MAE 207

Final report

Agent Path Planning and Tracking

Application of HJ Reachability

Group 14

Yuhao Chen
Shebo Jia
YaChe Shih
Xiaoran Zha

Abstract

Hamilton-Jacobi (HJ) reachability analysis is an important method for guaranteeing the performance and safety properties of dynamical systems; it has been applied to many systems in the past decade. Its advantages are compatibility with general nonlinear system dynamics and the availability of well-developed numerical tools. However, its exponential computational complexity costs much more time than other methods. In this project, we combine RRT path planning and PID path tracking with Hamilton Jacobian Reachability analysis to ensure the agent will avoid different types of obstacles while maintaining high efficiency. A simulation is conducted to test the viability of the combination. The obstacles setting includes known static obstacles, known dynamic obstacles, and unknown static obstacles. Using the combined method, the agent successfully passed these obstacles, and the time cost is significantly reduced compared to using Hamilton Jacobian Analysis alone.

Contents

1. Introduction

1.1 General ideas

Nowadays, more and more path planning and tracking methods are developed, making autonomous agent systems more intelligent. When designing a method, security should be the priority, which is the basis of an autonomous system. However, optimality is also an essential consideration for the system to strike a balance between optimal and safe trajectory. This project will develop a novel path planning and tracking method. The agent would avoid collision while following the optimal or suboptimal path to the terminal under this approach.

Generally speaking, the Hamilton-Jacobi (HJ) reachability would be applied to the safety problem because of its accuracy and stability [2]. One of the main disadvantages of the HJ method is the computational complexity, which would require a long-running time and significant computing power. Under such apparent disadvantages, the combination of the HJ method to other methods seems to be a reasonable solution. The goal of this project is collision avoidance, and some widely used approaches should be considered. The rapidly-exploring random tree (RRT) method is widely used in path planning, first introduced by Steven in 1998 [1]. This algorithm can efficiently handle scenes containing obstacles and differential motion constraints, so it is widely used in various robot motion planning scenarios. However, most RRT algorithms and their extended versions are suitable for the case of static obstacles, which inspires us that the RRT might be able to combine with the HJ method to develop a better algorithm.

1.2 Basic assumptions

As a result, the prototype of the project is as follows. Under a specific area, the starting point and the terminal point are given. There are three different types of obstacles: the static obstacle, the dynamic obstacle, and the unknown static obstacle. The obstacles are

considered in the shape of circles for the two-dimensional plane and cylinder in the three-dimensional plane, with the same radius R (safety bubble [3]). The location and radius of the static obstacles would be informed to the agent, based on which the path planning could be done. The dynamic system and the radius of the dynamic obstacles would also be informed, which allows the agent to detect the location of the dynamic obstacles for all the time span. The location of unknown obstacles would not be informed to the agent, which means the unknown obstacles could only be detected when they are close enough to the agent. The agent is supposed to move in a two-dimension plane, which results in no altitude change should be in consideration. Under these assumptions, a path will be planned by the RRT star, and then the agent should use the PID controller to follow the path and try to use the HJ controller to avoid collisions when obstacles are detected. This is the path-tracking process. However, due to the complexity of the calculation, the entire program will be offline, so the tracking process is a simulation, then disturbances and tracking errors will not be considered.

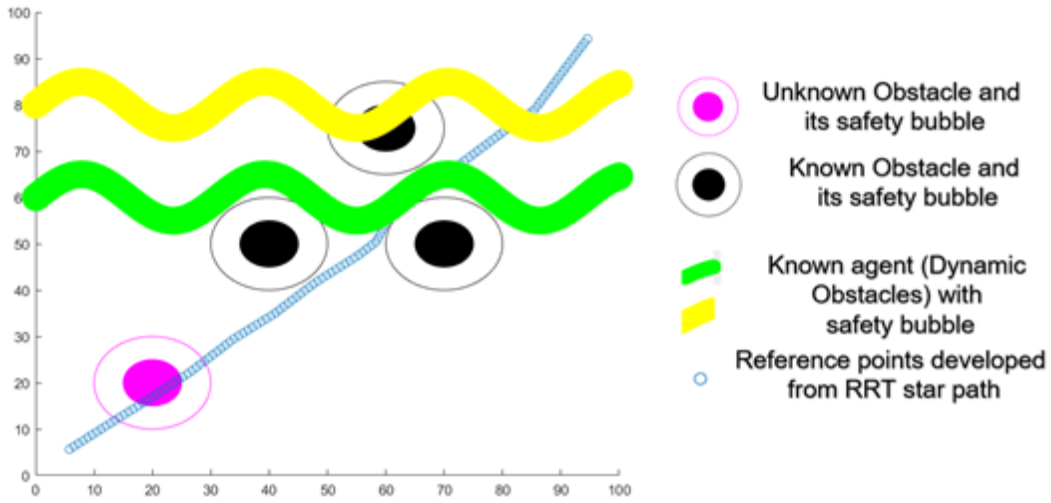

Figure 1 Simulation environment under basic assumptions and their definitions

Moreover, in this program, the agent would be considered constant velocity, and the maximum turning angle would be 10 degrees per unit time. The dynamic obstacles would also move in the constant velocity, overlapping with any other obstacles. As for the detection, assuming there is no time delay between the detection and the action, the controller could be switched when the agent detects the possible collided obstacles.

In the project, two controllers would be designed: tracking controller, which uses the proportional–integral–derivative (PID) control; safety controller, which is based on the

solution of HJ partial derivative equation (PDE). The switching process between them would simply be an event-triggered or so-called hard switch.

2. Objectives

This project aims to design an algorithm that allows the agent to plan a safe path and track it while avoiding possible collisions. Meanwhile, the suboptimal path is permitted, but no velocity change is permitted, and the turning back path should be avoided. The project is offline computation, so the running time would not serve as the judging standard.

3. Path planning

3.1 Introduction

Path planning is a subset of more extensive motion planning, but it is essential. A path is necessary if an agent goes from its initial set to the target set. The path sequence of post states connects the two sets with a relatively smooth route. To determine the sequence is path planning. In this project, path planning is created based on a graph-based method to numerically find a safe path with a length as short as possible.

3.2 RRT

RRT, rapidly-exploring random tree, is an efficient way to explore the potential paths in space and navigate the agent from its initial node to the destination. RRT grows a tree rooted at a start node and detects a possible route through its branches. Each node only has a single parent. RRT is a suitable method to find a path in the project. It looks for a valid path, with no obstacle in the way, instead of the shortest distance. Since the initial and target nodes are reachable, this method is guaranteed to find a path as the number of samples approaches infinity. In fact, the RRT can detect a solution with a smaller number of finite samples.

3.3 RRT Star

To make the solution of the path close to optimal, additional steps need to be added to RRT and make the method RRT*. After nodes are generated and connected, these nodes are also checked within a specified search zone and determine if the local nodes could be reconnected in a way that maintains the tree structure. This process also minimizes the total path length. Assuming a node could be connected in two different paths eliminates the longer path and keeps a relatively shorter path to remain the graph a tree structure.

3.4 Result

At the starting stage with fewer nodes, the path marked in black is tortuous and unsmooth, as shown in Figure-2. Comparing the graph in Figure-3, as more nodes are added, the existing paths are reconnected and refined over time. The path gets shorter and straighter.

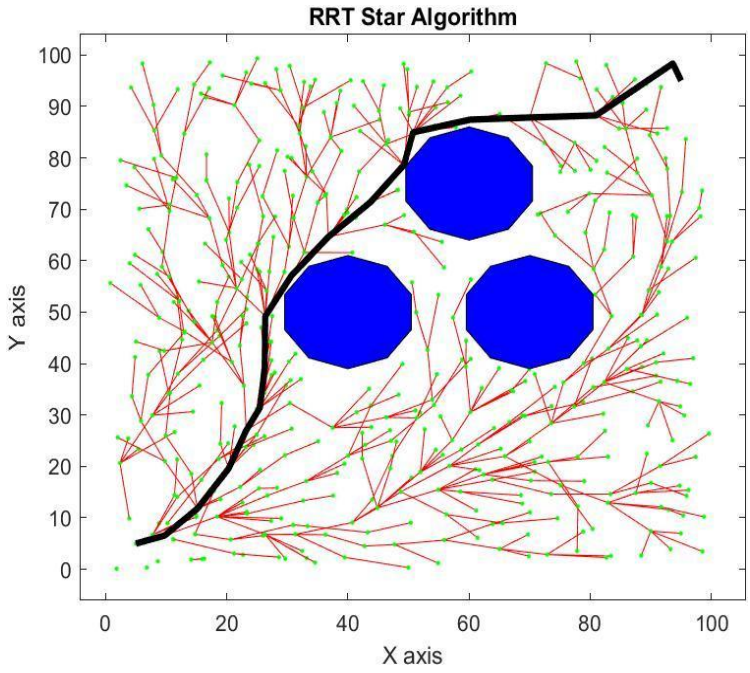

Figure 2 Initial Planned Path

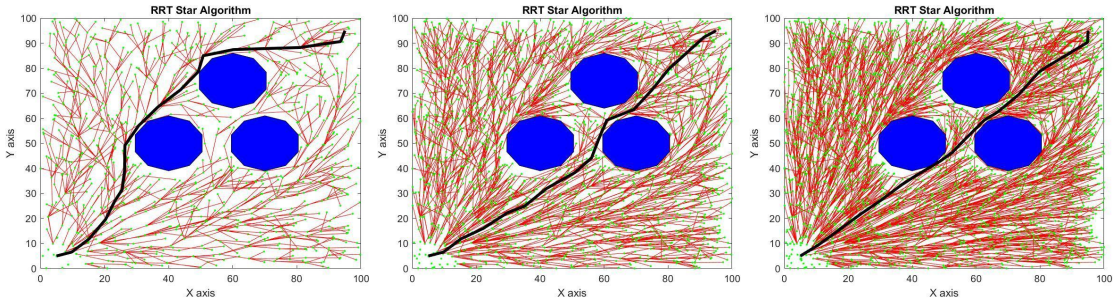

Figure 3 Refining Path

4. Path tracking

4.1 Tracking controller

4.1.1 PID controller

A PID controller is a controller that applies a correction based on P (proportional), I (integral), and D (derivative) to close the system. Hence the name.

Say we would like our state x in the following system to follow the desired path.

$$\frac{dx}{dt} = f(x, u)$$

First, we define

$x(t)$: state at time t

$x_d(t)$: desired state at time t

$e(t)$: error value of state at time $t = x_d(t) - x(t)$

u: control input to system at time t (function of x)

Let's now take a look at P, I and D respectively.

P (proportional): this term is proportional to the error value of state (e(t)). If e(t) is large and positive, this term will also be large and positive to accelerate our state.

I (integral): this term integrates all previous error values of state. $\int_0^T \delta e(t)dt$ at time T.

D (derivative): this term corrects our state by the derivative of the error value of state e(t).

All three terms have their own non-negative coefficients. K_p for P (proportional), K_i for (integral) and K_d for D (derivative). Therefore, our overall control function

$$u = K_p e(t) + K_i \int_0^T \delta e(t)dt + K_d \frac{de(t)}{dt}$$

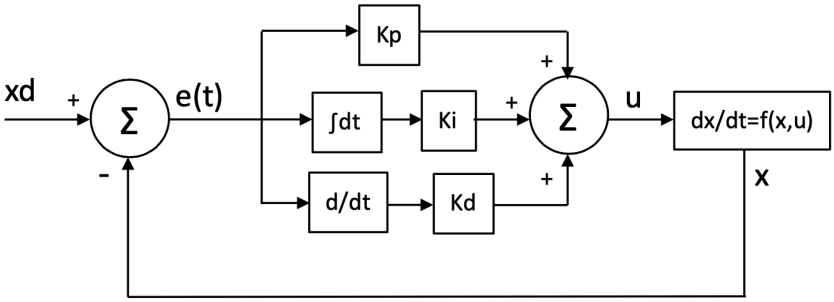

Figure 4 Block Diagram of PID controller

4.1.2 Tracking

We have our desired path from path planning, now we cut this route into several segments, and each segment has its own minor starting point and minor target set. Starting point for the next segment is in the target set for the previous segment. The agent can follow our planning path and pass through all static obstacles by arriving at all these minor target sets.

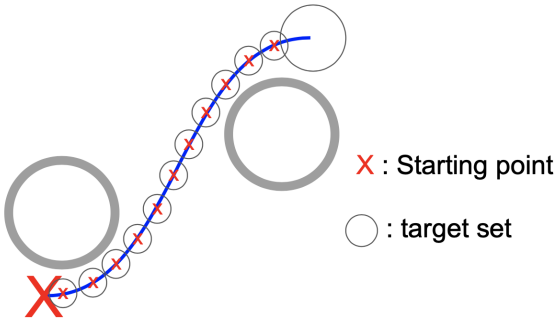

Figure 5 Schematic diagram for minor start point and target set

To track our agent, the dynamic is the following:

$$\frac{dx}{dt} = V \cdot cos\theta \cdot dt$$

$$\frac{dy}{dt} = V \cdot sin\theta \cdot dt$$

Where $\frac{dx}{dt}$ is the agent's velocity in x direction, $\frac{dy}{dt}$ is the agent's velocity in y direction and V is velocity of the agent. To simplify the simulation, we let V=1.

For the control part, since position in x and y direction can be presented by angle θ. Therefore, instead of let state feedback of system be $(x, y)^T$, we let the agent's direction θ be the feedback.

$$\frac{d\theta}{dt} = u \bullet dt$$

By applying PID controller, (desired direction θd) - (current direction θ) = error of direction δθ.

$$u = K_p \delta\theta(t) + K_i \int_0^T \delta\theta(t)dt + K_d \frac{d\delta\theta(t)}{dt}$$

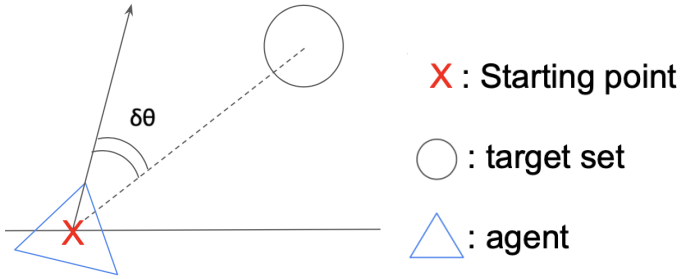

Figure 6 Schematic diagram for the agent and target set

When the agent's direction is way far from our desired direction, ||u|| may become really large, and this large input makes the agent rotate in unrealistic angular velocity. Therefore, to assure our simulation follows physical law, we bound ||u|| <= 10.

4.1.3 Results

From Figure 7, we notice that our agent's performance following our planning path is satisfying. However, there is a segment of twisting after passing the unknown obstacle. This twisting is caused by switching controllers between PID and HJI PDE solver (CH 5-2). When our agent tries to pass the unknown obstacle, our system applies an HJI PDE solver, and after it passes through the unknown object, our system switches back to the PID controller. The I (integral) accumulates all previous angle errors in the PID controller. When switching back to the PID controller from the HJI PDE solver, the error direction may be large and our system wants to rotate the agent quickly. However, we bound the value of ||u||, this makes angle error keep accumulating, and the agent's direction keeps being overcorrected (underdamped).

To prevent this twisting, we can modify our code from directly switching between the PDE controller and the HJI PDJ solver to slightly switching between each other. When the system wants to switch to the PID controller, we take half of u from HJI PDE solver

and half of u from PID controller. By this approach, the path of our agent can be smoother even when the controllers switched.

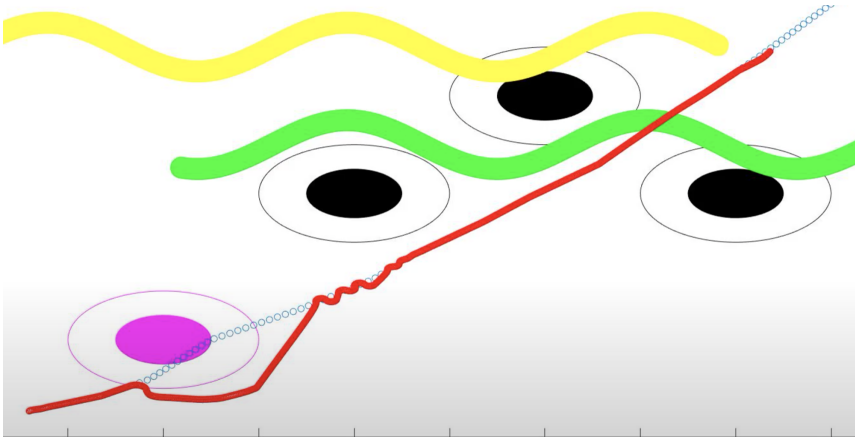

Figure 7 Twisting segments after switching from HJ to PID

4.2 Safety controller

4.2.1 HJI

HJ reachability analysis is a verification method for guaranteeing system performance and safety properties, overcoming some challenges in higher dimensional computation [4]. The backward reachability set (BRS) and backward reachability tube (BRT) would be two important concepts in the HJ PDE.

Generally speaking, the BRS stands for the sets of states that the system would reach in, and the BRT stands for the sets of states that the system would pass through or reach in. Then, since they are sets of states, they could be defined as maximal and minimal, and the formal definitions of them are given below by Mo and Tomlin [2]:

The maximal BRS:

$$R(t) = \{z:\ \forall\gamma\in\Gamma(t),\ \exists u(\bullet)\in U, \zeta(0; z, t, u(\bullet),\ \gamma[u](\cdot))\in T\}$$

The minimal BRS:

$$A(t) = \{z:\ \exists\gamma\in\Gamma(t),\ \forall u(\bullet)\in U, \zeta(0; z, t, u(\bullet),\ \gamma[u](\cdot))\in T\}$$

The maximal BRT:

$$\overline{R}(t) = \{z:\ \forall\gamma\in\Gamma(t),\ \exists u(\bullet)\in U, \exists s(\bullet)\in[t, 0], \zeta(s; z, t, u(\bullet),\ \gamma[u](\cdot))\in T\}$$

The minimal BRT:

$$\overline{A}(t) = \{z:\ \exists\gamma\in\Gamma(t),\ \forall u(\bullet)\in U, \exists s(\bullet)\in[t,0], \zeta(s;z,t,u(\bullet),\ \gamma[u](\cdot))\in T\}$$

There is a target set in the safety control system whose states the system would attempt to arrive or avoid. First, assuming the target set to be the enforceable one. Then, the BRT and the BRS should be maximal to find every possible path to arrive at the target set and obtain the optimal one from those paths. Then, the main difference between the BRT and BRS would be that the BRS would be inside the target set at the end of the time span for the proper input conditions, while BRT would not. Next, switch the target set to the inevitable one, then the BRT and BRS should be minimal, so the definitely risky path would be obtained. With those basic concepts, the HJ PDE solver could be applied to obtain a solution for the collision avoidance problem.

4.2.2 HJI PDE solver

In this section, the helperOC-master toolbox [5] and ToolboxLS [6] would be used to solve the value function and obtain the optimal trajectory. Based on the toolboxes for the MATLAB version, some important parameters should be set up before running the code. First, the Dubins car model would serve as the basic dynamic model of this system, which is defined as follows.

$$\dot{x} = v\cos\theta + d_x$$
$$\dot{y} = v\sin\theta + d_y$$
$$\dot{\theta} = \omega + d_\theta$$

Where $\omega\in[\omega_{min}, \omega_{max}]$, $d\in[d_{min}, d_{max}]$, thus, the system would be in two dimensions with three parameters: x-direction location x, y-direction location y, and the pointing angle ω.

Then, the program would start to make the grid, based on the input corner locations and the number of grid points, and with better grid points the longer running time involved. The target set would be assumed in the cylinder shape, with the given radius and location while the location of initial position and the initial pointing angle would serve as initial conditions. Next, the time vector should be set, which stands for the time span for the calculation. Here, the time vector should not be too large such that the computation process would be extremely long. On the contract, if the time vector is too small then the initial position would not be inside the BRT/BRS, and no trajectory could be obtained. Thus, the time vector is one of the important inputs for the solver. Last, the information about the obstacles should be ported to the solver, which would be

described in detail in the next section. With those parameters properly set in the HJ PDE solver, the optimal trajectory from initial point to target set would be obtained, if it exists.

In conclusion, the HJ PDE solver is a safety guaranteed and stable program, which helps to solve the reachability safety problems in this project, and the only difficulties remains would be determined the proper input and combine the output trajectories with the reference path, which would be described in 5.3 Controller switching.

4.2.3 Solver inputs

1) Obstacle detection

In reality, the obstacles could be detected by many different sensors, like the radars, the cameras, the ultrasonics et al. However, in the simulation, this process would be simplified as the detection of distance between the given location and the agent location. In this way, the detection would be done every time the agent moves.

Here is an example of unknown obstacles. The unknown obstacles are not considered by the RRT path, which means, it would only be avoided when detected. In the following figure, the blue circles stand for the original planned path, when the agent is close to the safety bubble of the obstacles, the safe controller would take the charge of the system. As for the known static obstacles, since the given radius of the obstacles would be the same as the detection module, when passing the known static obstacles along the RRT path, the system would not switch to the safe controller.

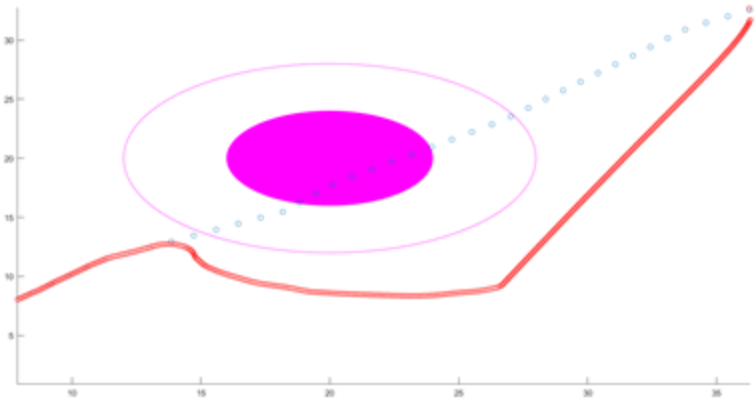

Figure 8 Example of unknown obstacle

2) Target set

When detected the obstacles, a target set should also be chosen. Since the system already obtains an optimal trajectory, recalculating another should not be a good option, which might be a time consuming process as well. Thus, the agent should

attempt to bypass the obstacles and get back to the RRT path or so-called reference path. Meanwhile, if the target position is too close to the initial position, then, it would be a waste of computation power. Under this consideration, the target should be set on or close to the reference path. Then, since the obstacles assumed the same size of safety bubble, we could just simply find a point from the reference path and located twice of the safety bubble radius away from the initial position where the possible collision was detected to occur. Then, this point is the so-called target point, the center of the target set. As for the radius of the target set, just simply set as 10. The green circle in the following figure shows what the target set looks like, the red circle stands for the obstacle, and the blue line is the region of an expanding BRS/BRT.

Nevertheless, it is possible for the obstacle to be located close to the end of the reference path. In this case, there would not exist a reference point that is located twice radius away from the initial position, and we could only simply set the target point as the terminal point. Under this situation, the safe value might not be negative, so some remedy approaches should be taken, which would explain in detail at the 5.2.4.

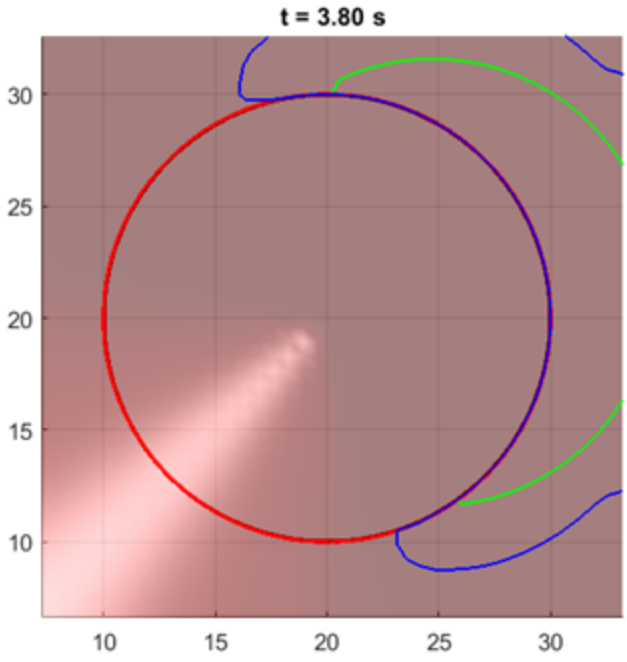

Figure 9 HJ PDE Solver Process

3) Time span

The time span is the initial running time of the solver. Normally, the new path would be slightly larger than the reference one, because more distance the agent should move. However, it still be considered as a proper initial time span, because the agent

is not supposed to arrive at the center of the target set, but only need to enter the target set, which results in a shorter distance. Then, we could simply estimate this time is equal to the time for the agent moving from the initial point to the target point. This time would be called running time.

4) Obstacle location

After obtaining the target set and time span, obstacle detection should be done again, but this time, the radius of the detection should be the distance between the initial position and the target point, such that all the possible collided obstacles, which are the potential unsafe sets, would be contained in the solver. In addition, since the dynamic obstacle is always moving, another detection should be performed at the target point, and the dynamic obstacle should be in the position where it should be after the running time. Here, since the running time is an approximation, the range of the so-called future detection should be slightly larger than the normal one.

With the given barrier information, all unsafe sets can be combined by the ShapeUnion function. If dynamic obstacles are included, static obstacles should be placed in chronological order without changing their positions. An example under this situation is as the following figure shows.

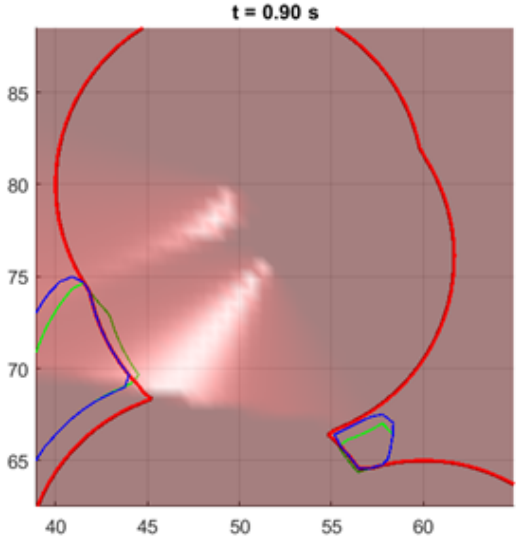

Figure 10 HJ PDE Solver Process

With those important inputs, the PDE could be solved.

4.2.4 Results

When the PDE is solved, the eval_u function would be used to judge whether the initial position is inside the BRT/BRS by resulting in a safe value, which is negative

when inside the BRT/BRS. Thus, the program has to handle both situations: when the value is less or equal than zero, then it would calculate the optimal trajectory and ask the agent to follow it; when the value is larger than zero, then some parameters should be reset.

As mentioned above, a target position that is too close may result in a positive value, not only because of space and steering angle limitations, but also because of too short running time. Thus, the program would analyze the current running time, if it is too short, then it would be enlarged and resolve the PDE; otherwise, the further target point would be given. Generally speaking, with few iterations, the value would then become negative, and the program would then back on track, and start to compute the trajectory. However, in some extreme cases, this process would be unexpectedly long. Therefore, obtaining the proper parameter could be an important improvement for this program.

4.3 Controller switching

Here, as mentioned above, it is some type of event-triggered control, when the obstacles are detected, the controller would directly switch to the safety one. However, in this section, the smoothness of the path needs to be discussed.

4.3.1 Switching to Safety controller

The switch from tracking controller to safety controller is quite smooth, for the result that when calculating the optimal trajectory, the steering angle is in consideration, which makes the angle difference smoother. Here are some example trajectories for this switch.

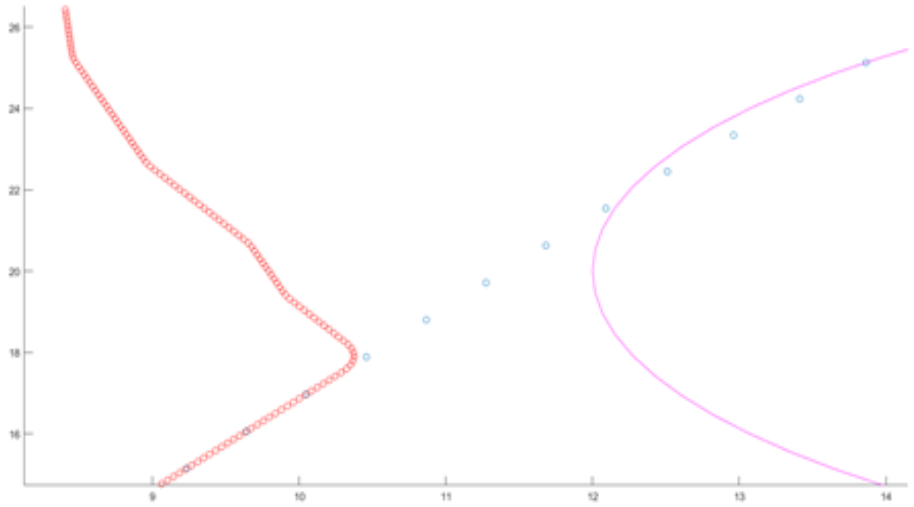

Figure 11 Trajectory Examples for controller switching

4.3.2 Back to Tracking controller

This switch would be slightly difficult, because the ending point of the safety trajectory is uncertain, which means the ending point would definitely be inside the target set, but the accurate location is unknown related to the reference path due to the running time difference. Thus, a reference point should be obtained by the agent, such that the agent could get back to the reference path.

First, the agent would find all the nearby reference points, then calculate the required steering angle change from current location to the reference point and obtain the minimum one as the reference point to get back to the path. The example is as follows. The blue line is the reference path, the red circle is the region to obtain the reference point, θ_1, θ_2, and θ_3 stand for three angle differences, and the black circles P_1, P_2, and P_3 are the candidate reference points. For this case, obviously the P_1 should be elected. Then, the tracking controller would track the straight line between the P_1 and the current location to drive the agent back to the reference path. An example for the whole trajectory for the controller switching is as follows.

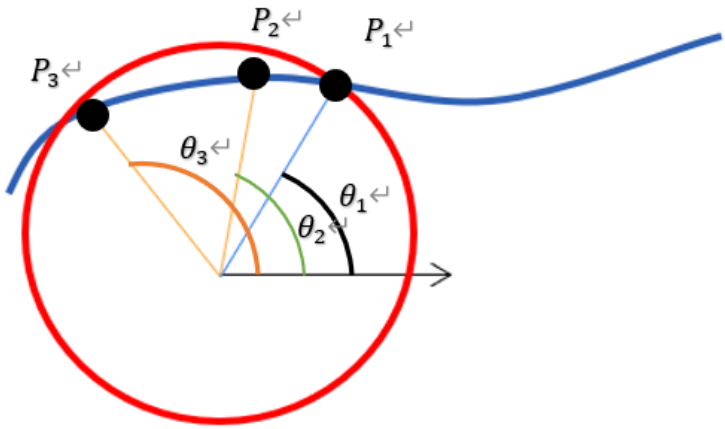

Figure 12 Reference point selection

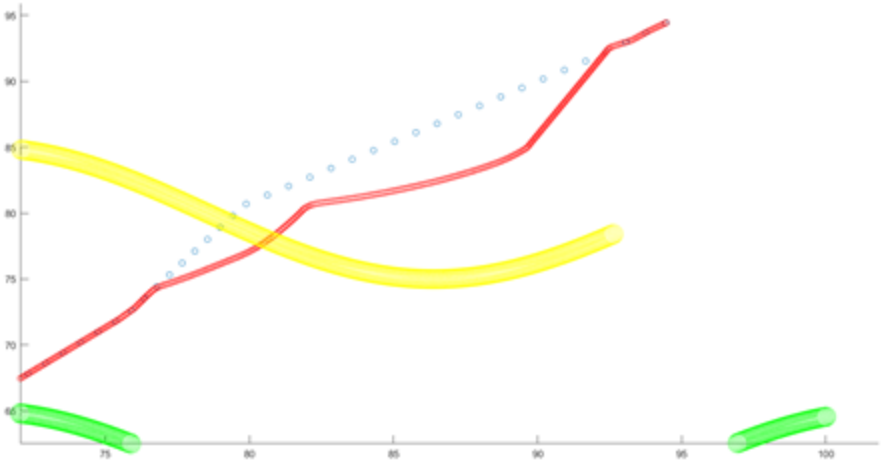

Figure 13 Example trajectory for the controller switching

5. Examples

With the techniques mentioned above, the whole path planning and tracking process could be done by the agent. Here are some detailed examples for the program. Also, the video of the examples would be provided in the attachment.

1) Without unknown obstacles

This case four static known obstacles are given, and two dynamic obstacles are moving across the area.

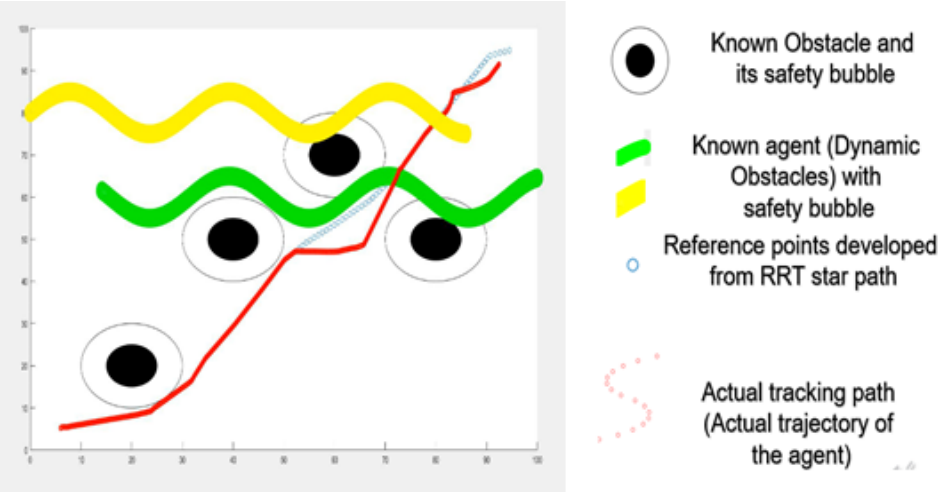

Figure 14 Example trajectory for case 1

2) With unknown obstacles

This case three static known obstacles are given, two dynamic obstacles are moving across the area, and one unknown obstacle located on the path.

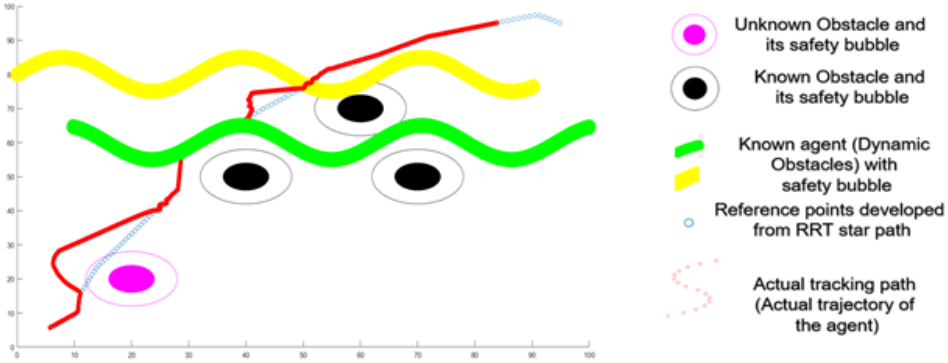

Figure 15 Example trajectory for case 1

6. Conclusions

6.1 Advantages

In conclusion, the project utilized three different methods: RRT path planning, PID path tracking, and HJ Reachability analysis for obstacle avoidance. The combination enables the agent to avoid static obstacles, unknown static obstacles, and known moving obstacles while significantly saving the computation time compared to using HJ Analysis along the whole process.

6.2 Disadvantages

First of all, the HJ Reachability analysis is still too long for real-time computation. Currently, the paths are still pre-computed and cannot apply to the real-time navigation needs for agents. Secondly, the current trajectory might have sharp turns in certain situations (figure 15). This might be caused by the hard switching method between PID and HJ analysis. Finally, disturbances are not considered in this simulation, and it is unknown that how this can affect the final trajectory.

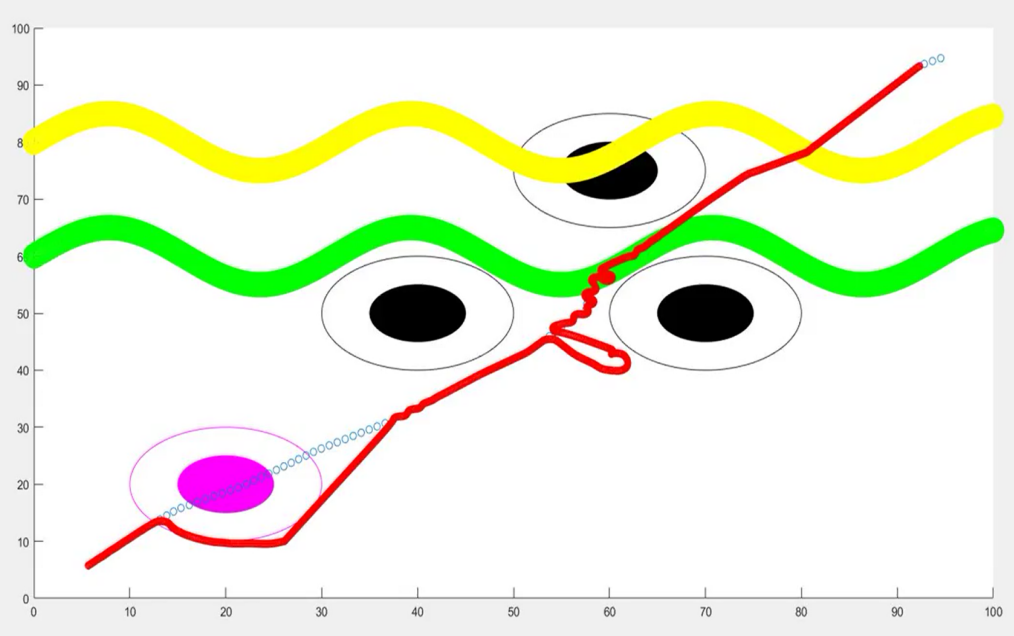

Figure 16 Sharp turns

6.3 Future works

The future works will gradually solve the issues discussed earlier. The first step is to add disturbances to the system to check its effect on the trajectory. Secondly, a better controller switching method needs to be implemented to solve the sharp-turn issue. So far, the current path planning & tracking simulation is complete. To solve the issue of real-time computation, different algorithms which suit better for real-time obstacle avoidance cases need to be tested and repeat all the steps mentioned above.

Reference

[1] LaValle, Steven M. "Rapidly-exploring random trees: A new tool for path planning." (1998): 98-11.

[2] Chen, Mo, and Claire J. Tomlin. "Hamilton–Jacobi reachability: Some recent theoretical advances and applications in unmanned airspace management." Annual Review of Control, Robotics, and Autonomous Systems 1 (2018): 333-358.

[3] Park, Chang-Su, Min-Jea Tahk, and Hyochoong Bang. "Multiple aerial vehicle formation using swarm intelligence." AIAA Guidance, Navigation, and Control Conference and Exhibit. 2003.

[4] Bansal, Somil, et al. "Hamilton-jacobi reachability: A brief overview and recent advances." 2017 IEEE 56th Annual Conference on Decision and Control (CDC). IEEE, 2017.

[5] Chen Mo (2016) helperOC,

https://github.com/HJReachability/helperOC.git

[6] Ian Mitchell (2007) the Toolbox of Level Set Method,

https://www.cs.ubc.ca/~mitchell/ToolboxLS/