Towards GPU-Accelerated PRM for Autonomous Navigation

Janelle Blankenburg[†], Richard Kelley[‖], David Feil-Seifer[†],
Rui Wu[§], Lee Barford[†‡], Frederick C Harris, Jr.[†]

[†]*Dept. of Computer Science and Engineering*
University of Nevada, Reno
Reno, NV

[§]*Dept. of Computer Science*
East Carolina University
Greenville, NC

[‡]*Keysight Laboratories*
Keysight Technologies
Reno, NV

[‖]*Nevada Center for Applied Research*
University of Nevada, Reno
Reno, NV

jjblankenburg@nevada.unr.edu, rkelley@unr.edu, dave@cse.unr.edu,
wur18@ecu.edu, lee.barford@ieee.org, fred.harris@cse.unr.edu

***Abstract*—Sampling based planning is an important step for long-range navigation for an autonomous vehicle. This work proposes a GPU-accelerated sampling based path planning algorithm which can be used as a global planner in autonomous navigation tasks. A modified version of the generation portion for the Probabilistic Road Map (PRM) algorithm is presented which reorders some steps of the algorithm in order to allow for parallelization and thus can benefit highly from utilization of a GPU. The GPU and CPU algorithms were compared using a simulated navigation environment with graph generation tasks of several different sizes. It was found that the GPU-accelerated version of the PRM algorithm had significant speedup over the CPU version (up to 78x). This results provides promising motivation towards implementation of a real-time autonomous navigation system in the future.**

keywords: path planning, autonomous vehicle, probabilistic roadmap, parallel computing, speedup

I. Introduction

The primary motivation of this work is to develop an end-to-end navigation system for an autonomous vehicle. Autonomous navigation can have a strong impact on society, enabling an industry that affects many people's daily lives. This booming industry involves both academic institutions and large enterprises competing together in order to develop a feasible autonomous car. Three of the major components autonomous navigation in uncertain environments include are: the controller, roadway and obstacle detection, and path planning. While these systems will work together, each component has its own set of engineering challenges in order for it to function in real time and in the real world. For the purposes of this work, we will focus on the path planning component.

In order for an autonomous system to perform in real-world environments, this system must be able to generate a path for navigation in real-time. This quick pace is one of the main challenges of path planning algorithms for autonomous vehicles. In recent years, many autonomous cars have begun to include embedded GPUs into their systems. These GPUs can be utilized towards the effort of developing a real-time navigation system. One of the other big issues developers face when choosing a path planning algorithm is how to incorporate the dynamics of the vehicle into the plan.

Various types of path planners and dynamics models have been used for autonomous navigation. This work focuses specifically on sampling based motion planners. These methods tend to utilize both a global and a local planner. The local planner is required to help the car navigate between different way points. The global planner it used to generate this set of way points from the start to the goal. Since a local planner handles the small scale navigation and vehicle dynamics, a simplified global sampling based path planning algorithm is sufficient for generating a long-range path. For this reason, this paper explores the use of Probabilistic Road Maps (PRMs) [1] for global planning. In order to ensure this system can be incorporated into a real-time autonomous navigation system in the future, the paper proposes a GPU-accelerated PRM algorithm.

The remainder of the paper is structured as follows: Section II presents state-of-the art approaches to GPU-accelerated sampling based planning for autonomous navigation, Section III describes the details of the proposed approach, Section IV describes the experimental setup and results, Section V presents new and related directions of research, and Section VI gives a concluding summary of the presented work.

II. Related Work

The eventual goal of this work is to create a real-time planner that will be integrated into the autonomous navigation system on the University of Nevada Reno's autonomous Lincoln MKZ vehicle. For the purposes of this work, this section explores two main areas of research: state of the art methods for autonomous navigation using sampling based planners and state of the art methods for accelerating sampling based planners using a GPU.

A. Autonomous Navigation Via Sampling Based Planning

Sampling based planning has been utilized as a means for path planning in autonomous systems for over a decade. The majority of these methods use a complicated path planning function, such as Rapidly-exploring Random Trees (RRT) [2], [3], [4], [5], [6]. However, the RRT algorithm requires most if not all of the steps to be implemented on the GPU all at once in order to avoid the high overhead cost of data transfer between the CPU and GPU. Therefore, this work uses a more simplistic algorithm (PRM) which allows us to incrementally determine feasibility of a path with minimal overhead.

Significantly fewer works utilize a simple sampling based planning method, such as PRM. The works in [7] and [8] used PRMs to perform navigation tasks for an autonomous unmanned aerial helicopter. Although this work performed planning in 3D, autonomous car navigation is similar as obstacles are 3-dimensional. The work proposed in [9] utilizes PRMs for multi-robot motion planning for car-like robots. The planning methods in this work are similar to planning for autonomous navigation tasks as well.

One recent work combines PRMs and reinforcement learning for long-range robotic navigation [10]. The application of this work is most similar to our intended application. In this work, an RL agent learns short-range, point-to-point navigation polices the capture the robot dynamics, which acts as the local planner for the autonomous navigation task. Similar to the proposed work, this paper uses PRM for the global planner for navigation.

Each of these papers are focused on creating a full autonomous navigation system. They focus not only on a global planner, but also utilize local planners to ensure the navigation adheres to the dynamics of the system. Therefore, these papers focus on a different application than the proposed work, as none of these methods utilize GPUs to accelerate the global planning. Instead of focusing on ensuring correct dynamics through a local planner, the proposed work focuses on speeding up the global planner by utilizing the GPU. Creating an end-to-end autonomous navigation system is currently out of the scope of this project, but the methods proposed in this work may be utilized in such a system in the future.

B. Sampling Based Planning using a GPU

Many works utilize the GPU to speed up various sampling based methods. Some works focus on complicated methods, such as RRT [11], [12], [13]. Other works focus on simpler methods such as PRMs [14], [15], [16]. However, all of these methods focus on a different problem formulation than that proposed in our future work on the autonomous car.

The work in [17] is most similar to our proposed method. This work developed a GPU-based parallel collision detection algorithm for motion planning using PRMs. Although this work develops a very thorough algorithm for GPU-accelerated PRMs, the applications discussed in this work are slightly different than those proposed in our work. Our proposed methods is focused on speeding up the global planner for a long-range autonomous navigation system for an autonomous car. Therefore, the overhead of the work proposed in [17] is unnecessary for our application, as we are only focusing on a small portion of the planning problem. This focus on long-range navigation requires planning in very large spaces, which are not addressed in [17].

III. Methodology

The proposed work is focused on speeding up the generation portion of the PRM algorithm by utilizing the computational efficiency of the GPU. The steps of the generation portion of the PRM algorithm are shown in Algorithm 1. The general idea

Algorithm 1 Main steps of the generation portion of the PRM algorithm.

```
1: Graph G is empty
2: while number of vertices in G < N do
3:     generate random configuration q
4:     for each q, select k closest neighbors do
5:         for each neighbor q' do
6:             local planner connects q to its neighbors
7:             if connection is collision free then
8:                 add edge (q, q') to G
9:                 add vertex q to G
```

behind the PRM algorithm is to generate a set of random points in the configuration space (i.e. generate a configuration) that are not in collision with the obstacles in the space. Then each point is connected to its k-nearest neighbors, thus generating a graph of the configuration space which can later be used for planning during the query phase of PRM. For the purposes of this work, we are not discussing the query phase of PRM, as we only provide modifications to the generation phase for PRM.

In order to develop a GPU-accelerated PRM algorithm, a few minor changes have to be made to this algorithm. These changes are focused on changing the order of the steps in order to allow for parallelization in the code. The GPU-accelerated PRM algorithm is shown in Algorithm 2.

Algorithm 2 Modified version of the generation portion of the PRM algorithm to allow for parallelization.

```
1: Graph G is empty
2: Target vector t is empty
3: generate N random configurations q
4: for each q, check for collisions do
5:     if point is collision free then
6:         save q into t
7: for each q in t, find k closest neighbors do
8:     if connection is collision free then
9:         add edge to G
10:        add vertex to G
```

Several important remarks about the implementation of the GPU-accelerated PRM algorithm are discussed in the following subsections.

A. Data Transfer Overhead

In the current version of the GPU-accelerated algorithm, the random configurations are generated on the CPU. They are then transferred over to the GPU, and are used on the GPU for the remainder of the algorithm. The rest of the algorithm is run on the GPU so there is almost no other data transfer overhead for generating the graph using this method. The only caveat here is discussed in Section III-C, where we see there is an additional minimal overhead to maintain the graph as it is generated.

B. Utilizing Thrust Functions

In order to gain the most efficiency from utilizing the GPU, several of the built in thrust functions were used. In order to perform the initial collision check, the following steps were taken:

1) use thrust::count_if to count the number of configurations that are not in collision with the obstacle in the scene, and
2) use thrust::copy_if to save the non-colliding configurations to the target vector t

The count function is necessary to allocate a target vector of the correct size to minimize the memory overhead required by the remainder of the algorithm. The utilization of the thrust functions allowed for a simple and efficient way to filter out the configurations that were in collision with the obstacle in the scene. Therefore, we are only left with configurations in the target vector which are possible candidates for nodes in the generated graph.

In order to find the nearest neighbors of the a given configuration q in the vector t, the following steps were taken:

1) use thrust::transform to calculate the distance from each configuration in t to q
2) use thrust::sort_by_key to sort the configurations by their distance to q, and
3) get the first k sorted configurations, which correspond to the k-nearest neighbors

By utilizing the transform function, we were able to get the distance from one configuration to the rest very quickly. By next applying a sorting function, we were able to quickly pull out the k-nearest neighbors by simply taking the portion of the data which corresponded to the closest configurations based on their distances to the given configuration. This manner of k-nearest neighbors search should be very efficient for large graphs, due to the parllelization of both the distance check and sorting via the thrust functions.

This process is then repeated for every configuration q in t, in order to get the k-nearest neighbors for each configuration in t. This repetition is done by iterating through this configurations, calling the above process each time. This iterative portion is the slowest part of the proposed algorithm. With additional data storage, this portion might benefit from further parallelization in the future.

C. Storage of the Graph

The last important remark about this algorithm is that the graph G is stored on the CPU. In order to do this, once we have the k-nearest neighbors for a given configuration q, we have to copy these from the GPU back to the CPU. This results in an additional data transfer overhead in our algorithm, as briefly mentioned in Section III-A. However, as we get to large graphs, this transfer becomes minimal since each configuration only transfers k neighbors. We can then store the edges between these neighbors and the given configuration q in the graph. Additionally, we store the given configuration q as a vertex in the graph. Thus, our graph is made up of a list of vertices and edges like normal. Storing the graph on the CPU is very useful for visualization purposes. Visualizing the generated graph is done on the CPU, so storing the graph directly to the CPU as it is generated has much less overhead since there is minimal data transfer between the CPU and the GPU.

IV. Experimental Validation

A. Experimental Setup

In order to evaluate the performance of the GPU-accelerated algorithm, we performed several different test cases in which we compared a CPU version of the algorithm with the GPU-accelerated version. These were done on an MSI MS-16H2 laptop with an Intel 4 core i7 CPU (i7-4710HQ) with 16GB of RAM and an NVIDIA GeForce GTX 860M with 4GB of NVRAM and 1152 cores. In order to allow for a more direct comparison, the CPU version also utilizes the parallelized version of the PRM algorithm. Due to the use of thrust vectors and functions, the only major differences between the CPU and GPU version is whether the thrust vectors are stored on the host or the device. In the CPU version, all of the thrust vectors are stored directly on the CPU and no data transfer occurs to the GPU. The details of the GPU version are discussed in Section III.

The CPU and GPU-accelerated algorithms were compared in a simulated navigation environment. This environment consists of a single large obstacle in the center of the image. The goal is to build a graph around the obstacle which can be used to generate a navigation path through the space later on. A sample of the environment is shown in Figure 1 with a generated graph of 1,000 nodes.

In order to evaluate the performance of the algorithms, we used them to generate graphs with different numbers of nodes. We looked at how long it took to generate a graph with 10, 100, 1,000, 10,000, and 100,000 nodes. However, due to the incredibly long runtime of the CPU algorithm with 100,000 nodes ($>$ 5 hours), we chose to omit running this scenario on the CPU. Therefore, the case for generating a graph with 100,000 nodes is only run and analyzed for the GPU-accelerated version of the algorithm. Thus, in all of the figures, the scenario for the CPU with 100,000 nodes is left blank, and in the tables it is marked as N/A. Within each case, we averaged the runtime over 5 trials. The base environment was consistent across all trials.

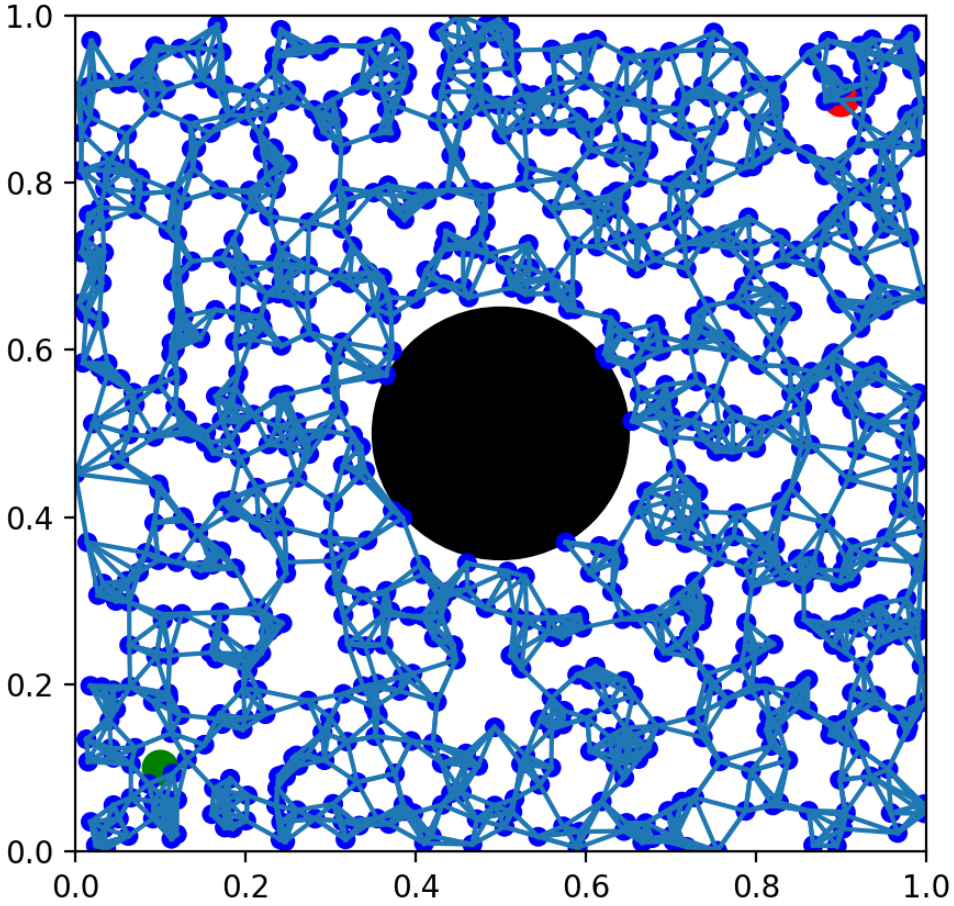

Fig. 1. A sample image of the simulated navigation task. The black sphere in the center is the obstacle in the scene. The green dot in the lower left is the starting point and the red dot in the upper right is the ending point for the navigation task. The dots and lines represented a graph consisting of 1,000 nodes that was generated by the GPU-accelerated algorithm.

B. Results and Discussion

The comparison of runtimes between the CPU and GPU-accelerated PRM algorithm for the different scenarios are shown in Table I and Figure 2. Additionally, we looked at a simple version of throughput across these different cases as well. In order to compute throughput, we simply divided the number of nodes generated by the runtime. This gives us a rough estimate of how many nodes can be generated per second. The throughput is shown in Table II and Figure 3. Lastly, we looked at the speedup of the GPU with respect to the CPU version. The speedup is shown in Table III and Figure 4.

From Table I we see that in large cases the GPU-accelerated version ran significantly faster, but in very small cases, the CPU version greatly outperformed the GPU version. This results is what we expected; as the graph grows in size, the GPU should perform faster than the CPU version. This result is further illustrated in Figure 2. We see that the slope of the runtime line for the GPU version is much less than the slope of the runtime line for the CPU version. This illustrates that as the number of nodes continue to grow, the difference in runtimes should continue to grow as well, implying that the GPU version performs better on larger graphs.

From Figure 3 we see strong trends in the CPU and GPU throughput. For small graphs, the CPU version has high throughput, but it quickly diminishes. On the other hand, the GPU version has small throughput to start and it gets better as the number of nodes increases, up until the 100,000 case. At this point, we see the throughput decreases slightly. Based on this information, the GPU algorithm works well for large graphs, but the performance decreases if the graphs get too large. The exact values of the throughput can be seen better in Table II.

In order to further analyze the performance of these methods, we compare them to calculate the speedup of the GPU-

TABLE I

CHART OF THE RUNTIME IN **SECONDS** AVERAGED OVER 5 TRIALS OF BOTH THE CPU AND GPU-ACCELERATED VERSION OF THE MODIFIED PRM ALGORITHM. THE FIRST ROW SHOWS WHICH SCENARIO WAS RUN (N = 10, ..., 100,000). THE NEXT TWO ROWS SHOW THE TIMES FOR THE CPU AND GPU RESPECTIVELY.

N	10	100	1,000	10,000	100,000
CPU	0.0042	0.0524	2.312	224.6414	N/A
GPU	0.1204	0.1326	0.396	2.8516	44.5028

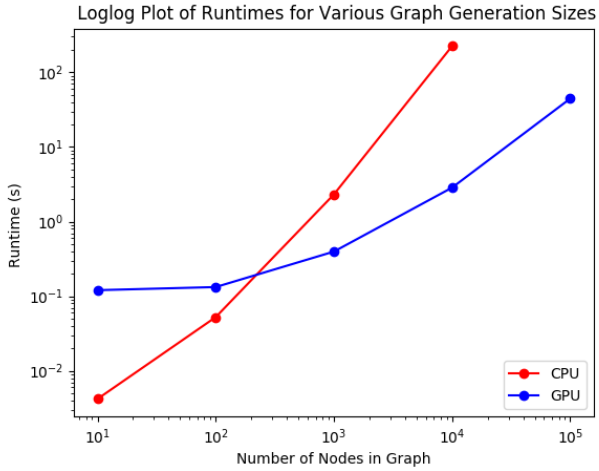

Fig. 2. A plot of the runtime in seconds of both the CPU and GPU-accelerated version of the modified PRM algorithm.

TABLE II

CHART OF THE THROUGHPUT OF BOTH THE CPU AND GPU-ACCELERATED VERSION OF THE MODIFIED PRM ALGORITHM. THROUGHPUT IS CALCULATED BY TAKING THE NUMBER OF NODES IN THE GRAPH DIVIDED BY THE RUNTIME.THE FIRST ROW SHOWS WHICH SCENARIO WAS RUN (N = 10, ..., 100,000). THE NEXT TWO ROWS SHOW THE THROUGHPUT FOR THE CPU AND GPU RESPECTIVELY.

N	10	100	1,000	10,000	100,000
CPU	2380.95	1908.40	432.53	44.51	N/A
GPU	83.06	754.15	2525.25	3506.80	2247.05

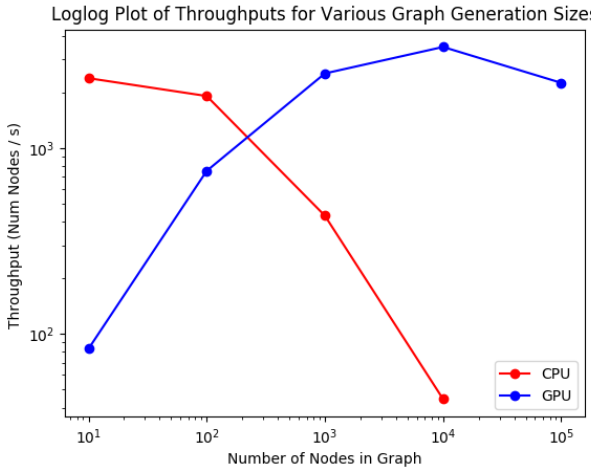

Fig. 3. A plot of the throughput of both the CPU and GPU-accelerated version of the modified PRM algorithm. Throughput is calculated by taking the number of nodes in the graph divided by the runtime.

accelerated version with respect to the CPU version. The speedup for each scenario is shown in Table III. The results in this table reflect what we saw in the runtime and throughput tables/plots. For small graphs, the GPU performs worse due to the overhead of the initial data transfer. As the graphs get larger, this overhead becomes minimal resulting in a significant speedup of the GPU version with respect to the CPU version. We see that for large graphs (N = 10,000) the GPU-accelerated version of the algorithm results in a speedup of 78x. From Figure 4 we see that we have nearly linear speedup on a loglog scale as the number of nodes in the graph increases. This shows that although the throughput went down a bit on the largest size graph, the speedup would probably still be significant for very large graphs. Unfortunately, due to extremely long run time, we were unable to run the CPU version for the 100,000 case so we cannot get a speedup to prove that the trend of a linear speedup on a loglog scale continues for this case.

Overall, from the experiments, we see that the GPU-accelerated version is much faster and more efficient for large graphs than the CPU version. Therefore, for the future application of long-range autonomous navigation, this algorithm will be very beneficial as the graphs necessary for path planning in these applications will be very large.

V. FUTURE WORK

This work presents several important directions for future work. In order to further increase the speedup from the GPU-accelerated PRM algorithm, the random configuration generation will be performed on the GPU instead of the CPU. This will eliminate the overhead of transferring the data from the CPU to the GPU in the first step of the algorithm. In the future, this algorithm will be incorporated into an end-to-end autonomous navigation system. This component will be the global planner used for long-rage navigation tasks performed by the University of Nevada Reno's autonomous Lincoln MKZ vehicle. This GPU-accelerated algorithm will help to ensure that the autonomous navigation system performs in real-time. If it is found that the speedup from this algorithm is not enough to perform navigation tasks in real-time, several other parallelizations of the algorithm could be programmed later such as those described in [17]. Additionally, the current version of the algorithm can be modified to include extra data storage to allow for the k-nearest neighbor search to be parallelized to run simultaneously across all configurations at once to help speed up the algorithm.

VI. CONCLUSION

This work is the first step towards a real-time autonomous navigation system. The focus is on speeding up a sampling

TABLE III

CHART OF THE SPEEDUP OF THE GPU-ACCELERATED VERSION OF THE MODIFIED PRM ALGORITHM WITH RESPECT TO THE CPU VERSION. SPEEDUP IS CALCULATED BY TAKING THE RUNTIME OF THE CPU VERSION / RUNTIME OF THE GPU VERSION. THE FIRST ROW SHOWS WHICH SCENARIO WAS RUN (N = 10, ..., 100,000). THE NEXT ROW SHOWS THE SPEEDUP OF THE GPU ALGORITHM.

N	10	100	1,000	10,000	100,000
Speedup	0.035	0.395	5.84	78.82	N/A

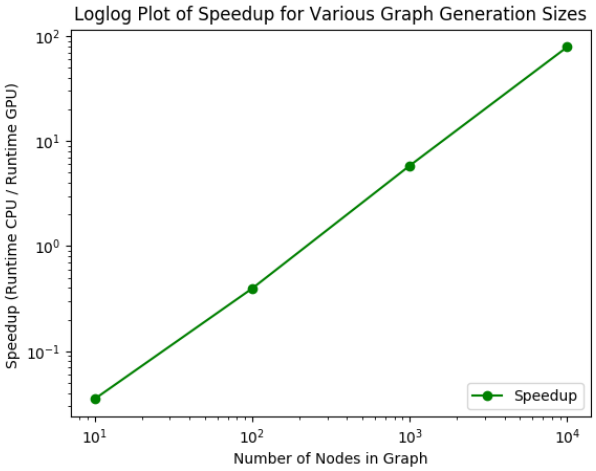

Fig. 4. A plot of the speedup of the GPU-accelerated version with respect to the CPU version of the modified PRM algorithm.

based path planning algorithm for long-range navigation. This work proposes a GPU-accelerated sampling based planner which can be used as a global planner in autonomous navigation tasks. The sampling based path planning algorithm explored in this work is the PRM algorithm. A modified version of the generation portion for the PRM algorithm is presented. Both a CPU and GPU-accelerated version of this algorithm were evaluated using a simulated navigation environment with graph generation tasks of several different sizes. Based on these experiments, we found that the GPU-accelerated version of the PRM algorithm had significant speedup (up to 78x) over the CPU version. This result is the first step towards the implementation of a real-time autonomous navigation system in the future.

VII. ACKNOWLEDGEMENTS

This material is based in part upon work supported by the National Science Foundation under grant numbers IIA-1301726 and IIS-1719027. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[2] L. Ma, J. Xue, K. Kawabata, J. Zhu, C. Ma, and N. Zheng, "Efficient sampling-based motion planning for on-road autonomous driving," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 4, pp. 1961–1976, 2015.

[3] J. hwan Jeon, R. V. Cowlagi, S. C. Peters, S. Karaman, E. Frazzoli, P. Tsiotras, and K. Iagnemma, "Optimal motion planning with the half-car dynamical model for autonomous high-speed driving," in *2013 American Control Conference*. IEEE, 2013, pp. 188–193.

[4] D. Braid, A. Broggi, and G. Schmiedel, "The terramax autonomous vehicle," *Journal of Field Robotics*, vol. 23, no. 9, pp. 693–708, 2006.

[5] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, and J. P. How, "Real-time motion planning with applications to autonomous urban driving," *IEEE Transactions on Control Systems Technology*, vol. 17, no. 5, pp. 1105–1118, 2009.

[6] J.-H. Ryu, D. Ogay, S. Bulavintsev, H. Kim, and J.-S. Park, "Development and experiences of an autonomous vehicle for high-speed navigation and obstacle avoidance," in *Frontiers of Intelligent Autonomous Systems*. Springer, 2013, pp. 105–116.

[7] P. O. Pettersson and P. Doherty, "Probabilistic roadmap based path planning for an autonomous unmanned aerial vehicle," in *Proc. of the ICAPS-04 Workshop on Connecting Planning Theory with Practice*, 2004.

[8] ——, "Probabilistic roadmap based path planning for an autonomous unmanned helicopter," *Journal of Intelligent & Fuzzy Systems*, vol. 17, no. 4, pp. 395–405, 2006.

[9] Z. Yan, N. Jouandeau, and A. A. Cherif, "Acs-prm: Adaptive cross sampling based probabilistic roadmap for multi-robot motion planning," in *Intelligent Autonomous Systems 12*. Springer, 2013, pp. 843–851.

[10] A. Faust, O. Ramirez, M. Fiser, K. Oslund, A. Francis, J. Davidson, and L. Tapia, "PRM-RL: long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning," *CoRR*, vol. abs/1710.03937, 2017. [Online]. Available: http://arxiv.org/abs/1710.03937

[11] C. Park, J. Pan, and D. Manocha, "Realtime gpu-based motion planning for task executions," in *IEEE International Conference on Robotics and Automation Workshop on Combining Task and Motion Planning (May 2013)*. Citeseer, 2013.

[12] J. Bialkowski, S. Karaman, and E. Frazzoli, "Massively parallelizing the rrt and the rrt," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2011, pp. 3513–3518.

[13] S. A. Jacobs, N. Stradford, C. Rodriguez, S. Thomas, and N. M. Amato, "A scalable distributed rrt for motion planning," in *2013 IEEE International Conference on Robotics and Automation*. IEEE, 2013, pp. 5088–5095.

[14] A. Bleiweiss, "Gpu accelerated pathfinding," in *Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*. Eurographics Association, 2008, pp. 65–74.

[15] L. G. Fischer, R. Silveira, and L. Nedel, "Gpu accelerated path-planning for multi-agents in virtual environments," in *2009 VIII Brazilian Symposium on Games and Digital Entertainment*. IEEE, 2009, pp. 101–110.

[16] J. T. Kider, M. Henderson, M. Likhachev, and A. Safonova, "High-dimensional planning on the gpu," in *2010 IEEE International Conference on Robotics and Automation*. IEEE, 2010, pp. 2515–2522.

[17] J. Pan and D. Manocha, "Gpu-based parallel collision detection for fast motion planning," *The International Journal of Robotics Research*, vol. 31, no. 2, pp. 187–200, 2012.