A Study Of Motion Planning Techniques For Autonomous Vehicle Navigation

Ananth Sriram Akshay Swaminathan Siddharth Viswanathan Allysun Hatton-Bannister

Abstract— **In recent years, the advancement of motion planning techniques has led to the development and realization of driverless vehicle technology. The key task to be accomplished by these systems is the formulation of an optimal, safe path from a start state to goal state capable of accounting for complex, dynamic environments while maintaining real-time functionality of the planner.**
In this paper, we introduce one of the most widely used planning libraries, the Open Motion Planning Library (OMPL) and elaborate a comparison of some of the planners available in this library within a user defined environment. The approach is evaluated by benchmarking the planners in OMPL with varied runtime parameters. Finally, we identify the strengths and limitations of the reviewed approaches and their problem dependent applications.

I. INTRODUCTION

In the last few decades, researchers have put considerable effort into developing autonomous driving technology. Autonomous vehicles have great potential to improve the performance and safety of currently existing transportation systems. To achieve this objective without affecting existing human drivers on the road, autonomous vehicles need to achieve a human-acceptable level of driving performance. The planner in the vehicle also needs to meet strict real-time requirements which enable it to react fast enough in emergency situations. In summary, it is important to develop a practical high-performance real-time motion planner for on-road driving.

Sampling based motion planners are powerful tools that employ sampling of the state space of the robot in order to quickly and effectively solve planning problems; especially for systems with a high number of degrees of freedom. Some of the limitations faced in previous works are constraints based on time, motion constraints or due to the size and dimension of the state space. Sampling methods give the possibility to quickly cover a much larger and more complex piece of the state space to get goal state from a state space through a valid and feasible path. Another interesting feature of the sampling-based methods is that most of them provide a probabilistic completeness, which means that if a solution exists, the probability of finding it converges to one as the number of samples goes to infinite. However, a point to note is that sampling-based approaches do not know if the problem has a solution solution or not.

There has been an increasing relevance of the sampling-based planning methods and their implementation in recent times; a lot of research and resources have been devoted to not just the original planners but also in the creation of a wide variety of extensions and variants of all of the different approaches. Usually all these variants have been carried by different researchers. That makes it difficult to compare the different techniques because they were tested on different environments, using different systems, and relying on the use of different libraries and languages. For this reason, we have been motivated to make use of the Open Motion Planning Library (OMPL), an extended library where most of the sampling-based methods have been implemented. To test these algorithms in a similar environment, we create our own 3-D model using Blender, a free open source modeling software. All the simulations can thus be run on the same computer, so the results will be easy to compare and reliable. The idea is to solve the same problem with different sampling based planning methods and elaborate a comparison between the obtained outputs. As a consequence of this we can obtain a comparison of some of the advantages and disadvantages of each method, as for future uses it could be useful to decide which one to choose depending on the problem to be solved.

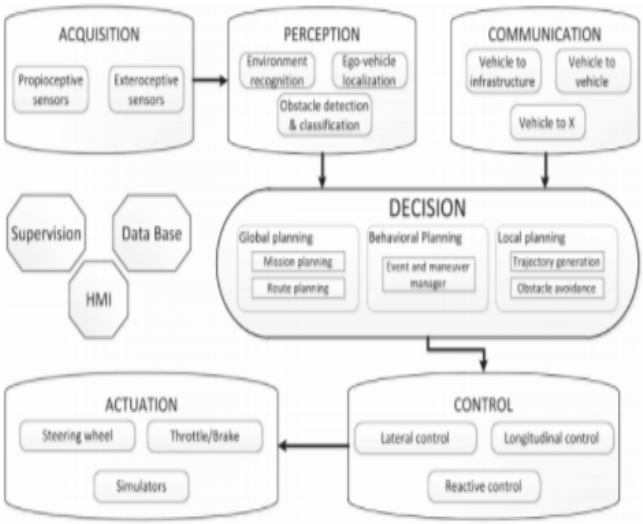

Fig. 1: Control Architecture of an Autonomous Vehicle

II. RELATED WORK

The Behavioural layer in Fig.1, comprising of some motion planner, is responsible for formulating a safe and dynamically feasible path from the vehicle's current configuration to some goal state while satisfying all local and global constraints. Depending on whether the quality of the solution path is considered, the terms feasible and optimal are used to describe this path. Feasible path planning refers to the problem of determining a path that satisfies some given problem constraints while the quality of the solution is ignored. Optimal path planning refers to the problem of finding a path that optimizes some objective function (e.g comfort) subject to some given constraints. In this section we introduce some of the related work done on motion planners and broadly classify them into three groups:

A. Sampling Based Techniques

In order to construct roadmaps in high-dimensional configuration spaces, Kavraki et al. [7] advocate the use of random sampling within the framework of Probabilistic Roadmaps (PRM) .This is because unlike grids, they can be naturally run in an anytime fashion. Since PRMs are formulated in a general fashion, they have been used for path planning for a variety of systems, including systems with differential constraints. However, the theoretical analyses of the algorithm have primarily been focused on the performance of the algorithm for systems without differential constraints, i.e. when a straight line is used to connect two configurations. Under such an assumption, PRMs have been shown in [7] to be probabilistically complete and asymptotically optimal. That is, the probability that the resulting graph contains a valid solution (if it exists) converges to one with increasing size of the graph and the cost of the shortest path in the graph converges to the optimal cost.

Karaman and Frazzoli [5] proposed an adaptation of batch PRM, called PRM*, that instead only connects neighboring vertices in a ball with a logarithmically shrinking radius with increasing number of samples to maintain both asymptotic optimality and computational efficiency. Schmerling et al. [10] propose a differential version of PRM* and prove asymptotic optimality of the algorithms for driftless control- affine dynamical systems, a class that includes models of non-slipping wheeled vehicles.

B. Graph Search Strategies

After the discretization of the free configuration space into the form of a graph, in order to obtain an actual optimal path in such a discretization, one must employ one of the graph search algorithms. In this section, we are going to review the graph search algorithms that are relevant for path planning.

Dijkstra's algorithm [6] is one of the most widely recognized algorithms used for finding the shortest paths in a graph. The algorithm performs the best first search to build a tree representing shortest paths from a given source vertex to all other vertices in the graph. When only a path to a single vertex is required, a heuristic can be used to guide the search process. The most prominent heuristic search algorithm is A* developed by Hart, Nilsson and Raphael [8]. If the provided heuristic function is admissible (i.e., it never overestimates the cost-to-go), A* has been shown to be optimally efficient and is guaranteed to return an optimal solution.

Often, as our sensory data from the world is updated, we find that the the shortest path from the vehicle's current configuration to the goal region is sought repeatedly. Since each such update usually affects only a minor part of the graph, it might be wasteful to run the search every time completely from scratch. The family of real-time replanning search algorithms such as D* [9], Focussed D* [11] and D* Lite [12] has been designed to efficiently recompute the shortest path every time the underlying graph changes, while making use of the information from previous search efforts.

A clear limitation of algorithms that search for a path on a graph discretization of the configuration space is that the resulting optimal path on such graph may be significantly longer than the true shortest path in the configuration space.

C. Incremental Search Techniques

One of the disadvantages of techniques that search over a fixed graph discretization is that they search only over the set of paths that can be constructed from primitives in the graph discretization. As a result, these techniques may fail to return a feasible path or may return a noticeably suboptimal one. To address this problem we introduce incremental feasible motion planners - typically, these methods incrementally build increasingly finer discretizations of the configuration space while simultaneously determining if a path from the initial configuration to the goal region exists in the discretization at each step. One of the first randomized tree-based incremental planners to be developed was the expansive spaces tree (EST) planner proposed by Hsu et al. [14]. The algorithm works by selecting a vertex for expansion randomly with a probability that is inversely proportional to the number of vertices in its neighborhood. This helps in promoting growth towards unexplored regions.

Rapidly-exploring Random Trees (RRT) [15] have been proposed by La Valle as an efficient method for finding feasible trajectories for high-dimensional non-holonomic systems. The rapid exploration is achieved by taking a random sample from the free configuration space and extending the tree in the direction of the random sample. Recently, it has been shown that using

RRT with heuristic steering over a fixed time step is not probabilistically complete [17]. Karaman and Frazzoli [5] also demonstrated that the RRT converges to a suboptimal solution with probability one and designed an asymptotically optimal adaptation of the RRT algorithm, called RRT*. Sufficient conditions for asymptotic optimality of RRT* under differential constraints are stated in [5] and demonstrated to be satisfiable for Dubins vehicle and double integrator systems. It has also been shown that for small-time locally attainable systems, the algorithm can be adapted to maintain not only asymptotic optimality, but also computational efficiency [18].

III. EXPERIMENTAL IMPLEMENTATION

This section details the experimental setup that was used to implement and compare the different motion planners in the same environment.

A. Environment Creation

Irrespective of the resolution of the state space e.g 2D in SE2, the environment to be used within the ompl.app must be 3D. This is due to the geometric representations that the GUI realizes internally in order to construct boundaries within which the planner is run. In order to develop this environment, we require a 3D software that is capable of creating COLLADA data files. In this project we have chosen to use Blender, a free open source software. A 2D view of our rendered environment is shown in Fig.2.

The *.blend* file is then exported as a *.dae* extension, which is integrable with the ompl GUI. The robot chosen (car1/car2 etc) is constrained to move inside a tight bounding box around the environment, the start pose, and the goal pose. The bounding box is visualized in OMPL.app as the white frame around the environment and robot and care must be taken so that the box fits the internal boundaries of the environment only.

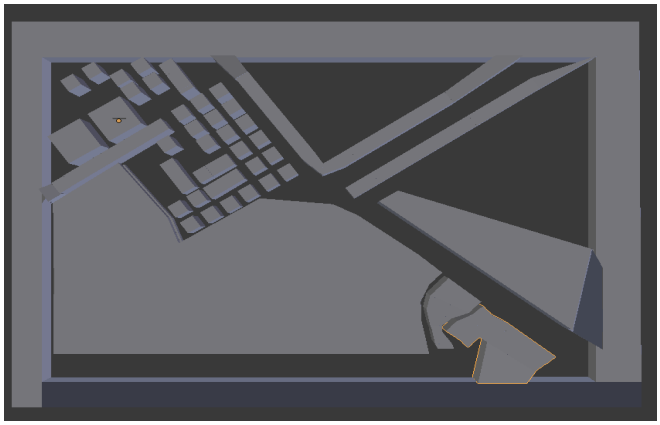

Fig. 2: The Environment as viewed in 2D

B. Open Motion Planning Library

OMPL, the Open Motion Planning Library, is a library comprising of some of the most important sampling-based motion planning algorithms. OMPL itself does not include any code related to collision checking or some plotting process. This part is left as a user's choice.

OMPL provides a way to represent all the important concepts of motion planning, such as the state space, control space, state validity, sampling, goal representations, and planners. Taking into account these variables, the user must select a computational representation for the robot and provide an explicit state validity/collision detection method. There does not exist a default collision checker in OMPL, which allows the library to operate with relatively few assumptions, allowing it to plan for a huge number of systems while remaining compact and portable.

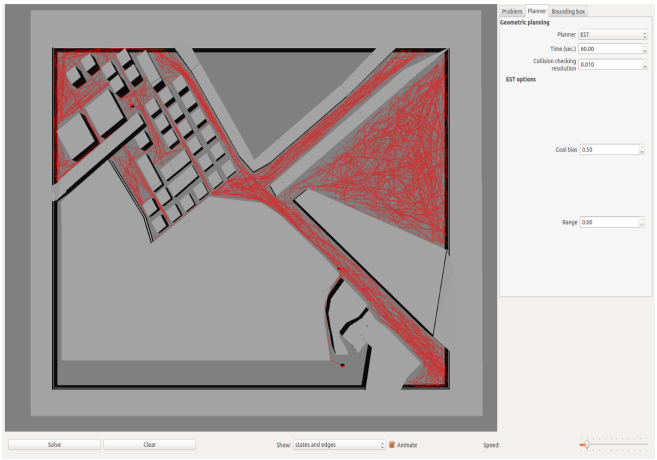

Fig. 3: The ompl GUI showing states and edges

The ompl GUI is a visual representation of OMPL, which can be applied to planning motion with rigid bodies and different vehicle types. It is based on the Assimp library, and uses a large variety of meshes to give a representation of robots and environments. It also provides a way of representing the dimensions and shape of the robot and the ability to transform these geometrics and the ones given by the environment into a collision checking routine through the library using a particular geometric representation. To utilize the app, first we have to select the type of robot, then load our environment, choose the robot model and lastly set the planner with suitable parameters. Finally, we have the possibility to see the planner results as the states, edges, or states and edges as shown in Fig.3. OMPL.app also has the ability to save the current solution path and reload that path for viewing in the future.

C. Planner Selection

In OMPL there are two categories of planners:

- Geometric planners
- Control based planners

Most of the planners were developed in both categories; usually first came the geometric planners which were then adapted to their control versions. The exceptions are KPIECE, and PDST, which were created as control planners and then converted to geometric ones. Gometric planners [21] compute a collision-free path connecting the initial and final configurations of a robot with a minimal number of waypoints. Control planners are used when the system is subject to differential constraints.For our experiments we have chosen to make use of the following planners: RRT, RRT*, KPiece, EST, PRM and PRM*.

It would be useful to obtain a common view of all of planners, so as to make an easier decision as to which one suits better with the relevant problem based on their features and the data obtained from the experiments. To do so we are going to run the same planning problem, the one described in section III.A. However, we make a slight alteration in that for geometric planners a comparison is elaborated between the geometric version of RRT, PRM, EST and KPIECE. In the case of control-based planners, the same environment and the same robot will be used but a state propagator must be added, according to the dynamics of the selected car. The planners selected are RRT, PRM and EST. Finally, we present three comparisons among the optimized versions of the RRT and PRM planners.

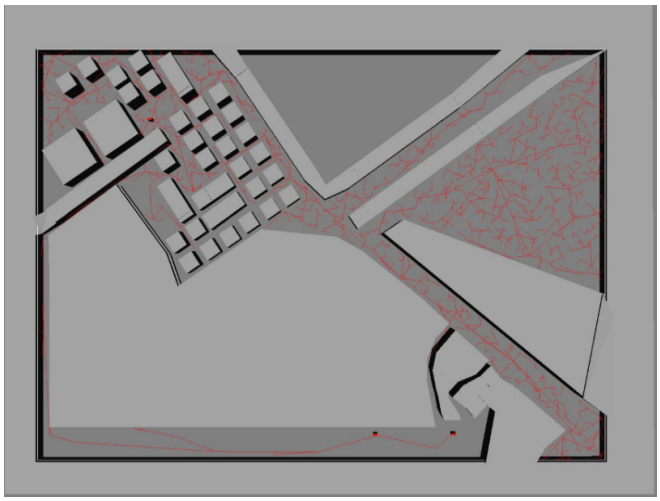

Fig. 4: The RRT Solver Output

IV. EXPERIMENTAL RESULTS

In this section we present the results obtained from running the planners which were divided into three categories: Geometric,Control-based and Optimizing planners. The tables below present a comparison of the planners based on different data: Time to find a solution, Number of states sampled, Number of states in the path, Length of the path and Number of edges for each one of the geometric planners. Since we are working with randomized planners, one experiment could vary too much from other simply because they are based on random samplings. As a consequence, we present an average value of our results which were obtained by running each solver ten times. The values are also normalized to make for ease of understanding.

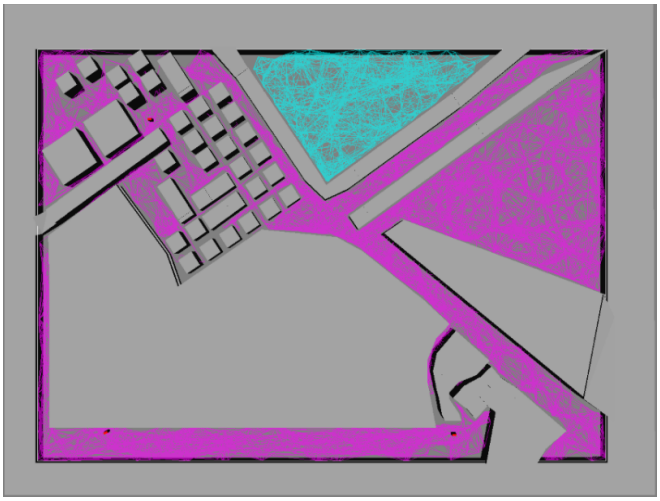

Fig. 5: The PRM Solver Output

A. Geometric Planners

In Table 1 it can be observed how some parameters change according to the planner being used to solve the problem. On the time spent to find a solution, we can see that, except PRM the other planners have a similar response time of approximately 2.3 ms. The reason why PRM has a comparatively high solution time is due to the way the planner algorithm works. PRM samples a vertex which is then connected to another within a fixed neighbor radius. That task produces an extra effort which results in an increment of the processing time. This can be validated by looking at the last row in Table 1 where it can be seen there is a large difference between the edges created by PRM and the edges created by the other planners. This comparison is also visually validated by looking at Fig 4 and Fig 5, which show the RRT and PRM solvers outputs after running for 60 seconds.

TABLE I: Comparison Of Geometric Planners

	RRT	PRM	KPIECE	EST
Time (s)	2.29	6.12	2.27	2.54
Samples	510	1305	1304	1891
Path States	73	112	126	116
Length (Cost)	1462	1213	2930	1690
Edges	2051	12321	1389	1899

B. Control-based Planners

In geometric planners the planner termination condition indicates that in case a solution is found the planner is to stop running and that solution is given as the path. In the case of control-based planners, we set the time within which the planner has to try to find a solution. For our experiment this time was set to 2 seconds. Reducing this set time did not always provide a solution for all planners. The trajectories generated by the planner won't be straight lines; as a result, where we had a proportional relationship between states of the path and length of the path in the previous section, here the relationship is inverse. In other words, as we are dealing with curved trajectories, the more and smaller curves, the less the path length.

As with with the geometric planner the sampling rate of EST is much bigger than the other planners due to its two trees growing. In this case that difference is even bigger because as the time increases the number of nodes from which the tree can grow increases and the number samples is not just doubled but has an exponential increase.

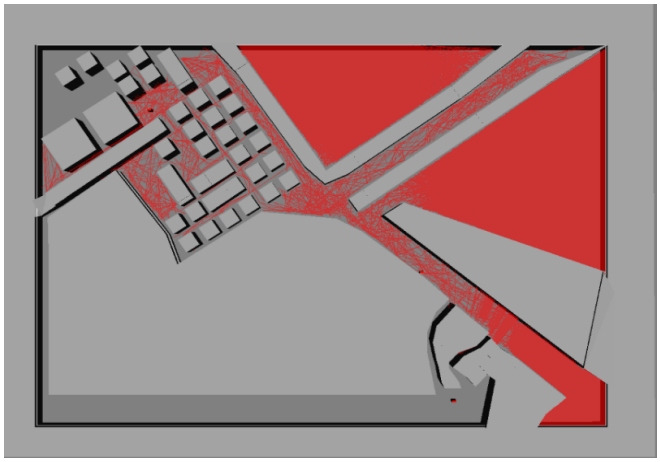

Fig. 6: The KPiece Solver Output

TABLE II: Comparison Of Control-based Planners

	RRT	KPIECE	EST
Time (s)	2	2	2
Samples	18,801	30,780	156,680
Path States	138	251	58
Length (Cost)	1884	1823	1853
Edges	17,950	29,993	156,240

C. Optimizing Planners

In ompl we have two optimal planners available: PRM* and RRT*. First we check that they really provide an optimal solution against their non-optimized versions. After that we attempt to decide which optimal planner gives the best solution. In ompl, two main optimization objectives have been implemented:

- Path Length: The goal is to find the shortest collision-free path possible
- Maximum clearance: This objective tries to steer the robot away from the obstacles

1) RRT vs PRM*:* The comparison between RRT* and PRM* is similar to the one made for their non-optimizing versions. The approach taken is slightly varied, this time we have introduced the other limitation mentioned before, the maximum number of nodes, while keeping the time always as 2 seconds. This modifications does not change the outputs of the PRM* solver, in table III since the limit is not reached but it does with the RRT*, which solves it earlier. In this case taking into account the data given in the table we can note that RRT* is quicker even if the sampling rate is smaller. With respect to the path length as both are optimal, this difference is negligible. It has been demonstrated that they give an optimal solution, so this could not be improved. The last aspect is the number of created edges, which is much bigger in the case of PRM*, that is the reason that makes it slower than RRT*.

TABLE III: RRT* vs PRM*

	RRT*	PRM*
Samples	9231	86410
Path States	209	204
Length (Cost)	1329	1326
Edges	9231	398769

2) PRM vs PRM:* The difference between the optimizing and non optimizing versions of PRM is that in the second phase of the algorithm (Connection phase) the neighbor radius, in which the algorithm looks for nodes to connect to, is not constant but given as a function of the number of samples. The radius decreases as the number of samples increases.

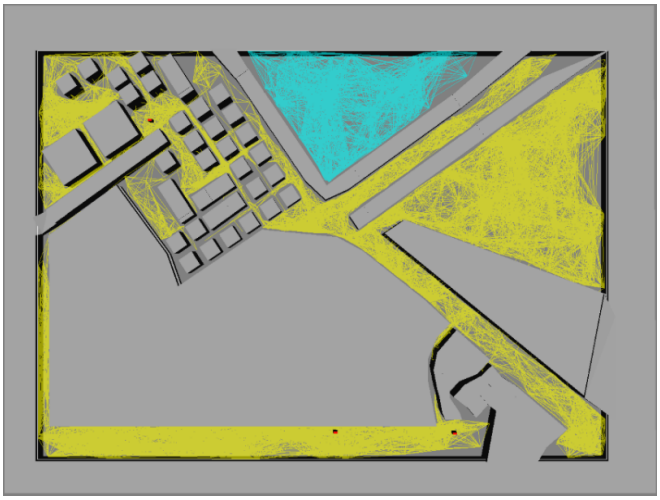

Fig. 7: The PRM* Solver Output

The most noticeable difference in Table IV is the number of edges generated. The connection stage in PRM is constant while in PRM* it decreases with time which allows a reduction of the computational effort. The path length obtained is approximately the same, so it is not a determining factor when making a decision.

TABLE IV: PRM vs PRM*

	PRM	PRM*
Samples	1305	86410
Path States	112	204
Length (Cost)	1213	1326
Edges	12321	398769

V. CONCLUSION

When taking into consideration geometric planners, from our obtained results when running the solvers in our environment we can neglect KPIECE and EST. When left with choosing between RRT and PRM it depends on the user requirements; although PRM is capable of providing shorter paths, the solver takes a longer time to run.

When we consider the control based planners, KPIECE cannot be ignored since it provides a feasible solution with a cost smaller than the other planners. This could result from the fact that KPIECE was first developed for problems involving dynamic constraints and then converted to its geometric version. Again it depends on the user's objective preference as to whether to reduce the cost or go for a more time efficient planner like RRT.

When dealing with optimizing planners, from our first experiment (RRT vs RRT*) we get the conclusion that RRT* is more suitable to problems where the user needs a high level of planning accuracy. If the user simply requires a path between to states then RRT is preferable since it is multiple times faster. Similarly, between PRM and PRM*, we choose the planner for the problem depending on the time and the computational effort. If a quicker solution is preferred over an optimal one, PRM is the ideal candidate while in the opposite case, the shortest path is obtained by using PRM*; however, it requires more computation time.

REFERENCES

[1] D. González, J. Pérez, V. Milanés and F. Nashashibi, "A Review of Motion Planning Techniques for Automated Vehicles," in IEEE Transactions on Intelligent Transportation Systems, vol. 17, no. 4, pp. 1135-1145, April 2016.

[2] Autonomous driving in urban environments: approaches, lessons and challenges Mark Campbell, Magnus Egerstedt, Jonathan P. How and Richard M. Murray Philosophical Transactions: Mathematical, Physical and Engineering Sciences Vol. 368, No. 1928, Traffic jams: dynamics and control (13 October 2010), pp. 4649-4672

[3] A Survey of Motion Planning and Control Techniques for Self-driving Urban Vehicles Brian Paden, Michal Cap, Sze Zheng Yong, Dmitry Yershov, Emilio Frazzoli

[4] A. Parodi, "Multi-goal real-time global path planning for an autonomous land vehicle using a high-speed graph search processor," Proceedings. 1985 IEEE International Conference on Robotics and Automation, 1985, pp. 161-16

[5] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," The International Journal of Robotics Research, vol. 30, pp. 846–894, 2011.

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

[7] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," Transactions on Robotics and Automation, vol. 12, pp. 566– 580, 1996.

[8] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," Transactions on Systems Science and Cybernetics, vol. 4, pp. 100–107, 1968.

[9] A. Stentz, "Optimal and efficient path planning for partially-known environments," in International Conference Robotics and Automation, pp. 3310–3317, IEEE, 1994.

[10] E. Schmerling, L. Janson, and M. Pavone, "Optimal sampling-based motion planning under differential constraints: the drift-less case," International Conference on Robotics and Automation, 2015.

[11] S. Anthony, "The focussed D* algorithm for real-time replanning.," in IJCAI, vol. 95, pp. 1652–1659, 1995.

[12] S. Koenig and M. Likhachev, "Fast replanning for navigation in unknown terrain," Transactions on Robotics, vol. 21, pp. 354–363, 2005.

[13] A Route Planning's Method for Unmanned Aerial Vehicles Based on Improved A-Star Algorithm LI Ji,SUN Xiuxia(Engineering College,Air Force Engineering University,Xi'an 710038,Shaanxi,China)

[14] D. Hsu, J.-C. Latombe, and R. Motwani, "Path planning in expansive configuration spaces," in International Conference on Robotics and Automation, vol. 3, pp. 2719–2726, IEEE, 1997.

[15] S. M. La Valle, "Rapidly-exploring random trees a new tool for path planning," tech. rep., Computer Science Dept., Iowa State University, 1998.

[16] L. E. Kavraki, M. N. Kolountzakis and J. C. Latombe, "Analysis of probabilistic roadmaps for path planning," in IEEE Transactions on Robotics and Automation, vol. 14, no. 1, pp. 166-171, Feb. 1998.

[17] T. Kunz and M. Stilman, "Kinodynamic RRTs with fixed time step and best-input extension are not probabilistically complete," in Algorithmic Foundations of Robotics XI, pp. 233–244, Springer, 2015.

[18] S. Karaman and E. Frazzoli, "Sampling-based optimal motion planning for non-holonomic dynamical systems," in International Conference on Robotics and Automation, pp. 5041–5047, IEEE, 2013.

[19] A. Colombo and D. Del Vecchio, "Efficient algorithms for collision avoidance at intersections," in Proceedings of the 15th ACM inter- national conference on Hybrid Systems: Computation and Control,

[20] J. Reeds and L. Shepp, "Optimal paths for a car that goes both forwards and backwards," Pacific journal of mathematics, 1990

[21] Z. Aljarboua "Geometric path planning for general robot manipulators," Proceedings of the World Congress on Engineering and Computer Science, vol. 02, 2009.