23 ± 10.5	116.81 $\pm$90.5
RRT*	25/50	3.29 ± 0.1	2001.00 $\pm$0.0	48.36 $\pm$6.5	5.99 ± 4.0	329.09 $\pm$10.9
RRV	50/50	0.11 ± 0.1	42.58 $\pm$19.5	46.19 $\pm$7.4	10.02 ± 15.8	6.61 ± 3.0
RRTx	50/50	0.37 ± 0.4	10.08 ± 7.3	50.30 $\pm$7.5	12.77 ± 11.9	14.91 $\pm$13.3

planners, all of which have performed consistently well both in simulation and real-life validation tests. Based on the performance results given in Tables II III, IV, V and their comparisons, RRTx and RRV appear as the most viable planning algorithms for exploitation in further analysis.

VII. CONCLUSIONS AND OUTLOOK

In this paper, we described a specific implementation of RRT-based algorithms for path planning purposes in low-speed autonomy use-cases. This can potentially be utilized in automated valet parking and retrieval applications. The presented work was conducted in the scope of the EU-funded project PRYSTINE[1]. The implementation started with the development of RRT-type algorithms as part of a realistic environment simulation framework. Consequently, scenario-based real-life validation of the algorithms was performed in a use case involving a static environment. The contributions of the paper are: a benchmark of the respective algorithms in simulation as well as the correlations of the results to real-life validation tests. Both the simulation and real-life tests indicate high success rates for the RRV and RRTx at very low computational overhead, which is promising for utilization in further analysis. The utilized use-case scenarios were selected in the interest of achieving a simple basis of comparison of the selected algorithms. On the other hand, these scenarios were overly simplistic to assess their performance in a typical usage cycle of the provided solution. The analysis and further development of the current solution will evolve in the direction of more complicated and challenging use-cases. These can include dynamic environmental factors for potential exploitation in a robotic taxi use case as part of future work.

ACKNOWLEDGEMENT

This project has received funding from the European Union's ECSEL Joint Undertaking, which funded the PRYSTINE project under grant agreement number 783190. The publication was written at Virtual Vehicle Research GmbH in Graz and partially funded within the COMET K2 Competence Centers for Excellent Technologies from the Austrian Federal Ministry for Climate Action (BMK), the Austrian Federal Ministry for Digital and Economic Affairs (BMDW), the Province of Styria (Dept. 12) and the Styrian Business Promotion Agency (SFG). The Austrian Research Promotion Agency (FFG) has been authorized for the program management. The authors extend their thanks to all the

funding organizations and associated partners as well as the technical team of the Prystine Project, namely: Elvir Crncevic, Muhamed Kuric, Eldar Kurtic, Haris Sikic, and Kenan Softic, whose efforts greatly contributed to the realization of this work. Furthermore, they would like to thank Prof. Bakir Lacevic (University of Sarajevo) for the supervision of the related Master's thesis of Rijad Muminovic, as well as Michael Stolz and Johannes Rumetshofer for their support in the finalization of this publication.

REFERENCES

[1] SAE-J3016, "Surface Vehicle Recommended Practice–(R) Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles," Society of Automotive Engineers, USA, Standard SAE J3016:JUN2018, 2018.

[2] J. J. Barriga, J. Sulca, J. L. León, A. Ulloa, D. Portero, R. Andrade, and S. G. Yoo, "Smart parking: A literature review from the technological perspective," *Applied Sciences*, vol. 9, no. 21, 2019. [Online]. Available: https://www.mdpi.com/2076-3417/9/21/4569

[3] H. Banzhaf, D. Nienhüser, S. Knoop, and J. M. Zöllner, "The future of parking: A survey on automated valet parking with an outlook on high density parking," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, 2017, pp. 1827–1834.

[4] U. Schwesinger, R. Siegwart, and P. Furgale, "Fast collision detection through bounding volume hierarchies in workspace-time space for sampling-based motion planners," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 63–68.

[5] B. Paden, M. Cap, S. Z. Yong, D. Yershov, and E. Frazzoli, "A Survey of Motion Planning and Control Techniques for Self-Driving Urban Vehicles," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 1, pp. 33–55, 2016.

[6] D. González, J. Pérez, V. Milanés, and F. Nashashibi, "A review of motion planning techniques for automated vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 4, pp. 1135–1145, 2016.

[7] Tsung-hua Hsu, Jing-Fu Liu, Pen-Ning Yu, Wang-Shuan Lee, and Jia-Sing Hsu, "Development of an automatic parking system for vehicle," in *2008 IEEE Vehicle Power and Propulsion Conference*, 2008, pp. 1–6.

[8] H. Vorobieva, S. Glaser, N. Minoiu-Enache, and S. Mammar, "Automatic parallel parking with geometric continuous-curvature path planning," in *2014 IEEE Intelligent Vehicles Symposium Proceedings*, 2014, pp. 465–471.

[9] ——, "Automatic parallel parking in tiny spots: Path planning and control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 1, pp. 396–410, 2015.

[10] Y. Qin, F. Liu, and P. Wang, "A feasible parking algorithm in form of path planning and following," in *Proceedings of the 2019 International Conference on Robotics, Intelligent Control and Artificial Intelligence*, ser. RICAI 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 6–11. [Online]. Available: https://doi.org/10.1145/3366194.3366196

[11] C. Chen, M. Rickert, and A. Knoll, "Path planning with orientation-aware space exploration guided heuristic search for autonomous parking and maneuvering," in *2015 IEEE Intelligent Vehicles Symposium (IV)*, 2015, pp. 1148–1153.

[12] U. Schwesinger, M. Bürki, J. Timpner, S. Rottmann, L. Wolf, L. M. Paz, H. Grimmett, I. Posner, P. Newman, C. Häne, L. Heng, G. H. Lee, T. Sattler, M. Pollefeys, M. Allodi, F. Valenti, K. Mimura, B. Goebelsmann, W. Derendarz, P. Mühlfellner, S. Wonneberger, R. Waldmann, S. Grysczyk, C. Last, S. Brüning, S. Horstmann, M. Bartholomäus, C. Brummer, M. Stellmacher, F. Pucks, M. Nicklas, and R. Siegwart, "Automated valet parking and charging for e-mobility," in *2016 IEEE Intelligent Vehicles Symposium (IV)*, 2016, pp. 157–164.

[13] L. Han, Q. H. Do, and S. Mita, "Unified path planner for parking an autonomous vehicle based on rrt," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 5622–5627.

[14] S. Klemm, J. Oberlander, A. Hermann, A. Roennau, T. Schamm, J. M. Zollner, and R. Dillmann, "RRT*-Connect: Faster, asymptotically optimal motion planning," *2015 IEEE International Conference on Robotics and Biomimetics, IEEE-ROBIO 2015*, pp. 1670–1677, 2015.

[15] D. Kiss and G. Tevesz, "Autonomous path planning for road vehicles in narrow environments: An efficient continuous curvature approach," *Journal of Advanced Transportation*, vol. 2017, no. 2521638, pp. 1–27, 2017.

[16] K. Zheng and S. Liu, "Rrt based path planning for autonomous parking of vehicle," in *2018 IEEE 7th Data Driven Control and Learning Systems Conference (DDCLS)*, 2018, pp. 627–632.

[17] J. Vlasak, M. Sojka, and Z. Hanzálek, "Accelerated rrt* and its evaluation on autonomous parking," *Proceedings of the 5th International Conference on Vehicle Technology and Intelligent Transport Systems*, 2019. [Online]. Available: http://dx.doi.org/10.5220/0007679500860094

[18] C. Chen, B. Wu, L. Xuan, J. Chen, T. Wang, and L. Qian, "A trajectory planning method for autonomous valet parking via solving an optimal control problem," *Sensors*, vol. 20, no. 22, 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/22/6435

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

[20] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, vol. 1, 1999, pp. 473–479 vol.1.

[21] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed rrt*: Optimal incremental path planning focused through an admissible ellipsoidal heuristic," *CoRR*, vol. abs/1404.2334, 2014. [Online]. Available: http://arxiv.org/abs/1404.2334

[22] M. Otte and E. Frazzoli, "Rrtx: Asymptotically optimal single-query sampling-based motion planning with quick replanning," *The International Journal of Robotics Research*, vol. 35, pp. 797 – 822, 2016.

[23] I.-B. Jeong, S.-J. Lee, and J.-H. Kim, "Quick-rrt*: Triangular inequality-based implementation of rrt* with improved initial solution and convergence rate," *Expert Systems with Applications*, vol. 123, pp. 82–90, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417419300326

[24] A. Tahirovic and M. Ferizbegovic, "Rapidly-exploring random vines (rrv) for motion planning in configuration spaces with narrow passages," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 7055–7062.

[25] P. Polack, F. Altché, B. d'Andréa-Novel, and A. de La Fortelle, "The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?" in *2017 IEEE Intelligent Vehicles Symposium (IV)*, 2017, pp. 812–818.

[26] J. Ziegler, P. Bender, T. Dang, and C. Stiller, "Trajectory planning for bertha — a local, continuous method," *2014 IEEE Intelligent Vehicles Symposium Proceedings*, pp. 450–457, 2014.

[27] J. A. Reeds and L. A. Shepp, "Optimal paths for a car that goes both forwards and backwards." *Pacific J. Math.*, vol. 145, no. 2, pp. 367–393, 1990. [Online]. Available: https://projecteuclid.org:443/euclid.pjm/1102645450

[28] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of Mathematics*, vol. 79, no. 3, pp. 497–516, 1957. [Online]. Available: http://www.jstor.org/stable/2372560

[29] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, p. 509–517, Sep. 1975. [Online]. Available: https://doi.org/10.1145/361002.361007

[30] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[31] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda, and T. Hamada, "An open approach to autonomous vehicles," *IEEE Micro*, vol. 35, no. 6, pp. 60–68, 2015.

[32] DataSpeed Inc., "We make robots move and cars drive by themselves," http://dataspeedinc.com/, accessed: 2018-11-04.

[33] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi, "Autoware on board: Enabling autonomous vehicles with embedded systems," in *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, 2018, pp. 287–296.

[34] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "Ros: an open-source robot operating system," in *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan, May 2009.