Trajectory Planning for Automated Vehicles using Quantum Computing

Lilja, Juha

Published in:
2023 IEEE 19th International Conference on Intelligent Computer Communication and Processing (ICCP)

Accepted/In press: 01/01/2023

Document Version
Peer reviewed version

Link to publication

Please cite the original version:
Lilja, J. (in press). Trajectory Planning for Automated Vehicles using Quantum Computing. In *2023 IEEE 19th International Conference on Intelligent Computer Communication and Processing (ICCP)* IEEE Institute of Electrical and Electronic Engineers.

Trajectory Planning for Automated Vehicles using Quantum Computing

Juha Lilja
VTT Technical Research Centre of Finland Ltd.
Tampere, Finland
juha.lilja@vtt.fi

***Abstract*—This paper presents a method for automated vehicle trajectory planning using quantum computing. The use of quantum computing for the trajectory planning problem was studied because the current classical methods for solving the problem are computationally expensive. Simulation tests with the presented method were performed in more complex test cases than what has previously been tested with this method. Simulation tests show, that the method scales also for more complex problems. The method presented here can theoretically outperform classical methods in complexity in certain conditions, although it is not guaranteed in every condition.**

***Index Terms*—trajectory planning, quantum computing, automated vehicles**

I. Introduction

Finding smooth, collision-free, and efficient trajectories that a vehicle can follow and use to navigate through its environment autonomously is an important task in designing automated vehicles. While there are multiple methods that can be used for trajectory planning, solving the problem is a computationally expensive task. Therefore computational resources available can be a limiting factor in finding an optimal trajectory.

In quantum computing, quantum phenomena such as superposition and entanglement are used to enable new ways of performing computational tasks. Quantum computing cannot provide an advantage over classical computing in every problem, but it can bring an advantage for certain problems. While quantum computers are still in an early development stage, possible use cases have already been considered for a long time.

In this paper, a method for trajectory planning using quantum computing is presented. The method is based on my previous work [1], which introduces a quantum algorithm for trajectory planning. This paper shows by simulation that the algorithm scales and it is possible to use the algorithm in a more complex environment than what was used in the previous work.

II. Background

In this section, the basics of state lattices and quantum computing are presented to form a basis for the method used in this paper.

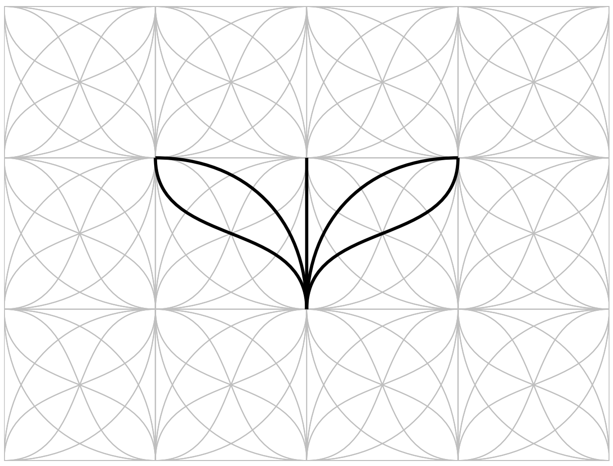
Fig. 1. An illustration of a state lattice. The basic set of primitives is highlighted in black. This set of primitives is copied and shifted to start also from other configurations, as shown in grey, which creates a regular pattern. Configurations are the nodes where individual primitives start and end.

A. Trajectory Planning using State Lattices

There are multiple methods for trajectory planning with classical computers. These can be divided generally into two approaches which are decoupled methods and direct methods [2]. In decoupled methods, trajectories are created in two steps. In the first step, a path is created. While a path is a geometrical representation of a route, a trajectory should also include information about timing, that is velocities at which the vehicle should follow the trajectory. Thus, the second step is to add information about timing to the path so that it is transformed into a trajectory. In direct methods a single step is used to generate the trajectory and no separate path planning step is used. Within these two categories, multiple methods such as using potential fields, rapidly-exploring random trees, or state lattices can be used. The method presented in this paper is also based on state lattices so this method is briefly presented next.

The state lattice can be viewed as a generalization of a grid [3]. To create a state lattice, a configuration space C must first be defined. The C-space consists of all different configurations (states) the vehicle can exist in. Each configuration includes parameters such as x- and y-coordinates, heading, curvature, and velocity of the vehicle. Also, higher-order parameters can be added and the specific set of parameters can be chosen depending on the problem. Adding more parameters may lead to more accurate and optimal trajectories as the capabilities of the vehicle can be taken better into account. However, each parameter within a configuration is an additional dimension in the C-space, which increases the computational complexity of the problem. The parameters of which configuration consists can be chosen depending on the requirements of the vehicle, how accurate trajectories are needed, and how much computing resources are available. If the C-space consists only of the spatial coordinates, the problem is essentially a path planning problem. When a velocity is introduced as a dimension to the C-space, the problem is transformed into a trajectory planning problem as the resulting paths include timing information from velocities. To account for dynamic environments with moving obstacles, time can be added as a dimension to the C-space to create a spatiotemporal state lattice [4].

For obstacle avoidance, two subspaces C_{obs} and C_{free} are constructed from the C-space. The subspace C_{obs} consists of all configurations that are occupied by an obstacle and C_{free} consists of every configuration that is not occupied by an obstacle. Valid trajectories can only use configurations within C_{free} so that collisions are avoided.

To create a state lattice, the C-space is discretized along each dimension. The discrete configurations near each other are connected with short trajectories, primitives. Each primitive is designed so that it can be accurately followed by the vehicle and each primitive is leading the vehicle from one configuration to another configuration in the discrete C-space. The same set of primitives can be shifted to start from different configurations, which creates a regular pattern, a state lattice. Longer trajectories are then created by applying multiple primitives after each other. In the Fig. 1, a simple illustration of a 3-dimensional state lattice can be seen, where the dimensions are spatial coordinates x and y as well as heading. The heading dimension in the figure is projected into the x-y plane. State lattices can also be represented as a graph where configurations within the C-space are nodes and those are connected by primitives which represent edges in the graph. From a starting configuration c_s, any graph search algorithm can then be applied to find a valid trajectory to a target configuration c_t within the state lattice [3]. This paper presents a method of finding a valid trajectory using quantum computing.

B. Quantum Computing

In quantum computing, instead of bits, quantum bits or qubits are used. A state of a qubit v can be written using bra-ket notation as $|v\rangle$ [5]. Qubits can exist in so called computational basis states $|0\rangle$ and $|1\rangle$, which are corresponding to 0 and 1 states of classical bits. Qubits are different from classical bits in that they can also exist in a superposition of these basis states. The state of a qubit can be presented in a general form of

$$|v\rangle = c_0 |0\rangle + c_1 |1\rangle , \tag{1}$$

where $[c_0, c_1] \in \mathbb{C}$ and $|c_0|^2 + |c_1|^2 = 1$. The qubit may exist in a superposition state only when its state is not observed. Measuring the state of a qubit affects the state of the qubit so that it collapses into one of the basis states $|i\rangle$ with a probability of $|c_i|^2$ [6].

A quantum register consists of multiple qubits. The basis states of an n-qubit quantum register are

$$\{|00\ldots00\rangle , |00\ldots01\rangle , \ldots , |11\ldots10\rangle , |11\ldots11\rangle\}. \tag{2}$$

A quantum register can exist in any superposition between these basis states, the state being then

$$c_0 |00\ldots00\rangle + c_1 |00\ldots01\rangle + \cdots + c_{2^n-1} |11\ldots11\rangle , \tag{3}$$

where $[c_0, \ldots, c_{2^n-1}] \in \mathbb{C}$ and $|c_0|^2 + \cdots + |c_{2^n-1}|^2 = 1$. The measurement of a quantum register affects the register the same way as with a single qubit so that the register collapses into one of the basis states $|i\rangle$ with the probability of $|c_i|^2$.

Quantum computation is performed by feeding qubits through a quantum circuit consisting of quantum gates. A quantum gate is an operation in quantum computing that operates on one or multiple qubits and changes the state of those qubits according to the definition of the gate. Quantum gates are always reversible, meaning that for every quantum gate U, there exists a quantum gate $U^\dagger$, which reverts the effect of U [6]. So applying U and $U^\dagger$ immediately after each other leaves the state of the qubits they are operating on unchanged. Quantum circuits are created by applying multiple quantum gates to create the desired computation. A quantum circuit usually ends in a measurement of the qubits to read out the result of the computation. The measurement operation is not reversible, i.e. the state of the system before measurement cannot be restored after the measurement is performed [6].

C. Grover's Algorithm

The method in this paper uses a well known quantum algorithm called Grover's algorithm. Grover's algorithm can be used for unstructured search and it was introduced by Lov Grover in 1996 [7]. Two main operations, oracle O and diffuser D, are used to implement Grover's algorithm.

The oracle O takes in a quantum register x and one additional qubit s. The items in the list in which the search is performed, are indexed and x represents the index of an item. The items that one is trying to find are referred to here as solutions. The O operation is designed uniquely for each problem so that, when x is set to a basis state corresponding to an item in a list and s is initialized as $|0\rangle$, the qubit s is flipped to $|1\rangle$ if x is a solution state, i.e. item that one is trying to find, or s is left as $|0\rangle$ otherwise. The O is used in Grover's algorithm by initializing x into an equal superposition of all the basis states and initializing s as $|-\rangle = \frac{1}{\sqrt{2}} |0\rangle - \frac{1}{\sqrt{2}} |1\rangle$. With these initial states, the O affects the superposition state of x so that for each solution state the coefficient c_i as in (3) becomes $-c_i$, which is sometimes called phase kickback effect [8]. So as a result, x is now in a similar superposition as before O operation but with coefficients c_i of each solution state being negative.

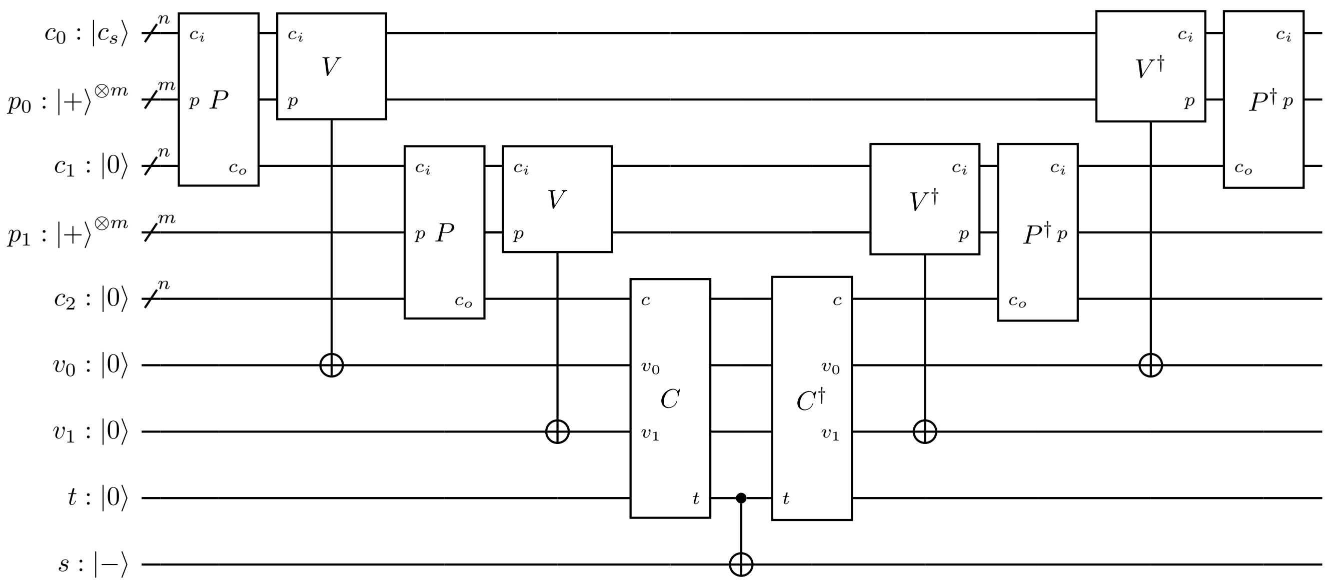

Fig. 2. Circuit of an oracle O.

After applying the oracle O, diffuser D is applied. While O is uniquely designed for each problem, D operation is implemented the same way regardless of the problem. The D operation takes the x register from O and one additional qubit in state $|-\rangle$. The effect of a diffuser is that it amplifies the coefficients c_i which are negative after O. So as a result, x is in some superposition state which is the form of (3), and the coefficients c_i of solution states after applying D are greater than coefficients of other basis states. This means, that for each solution state, the probability of it to become a result of a measurement is greater than the corresponding probability of a non-solution state.

The oracle and diffuser can be applied multiple times to further increase the probability of a solution state being measured by amplifying the coefficients of the solution states. The optimal number for iterations is

$$T \approx \frac{\pi}{4}\sqrt{\frac{2^n}{M}}, \tag{4}$$

where M is the number of solutions and n is the number of qubits in the x register [6], [9]. If O and D are applied more than T times, the probability of a solution state to be measured starts to decrease. The complexity is $O(\sqrt{N})$ for Grover's algorithm when considering the number of evaluations of the oracle [7]. Because of this, Grover's algorithm could outperform classical algorithms in certain problems.

III. Method

The method for trajectory planning in this paper uses state lattices and for finding a valid trajectory, Grover's algorithm is used. A trajectory is considered here as valid when it reaches a target configuration c_t from the starting configuration c_s without colliding with obstacles or going outside of the finite C-space. Target configuration c_t is a configuration within the C-space which is a desired configuration for the vehicle to end up. There can be multiple target configurations set at the same time to any of which the vehicle should end up.

The method presented here is based on my previous work on trajectory planning with quantum computing [1], which in turn is building on [10], which presents a way to solve a path planning problem using quantum computing.

A. Circuit

The oracle O can be designed using an approach presented in [11] for production systems. Here, the oracle consists of three different operations, named P, V, and C operations. The P operation is used to compute a resulting configuration based on a previous configuration and a primitive. The V operation checks if following the latest primitive can be done, i.e. vehicle does not collide with any obstacle and does not leave the finite C-space while following the primitive. In [1], P and V operations are one operation, but here these two are separated into their own operations to keep the implementation more clear even with more complex C-space. The C operation checks whether a target configuration was achieved with a valid trajectory. Each operation is explained in more detail in the next paragraphs.

The P operation is the main operation in constructing the trajectory from multiple primitives. As every quantum operation needs to be reversible, they need to have an equal number of qubits going in and coming out of the operation. However, in many cases, qubits can be divided into logical inputs and outputs based on their purpose, and from this point on, when referring to input or output registers, logical input and output are meant. The P-operation takes as an input an n-qubit register c_i, which is an input configuration, and an m-qubit register p, which represents a primitive. As an output, P operation gives a configuration c_o, which is an n-qubit register. The c_o is computed based on the input configuration c_i and the primitive p, so that p leads the vehicle to c_o, when starting from

c_i. Multiple P operations can be applied one after another by feeding c_o from the previous P-operation as c_i to the next P operation as shown in Fig. 2 with two P operations. Naturally, more P operations can be applied the same way to create longer trajectories, as one primitive is added to the trajectory with each P operation.

After every P operation, a V operation is applied. The V operation takes the same inputs as the P-operation, which are c_i and p, and gives as an output one qubit v. The V operation checks if primitive p can be followed from c_i and sets v as the result of the evaluation. If following p from c_i would cause a collision with an obstacle or lead the vehicle outside of the finite C-space, v would be set to $|1\rangle$. Otherwise v is set to $|0\rangle$. While designing primitives, it is analyzed to which configurations in relation to the starting configuration, the vehicle collides with. The V operation then uses this information about primitives to check whether the vehicle collides or not.

After applying all primitives, a C operation is then applied to check if the trajectory leads to a target configuration. The C operation takes as an input c_o configuration from the last P operation and all v qubits from V operations as is shown in Fig. 2. The C operation then outputs one qubit t which is set to $|1\rangle$ if c_t is reached with no collisions or invalid primitives. Otherwise, if c_t is not reached, or the trajectory is invalid otherwise, t is set to $|0\rangle$.

After C operation, the t and s qubits are connected with a $CNOT$ gate. The $CNOT$ gate operates so that when the control qubit marked with $\bullet$ is $|1\rangle$, the gate flips the target qubit marked with $\oplus$. When the target qubit, in this case s, is in state $|-\rangle$, the $CNOT$ gate causes the phase kickback which is the desired effect of the oracle. After this, everything is uncomputed by applying reverse operation of every applied operation in the reverse order. The inverse operations are needed to propagate the kickback effect of the $CNOT$ operation to each of the primitive registers. The full circuit of O can be seen in the Fig. 2 in a case with two primitives. More P and V operations can, and should, be added to create longer trajectories. The O can now be used in Grover's algorithm as explained in the section II-C. All primitive registers are set into an equal superposition between all basis states, denoted as $|+\rangle^{\otimes m}$ in Fig 2, and s is initialized as $|-\rangle$. The oracle O now affects primitive registers so that for each solution state, the coefficient c_i is flipped to $-c_i$.

The exact construction of P, V and C operations depend on the C-space and primitives chosen for each case. These operations can be designed using $CNOT$ gates as described in [12].

After the oracle, a standard diffuser D is applied to complete the amplitude amplification. The D takes as input all primitive registers, as well as the s qubit. Applying O and D can then be repeated to further amplify the coefficients of the solution states as discussed about iterations in section II-C. Finally, the primitive registers can be measured. Because of the effects of O and D, a set of primitives that form a valid trajectory is coming as a result with high probability. If the result is still an invalid trajectory, the full process of iterating O and D is repeated until a valid one is found.

B. Details

For finding a trajectory from a state lattice, this method could outperform classical graph search algorithms in computation performance if $b_{avg} > \sqrt{b_{max}}$, where b_{avg} is the average branching factor of the classical graph search algorithm and b_{max} is the maximum branching factor, i.e. the number of primitives starting from each configuration in the C-space [10]. This follows from the complexity of $O(\sqrt{N})$ of Grover's algorithm [7]. The b_{avg} may be smaller than b_{max} in the implementation of a graph search algorithm because the graph search algorithm does not necessarily need to check every possible primitive starting from a configuration.

This method does not guarantee any optimality of the resulting trajectory so far. The resulting trajectory is merely a random one from all the possible valid trajectories from c_s to c_t. In [13], a method for optimizing the solution is presented. Optimality is increased by first running the search as usual and when a valid trajectory is found, a cost is calculated for it. In this case, the C operation is then modified so that it computes a cost for trajectories and only marks trajectories as valid if they have a smaller cost than the previously found best solution. This way, a more optimal trajectory can be found with the cost of running the search multiple times.

The number d of P operations used in the circuit determines how many primitives each resulting trajectory is consisting of. However, a target configuration may be reached also with a trajectory consisting of fewer number of primitives than d. This is why a null primitive should always be included in the set of primitives used. Applying a null primitive keeps the vehicle's configuration unaltered and it can just be discarded from the trajectory if the resulting trajectory includes any null primitives. This way, by including null primitives, the number of non-null primitives in the trajectory can be less than d and the method can find shorter than d valid trajectories. This actually has a side effect of shorter trajectories having a greater probability of becoming as a result of a measurement. This is, because if a trajectory contains for example 1 null primitive, the null primitive may exist in d different places along the trajectory. Now, each of the trajectories that differ from each other only by the position of the null primitive along the trajectory, are considered unique by Grover's algorithm and they have the same high probability of becoming as a result. However, after discarding null primitives from the result, these trajectories are exactly equal to each other and so many solutions lead to the same result trajectory when null primitives are discarded. This means that it is more probable for a trajectory consisting of few non-null primitives to become a result than a trajectory with many non-null primitives. This may usually be also a desired property, because many times a trajectory with fewer non-null primitives may be more optimal in the sense of distance or elapsed time. However, it is not universally true that a trajectory with fewer primitives is more optimal.

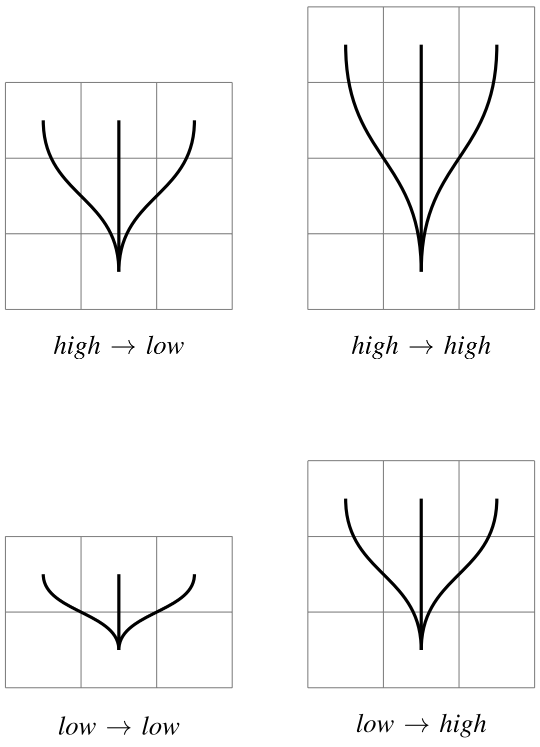

Fig. 3. Primitives used in tests. Each box represents a configuration and black lines between these configurations are the primitives. Below each set is written a velocity at the start of the corresponding primitives and a velocity the primitive leads to. Also null primitive is included in the set of primitives.

It may not be straightforward to determine a reasonable number d of P-operations to apply. By choosing d to be too low, there might not exist a solution from c_s to c_t because of too few primitives. However, choosing a large d may lead to unnecessarily high computational cost for the problem. It may be possible to estimate a reasonable d based on the distance between c_s and c_t within the C-space and information about obstacles within C-space. As this depends heavily on the chosen C-space and the chosen set of primitives, considering solutions to this problem is left out of this paper and the problem is merely addressed here.

Usually, the exact number M of valid trajectories from c_s to c_t is not known. This raises a question of how should the number of iterations be decided as it is not possible to calculate it directly with (4) when the M is not known. An algorithm presented in [14] can be used to solve this. First, a small number of iterations is used. If no valid solution is found, the number of iterations is chosen at random from a larger range of numbers and the search is repeated. This is repeated until a valid solution is found. The complexity of the algorithm still stays as $O(\sqrt{N})$ [14].

IV. Tests

For testing, a circuit for a simple C-space was implemented using Qiskit [15]. The circuit was then simulated on classical hardware using a matrix product state simulator. The C-space used in tests was a 3-dimensional space with spatial coordinates x and y as well as velocity as dimensions. As simulating quantum circuits on classical hardware is a computationally expensive task, the C-space had to be small. Therefore, the C-space included x-coordinates in range $[0\dots3]$, y-coordinates in range $[0\dots7]$ and two different velocities, *low* and *high*.

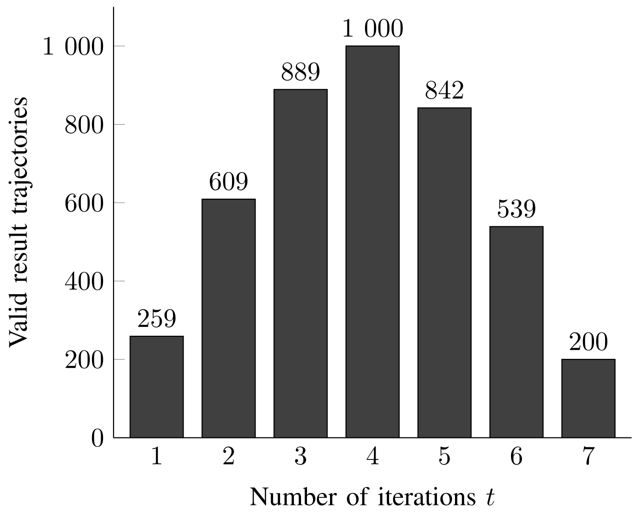

Fig. 4. Simulation test results.

The number of primitives starting from each configuration was 7, including the null primitive. For representing primitives, 3 qubits were used. Depending on if the starting configuration was a *low* velocity configuration or *high* velocity configuration, primitives used were different. The set of primitives is shown in Fig. 3.

In the test scenario, starting configuration $x = 0, y = 0$ and *low* velocity was used. For target configurations, $x = 1, y = 5$ with any velocity were chosen. Depth was chosen to be 3, that is, each resulting trajectory consists of 3 primitives. To keep the simulation simple enough for the used hardware to compute, no obstacles were used. With this test arrangement, there were 16 valid trajectories from c_s to c_t, when counting trajectories with null primitives in different places along the trajectory as unique trajectories. Using (4), it can be calculated that the optimal number of iterations in this case was 4, as 3 qubits were used for each of the 3 primitives. Tests were run with the number t of iterations in the range $[1\dots7]$. The test case with each iteration value was run 1000 times without repeating the computation if the resulting trajectory was invalid.

Counts, when a valid trajectory became as a result of the measurement are shown in Fig. 4 for each iteration value t. From the results, it can be seen that with an optimal number of iterations 4, the result was always a valid trajectory. While $t \neq 4$, it can be clearly seen that the number of valid results decreased when t was farther from 4.

For this test case, there were 2 solution trajectories with only 2 non-null primitives. As $d = 3$, these trajectories all included one null primitive. This meant that 3 different primitive sequences corresponded to each of these trajectories because the null primitive could exist in 3 different places along the trajectory. This led to a situation, where when discarding null primitives from the result trajectories in the case when $t = 4$, the other trajectory with 2 primitives came as a result 197 times and the other 190 times. Trajectories with 3 non-null primitives came as a result on average 61.3 times. This result supports the effect of null primitives discussed in the section III-B, that trajectories with fewer non-null primitives are more likely to become as a result than trajectories with more non-null primitives.

These results with the method scaled to a 3-dimensional C-space including velocity dimension, are aligned with results in [1], where simulation tests were performed in a 2-dimensional C-space with obstacles. This method has also been tested on quantum hardware, but it did not perform any better than random selection of a resulting trajectory would, probably because the used quantum hardware was not able to sustain the state of the qubits for the duration of the computation [1].

V. Conclusion

This paper presented a method for trajectory planning using quantum computing. By simulation, it was shown that the method scales as designed, and can be applied in a more complex environment than what was previously tested. This method can theoretically outperform current classical methods, but it is not guaranteed in every condition. While current implementations of quantum computers are not yet capable enough for a task this complex, it is a topic of future research to test and evaluate the performance of the method using a quantum computer.

References

[1] J. Lilja, "Quantum Computing for Automated Vehicle Trajectory Planning," Bachelor's thesis, Tampere University, 2023. [Online]. Available: https://urn.fi/URN:NBN:fi:tuni-202301191558

[2] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, and W. Burgard, *Principles of robot motion: theory, algorithms, and implementations*. MIT press, 2005.

[3] M. Pivtoraiko and A. Kelly, "Efficient constrained path planning via search in state lattices," in *International Symposium on Artificial Intelligence, Robotics, and Automation in Space*. Munich Germany, 2005, pp. 1–7.

[4] J. Ziegler and C. Stiller, "Spatiotemporal state lattices for fast trajectory planning in dynamic on-road driving scenarios," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009, pp. 1879–1884.

[5] P. A. M. Dirac, "A new notation for quantum mechanics," *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 35, no. 3, p. 416–418, 1939.

[6] E. G. Rieffel and W. H. Polak, *Quantum computing: A gentle introduction*. MIT Press, 2011.

[7] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, 1996, pp. 212–219.

[8] R. Jozsa, "Searching in Grover's Algorithm," 1999.

[9] C. Zalka, "Grover's quantum searching algorithm is optimal," *Phys. Rev. A*, vol. 60, pp. 2746–2751, Oct 1999. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.60.2746

[10] A. Chella, S. Gaglio, G. Pilato, F. Vella, and S. Zammuto, "A Quantum Planner for Robot Motion," *Mathematics*, vol. 10, no. 14, 2022. [Online]. Available: https://www.mdpi.com/2227-7390/10/14/2475

[11] L. Tarrataca and A. Wichert, "Problem-solving and quantum computation," *Cognitive computation*, vol. 3, pp. 510–524, 2011.

[12] A. Younes and J. Miller, "Automated Method for Building CNOT Based Quantum Circuits for Boolean Functions," 2003.

[13] W. P. Baritompa, D. W. Bulger, and G. R. Wood, "Grover's Quantum Algorithm Applied to Global Optimization," *SIAM Journal on Optimization*, vol. 15, no. 4, pp. 1170–1184, 2005. [Online]. Available: https://doi.org/10.1137/040605072

[14] M. Boyer, G. Brassard, P. Høyer, and A. Tapp, "Tight Bounds on Quantum Searching," *Fortschritte der Physik*, vol. 46, no. 4-5, pp. 493–505, 1998.

[15] Qiskit contributors, "Qiskit: An Open-source Framework for Quantum Computing," 2023.