A new Approach for Robot Motion Planning using Rapidly-exploring Randomized Trees

Lukas Krammer, Wolfgang Granzer, Wolfgang Kastner
Vienna University of Technology
Automation Systems Group
Email: {lkrammer, w, k}@auto.tuwien.ac.at

***Abstract*—In the last few years, car-like robots became increasingly important. Thus, motion planning algorithms for this kind of problem are needed more than ever. Unfortunately, this problem is computational difficult and so probabilistic approaches like Probabilistic Roadmaps or Rapidly-exploring Randomized Trees are often used in this context.**

This paper introduces a new concept for robot motion planning especially for car-like robots based on Rapidly-exploring Randomized Trees. In contrast to the conventional method, the presented approach uses a pre-computed auxiliary path to improve the distribution of random states. The main contribution of this approach is the significantly increased quality of the computed path. A proof-of-concept implementation evaluates the quality and performance of the proposed concept.

I. Introduction

In the past two decades, robots became more and more present in our daily life. Primarily, robots were invented to save human resources in industry. Since a few years the use of robots is not only limited to the industrial domain. Due to the introduction of precise navigation systems like the Global Positing System (GPS) and well established sensor technologies (e.g., for obstacle detection), robots can also be used in *urban terrain* and *agriculture* to perform fully autonomous applications (e.g., mowing of large areas). For this kind of applications, *motion planning* is of utmost importance. The aim of motion planning for these applications is to find a feasible path for a so called *Car-like Robot (CLR)* between two arbitrary points.

Figure 1 shows an example for such a *working area* where a feasible path has to be found. The working area (red polygon) defines the area where the CLR is allowed to work independently. It may contain an arbitrary amount of obstacles (blue polygons) which must be taken into account by the CLR.

In general, it is computational difficult to find a proper path for such arbitrary areas. Unfortunately, finding a complete or even complete and optimal solution, requires highly complex algorithms (e.g., NP, PSPACE) and therefore it is not possible to completely solve such problems in practice. Some results of complexity analyses are presented in [1], [2].

In the context of motion planning, many different approaches exist (cf. [3], [4]). For example, roadmap based methods like *Visibility Graphs* or *Cell decomposition* can be used. Furthermore, *Potential Fields* are used by different motion planning approaches as described in [4].

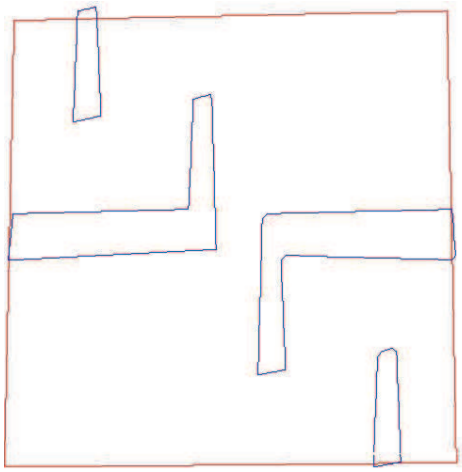


Fig. 1. Example of a working area and two obstacles

A special class of algorithms that are suitable for solving this kind of motion planning problem are *randomized algorithms*. Obviously, such algorithms are not complete or optimal, but they provide pretty good results with a high performance so they are also usable for practical implementations. One class of randomized algorithms is based on so called *Probabilistic Roadmaps (PRMs)* (cf. [5]–[7]). The basic idea of this approach is to construct a roadmap (i.e., a graph) based on randomly sampled states. Using such a roadmap, a path between two states (i.e., start and goal point) can be found by searching the nearest neighbor of the two points and using a graph search algorithm. Unfortunately, this approach requires an enormous amount of sample points in case of large areas.

Another probabilistic approach uses a special kind of trees which discover given spaces rapidly. These trees are called *Rapidly-exploring Randomized Trees (RRTs)*. The proposed approach uses the basic structure of this algorithm to compute paths in presence of arbitrary obstacles. While path-finding algorithms based on RRTs are very promising for the given problem instance, conventional RRT algorithms have several disadvantages. First, the path length of the solutions as well as the standard deviation to the optimal solution may be very large. Second, the time to find a solution may vary, too. Therefore, this paper presents an approach that significantly improves the quality of the solutions. After a brief

978-1-4577-0434-5/11/$26.00 ©2011 IEEE

introduction to the basic concepts of RRTs (cf. Section II), Section III describes the improvements to conventional RRT algorithms. Afterwards, Section IV discusses the special kind of tree expansion. To evaluate the proposed improvements, the developed approach has been implemented within a simulation framework. Details of the simulation framework as well as the experimental results of the evaluation are given in SectionV.

II. Rapidly-exploring randomized trees

As introduced in [8], RRTs are special trees which discover given spaces rapidly. Due to this fact, the algorithm is well suited for path planning, especially for finding a path between large distances. In contrast to other motion planning approaches, RRTs can be used for high dimensional spaces and do not need any preprocessing. Thus, the approaches are well suited for single query purposes [9].

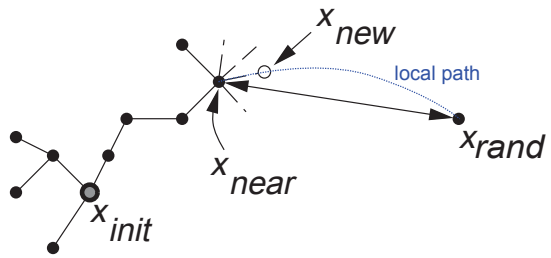


Fig. 2. Visualization of function `EXTEND()` (taken from [10])

The construction algorithm of an RRT is quite easy to implement. Beginning from an arbitrary start point, the algorithm selects a random state within the free portion of the space. Subsequently, it searches for the closest state in the current tree using a feasible metric. Furthermore, the algorithm uses a local planner to get towards the random state. In a predefined or random distance from the nearest node, the algorithm generates a new node, if the path and the new state do not collide with obstacles.

For efficient path planning, it is useful to generate two different trees starting from the start point (x_{init}) and the end point (x_{goal}). If the trees intersect or meet each other in a feasible way, a path is found. Algorithm 1 formally describes the basic steps. However, essential steps (i.e., sub procedures) are left open in this algorithm and depend on the actual problem and space. Thus, these procedures are described as follows.

- `RANDOM_STATE()`
  This procedure is used to generate a random state out of the space. The space for a CLR depends on the used system model. The simple system model only considers the position and the heading of the CLR and therefore it can be defined as $\mathcal{C} = \mathbb{R}^2 \times [0, 2\pi[$. If the simple system also considers the steering angle as state variable, the space extends to $\mathcal{C} = \mathbb{R}^2 \times [0, 2\pi[\times [-\varphi_{max}, \varphi_{max}]$.
  It is important that the random function generates a uniform distributed random number in every dimension to guarantee probabilistic completeness. Furthermore, this procedure must also perform a collision check to avoid that the random states are inside obstacles.
- `EXTEND` $(\mathcal{T}, x_{rand})$
  This procedure extends the tree by a node based on a given random point. At the beginning, the procedure searches for the nearest neighbor of the given random state. Therefore, the distance of every node of tree to the random state has to be computed and the shortest one is chosen. Unfortunately, the exact computation of the real distance (or costs) between two states can be as hard as the motion planning problem itself. However, it is not necessary to know the exact distance. It is sufficient to estimate the distance using a heuristic or simply the Euclidean distance.
  A very fast method is to consider only the Euclidean distance of the projection on $\mathbb{R}^2$, but this is not optimal in general. Consider the case that two states with nearly the same coordinates in $\mathbb{R}^2$, but two different headings. The Euclidean distance between the two states is nearly zero, but the real distance is much higher since the CLR has to turn.
  If the nearest neighbor is determined, a path has to be

Algorithm 1 Bidirectional planning algorithm using an RRT (adapted from [10])

RRT_CONNECT_PLANNER (x_{init},x_{goal})

```
1:  T_a.init(x_init)
2:  T_b.init(x_goal)
3:  for k = 1 to K do
4:     x_rand ← RANDOM_STATE ()
5:     if not EXTEND (T_a, x_rand) = Trapped then
6:        if CONNECT (T_b, x_new) = Reached then
7:           Return PATH(T_a, T_b)
8:        end if
9:     end if
10:    SWAP (T_a, T_b)
11: end for
12: Return Failure
```

EXTEND($\mathcal{T}$, x)

```
1:  x_near ← NEAREST_NEIGHBOR (x,T)
2:  if NEW_STATE(x, x_near, x_new, u_new) then
3:     T.add_vertex (x_new)
4:     T.add_edge (x_near, x_new, u_new)
5:     if x_new = x then
6:        Return Reached
7:     else
8:        Return Advanced
9:     end if
10: end if
11: Return Trapped
```

CONNECT($\mathcal{T}$, x)

```
1: repeat
2:    S ← EXTEND (T, x)
3: until not (S = Advanced)
```

generated towards the random points. Note that it is not necessary that the new state reaches the random point. It is only necessary that the path from the nearest node to the new node is feasible. It is possible to use the result of a local planner and compute the new node in a fixed or random distance from the nearest node. However, using a local planner does not guarantee that the resulting path is navigable. Therefore, finding an adequate path to a new node within a reasonable time is a critical point. Section IV presents an approach of an effective solution to this problem.

- `CONNECT` $(\mathcal{T}, x_{new})$
  This function is used to check if the new node computed by `EXTEND` can be feasibly connected to some node of the other tree. The procedure has to decide whether two states are feasible considering their heading and the obstacles. Furthermore, the system model of the CLR has to be considered (i.e. the curvature constraint). For this purpose, it is possible that a local planner is used to connect two states. Obviously, a collision check has to be performed.
- `PATH` $(\mathcal{T}, \mathcal{T})$ Based on two distinct states of the two trees, the `PATH` function computes a path from x_{init} to x_{goal}. If every node of a tree stores the path to its predecessor, it is simple to compute a path from the connecting nodes to the root nodes. Furthermore, both paths must be combined with the result of the local planner to return the complete path.

III. A Hybrid Approach for Distributing Random Points

Unfortunately, the average length of the paths generated by the conventional RRT approach may be very large. Moreover, the standard deviation of a set of computations is also very high. To improve the quality of the computed paths, a new approach for distributing the random points is introduced. In contrast to the guided RRT approach presented in [11], this approach uses a special distribution of the random points to guide the search. In addition to the uniform distribution of the random points which guarantees probabilistic completeness, a normal distributed channel around an *auxiliary path* is generated. Within this channel the density of random points is much higher than in other regions. However, the normal distributed function guarantees that the density in the center of the channel is higher than in the outer regions of the channel. Note that since the auxiliary path is only used to guide the random search, the auxiliary path does not need to suffice the constraints of the system i.e., the path does not need to be "navigable". Therefore, this auxiliary path can be computed using a fast path-finding algorithm (e.g., a heuristic).

Figure 3 shows a sketch of an environment. In this figure, the auxiliary path is marked red. Furthermore, the density around this path is high. The orange areas in the figure denote a medium density and the yellow areas denote a low density. Finally, the white region denotes the uniform distributed area where the density is lower than in all other regions. The blue curves along the path sketch the normal distribution of the random points along the path. Note that the density decreases continuously.

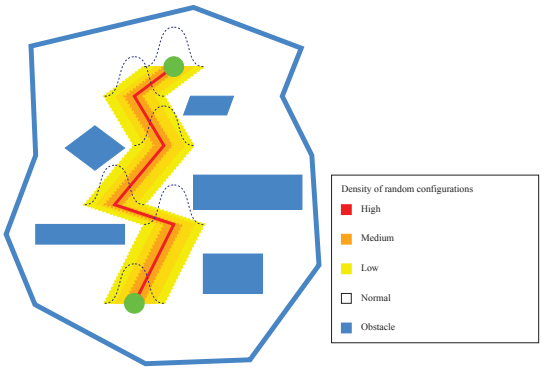


Fig. 3. Visualization of the Density of random points in a CS

In most cases this approach computes a path very fast in contrast to the basic RRT approach. Furthermore, the quality of this path is much higher due to the concentration of random points along the auxiliary path. However, there are some special environments where this approach may take longer than the conventional approach. Consider, for example, an environment with a long narrow and infeasible passage. The hybrid approach will try to find a path along this passage, even if no feasible one exists along the auxiliary path. Since the density of random points in the other regions is quite lower, this approach will take a longer time to find a feasible path, but a path will be found, anyway.

IV. Simulation Based Tree Expansion

Another critical point is the extension of the tree. After a new random point has been selected and the nearest node in the tree has been determined, a feasible path has to be computed which moves the virtual CLR towards the random point. This can be done by different methods depending on the physical system.

A very flexible approach is the simulation of the desired system to compute a feasible path. In this case, a combination of a system model and a controller is used.

Figure 4 shows the concept of this approach. The system starts simulating from the closest node of the tree (w.r.t. the random point). The input of the controller is the desired random point. A feedback loop between the controller and the virtual system is used to compute a path towards the random point. The most important property of this process is that the virtual system model generates a feasible trajectory.

Since time discrete models are used, the control process is iterated by a given number of steps. The last output of the system model is defined as the new node of the tree. The following two paragraphs describe the system model and the controller.

A. Simulation model

Basically, the simulation model is a discretization of the simplified *single track model* as described in the following

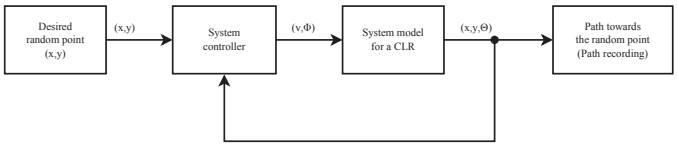


Fig. 4. Block diagram of the trajectory generator

equation:

$$\begin{aligned} X_{k+1} &= X_k + (v \cdot \Delta t) \cdot \cos(\theta_k + \tfrac{\psi \cdot \Delta t}{2}) \\ Y_{k+1} &= Y_k + (v \cdot \Delta t) \cdot \sin(\theta_k + \tfrac{\psi \cdot \Delta t}{2}) \\ \theta_{k+1} &= \theta_k + (\psi \cdot \Delta t) \end{aligned} \quad (1)$$

where X_k and Y_k denote the Cartesian coordinates of the current position, θ_k the current orientation, and v the speed of the CRL. X_{k+1},Y_{k+1}, and θ_{k+1} represent the next state. ψ can be determined as

$$\psi = \frac{v}{L} \tan(\varphi). \quad (2)$$

where φ denotes the steering angle.

Since the path computation starts from the start position and from the end position, two different system models have to be used in general. In this case, only the sign of the speed v has to be changed.

B. Controller

While the system model simulates a real system (i.e., the CLR), the system controller is used to move the CLR towards the desired point (i.e., the random point). Generally, lots of motion controllers for CLRs exist. For this purpose a controller called *pure pursuit controller* (cf. [12]) is used. This controller can be used for forward driving or with little adaptions even for backward driving. Figure 5 shows the geometric model of the CLR and the concept of the controller. Based on the angle $\eta_{\{fw,rv\}}$ (which is the angle between the line segment and the main axis of the CLR), the steering angle is computed. However, the distance between the point on the path and the CLR depends on the speed. Note that this controller does not control the speed of the CLR.

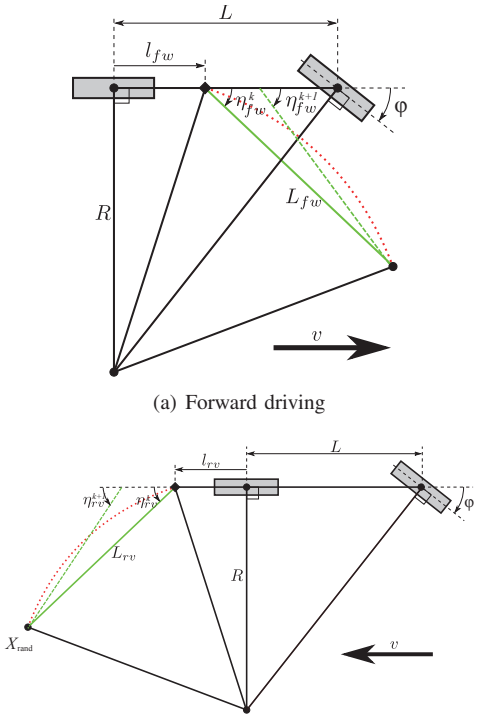
To use the *pure pursuit controller* for this purpose, it has to be adapted since no path or trajectory is given. Thus, the line segment is computed between the CLR and the desired static point. Similar to the conventional *pure pursuit controller*, the steering angle is computed. Equation 3 defines the corresponding control law for the system either in forward or in reverse mode. Note that only the steering angle is controlled, the speed of the CLR has to be kept constant.

$$\varphi_{\{fw,rv\}} = -\arctan\left(\frac{L \sin \eta_{\{fw,rv\}}}{\frac{L}{2} + l_{\{fw,rv\}} \cos \eta_{\{fw,rv\}}}\right) \quad (3)$$

V. Implementation and Experiments

A. Environment

To evaluate the previously described approach in contrast to a conventional RRT approach, a simulation framework was implemented. Using this framework, it is possible to specify arbitrary environments with an arbitrary amount of obstacles. An example of such an environment is shown in Figure 1. These objects are specified by using either planar coordinates or even GPS coordinates. Based on two arbitrary states (i.e., start state and goal state), the implementation tries to find a path between these states considering the obstacles.



(a) Forward driving

(b) Reverse driving

Fig. 5. Pure pursuit controller which moves the CLR towards X_{rand} (taken from [12])

B. Auxiliary Path

The auxiliary path is computed by using a mesh grid in the free area of the space. Based on this grid, a graph is computed. To compute the auxiliary path, the nearest nodes in the graph to the end points (i.e., start and end point) are searched w.r.t. the Euclidean distance. Finally, an implementation of Dijkstra's algorithm [13] is used to compute the path.

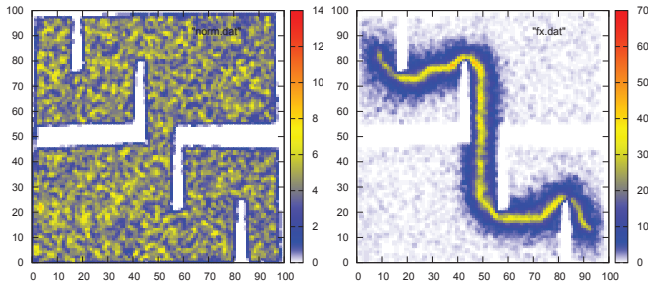
C. Tree computation

As described in Section IV, a simulation process is used to compute the path. Thus, the system differential equation has to be solved. Instead of the use of an Ordinary Differential Equation (ODE) solver, the system differential equation was discretized and executed step-by-step.

The CLR is simulated with a constant speed and the steering angle is set after a number of adjustable steps.

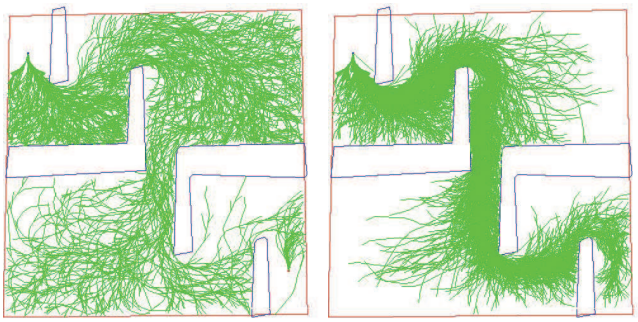
D. Results

To prove the presented approach and to compare it with the conventional approach, it was implemented and evaluated. The following results for a problem instance will illustrate the



(a) Conventional RRT (b) Hybrid RRT

Fig. 6. Comparison of the random point generation between the conventional and the new hybrid RRT approach



(a) Conventional RRT (b) Hybrid RRT

Fig. 7. Comparison of the tree exploration between the conventional and the hybrid RRT approach

computation process of the paths and the differences between these approaches. To evaluate the described test series, the test application was executed on a *Sun Java* JVM_x64 with Windows 7 64 Bit operating system. The hardware platform is based on a dual core CPU (Intel®Core ™2 Duo E8400, 2x3.0 GHz, 4 GB RAM).

1) Random point generation: As described in Section III, the hybrid RRT approach differs from the conventional RRT approach in the generation of the random points. Figure 6 shows the absolute density of random points per square meter represented by color gradients in an example environment. Note that 15.000 random point were computed in each case. In Figure 6(a) the density of random points of the conventional RRT is shown. Obviously, the density is equal for every region except the obstacle regions. In contrast to the conventional approach, the hybrid approach computes a channel of normal distributed random points in addition to the uniformly distributed random points. Figure 6(b) shows the density of random points for the hybrid RRT. It is easy to see that the density of random points along the auxiliary path is much higher than in all other regions of the environment.

2) Tree exploration: In the previous paragraph, the result of the random point generation was illustrated. Based on these random points, two trees are computed starting from the start- and the endpoint. In Figure 7(a) it is shown that the trees of the conventional RRT approach explore the whole environment except the obstacles. Figure 7(b) shows that the trees of the hybrid approach explore the free space of the area, but they concentrate on the auxiliary path.

Note that a path can be found with less tree nodes as shown in Figure 7, but the tree computations were continued anyway to illustrate the behavior of both approaches.

3) Performance and quality of the path computation: To show the quality and the performance of the new hybrid RRT approach, a quantitative analysis of the runtime and the length of the computed path was made. This analysis is based on a fixed environment and fixed initial and goal states. To increase the significance of this analysis, 50 executions of the same but independent problem instance were examined. Thereby, once the conventional RRT approach and once the hybrid RRT approach were used.

Based on these test series, the mean value and the standard deviation are listed in Table I. To illustrate the differences between the two approaches, the results of the test series were evaluated by an Empirical Distribution Function (EDF) regarding the path length and the computation time as shown in Figure 8. The diagrams show that the paths computed with the hybrid approach are much shorter than those which are computed with the conventional approach. Moreover, the diagram indicates that the standard deviation (w.r.t. the path length) of the hybrid approach is smaller than the standard deviation of the conventional approach. The hybrid RRT approach has also better performance (i.e., shorter computation time) than the conventional RRT. Similarly to the path length, the standard deviation of the hybrid RRT is much smaller than that of the conventional RRT.

*	*	*	Conv. RRT	Hybrid RRT
Duration	Avg.	ms	3953,4	1676,7
Duration	Std.dev.	ms	1967,7	642,0
Path length	Avg.	m	267,2	205,5
Path length	Std.dev.	m	33,6	19,5

TABLE I
COMPARISON OF THE CONVENTIONAL RRT APPROACH WITH THE NEW HYBRID RRT APPROACH

To visualize the path computation, Figure 9 shows the computation of 50 paths with the same initial and goal state. The left figure shows the computation using the conventional approach and the right figure shows the paths computed by the new hybrid approach.

VI. CONCLUSION AND OUTLOOK

A lot of different motion planning concepts for CLRs exist with different benefits and drawbacks. This paper presented a probabilistic algorithm which promises high quality paths. Moreover, the algorithm stands out due to its high computational performance. This paper also presented a proof-of-concept implementation that was integrated into a simulation framework. Using this simulation framework, the improvements of the proposed approach were evaluated. As shown by the results of the simulation, both the quality of the computed

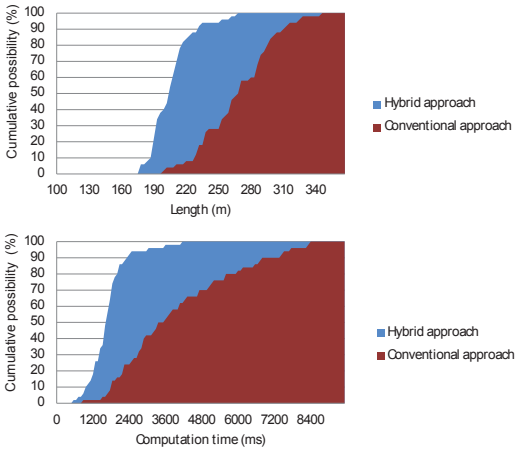
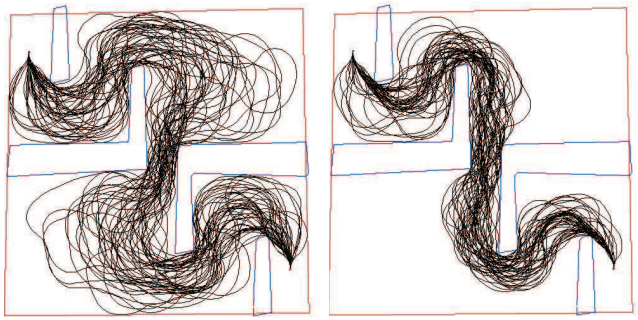


Fig. 8. EDF of the path length and the computation time



(a) Conventional RRT (b) Hybrid RRT

Fig. 9. Comparison of the conventional and the hybrid RRT approach computing 50 paths

path as well as the average runtime of the algorithm were enhanced.

The simulation results show that the proposed motion planning approach provides a good solution with an acceptable runtime which makes the approach suitable for real-world application. Therefore, as a next step, the implementation is going to be ported to a CLR used for mowing applications in industrial agriculture. First field tests are already underway.

REFERENCES

[1] J. Reif, , J. Reif, and H. Wang, "The complexity of the two dimensional curvature-constrained shortest-path problem," in *Proc. 3rd International Workshop on the Algorithmic Foundations of Robotics*, 1998, pp. 49–57.

[2] J. H. Reif, "Complexity of the mover's problem and generalizations," in *SFCS '79: Proc. of the 20th Annual Symposium on Foundations of Computer Science*. Washington, DC, USA: IEEE Computer Society, 1979, pp. 421–427.

[3] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006, available at http://planning.cs.uiuc.edu/.

[4] J. C. Latombe, *Robot Motion Planning*. Norwell, MA, USA: Kluwer Academic Publishers, 2003.

[5] L. Kavraki and J. C. Latombe, *Practical Motion Planning in Robotics: Current Approaches and Future Directions*. John Wiley and Sons, 1998, ch. Probabilistic Roadmaps for Robot Path Planning, pp. 33–53.

[6] L. Kavraki, P. Svestka, J. C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," in *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, 1996, pp. 566–580.

[7] G. Song and N. M. Amato, "Randomized Motion Planning for Car-like Robots with C-PRM," Department of Computer Science, Texas A&M University, Tech. Rep., 2001.

[8] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," Iowa State University, Tech. Rep., 1998.

[9] J. J. Kuffner Jr. and S. M. Lavalle, "RRT-Connect: An efficient approach to single-query path planning," in *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2000, pp. 995–1001.

[10] S. M. L. Valle and J. J. Kuffner, "Randomized kinodynamic planning," *Journal of Robotics Research*, vol. 20, pp. 378–400, 2001.

[11] V. Vonàsek, J. Faigl, T. Krajnìk, and L. Přeučil, "RRT-path - A Guided Rapidly Exploring Random Tree," in *Robot Motion and Control 2009*, K. R. Kozlowski, Ed. Springer, 2009.

[12] Y. Kuwata, J. Teo, S. Karaman, G. Fiore, E. Frazzoli, and J. P. How, "Motion planning in complex environments using closed-loop prediction," 2007.

[13] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[14] L. Krammer, "Motion Planning for Car-like Robots," Master's thesis, Vienna University of Technology, Institute of Computer Aided Automation, Automation Systems Group, Jul. 2010.

[15] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations (Intelligent Robotics and Autonomous Agents)*. The MIT Press, June 2005.

[16] J. P. Laumond, S. Sekhavat, and F. Lamiraux, *Robot Motion Planning*. Springer, 1998, ch. Guidelines in Nonholonomic Motion Planning for Mobile Robots.

[17] A. DeLuca, G. Oriolo, and C. Samson, *Robot Motion Planning*. Springer, 1998, ch. Feedback Control of a Nonholonomic Car-Like Robot.

[18] P. Svestka and M. H. Overmars, *Robot Motion Planning*. Springer, 1998, ch. Probabilistic Path Planning.

[19] S. Kammel, J. Ziegler, B. Pitzer, M. Werling, T. Gindele, D. Jagzent, J. Schröder, M. Thuy, M. Goebl, F. v. Hundelshausen, O. Pink, C. Frese, and C. Stiller, "Team annieway's autonomous system for the 2007 darpa urban challenge," *J. Field Robot.*, vol. 25, no. 9, pp. 615–639, 2008.

[20] Y. Kuwata, G. A. Fiore, J. Teo, E. Frazzoli, and J. P. How, "Motion planning for urban driving using RRT," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008, pp. 1681–1686.

[21] J. A. Reeds and L. A. Shepp, "Optimal path for a car that goes both forwards and backwards," *Pacific Journal of Mathematics*, vol. 145, no. 2, 1990.

[22] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney, "Stanley: The robot that won the DARPA Grand Challenge: Research Articles," *Journal of Robotic Systems*, vol. 23, no. 9, pp. 661–692, 2006.