Motion Planning in Dynamic Environment

2100017408 王翼飞

2023.7.2

1 Introduction

This work is concerned with motion planning in dynamic, uncertain environments, such as robot manipulators avoiding moving obstacles, and Non-player characters interacting with players.

Motion planning in dynamic environments is a difficult problem since it requires planning in the state space, i.e., simultaneously solving the path planning and the velocity planning problems. Before reinforcement learning became the mainstream method to solve such problems, traditional methods for local path planning that focus on velocity obstacles(VO) are introduced.[2] The VO method produced a hierarchical path planning framework that involves the global path planner and local path planner.

In this project, I present an implementation of velocity obstacles. My work is based on the github repository of atb033, which can be found in https://github.com/atb033/multi_agent_path_planning. However, the original project presents a toy model which only considers one robot and obstacles with constant velocity. My contribution to the project can be summarized below:

- I enhanced the problem to multi-agents with multi-obstacles. Such a problem is more complicated since agents are required to avoid collision with other obstacles that do not have constant velocities. Users can customize the velocity as well as the number of agents.
- I observed that the robot may "jitter" after applying the original method, such behavior is time-consuming and energy inefficient. To tackle this problem, I introduced a distance threshold to each robot and achieved an effective result.

2 Related Work

In this section, I will elaborate on the VO method. In this project, I focus on two dimensions with disc-shaped robots and obstacles.

2.1 VO

The velocity obstacle(VO) was introduced for local collision avoidance and navigation of a robot amongst multiple moving obstacles.[2][3] It can be defined as follows:

Let A be a robot and B be a dynamic obstacle. p_A and p_B donate the current positions of A and B, respectively. The velocity obstacle for robot A induced by obstacle B, written $VO_{A|B}$, defines the set of A's velocities which will result in a collision between A and B.

To be more specific,let p_A and p_B denote the current positions of A and B, $D(p, r)$ be a open disc of radius r centered at p, then:

$$VO_{A|B} = \{v|\exists t > 0 :: t(v - vB) \in D(p_B - p_A, r_a + r_B)\} \quad (1)$$

However, the velocity obstacle method may result in oscillations, which is the phenomenon of A and B selecting their old velocities as their new velocities.

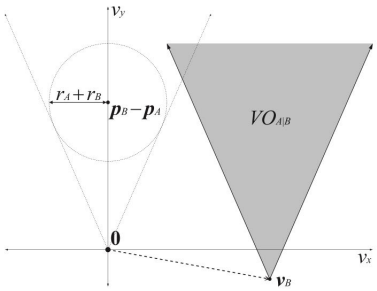

fig 1: The velocity obstacle $VO_{A|B}$ for robot A induced by B.

2.2 RVO

The reciprocal velocity obstacle(RVO)[6] focuses on the problem of oscillations caused by the velocity obstacle by incorporating the reactive nature of other robots. The robot assumes the other robot will react too to avoid a collision. As a result, it only takes half of the responsibility for avoiding a collision. The reciprocal velocity obstacle for robot A induced by B, written $RVO_{A|B}$, is defined as

$$RVO_{A|B} = \{v|2v - v_A \in VO_{A|B}\} \qquad (2)$$

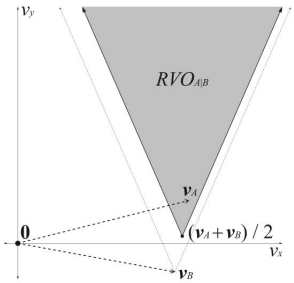

fig 2: The reciprocal velocity obstacle $RVO_{A|B}$ for robot A induced by B.

2.3 HRVO

The hybrid reciprocal velocity obstacle(HRVO) formulation has the consequence that if robot A attempts to pass on the wrong side of robot B, then it has to give full priority to robot B, as with the velocity obstacle.[5] However, if it chooses the correct side, it can assume the cooperation of robot B and retains equal priority, as for the RVO. This greatly reduces the possibility of oscillations.

To be more specific, for robots A and B, if v_A is to the right of the center line of $RVO_{A|B}$, we wish robot A to choose a velocity to the right of $RVO_{A|B}$. To encourage this, the reciprocal velocity obstacle is enlarged by replacing the edge on the side we do not wish the robot to pass, in this instance the left side, by the edge of the velocity obstacle $VO_{A|B}$. The apex of the resulting obstacle corresponds to the point of the intersection between the right side of $RVO_{A|B}$. If v_A is to the left of the centerline, we mirror the procedure, exchanging left and right.

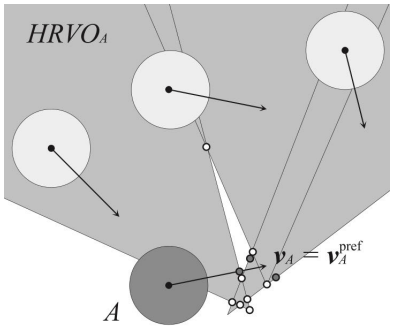

fig 3: The combined hybrid reciprocal velocity obstacle for robot A is the union of all HRVC induced by the other robots.

3 Problem formulation

In this project, I considered the following definition of the problem of navigating multiple agents.

Let there be a set of disc-shaped robots sharing an environment in the plane. The environment may also contain dynamic obstacles and static obstacles. Obstacles have constant speed while robots' velocity changes based on others'

behavior. Each robot A has a fixed radius r_A, a current position p_A, and a current velocity v_A. Let each robot also have a goal located at p_A^{goal} and a preferred speed v_A^{pref}. The preferred speed is set as V_{max} and the goal may simply be a fixed point chosen in the plane.[5]

The objective of each robot is to independently and simultaneously choose a new velocity at each time step to compute a trajectory toward its goal without collisions with any other robots or obstacles and with as few oscillations as possible. To simplify the problem, robots select their speed in a fixed order.

4 Proposed method

I selected the VO method as the basic idea of the project. I also introduced the idea of RVO and HRVO into the project to help with motion. The whole process can be elaborate below:

4.1 Single agent path planning

In the original project, the author only considered one agent with 4 moving obstacles that have a constant speed. The whole process was broken into 50 time-steps. In each time step, the agent was required to select the velocity which is outside the velocity obstacle and has the closest distance to the desired velocity. The desired velocity is simply v_{max} with the direction from the current position to the goal. For the computation of the velocity obstacle space, specific code is in the attached files.

Algorithm 1 Single agent path planning

```
obstacles = create_obstacles()
for timestep in timestpes do
    desired_v = desired_velocity(pos, goal)
    v = compute_velocity()
    update
end for
```

4.2 Multi-agent path planning

To generalize the problem, I introduced multi-agents and multi-obstacles to the project. Agents can choose their velocity based on the current situation while obstacles travel at a constant velocity. When computing the velocity of a specific agent, I treated other agents as obstacles. In each time step, the velocities of agents were computed in a fixed order. Based on this idea, I present the algorithm below:

Algorithm 2 Multi-agent path planning

```
obstacles = create_obstacles()
for agent in agents do
    for timestep in timestpes do
        des_v = desired_velocity(pos, goal)
        v = compute_velocity()
        update
    end for
end for
```

As we can see, the velocities of each round were not computed simultaneously. This adaption was made to simplify the problem. The altered algorithm still showed state-of-the-art performance.

4.3 Distance threshold

Often, the robot's motion may result in oscillation after applying the algorithm. To solve this problem, I introduced a distance threshold mechanism to my project, which only allows the robot to change its desired velocity when there exists a moving object whose distance is smaller than the threshold. The idea can be shown as follows:

5 Simulation

I presented several simulations to show the result of the project. The algorithm can easily find paths for single agents, multiple agents with

Algorithm 3 Distance threshold

```
for robot in robots do
    flag = 1
    v = compute_velocity()
    for obstacle in obstacles do
        if dis(robot,obstacle)<threshold then
            flag = 0
        end if
    end for
    if flag then
        v = v_desired
    end if
end for
```

walls, and multiple agents with moving obstacles(fig 4,5,6). In the figures shown below, the green circle stands for robots, which selects their velocity based on the current situation. Blue circles stand for obstacles, which move at a constant speed. Obstacle collisions were not taken into consideration. A better understanding of the path is visualized through the gif images in the attached files.

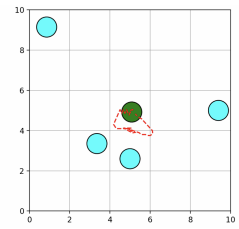

Fig 4: Single agent

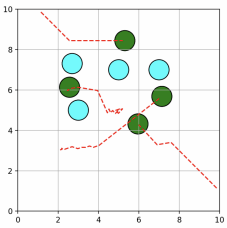

Fig 5: obstacles with zero velocities(walls)

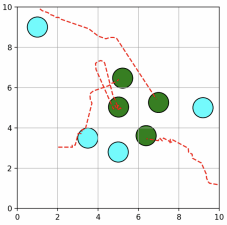

Fig 6: multi-agents with moving obstacles

5.1 Distance threshold

Fig 7 and Fig 8 show the differences between using and not using the distance threshold. We can see that the center robot's motion "jitters" in the beginning, and had a more smoothed path after I used distance thresholding.

More surprisingly, the right corner robot's path lengthened after thresholding, which means fifty time-steps are not enough for the robot to reach the goal. But thresholding produces a more efficient path for the agent.

However, there are a few drawbacks to this method. To start with, the chances that pathfinding fails to increase. Due to this problem, how much should be set as a threshold needs to be carefully considered? Currently, I choose 2 and successfully find the correct path most of the time.

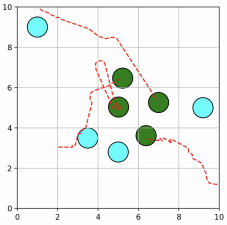

Fig 7: Before thresholding

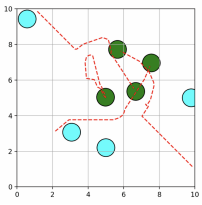

Fig 8: After thresholding

More examples after thresholding are listed

here:

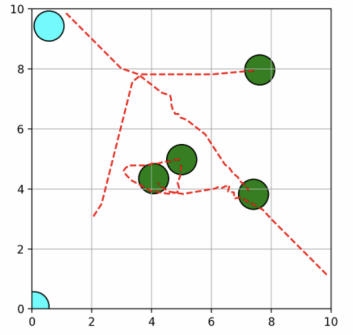

Fig 9

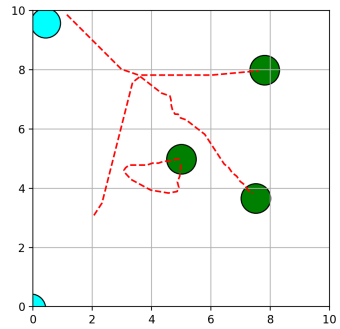

Fig 10

6 Conclusion

In this project, I used Python to implement the hybrid reciprocal velocity obstacles for navigating multiple mobile robots and did thorough research on this method using different numbers of robots as well as different velocities. I also added a distance threshold so that robots can move more efficiently. Direct, collision-free, and oscillation-free navigation were shown. However, more theoretical analysis needs to be done on the distance threshold method.

In the future, I would like to add more sophisticated obstacles to the project. Such as a rectangle wall or moving square obstacles. The size of the obstacles may also be altered to test the robustness of the algorithm.[4]

To tackle a more complicated problem, global search such as A* algorithm or RRT should be added. Combined with the VO method, I may present a universal path-finding framework.[1] The problem is of great significance due to its wide application in today's world.

7 Appendix

This section briefly explains the components of the project. The project can be found at https://github.com/a-little-hoof/robotics.

- Python files: vo.py contains the main algorithm of velocity obstacles, and provides parameters for users to define. utils.py contains helper functions. draw.py helps us visualize the result.
- Gif files: simulations using different parameters.
- report.pdf: Project report
- requirements.txt: essential python libraries

References

[1] Kuanqi Cai et al. "Mobile robot path planning in dynamic environments: A survey". In: *arXiv preprint arXiv:2006.14195* (2020).

[2] Paolo Fiorini and Zvi Shiller. "Motion planning in dynamic environments using velocity obstacles". In: *The international journal of robotics research* 17.7 (1998), pp. 760–772.

[3] MG Mohanan and Ambuja Salgoankar. "A survey of robotic motion planning in dynamic environments". In: *Robotics and Autonomous Systems* 100 (2018), pp. 171–185.

[4] S. Petti and T. Fraichard. "Safe motion planning in dynamic environments". In: *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2005, pp. 2210–2215. DOI: 10.1109/IROS.2005.1545549.

[5] Jamie Snape et al. “The hybrid reciprocal velocity obstacle”. In: *IEEE Transactions on Robotics* 27.4 (2011), pp. 696–706.

[6] Jur Van den Berg, Ming Lin, and Dinesh Manocha. “Reciprocal velocity obstacles for real-time multi-agent navigation”. In: *2008 IEEE international conference on robotics and automation*. Ieee. 2008, pp. 1928–1935.