EC601 Project 1

Name: Yitian Ren   BUID: U75800956

Problem Statement

For simple dynamic systems, it is easy for the agent to accomplish a goal, for instance, reaching a goal point while avoiding the collision with obstacles. However, for multiagent system with more complicated environment, we cannot get intuitive answer of trajectory for the dynamic system. These years have witnessed the combination of Control Lyapunov Function (CLF), Control Barrier Function (CBF) and dynamic programming, especially quadratic programming be a strong tool to describe and solve the problem.

$$\begin{aligned} u(x) = \underset{u\in\mathbb{R}^n, \delta\in\mathbb{R}}{\arg\min} \frac{1}{2}(u-u_{ref})^T H(u-u_{ref}) + p\delta^2, \\ s.t.\ \dot{V}(x) + \alpha(V(x)) \leq \delta, \\ \dot{h}(x) + \beta(h(x)) \geq 0 \end{aligned}$$

Structure of CLF-CBF-QP

The emergence of High Order Control Lyapunov Function (HOCLF) and High Order Control Barrier Function (HOCBF) enables us to solve systems with more types of control model which describes various environments and control inputs, for instance, double-integrator model.

$$L_f^m V(x) + L_g L_f^{m-1} V(x) u + O(V(x)) \leq -\alpha_m(\Phi_{m-1}(x))$$

HOCLF

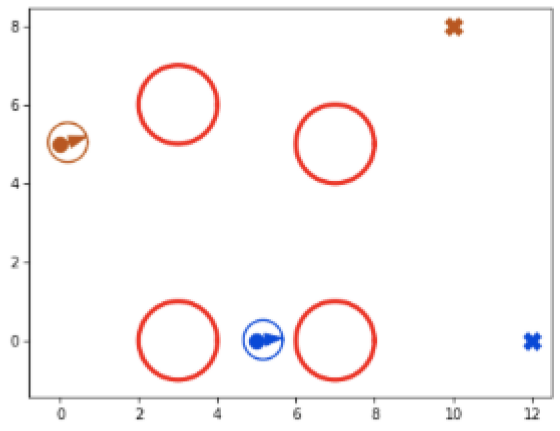
$$L_f^m h(x) + L_g L_f^{m-1} h(x) u + O(h(x)) \geq -\alpha_m(\Phi_{m-1}(x))$$

HOCBF

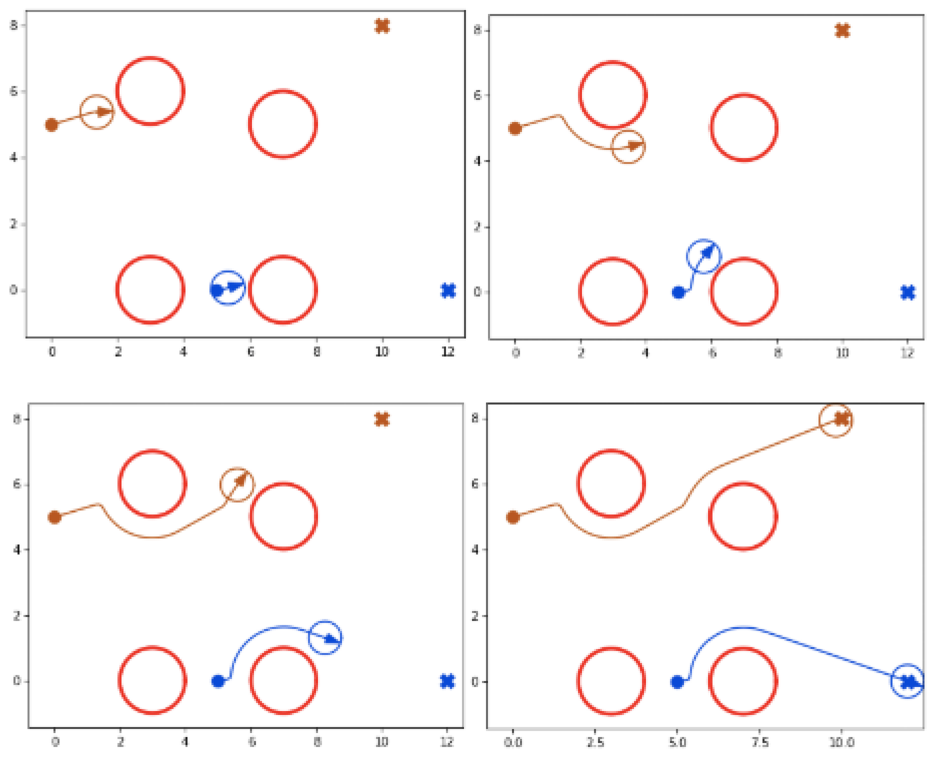
And there is no standard software to quickly solve this whole process and generate the trajectory for the robotic systems. Therefore, I want to develop a toolbox to realize it in python. I also want to investigate performance for various dynamic programming solvers and its different use. There are some QP solvers that people usually uses, which are attached in section “QP Solvers”

Usage

It can be used in many robotic systems, either for ROS or some real applications, for instance, autonomous driving, and any robots which needs real-time navigation. One advantage of this tool against learning based model is that it can be used in very complicated real-time environments, while learning-based method can only works on the situation that the model already have experience with. And furthermore, it can serve as a package and there is no need for building the environment and define the whole process every time we do research topics which need this basic CLF-CBF-QP structure. Different formats of outputs is also useful for different use. For example, some users might need some animations to demo, while some might only need the trajectory list and want the toolbox to execute as fast as possible. Here I implement a simple example to show that this structure can be used in multi-agent path planning problem.



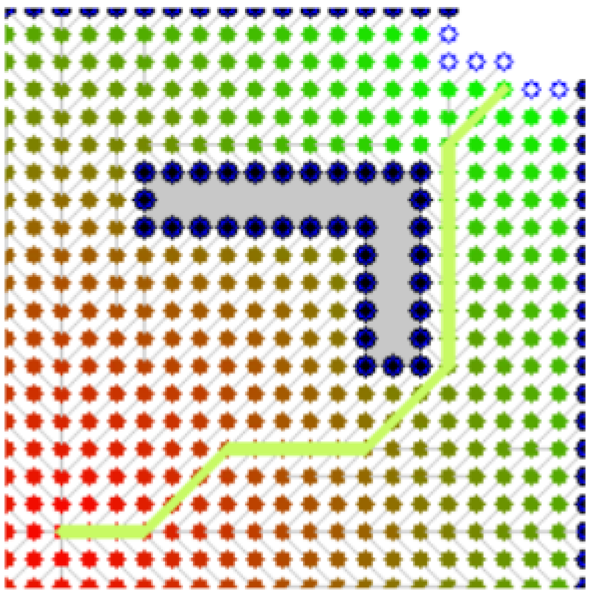
Initial Environment


Screenshots of the live plots for Example

Literature Review

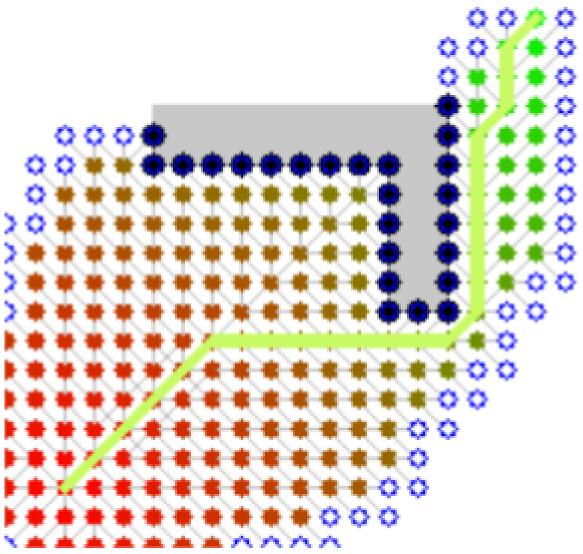
Path-Planning Algorithms

One of the earliest and simplest algorithms is Dijkstra’s algorithm. Starting from the initial vertex where the path should start, the algorithm marks all direct neighbors of the initial vertex with the cost to get there. It then proceeds from the vertex with the lowest cost to all of its adjacent vertices and marks them with the cost to get to them via itself if this cost is lower. Once all neighbors of a vertex have been checked, the algorithm proceeds to the vertex with the next lowest cost. Once the algorithm reaches the goal vertex, it terminates and the robot can follow the edges pointing towards the lowest edge cost.

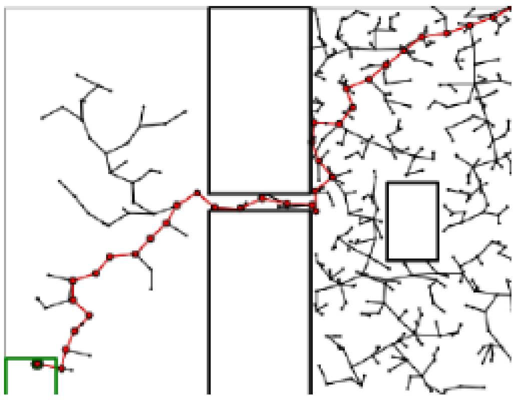

Dijkstra’s Algorithm Implementation

We can see from the figure that the algorithm performs a lot of computations that are “obviously” not going in the right direction. Then comes the A* algorithm.

A* operates similarly to Dijkstra’s algorithm except that it guides its search towards the most promising states, potentially saving a significant amount of computation time by setting a heuristic function. For example, we could give priority to nodes that have a lower estimated distance to the goal than others.


A* Algorithm Implementation

Rapidly-exploring Random Tree algorithms are a common sampling based method for robotic path planning. In its simplest form, RRTs initialize with a starting state and a goal state with the starting state being the base node on a graph. A random state is taken from the state space and the algorithm finds the nearest node and connects it to a node one step in the direction of the randomly sampled state. With just this consideration and no modifications, the resulting path and plot will look like the below Figure.


RRT Algorithm Implementation

CLF-CBF-QP:

This structure is first developed by AD Ames in his paper “Control Barrier Function based Quadratic Programs with Application to Adaptive Cruise Control” in 2014. And these years has witnessed CBF and this structure being a hot research topic. Wei, Xiao in his paper “Control barrier functions for systems with high relative degree”, 2019, proposed the HOCBF, which enables this structure solve system of higher relative degree and with more complicated control input. And there are also many researches which focused on noisy control and on different situations that this structure can be used. For instance, Wei Xiao, in his paper “Rule-based Optimal Control for Autonomous Driving”, 2021, illustrates how this structure can be applied in autonomous driving while obeying the traffic rules in different level of importance.

Open Source research

Usually, every researcher will build their own CLF-CBF-QP structure based on their own research project and specific environments and control affine system, and there are no such a universal software which contains different kinds of dynamic model, types of obstacles, quadratic programming solver and formats of output. For example, team in Hybrid Robotics Lab of UCB built a Matlab Interface, which can solve basic CLF-QP, CBF-QP and CLF-CBF-QP structure (https://github.com/HybridRobotics/CBF-CLF-Helper), and in their demonstration, they use the traditional Adaptive Cruise Control (ACC) problem as the example. However, this still has many constraints like their input is still complicated, and they do not have many output formats. If people want to use this toolbox, for example, to generate the train set of a neural network. It would really be inefficient since actually most executing time is used for plotting which is useless for building the train set.

Relevant Papers And Websites

QP Solvers

1. cvxopt: http://cvxopt.org/userguide/index.html
2. qpOASES: https://usermanual.wiki/Pdf/manual.2007140309/view
3. quadprog: https://github.com/quadprog/quadprog
4. cvxpy: https://www.cvxpy.org/
5. Gurobi: https://www.gurobi.com/documentation/9.5/refman/
6. Mosek: https://www.mosek.com/documentation/

CLF & CBF Theoretical Background

1. Ames, Aaron D., et al. "Control barrier function based quadratic programs for safety critical systems." *IEEE Transactions on Automatic Control* 62.8 (2016): 3861-3876.
2. Xiao, Wei, and Calin Belta. "Control barrier functions for systems with high relative degree." *2019 IEEE 58th conference on decision and control (CDC)*. IEEE, 2019.
3. Ames, Aaron D., et al. "Control barrier functions: Theory and applications." *2019 18th European control conference (ECC)*. IEEE, 2019.
4. Xiao, Wei, et al. "Rule-based optimal control for autonomous driving." *Proceedings of the ACM/IEEE 12th International Conference on Cyber-Physical Systems*. 2021.