Time Constrained Multi-Agent Path Planning Around Static and Dynamic Obstacles Using Graphs of Convex Sets

Matthew Osburn
ECEn 682R
osburnm@byu.edu

***Abstract*—This paper presents a method for finding trajectories between states subject to strict timing constraints using graphs of convex sets. Scenarios are decomposed into convex sets of safe configurations with respect to space and time, then navigated using the A* graph traversal algorithm. This method can be applied to multiple agents using a priority queue and geometric guarantees from the convex sets. Dynamic and static obstacles, as well as other agents, are mathematically equivalent under the priority queue assumption. We show that safe paths can be found for multiple agents without violating timing constraints.**

I. Introduction

Computing trajectories that avoid static and dynamic obstacles is a difficult problem that many applications must address. The underlying assumptions of the scenario, the notion of optimally used, and the physical constraints of the system result in a variety of solutions for collision avoidance [1]. Solutions for path planning and collision avoidance include randomly exploring random trees (RRT) [2], graph search algorithms such as A* [3], spline based planning [4], model predictive control [5], and even machine learning methods [6].

The path planning problem becomes even more difficult when strict timelines must be adhered to along the path e.g. generating a safe path from state A at starting time t_0 to state B at specific time t_f while minimizing a cost function. [7][8]. It may be difficult to find a feasible path around dynamic obstacles because resolving collisions may result in breaking the strict time constraint.

Two recent papers discuss a method that can be applied to trajectory formulation problems that rely on graphs of convex sets [9] [10]. The configuration space of an agent is decomposed into convex safe sets and then connected in a graph. Convex optimization techniques are then applied to traverse the graph and optimize the path simultaneously. The papers claim that trajectories with many dimensional configuration spaces can be generated through convex approximation.

Once trajectories are spatially defined an optimization program defines the timing of the path. When multiple agents or dynamic obstacles are present the configuration spaces of each agent and obstacle are combined. This ensures safety in agents but causes the dimension of the convex safe sets to grow with each additional agent or dynamic obstacle, quickly increasing the time complexity to optimize.

In this paper we will expand on the idea of using graphs of convex sets by introducing time explicitly into the configuration space, instead of solving for timing at the end of the generation process as in [10]. By adding time as a configuration dimension we will also be enforcing the strict timing constraints on the trajectories that we wish to generate. Adding time as a dimension also allows us to represent dynamic obstacles as static in the configuration space. In this paper we will show that this will allow us to (1) navigate around dynamic obstacles and navigate other agents without increasing the number of dimensions in the convex sets, and (2) maintain strict time constraints in the solution trajectory.

II. Methods

Time is often the parameter in state differential equations and path formulations, often appearing in the differential equation $\dot{x}(t) = f(x(t), u(t))$ or the path reference function $B(t)$. The units associated with time are essential for providing physical meaning to the quantities described in such equations. However, time as a state is often excluded because there is no way to control the flow of time and in non relativistic frames changes at a linear rate.

We introduce time as a part of the state so that we can leverage convex sets to ensure collision avoidance as well as arrival at our destination at the specified time. To model a 2D environment with time we define the configuration space $\mathbb{P} \subseteq \mathbb{R}^3$ so that each unique state is defined by the coordinate (x, y, t). In other words, each point in the configuration space has a unique combination of geometric coordinates (x, y) and a time coordinate (t).

The graph of convex sets optimization scheme as outlined in [9] and [10] creates a graph containing convex regions that are safe for an agent to traverse. These regions can be the physical location of the agent or internal dynamics states such as rotation or velocity. Once this is created they claim that the graph can be traversed from state A to state B and a physical path optimized simultaneously. However, in practice we were not able to replicate the results of this paper. This could be caused by user error, using a different optimization package, and limited time to complete this project. Instead, we chose to traverse the graph and then solve for the final trajectory in two distinct steps. The ramifications of this decision will be discussed later in the paper.

To create the graph of convex sets we must first start with a configuration space bound. This bound is assumed to be a convex polyhedron comprised of vertices $v \in \mathbb{P}$ and faces f connecting vertices. This bounding region is repeatedly cut to form a list of convex polyhedrons that are exterior to the obstacles and therefore safe. Finally, adjacent polyhedrons are connected to form a graph of convex sets. The algorithm psuedocode is shown in algorithm 1.

Obstacles in 2D can be represented in time by connecting the 2D polygonal shapes through time to create a 3D polyhedron (fig. 1). The motion of the obstacle is most likely non-convex, so the path is discretized into segments of convex shapes describing the motion of the object through time. These shapes are used as the $\mathscr{C}$ term in algorithm 1 to cut the bounding region into convex safe sets. The polyhedron cutting process at line 8 of algorithm 1 could theoretically allow for non-convex cutting shapes, but due to time limitations on this project that could not be implemented.

Additionally, obstacles can be augmented with the shape of an agents hull using a minkowski sum. This augmentation ensures that the entire hull of the agent will fit within the safe set. Traversing the graph of convex sets from the starting node to the ending node provides a set of polyhedrons that the final trajectory must pass through to reach the ending state.

To traverse the graph of convex sets we used the A* algorithm. This algorithm relies on a heuristic to score nodes and find an optimal solution to traversing between start and ending nodes in a graph. The A* algorithm is only guaranteed to find an optimal path if the heuristic used does not over-estimate the true score of the node. In our case, the path-length from the starting node to ending node had to be approximated and is an over-estimate of the true path-length. Because of this we cannot ensure that the path returned by A* is the optimal one.

Once a list of nodes was returned by A*, we optimized a path through adjacent nodes attempting to minimize path length. Each node in the A* gets a line segment that is optimized. The points of the line segment must remain within their respective nodes and must join with connecting nodes in the A* solution. Line segments are constrained to be causal, meaning that the time component of the line must be strictly increasing. Line segments in the start and end nodes must connect with start and end states. Finally, the velocity of the segments cannot exceed a maximum velocity constraint. This optimization is formulated as follows:

$$\arg\min_{p} \sum_{n \in \text{Nodes}} \|p_{n,2,xy} - p_{n,1,xy}\|_2 \quad \textit{(Path Length)}$$

$$\text{s.t}$$

$$p_n \in \mathscr{H}_n \quad \textit{(Points Are Within Convex Hulls)}$$

$$p_{n,1} - p_{n-1,2} = 0 \quad \textit{(Segments Connect)}$$

$$p_{n,2,t} > p_{n,1,t} \quad \textit{(Segments Are Causal)}$$

$$\frac{\|p_{n,2,xy} - p_{n,1,xy}\|_2}{p_{n,2,t} - p_{n,1,t}} < V_{max} \quad \textit{(Velocity Constraint)}$$

$$p_{0,1} = [x_0, y_0, t_0] \quad \textit{(Starting State)}$$

$$p_{n,2} = [x_f, y_f, t_f] \quad \textit{(Ending State)}$$

After a path is optimized, the agent can be treated as a dynamic obstacle for all other agents. The agent's path can be converted into a list of convex shapes and treated as an obstacle by all other agents. Each agent was put in a priority queue and paths were optimized in order of priority. This means that the last agent in the queue had to plan around all previous agents paths to reach the desired end state.

Optimization and simulations were run in Python on a Windows 11 computer with a Intel i7, 12th generation. The sci-py optimization package was used to create and solve the optimization problem. Python's matplotlib package was used for all graphs and visualization. Blender, the open source 3D program was used for 3D visualization and debugging.

Algorithm 1 Graph of Convex Sets Generation

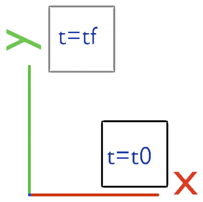
```
Require: $\mathscr{B}$ {Polyhedron Bound}, $\mathscr{C}$ {Convex Obstacle Polyhedrons}
 1: {Define front list, back shape, and list of cutting sets}
 2: $f \leftarrow \{\}$
 3: $b \leftarrow \mathscr{B}$
 4: $c \leftarrow \{\mathscr{C}\}$
 5: for all $c_i \in c$ do
 6:     if $c_i$ intersects $b$ then
 7:         for all faces $\in c_i$ do
 8:             cut $b$ into front and back polyhedrons
 9:             $f$ appends front polyhedron
10:             $b \leftarrow$ back polyhedron
11:         end for
12:     end if
13: end for
14: $e \leftarrow$ connections between polyhedrons in f
15: return $G(f, e)$ {Graph nodes are the elements of f and edges are e}
```

III. Results

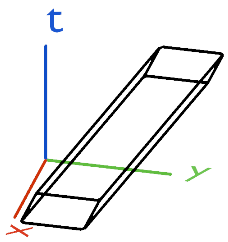
Multiple scenarios were devised to test the optimization and geometric methods outlined in the previous section. These experiments include: A single agent navigating around a static obstacle and multiple dynamic obstacles and reaching the end in a fixed time, multiple agents path planning through a tight hallway and re-forming a formation in a fixed time, multiple agents path planning through the same hallway but in a different order and with collision hulls enabled, and an example of how the priority ordering of this method can cause idling or other strange behaviors. Each experiment and their results will be discussed in their own respective subsection.

A. Single Agent, Many Obstacles

The first scenario is an agent that starts at state $[x_0 = 2, y_0 = 0, t_0 = 0]$ and must reach the state $[x_f = 2, y_f = 3.5, t_f = 6s]$ avoiding the static obstacle in the middle (black rectangle) and the three dynamic obstacles (red squares). The



(a) (x,y) representation of an obstacle at two different times



(b) (x,y,t) representation of motion. The geometry of a dynamic obstacle is static in this representation

Fig. 1: Motion Representation In The Configuration Space $\mathbb{P}$

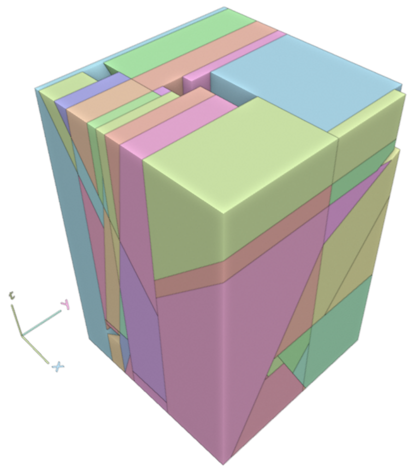


Fig. 2: The convex sets of a scenario with dynamic obstacles resulting from algorithm 1

convex hull breakdown of this scenario is shown in figure 2. The A* algorithm return cells that allowed for the minimization of the path length between start and end state. This was a coincidental and resulted from the specific combination of starting and ending states. Some frames of this simulation can be seen in figure 3.

B. Multi Agent

The second scenario was a group of multiple agents that needed to pass through a tight hallway and remake the formation on the other side. The order of priority for planning was lower-left, lower-right, upper-left, upper-right. All agents needed to be to their target positions 9 seconds after the simulation started. Some frames of this progression are shown in figure 4. Agents successfully navigated the corridor and reached the end states on time.

C. Multi Agent, Collision Hulls Incorporated

The minkowski sum of an agents collision hull with the planned trajectories of previous agents and the static corridor obstacles was used to create sets that are safe and allow for the entire agent to fit without collision. The agents must reform on the other side of the corridor 10 seconds after the start of the simulation, but this time with stricter spatial constraints due to the collision hulls. The agents successfully move out of the way to let higher priority agents pass through first and reach formation on the other side. Some frames of this simulation are shown in figure 5.

D. Multi Agent, Geometric Idling Example

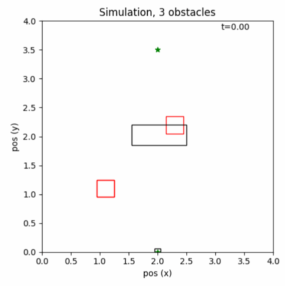
The same multi-agent scenario with collision hulls is repeated, except that the priority queue is shuffled. The planned paths require far more time and extreme velocities for agents with less priority. Compare the agents on the left side of the corridor in figure 6 (c) at time=6.25s to the agents on the left side of the corridor in figure 5 (c) at time=4.65s. The agents in figure 5 make much more progress quicker than the agents in figure 6. This is due to the geometry that results from algorithm 1 and the sub-optimal nature of the A* algorithm that we are using. End velocities are extremely large for the fourth agent. Some frames of this simulation are shown in figure 6.

IV. Conclusion

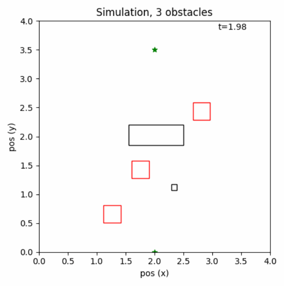
Because we did not solve the trajectory generation problem simultaneously with traversing the graph there were some key shortcomings with our method. These changes were necessary to finish the project before the deadline but would need to be addressed before this work could be published in an academic journal.

The first shortcoming was with the use of A* to find traverse the graph of convex sets before optimizing a minimum path length trajectory. A* is a greedy search that has no guarantees that the solution it finds will be a global minimum. Additionally, it is hard to assign a metric or score to a node that effectively encapsulates the path-length cost function, because that is heavily dependant on the positions of previous and following nodes in the solution.

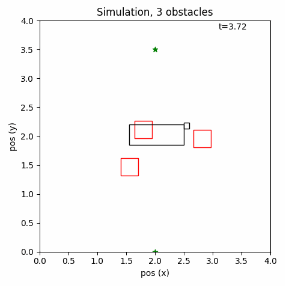
Another glaring issue with the A* algorithm is that it is impossible to force causality in the graph traversal algorithm. There may be many paths between two connecting nodes that are causal and many paths that are non-causal, and



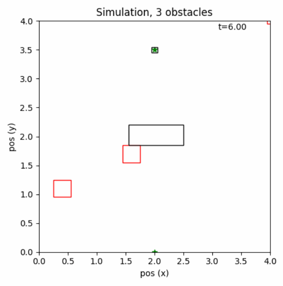
(a) (t=0s) Agent in starting position



(b) (t=1.98s)

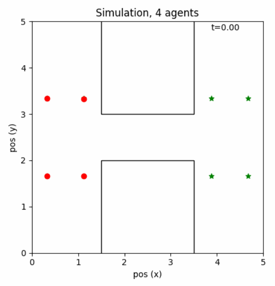


(c) (t=3.72s)

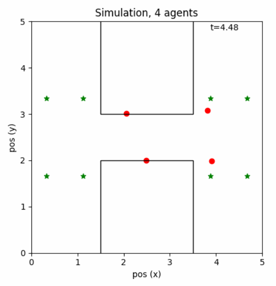


(d) (t=6s) Agent reaches the end state on time

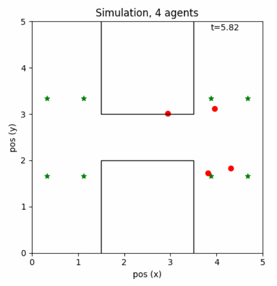
Fig. 3: Single Agent Motion Through Time



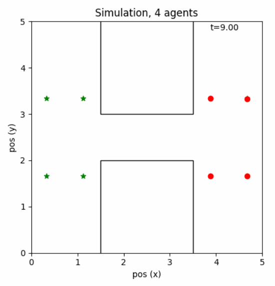
(a) (t=0s) Agent in starting position



(b) (t=4.48s)

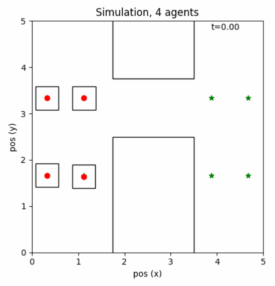


(c) (t=5.82s)

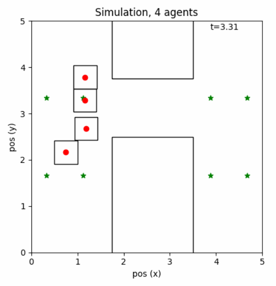


(d) (t=9s) Agent reaches the end state on time

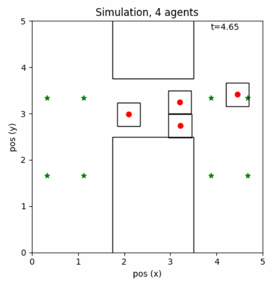
Fig. 4: Multiple Agents



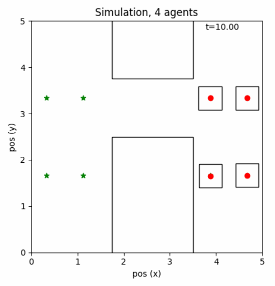
(a) (t=0s) Agent in starting position



(b) (t=3.31s)

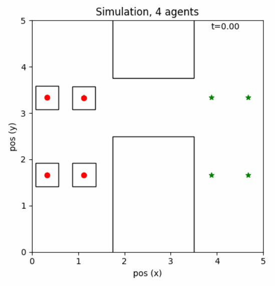


(c) (t=4.65s)

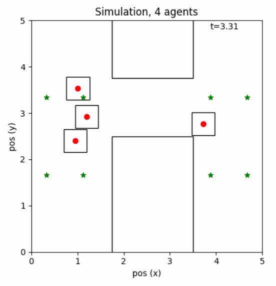


(d) (t=10s) Agent reaches the end state on time

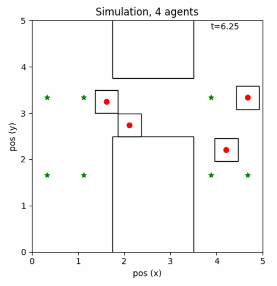
Fig. 5: Multiple Agents, Collision Hulls



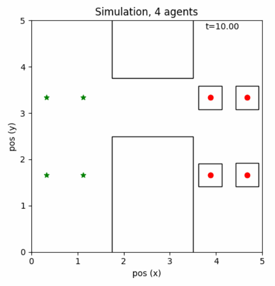
(a) (t=0s) Agent in starting position



(b) (t=3.31s)



(c) (t=6.25s)



(d) (t=10s) Agent reaches the end state on time

Fig. 6: Multiple Agent, Geometry Idling

punishing transitions to nodes because it has many more non-causal routes may end up eliminating the global solution to the problem. This issue also leads to problems enforcing velocity constraints in the graph traversal process, leading to a high likelihood that A* will return a dynamically infeasible path. A* could potential return solutions that require infinite velocities to achieve.

Another problem was the use of constant velocity path segments. The simulation results would have been far more intuitive if the agents did not jump between very small and very large velocities instantaneously. Paths with acceleration and jerk would represent the dynamics of a real world scenario far better.

If it had been possible to fully integrate MITs optimization method before the deadline of this project the issues outlined above could have been entirely avoided. Paths could be optimized in tandem with the graph traversal, allowing for causality constraints and velocity constraints to be better integrated into the process. It would have also potentially made it possible for multiple agents to optimize their paths against each other, rather than relying on a priority queue and re-calculation of the geometry to ensure safety.

In future work we plan to fully implement MITs optimization scheme and investigate how well causality and velocity constraints can be met. This will probably require the use of a different optimization package that is better equipped to solve this problem than the scipy optimize package. It may also be helpful to investigate more ways to partition the configuration space that would enforce causality and kinematic constraints.

To conclude, we were able to implement geometric methods that allow us to (1) navigate around dynamic obstacles and navigate other agents without increasing the number of dimensions in the convex sets, and (2) maintain strict time constraints in the solution trajectory. However, key assumptions needed to be violated to achieve this within the project deadline. Additional work is required to make these geometric methods provably safe and feasible.

References

[1] Michael Hoy, Alexey S. Matveev, and Andrey V. Savkin. “Algorithms for collision-free navigation of mobile robots in complex cluttered environments: a survey”. en. In: *Robotica* 33.3 (Mar. 2015), pp. 463–497. ISSN: 0263-5747, 1469-8668. DOI: 10.1017/S0263574714000289. URL: https://www.cambridge.org/core/product/identifier/S0263574714000289/type/journal_article (visited on 03/27/2024).

[2] Steven LaValle. “Rapidly Exploring Random Trees: A New Tool For Path Planning”. In: *Research Report* 9811 (Oct. 1998).

[3] Peter Hart, Nils Nilsson, and Bertram Raphael. “A Formal Basis for the Heuristic Determination of Minimum Cost Paths”. en. In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107. ISSN: 0536-1567. DOI: 10.1109/TSSC.1968.300136. URL: http://ieeexplore.ieee.org/document/4082128/ (visited on 04/11/2024).

[4] Giuseppe Carbone and Fernando Gomez-Bravo, eds. *Motion and Operation Planning of Robotic Systems: Background and Practical Approaches*. en. Vol. 29. Mechanisms and Machine Science. Cham: Springer International Publishing, 2015. ISBN: 978-3-319-14704-8 978-3-319-14705-5. DOI: 10.1007/978-3-319-14705-5. URL: https://link.springer.com/10.1007/978-3-319-14705-5 (visited on 04/11/2024).

[5] Muhammad Awais Abbas, Ruth Milman, and J. Mikael Eklund. “Obstacle Avoidance in Real Time With Non-linear Model Predictive Control of Autonomous Vehicles”. en. In: *Canadian Journal of Electrical and Computer Engineering* 40.1 (2017), pp. 12–22. ISSN: 0840-8688. DOI: 10.1109/CJECE.2016.2609803. URL: https://ieeexplore.ieee.org/document/7869433/ (visited on 04/11/2024).

[6] Xuesu Xiao et al. “Motion planning and control for mobile robot navigation using machine learning: a survey”. en. In: *Autonomous Robots* 46.5 (June 2022), pp. 569–597. ISSN: 0929-5593, 1573-7527. DOI: 10.1007/s10514-022-10039-8. URL: https://link.springer.com/10.1007/s10514-022-10039-8 (visited on 04/11/2024).

[7] Kawser Ahmed and Kouamana Bousson. “Generating Time Optimal Trajectory from Predefined 4D waypoint Networks”. en. In: *International Review of Aerospace Engineering (IREASE)* 10.4 (Aug. 2017), p. 207. ISSN: 1973-7440, 1973-7459. DOI: 10.15866/irease.v10i4.12570. URL: http://www.praiseworthyprize.org/jsm/index.php?journal=irease&page=article&op=view&path[]=20854 (visited on 03/27/2024).

[8] Vittorio Di Vito et al. “An Overview on Systems and Algorithms for On-Board 3D/4D Trajectory Management”. en. In: *Recent Patents on Engineering* 3.3 (Nov. 2009), pp. 149–169. ISSN: 18722121. DOI: 10.2174/187221209789117744. URL: http://www.eurekaselect.com/openurl/content.php?genre=article&issn=1872-2121&volume=3&issue=3&spage=149 (visited on 03/27/2024).

[9] Tobia Marcucci et al. *Shortest Paths in Graphs of Convex Sets*. en. arXiv:2101.11565 [cs, math]. July 2023. URL: http://arxiv.org/abs/2101.11565 (visited on 03/25/2024).

[10] Tobia Marcucci et al. “Motion planning around obstacles with convex optimization”. en. In: *Science Robotics* 8.84 (Nov. 2023), eadf7843. ISSN: 2470-9476. DOI: 10.1126/scirobotics.adf7843. URL: https://www.science.org/doi/10.1126/scirobotics.adf7843 (visited on 03/25/2024).