Anytime path planning algorithm for obstacle avoidance in dynamic environment

Benjamin Russ
Mechanical Engineering
University of Cincinnati
russbj@mail.uc.edu

Remi Carteret
Mechanical Engineering
University of Cincinnati
carter5@mail.uc.edu

Dipin Nair
Mechanical Engineering
University of Cincinnati
nairdk@mail.uc.edu

***Abstract*—Mobile robots have seen significant development over the years. Path planning in predictable or static environments have largely been solved, while dynamic environments pose significant practical challenges. This project looks to add a possible development path of a hybrid approach. The agent progressing through the environment will stick to an initial path generated from the global frame by central computation. Then, the path will be recalculated only when an obstacle is close and interfering with the current path. Once past the interfering obstacle, the agent will reconnect with the original path. This hopes to distribute the computation burden between the central computer and agent which will free up some energy on the agent and less central management**

***Index Terms*—Path planning, Obstacle avoidance, Dijkstra, Dynamic**

I. Introduction

Mobile robots have been the recent tool for navigation and various dynamic tasks. The dynamic environments that are typically posed include many unpredictable and infinite variation of shapes. The combination of these two factors provides the greatest source of challenge in these systems.

A. Problem Formulation

The most common application includes a warehouse(Fig. 1) setting in which there are some general constraints such as organized racks and planned synergies. Additionally modern warehouses are typically outfitted with some management systems which control the flow of product and track the positions of the workers and in/out going packages. Clearly, the primary benchmark to any major system for these businesses is product throughput. Thus, in the case of mobile robots, speed and responsiveness will drive this value in this specific application. To facilitate these two parameters, there are mechanical

Fig. 1: Self driving car and Amazon Warehouse robots autonomously carrying out tasks

design and component level challenges which are assumed to be optimal in this project. Rather, this project looks at how the process of path planning is conducted and possible optimizations.Transferring the obstacle avoidance computation will reduce the latency for decision making and response to obstacle interference. While the full initial path computed in the central control system required minimal system resources.

The expected result from this combination is an initial path which properly avoids static objects such as racks and static machinery while appropriately responding the distance triggers and rerouting. Ultimately this algorithm could be optimized further using an anytime-algorithm type approach. [1] [2]. This would mean that not only would the re planning of the path triggered by distance, but the solution time could be limited and would require probabilities to be assigned to the various paths.

The following project assesses this warehouse scenario and hopes to examine the validity of spreading the computation burden with the goal of reducing response latency to the dynamic objects.

II. Methodology

Real time path planning proposed method consists of 3 main components which are environment, solver, agent. Here we use the D* solver to get initial optimal path at t = 0, then plan the route in dynamic environment actively avoiding obstacle

A. Environment setup

Environment of $N \times N$ grid matrix is created with walls as boundary. Logical values used to created both wall and obstacles. Free cells are marked as 0 and other as 1. The dynamic obstacles are then integrated to the environment with collision dynamics. When moving obstacles are collided with walls, it rebounds back and goes in opposite direction. The environment we used for this project can be seen from the figure 2

B. Solver

After the introduction of autonomous robots, researches in path planning field has become very relevant. There are many path planning algorithm out there widely used which are: A*, D*, RRTs, depth-first search (DFS), etc. In our study we chose Dijskstra algorithm(D*) which searches the entire space and provides global minimum.

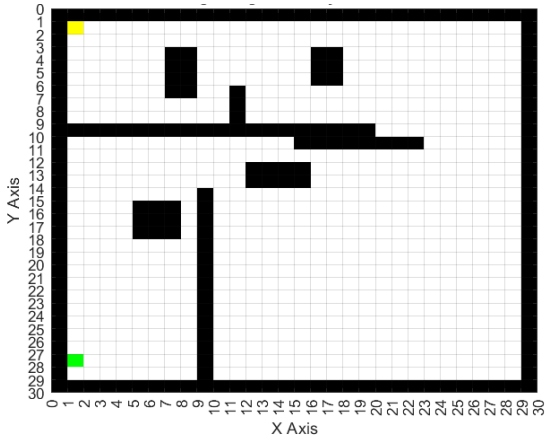


Fig. 2: Environment at t=0, Start (green) position [28,2] , Goal (yellow) [2,2]

Algorithm 1: Dijkstra's algorithm

Input : Environment, start, goal
Output: route

for *each node* $n \in N \times N$ **do**
  $distFromStart \leftarrow \infty$
  $parent \leftarrow NIL$
  $map \leftarrow 0\&1$
end
Current = start
$distFromStart(current) = 0$
while $Current \neq goal$ **do**
  $minDist = min(distFromStart)$
  $[i, j] = index(minDist)$
  for *each near node of [i,j]* **do**
    if *map(near node)* $>$ *minDist + cost* **then**
      if *no parent & no obstacle* **then**
        distFromStart = cost
        parent(near node) = current
      end
    end
  end
  distFromStart(current) = ∞
end
route = goal
while *parent(route(1)) = 0)* **do**
  route = [parent(route(1)), route]
end

D* algorithm provides the best route possible avoiding both static obstacle and dynamic obstacle from $t = 0$ and provides to the agent. The same D* also being used when agent tries to avoid obstacle in local scope which we will discuss more in next section.

C. Agent

Agent starts to move through initial optimized path after other initialization processes are complete. If there are no disturbances along the path it will follow it until the destination is met. This means that no computation was done by our agent other than following the path. This frees the system for other tasks and can trust that there won't be any collisions if an obstacle isn't detected within a specified range.

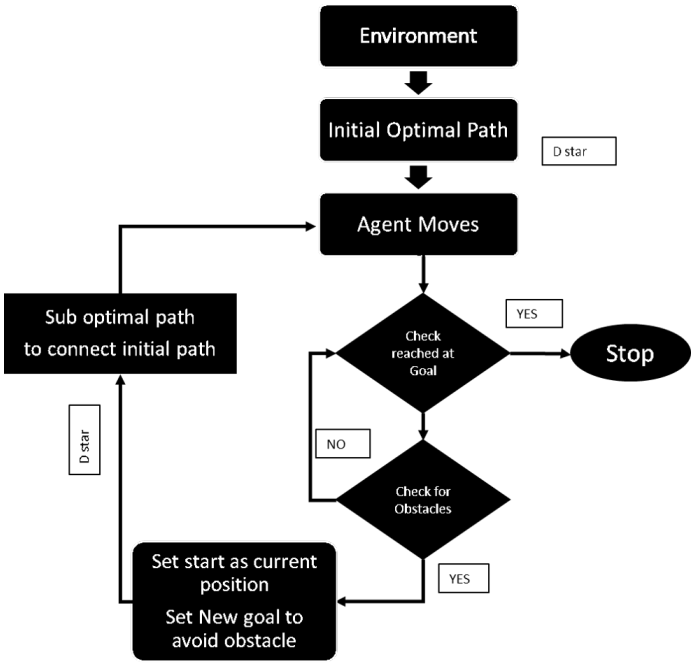
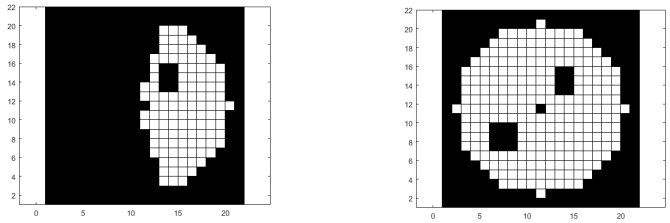


Fig. 3: Block diagram of Path planning



(a) Computation space (b) Full vision

Fig. 4: Agent vision in local frame

When an object is detected, it must be determined if action is required. First, the assessment is made whether the object is actually obstructing the path. If it is, then the agent looks to the farthest point in the opposite direction of travel for the object, and it plots a new path up to that point. If the agent can not see the end of the object, it will plot as far as it can "see". Once the agent has successfully navigated around the object, it will find the closest point of the original optimal path and proceed as discussed prior. The position will be given the entire process until the agent has found the destination.

The last situation considered is whether the current path is valid based on the velocity of the object. If the object is within range to take action but the agent can still make it passed based on the velocity of the incoming object, the path will be continued. If the object will cross the path and result in a collision if the path is continued, action will be taken. The same method is applied by finding the farthest point on the object and going towards that temporary destination.

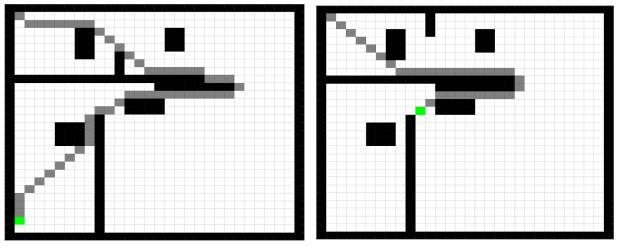
To ease computation for the agent further, what is captured in vision is not fully computed for a new path. Rather, the vision is reduced by an angle theta until a minimum field of view (FOV) is met (Fig 4). The amount of the vision space to compute is then determined by the distance to the closest object.

To satisfy this algorithm, the range before avoidance is required as a parameter and can be tuned depending on the danger of collision and specific environmental conditions. The angle of the computation space will then scale squared to that distance to avoid getting stuck in computation and unable to pass an object.

III. Results

a) Expected result: A random map can be generated having static and dynamic obstacles. Every time step, the environment provides detailed information about the velocity, size and direction of each obstacle presented in the agent vision range. Then, the agent takes the decision when and where to run the path planning algorithm to avoid obstacles next to it.

b) Actual result: The vision process that was supposed to decide when and where to re-run the algorithm was developed but could not correctly communicate with the environment. Therefore, its capacity to avoid obstacle could not be correctly tested. At this point, the D* algorithm is relevant regarding the size of the map. It's running every time and the path is updated considering the fixed obstacle positions and the dynamic obstacle latest positions. The agent moves then to the next position of the current path. The agent can reach the goal in a map having any random fixed obstacles and random dynamic obstacles moving along the X or Y axis.



(a) Agent location at $t = 0$ (b) Agent location at $t = 14$

Fig. 5: Agent (green) movement in dynamic environment

The time taken for the simulation 65.51 sec on core i5 gen-10 processor.

IV. Conclusion

Given the developmental challenges of this project, a few perspectives can be gleamed. First, a centralized computation approach with global knowledge will be the simplest to implement. The target and possible outcomes being in view and tracking the dynamic obstacle throughout provides the most robust response to this problem. However, given the practical challenges mentioned prior, our investigation did not fully uncover the benefits of local computation. While it has been demonstrated that it is possible to develop a hybrid computation system and distribute the burden, the complications became readily apparent. The decision process to redirect the path likely required more central input to limit the complexity such as grid valuation and a utility approach. Overall, the base central control system was robust and yielded some results which demonstrate the effectiveness given this scenario.

A. Challenges

Path planning using anytime algorithm is very challenging since the agent has to find a sub optimal route to avoid the obstacles. These are the main challenges we encountered during this project:

- Setting up dynamic environment with collision physics.
- The Genetic algorithm was the initial choice of the solver, but developing GA including diagonal movement was found to be very difficult.
- During the movement there is a chance that obstacle can come towards the agent and collide with it.
- Re-planning locally and reconnect the initial path

B. Future work

Our current methods was able to produce optimal path at each instance whenever it gets interrupted by dynamic obstacle effectively. But it takes more computational time because the D* had to search the entire map and find optimal path. In future work, we plan to effectively integrate the agent. On the other hand, complete all functionality of agent like collision mechanics, adding object occlusion,and effective communication by transforming the local and global frames. In addition to this, scaling environment and adding more obstacle with different shape is also an area where it can be improved.

References

[1] S. Zilberstein, "Using Anytime Algorithms in Intelligent Systems" AI MAGAZINE., pp 73-83, FALL 1996.

[2] A. Jesus, A. Liefooghe, B. Derbel, L. Paquete, "Algorithm Selection of Anytime Algorithms" in Computation Conference (GECCO '20), July 8–12, 2020, Cancún, Mexico.ACM,New York, NY, USA, 9 page

[3] https://github.com/YashBansod