Multi-Robot Motion Planning for Warehouse Management

Dushyant Patil, Omkar Bharambe, Ankit Talele

I. Introduction

Motion planning and navigation are currently a very important part of robotics and this is an ever-growing field of robotics. Algorithms are getting more efficient day by day and the problem solutions are reducing with it. One such problem which we plan to tackle is the path planning of multiple robots in warehouse automation. It is a known fact that the warehouses are growing more and more towards an automated workspace where humans are not needed anymore. This increases its efficiency as well as the number of human errors. A warehouse robot is an autonomous machine designed to replace or augment human effort in a factory environment as a form of automation. Some warehouse robots are solely designed to transfer material between warehouse slots or between slots and the loading dock. Other robots can reconfigure the warehouse itself, moving entire racks around to create a more efficient space. Although the warehouse industry has made huge strides to achieve certain level of autonomy, a fully autonomous warehouse is still a distant goal which can be achieved with optimization of existing technology and exploration of new frontiers. In this project we try to explore existing as well as novel motion planning algorithms to solve warehouse management problem.

II. Background Problem

Warehouse and manufacturing involves day to day tasks which might be risky and repetitive for human labor. As robotics engineers, we strive to eliminate 3D's of daily tasks - dirty, dangerous and dull. Through this project, we aim to get rid of the 2D's - dangerous and dull from the warehouse of manufacturing industry. One of the main objectives of warehouses along with worker safety is to improve productivity, logistic enhancement and storage optimization. Warehouse robotics offers optimal solutions to these problems. Due to these benefits of warehouse robotics, various manufacturing industries are adopting the use of robots for warehouse management.

Warehouse automation can be implemented with various techniques such as Rail navigation, Wire-guided navigation, Label-based navigation, Vision-based navigation, etc. While each method satisfies the goals of the industry, it does so with different accuracy, efficiency and costs. Techniques like Rail navigation and Wire-guided navigation need infrastructure to be set up and generally cost more. Hence, we chose Vision-based navigation for this project which is cost-effective, structured and productive. We aim to use vision-based navigation to implement multi-robot path planning for non-holonomic robots in warehouse.

Application

As of 2023, Amazon, for example, operated over 500,000 robots in its warehouses, and other e-commerce giants have been quick to follow suit. Walmart, for example, is adding autonomous robots to 25 of its distribution centers starting in 2021, according to TechCrunch.[1]

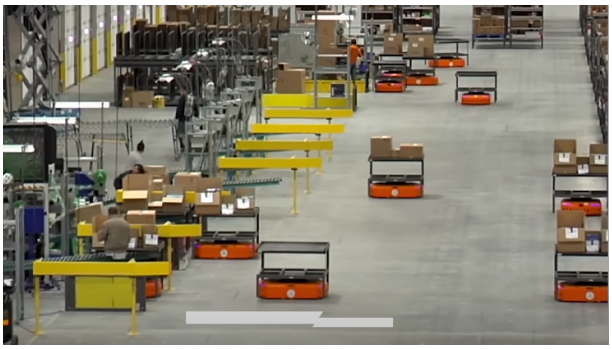

Fig. 1: Amazon Robotics Warehouse - Multi Robot System

Fig. 2: Multi-robot rail navigation system at Ocado Technology

Objective

Our objective for this project was to implement warehouse path planning for rail guided navigation system like the one shown in Fig.2. We aim to use robots to collect items for shipment/ dispatch from multiple shelves/ racks and collect them at a loading dock from where they will be dispatched. We studied different algorithms to achieve this objective. We decided to move ahead with the Cooperative A-Star algorithm[2] and its variant for their simplicity and effectiveness.

III. Proposed Solution and Algorithms

A. *Environment Setup*

We used a 2D grid environment to emulate a standard warehouse setup. The warehouse setup has 2 rows of shelves. Each row has 4 shelves. Each shelf contains different objects. The environment setup is shown in figure below.

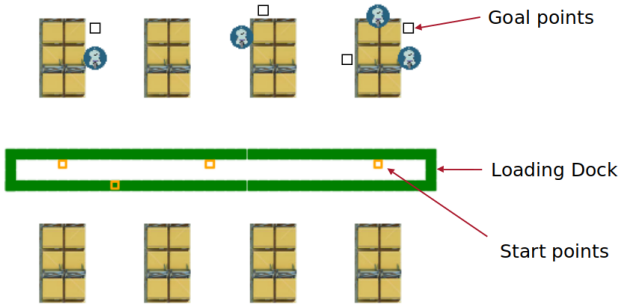

Fig. 3: Warehouse environment

As shown in the figure above, the robots start from the loading dock (Green box) and go towards specific shelf depending on which objects they need to retrieve. Then the robots bring back the objects to the loading dock. The loading dock area is not an obstacle for robots (We are assuming timely clearance of objects from loading dock for dispatch). Thus the only obstacles for robots in this project are the shelves and other robots. The shelves act as static obstacles while other robots being mobile act as dynamic obstacles. The robots start from the stat points shown in orange box in Fig. 3 and move to goal points shown in black boxes in Fig. 3 while avoiding other robots. We achieve this with the help of Cooperative A* algorithm.

B. *Cooperative A**

Our global path planning is done using Cooperative A* . Cooperative A* (CA*) is a algorithm designed to solve the Cooperative Pathfinding problem. The problem is divided into a series of individual searches, where each agent's planned route is taken into account. Each robot can perform 5 possible motion - [go up, go down, go right, go left, stay at the same point]. The algorithm operates in a three-dimensional space-time, where the agents can wait in one place without moving. Each individual robot's traversed path is shared with all other robots. After each agent calculates their route, the states along the route are marked into a reservation table, and subsequent agents avoid these areas. This table represents shared knowledge about each other's planned routes and is a sparse data structure that marks off regions of space-time.

The reservation table is treated as a 3-dimensional grid, with each cell that is intersected by an agent's planned route marked as impassable. The grid is efficiently implemented as a hash table, where only a small proportion of grid locations are touched. The algorithm is not sensitive to the ordering of the agents and can work with agents of varying speeds or sizes.

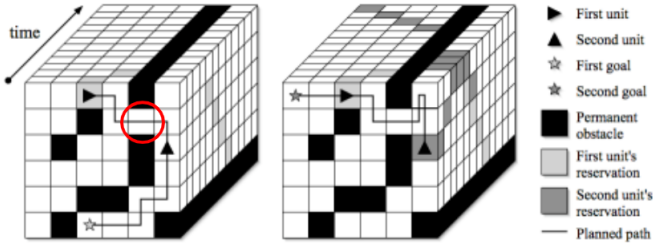

Fig. 4: Reservation table[3]

The reservation table shown in figure above is used for path planning of 2 robots. The table contain (x,y,t) information of each robot i.e. the position of the robots at each time step. In above case, both the robots have same (x,y,t) value for the point shown by red circle. As the robots are bound to collide with each other in the area marked by red circle, the cooperative algorithm forces second robot to reroute its path till the first robot passes through this bottleneck region.
We planned on using Manhattan distance as heuristic function for the A* search of individual robots. However, this can lead to poor performance in more challenging environments, and a better heuristic can be used to help reduce computation. Thus we switched to Euclidean distance as a heuristic.

Drawbacks of CA*:

It is important to note that any the CA* algorithms can use prioritized planning should be used for good performance but it still faces some issues in corner cases. The CA* algorithm suffers from head-on collision problem. For example shown in Fig.5, the robot1 is in between the start point and goal point of robot2 and vice versa. Both the robots travel on the same line to reach their goals in greedy manner. But this leads to a collision. The cooperative A* does not give a solution to this problem and both robots get stuck in their path. Another problem which CA* is not able to solve is of cooperation after reaching destination. When robot R_i reaches its destination and another robot R_j has their rerouted path which goes through the destination of R_i, the robot R_i does not move away from its destination and forces R_j to find a new path.

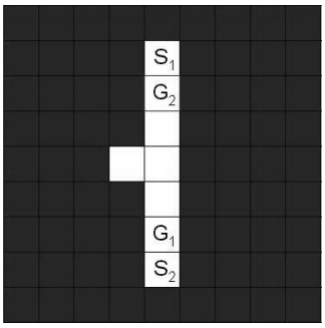

Fig. 5: Cooperative A* unable to solve this planning problem.

In summary, CA* is a promising algorithm for solving the Cooperative Pathfinding problem, which efficiently uses shared knowledge about planned routes of agents. The algorithm is not sensitive to the ordering of agents and is capable of working with agents of varying speeds or sizes. But it is prone to a few corner cases which increase execution time and might result in failure in a few cases. To solve this, we performed a path rerouting using simplistic Delayed Rerouting, Hierarchical Cooperative A Star, Windowed Hierarchical Cooperative A Star algorithms and selected a more suitable algorithms with optimal performance.

C. *Simplistic Delayed Rerouting*

In this algorithm, we determined the equation of line for each robot for 2 or more consecutive time steps. Using these equations, we determine if 2 robots travel the same line. If the 2 lines overlap for a nonzero time duration, we detect a possible collision. Using the line equation and time data for each robot, we determine the duration of overlap for colliding robots. The robot which whose global path was planned later is given lower importance for continuing the global CA* path. The lower importance robot has to reroute its path before the overlap of lines start. The robot will wait till the primary robot completes the line traversal. If waiting does not resolve the issue, we reroute the robot to different path avoiding primary robot's path altogether. This algorithm solves the head on collision issue. But as the preference is given to waiting, the total execution time becomes very lengthy as the number of robots increases.

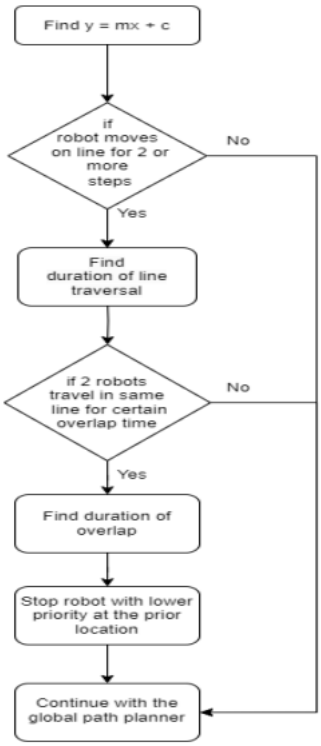

Fig. 6: Simplistic Delayed Rerouting Algorithm

D. *Hierarchical Cooperative A**

We also looked into a method called Hierarchical Cooperative A* (HCA*) for solving the problem of cooperative pathfinding. HCA* uses a simple hierarchy with a single domain abstraction, which is a 2-dimensional map with all agents removed, and computes abstract distances on demand using a modified A* search algorithm called Reverse Resumable A* (RRA*). The RRA* algorithm executes a modified A* search in a reverse direction to calculate the optimal distance from a specified node to the goal. The pseudocode for RRA* algorithm is shown in Algorithm 1. The HCA* is similar to CA* but is uses a better heuristic in the form of the RRA* abstract distance. Thus, HCA* algorithm is able to overcome the head-on collision drawback of CA*. But the algorithms stops finding path for other robots when they reach their destination. Thus the problem of 'no cooperation after reaching destination' persists even in HCA*.

Algorithm 1 Reverse Resumable A*

procedure INITIALISERRA*(O, G)
  $G.g \leftarrow 0$
  $G.h \leftarrow$ MANHATTAN(G, O)
  $Open \leftarrow \{G\}$
  $Closed \leftarrow \emptyset$
  RESUMERRA*(O)
end procedure

procedure RESUMERRA*(N)
  while Open $\neq \emptyset$ **do**
    $P \leftarrow pop(Open)$
    $Closed \xleftarrow{add} P$
    if $P = N$ **then**
      return success
    end if
    for all $Q \in reverse($SUCCESSORS$(P))$ **do**
      $Q.g \leftarrow P.g +$ COST(P, Q)
      $Q.h \leftarrow$ MANHATTAN(Q, O)
      if $Q \notin Open$ **and** $Q \notin Closed$ **then**
        $Open \xleftarrow{add} Q$
      end if
      if $Q \in Open$ **and** $f(Q) < f(Q$ in $Open)$ **then**
        $Open \xleftarrow{update} Q$
      end if
    end for
  end while
  return failure
end procedure

procedure ABSTRACTDIST(N, G)
  if $N \in Closed$ **then**
    return g(N)
  end if
  if RESUMERRA*$(N) =$ **success then**
    return g(N)
  end if
  return $+\infty$
end procedure

Fig. 7: RRA* algorithm

E. *Windowed Hierarchical Cooperative A**

We have implemented the WHCA* algorithm, which is a cooperative pathfinding algorithm for multi-agent systems. The algorithm addresses several issues present in previous pathfinding algorithms, including blocking of paths, sensitivity to agent ordering, and the need to calculate a complete route to the destination.

One solution to these issues is windowing the search. This means that each agent searches for a partial route to its destination, and the search is limited to a fixed depth specified by the current window. At regular intervals, the window is shifted forward and a new partial route is computed. This approach reduces the complexity of the search, as agents are only considered for a limited time, and processing time can be spread across all agents.

To search the new search space efficiently, a trick is used. Once the window has elapsed, agents are ignored, and the search space becomes identical to the abstract search space. This means that the abstract distance provides the same information as completing the search. For each node reached after the window, a special terminal edge is introduced, going directly from the node to the destination, with a cost equal to the abstract distance from the node to the destination. The search is reduced to a window of size w using the abstract distance heuristic introduced for HCA*.

In general, the edge cost function for WHCA* is:

$$\text{COST}(P, Q) = \begin{cases} 0 & \text{if } P = Q = G, t < w \\ \text{ABSTRACTDIST}(P, G) & \text{if } t = w \\ 1 & \text{otherwise} \end{cases}$$

Fig. 8: WHCA* edge cost function

In addition, the windowed search can continue once the agent has reached its destination. The agent's goal is no longer to reach the destination but to complete the window via a terminal edge. Any sequence of w moves will reach the goal, but the WHCA* search will efficiently find the lowest cost sequence. This optimal sequence represents the partial route that will take the agent closest to its destination, and once there, to stay on the destination for as much time as possible.

Finally, the RRA* search results can be reused for each consecutive window. This requires each agent to store its own Open and Closed lists. An initial RRA* search is performed for each agent from its original position to its goal. For each subsequent window, the RRA* search is resumed to take account of any new nodes encountered. For consistency, it must continue to search towards the agent's original position and not the agent's current position.

In conclusion, the WHCA* algorithm is a cooperative pathfinding algorithm that addresses several issues present in previous pathfinding algorithms. The algorithm window the search, uses a trick to reduce the search space, and allows the windowed search to continue once the agent has reached its destination. The RRA* search results can be reused for each consecutive window, reducing the complexity of the search.

IV. Results and Observations

The performance checks for A* and CA* were simulated in special environments. This was done to create certain situations which demonstrate the drawbacks of the two algorithms in avoiding dynamics obstacles. While A* does not take into account the dynamic obstacles at all, CA* does a better job at avoiding them but fails in situations where there is a possibility of head-on collision or when a robot already reaches it goal and is not considered an obstacle altogether. The drawbacks of CA* were tackled by performing live rerouting with the help of WHCA* algorithms. We evaluated the performance of WHCA* by varying number of robots and window sizes in the warehouse environment as well as specially curated environments which showcase drawback of previous algorithms. The demonstrations of all the test cases are linked below for reader's reference.

Demonstrations Performed

1) A* performance and drawbacks
2) CA* performance and drawbacks
3) Simplistic Delayed Rerouting Algorithm
4) WHCA* demo with window size = 8 and number of robots = 4
5) WHCA* demo with window size = 8 and number of robots = 6

WHCA* Results:

We evaluated the performance of WHCA* for window sizes of 2, 4, 8, 16 respectively for total 4 and 6 robots for warehouse management. The key metrics for evaluation were total path planning time and success rate to ensure efficiency and repeatability of selected optimal window sizes. We observed following results.

A. *Window Size vs Path Planning Time*

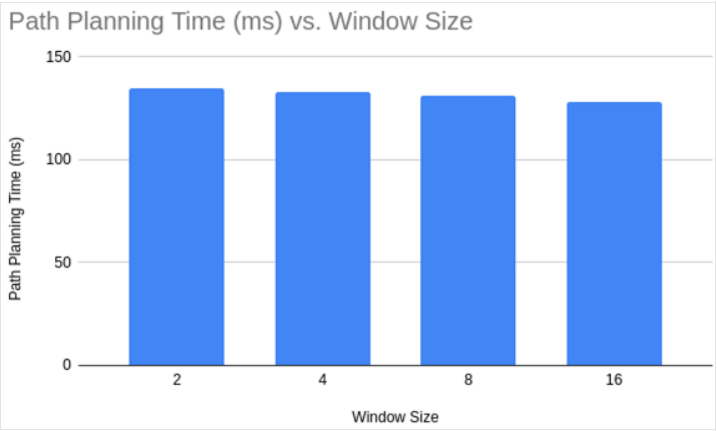

Fig. 9: For 4 Robots

The path planning time with 4 number of robots:
Window size 2 - 135 ms
Window size 4 - 133 ms
Window size 8 - 130 ms
Window size 16 - 129 ms

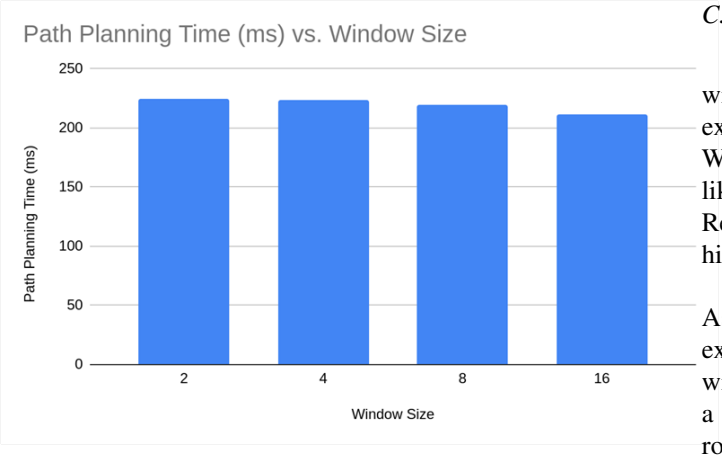

Fig. 10: For 6 Robots

The path planning time with 6 number of robots:
Window size 2 - 224 ms
Window size 4 - 220 ms
Window size 8 - 213 ms
Window size 16 - 211 ms

B. Number of Robots vs Success Percentage

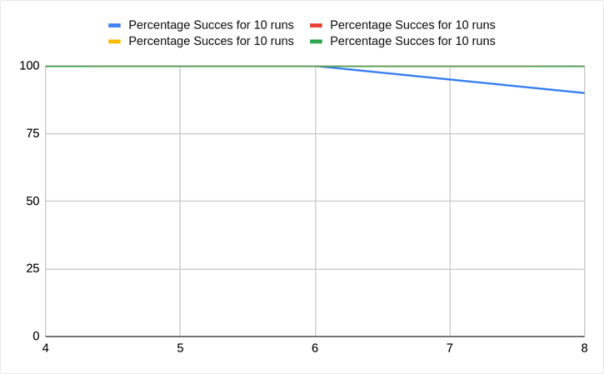

Fig. 11: For 4 Robots

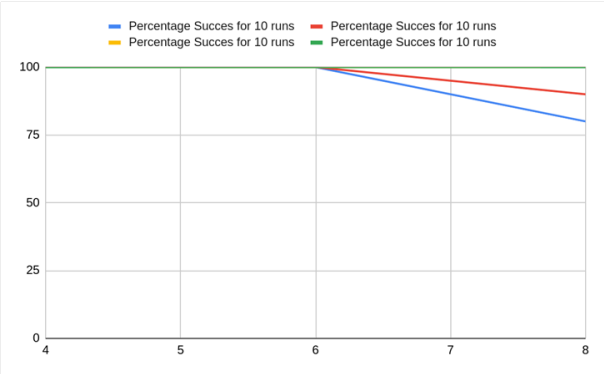

Fig. 12: For 6 Robots

C. Observations

The efficiency of WHCA* algorithm depends on the window size chosen. A smaller window size requires more execution time as it performs rerouting quite frequently. When window size is very small, the WHCA* algorithm acts like individual A* at all time steps (also known as Local Repair A*). A larger window size reroutes less often but with higher cost.

As seen in results above, for 4 robots (Fig. 9), the total execution time for window size = 2 is 135 ms. While for window size = 16 the execution time = 128 ms. This gives a 7 ms difference in path planning. When the number of robots was increased to 6 (Fig. 10), a window size of 2 takes around 224 ms while a window size of 16 takes 211 ms. This gives an execution delay of 13ms which is quite significant for real time material handling. Window size of 8 also gives a similar performance to that of window size 16. Thus, for our application, from the standpoint of total path planning time, we found 8 and 16 to be optimal window sizes. Bigger window sizes (> 32) were not considered for evaluation as they were not necessary for the small environment size.

For validating the repeatability of the WHCA* algorithm, we used a criteria of successful path planning vs failure in path planning. If WHCA* gives correct global+local path (with obstacle and robot avoidance) and does not get stuck in a local loop, we consider it as a successful simulation and if there is no path or if there are loop or robot collisions, it will be considered as a failure. We ran the WHCA* 10 times for each window size mentioned in the results section. It was observed that the WHCA* algorithm gave a success rate of 100% for window sizes 4, 8, 16 for 4 robots and for window sizes 8, 16 for 6 number of robots. The 2 times we got a failure with window size 2 and one with window size 4 was due to robots getting stuck in a local loop because of high occupancy of obstacles and other robots in same local area.
Overall the WHCA* algorithm performed very successfully for the warehouse environment. from these results and observations, we found that the window size of 8 and 16 were optimal for our application with 4 and 6 number of robots to handle the material between shelves and dock.

V. Conclusion

Individual A* (Also known as Local Repair A star) is suitable for simple environments with few bottlenecks and agents, but inadequate for more challenging environments. Cooperative A* algorithms, gave a better performance for global path planning but suffered when corner cases were present in path planning due to relatively worse heuristic. Variants such as WHCA*, have been developed with abstract distance based Hierarchical rerouting to improve the heuristic. Windowing is added to remove the inefficiency of the search. WHCA* finds successful routes with shorter paths and fewer cycles. The algorithms are applicable to more general pathfinding domains, and has potential application in differnet industries.

VI. Future Work

The cooperative algorithms studied (CA*, WHCA*) can also be used in dynamic environments. This approach will need the agent's route to be recomputed when the map changes due to dynamic obstacles. The current implementation was used with rail guided robot navigation. This can be extended in future to wheeled robots with kinematic constaints (non-holonomic constraints).

VII. References

[1] Abby Jenkins, 'What is warehouse robotics? A 2022 guide'
[2] Cooperative Pathfinding using A*, David Silver, University of Alberta, Canada
[3] Cooperative Pathfinding, David Silver, University of Alberta, Canada
[4] Pritam Ojha, Atul Thakur, 'Real-Time Obstacle Avoidance Algorithm for Dynamic Environment on Probabilistic Road Map'
[5] Inside Amazon's smart warehouse
[6] Motion Planning for a Chain of Mobile Robots Using A* and Potential Field
[7] Steven M. LaValle - Planning Algorithms-Cambridge University Press (2006)
[8] Gil Torass, 'Multi Stop Route Planning, Universitat Salzburg, Summer 2020'
[9] A Review of Motion Planning Techniques for Automated Vehicles