# Cooperative Path Planning for Automated Warehouse Management

Parthsarthi Rawat, Sayan Das and Kishor Sabarish G.

*Abstract*— **Autonomy for industrial tasks has been an active research topic in recent years. In this project, we focus on the motion planning algorithms used by autonomous pods to transport goods from one point to another in large warehouses. We implement cooperative path planning algorithms like Local Repair A*, Hierarchical Cooperative A* and Windowed Hierarchical Cooperative A* and test their performance in 2 dimensional grids. We then implement, benchmark and analyze WHCA* and Collision Based Search (CBS) on a 2 and 3 dimensional warehouse setting.**

***Index Terms*—Motion Planning, WHCA*, CBS, Automated Warehouse, 3D environment**

## I. Introduction

Most warehouse tasks are somewhat repetitive in nature. Therefore, it makes sense for companies to invest in automation in order to augment their profitability and make their business models crisis resilient. Amazon, for instance, has invested heavily in warehouse automation and has hundreds of intelligent robots working in their massive warehouses. This enabled them to maintain their service level even during the Covid crisis. In our project, we are specifically looking into the motion planning algorithms of autonomous pods that transport stacks of items from the storage area to the human operated processing booths and vice versa. With regards to this, we will implement a multi robot cooperative global planner to generate efficient routes between current position and final destination.

## II. Background

Motion planning is a crucial component of any robotic system. A*, D* and Dijkstra's algorithm are some of the popular motion planning algorithms used in many robotic systems [1]. A* is an informed version of Dijkstra's algorithm. A* considers a "best match first" because it greedily chooses which vertex to explore next. If the heuristic function $h$ in A* is replaced with non-informative heuristic $h = 0$, A* will choose which vertex to develop next using only "so far cost" i.e., function $g$. This is similar to Dijkstra's. A lot of motion planning algorithms are available which are versions of some basic algorithms modified for specific tasks.

For multi agent path planning, cooperative algorithms like Cooperative A*(CA*) searches space for non-colliding routes, Hierarchical Cooperative A*(HCA*) uses an abstract heuristic to boost performance and Windowed Hierarchical Cooperative A*(WHCA*) limits the space-time search to dynamic window, spreading computation over the duration of the route [2]. In [3], a robot traffic density model is developed to predict spatial robot density in a future horizon. This information is then utilized by the motion planner to determine congestion free routes for the robots. In [5], the authors develop Conflict Based Search (CBS) with the objective of combining the advantages of coupled and decoupled planning algorithms.

The pods we will use in our simulations will be similar to Kiva pods which have been used extensively by Amazon. Kiva pods scan QR codes on the ground below them to map their position inside the warehouse and have additional sensors for detecting obstacles in close proximity to them.

## III. Algorithms

We discuss various multi robot path planning algorithms which we have considered for our application in this section.

### *3.1 Local Repair A-star*

Local Repair A* (LRA*) describes a set of algorithms in which agents plan their paths individually using A* until a collision is imminent. Essentially, each agent calculates a route to its target ignoring all other agents. The agents keep following their predetermined path until a collision with another agent is imminent on the next step. The agents whose path are obstructed then attempt to re-plan a new path at that instant considering the new obstacles near them. The algorithm is essentially non cooperative in

nature since an agent's planning procedure does not consider the paths determined by other agents. Hence, cycles and bottlenecks are common in this algorithm. In cluttered environments, the A* algorithm may need to be invoked multiple times which greatly increases the computational complexity of this algorithm.

### *3.2 Cooperative A-star*

In cooperative A*, the multi robot path planning task is decoupled into a sequence of single agent searches. The path of the first agent is calculated while ignoring all other agents. Its path is then stored in a 3 dimensional reservation table whose two dimensions are position coordinates and time is the third dimension. The path of the second agent is determined while avoiding the first agent's path in the reservation table and so on. Additionally, a wait move is also incorporated along with the other moves which transport the robot to adjacent nodes. Once a set of paths is found, the algorithm does not need to rerun during the actual path following. It is, however, important to note that any decoupled, greedy algorithm that pre calculates the optimal path will not be able to solve certain problems [2]. Additionally, the algorithm's performance is affected by the ordering of the agents' path planning steps.

### *3.3 Hierarchical Cooperative A-star*

HCA* replaces the Manhattan Distance heuristic of CA* with an abstract heuristic, which ignores both the time dimension and reservation table. removed. Abstract distances can therefore be visualized as perfect estimates of the distance to the respective destinations, ignoring any potential interactions with other agents. HCA* uses Reverse Resumable A* (RRA*) to calculate the abstract distance. It is a way to calculate abstract distances from the current robot node to its destination while reusing previous search data. At each node, the minimum distance from the node to the goal is returned by RRA* but all expanded nodes from previous iterations is saved in a list to prevent any excess computations. RRA* can be used to compute distances on demand and is more computationally efficient than CA*. However, it is also sensitive to the ordering of the agents.

### *3.4 Windowed Hierarchical Cooperative A-star*

WHCA* is similar to HCA* but it executes its search in small windows of some step length instead of computing the full path at the beginning. This gives it additional flexibility in determining multi agent paths. In WHCA*, the planning does not end when any single agent reaches its goal. It can still move out of its target position to give way for other robots as long as it is able to reach back at its target at the end of the window. The ordering of agents is also altered after each window to decrease the sensitivity of the algorithm to the ordering of agents. Such a style of execution makes the planning more resilient to new developments in the grid map as well. However, only the cooperative search is limited to a fixed depth specified by the window size whilst the abstract search is executed to its full depth. Due to these characteristics, WHCA* has the highest case solving percentage relative to the previous algorithms.

### *3.5 Conflict Based Search*

CBS is a two level planning algorithm that attempts to combine the computational efficiency of decoupled algorithms and the robustness of coupled algorithms. At the high level, a search is performed on a Conflict Tree which is created based upon conflicts between individual agents. Each node of the tree represents a complete solution. If there is any collision between the agents in that solution, it is labelled as invalid, and two child nodes are created. Each child node is given an additional constraint whereby one of the colliding agents has to modify its path due to the previous conflict point being treated as a constraint. Finally, the node which represents a valid solution and has the minimum cost is returned as a solution. The solution in each node is obtained by finding paths for each agent with associated constraints using local planning algorithms like A*. The reason for inclusion of CBS in our list of algorithms is that its performance increases in constrained spaces like warehouses due to the decrease in number of nodes to be expanded [5]. It is a late addition to our list, and we are skipping over its implementation in the 2D maze in the next section. We directly test it in the warehouse environment.

## IV. IMPLEMENTATION

We adapted our algorithm from Ireneusz Szulc's WHCA* implementation. We utilize LRA*, HCA* and WHCA* to solve a similar maze to analyze the strengths and weaknesses of the algorithms. We don't separately analyze CA* because it will give similar results as HCA* at additional computational expense.

### *4.1 Local Repair A-star*

We initiate a grid world as shown in Fig. 1.0(a). The circles represent the agents, shown at their starting points. Each agent has been assigned a target represented by a cross sign of the corresponding color. LRA* is able to function properly in this setting and reaches the target as shown in Fig. 1.0(b) Next, we complicate the problem by

adding additional agents for a total of seven distinct agents as shown in Fig.1.0(c). In this setting, LRA* fails to generate proper paths for all the agents to reach their goals due to the absence of any cooperative aspect to its algorithm. The algorithm is not able to proceed beyond the setting shown in Fig.1.0(d) where agents 3 and 6 are at their targets but are preventing the rest of the agents from reaching their goals.

*4.2 Hierarchical Cooperative A-star*

We use HCA* to solve the same multi agent path planning problem as shown in Fig.1.0(c). HCA* has a much higher chance of success in this setting due to its cooperative nature induced by the use of a reservation table. The entire paths are generated at the start of the run as shown in Fig. 2.0(a). Fig. 2.0(b) shows some of the agents nearing their targets while the other agents are maneuvering around each other. Note that the agents are assured of collision avoidance by the use of the reservation table which ensures that any position in the grid is occupied by a maximum of only one agent at any time. However, this success is dependent on the ordering of agents. We have noted that HCA* fails in this setting for various other agent orders, especially due to the fact that since HCA* does not follow a windowed approach, an agent does not change its position once it reaches its target.

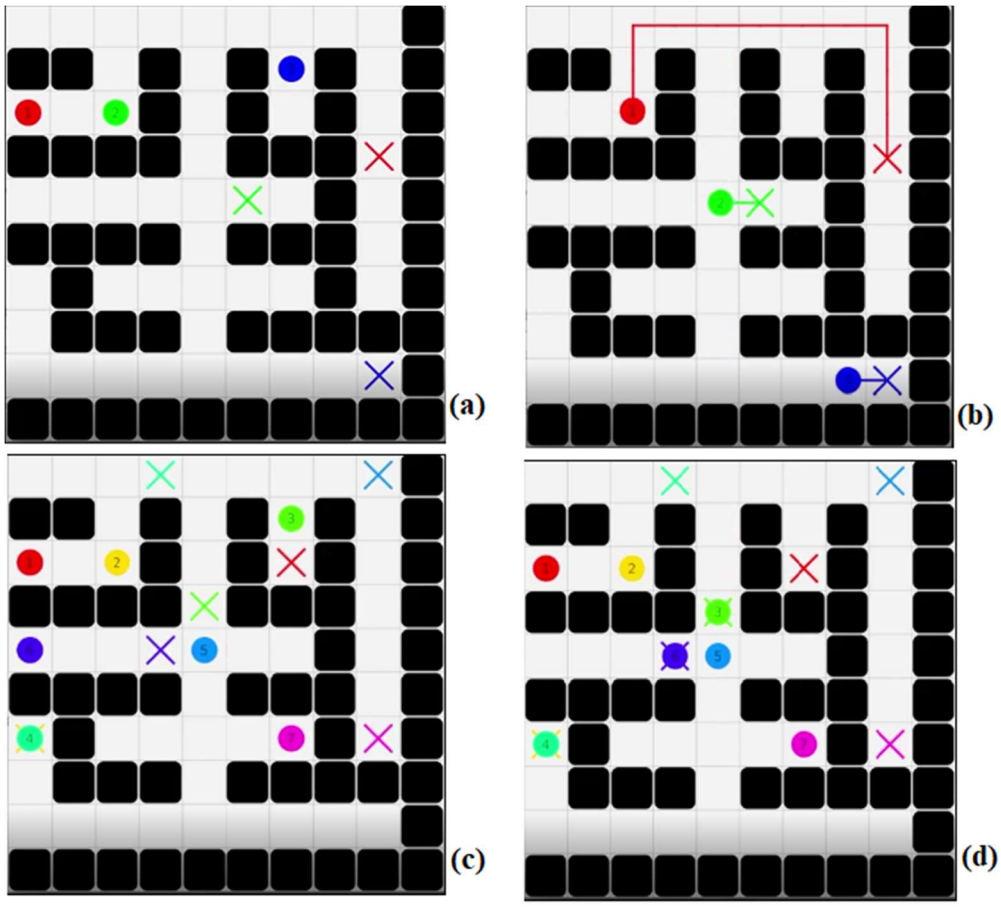


**Fig 1.0 (a)**LRA* initialized with 3 bots (less complexity). **(b)** LRA* successfully reaches the target positions. **(c)** The same environment regenerated with 7 bots (high complexity). **(d)** LRA* fails to reach the desired destination.

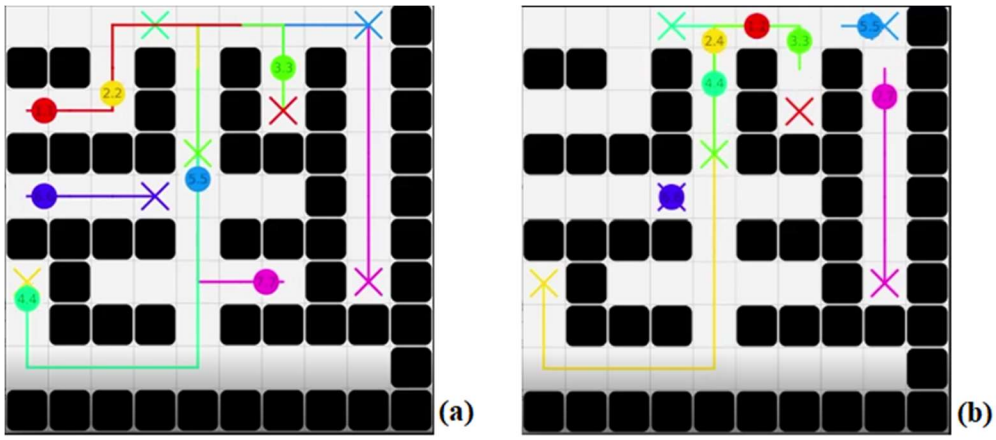


**Fig 2.0 (a)**HCA* initialized with 7 bots. **(b)** HCA* successfully reaches the target positions.

*4.3 Windowed Hierarchical Cooperative A-star*

Next, we analyze the strength and weaknesses of the WHCA* algorithm. On receiving the same problem as in Fig. 1.0(c), WHCA* forms partial paths as seen in Fig. 3.0(a). WHCA* is much more robust to agent ordering as the agent order changes after each window. Additionally, after an agent reaches its target, it is free to move to allow higher priority agents to pass through as long as reaches its target at the end of the window. The window size can also be gradually increased to allow more complex paths for agents if it is observed that the agents are getting stuck in some place. However, WHCA* can also partially fail in certain settings as seen in Fig. 3.0(b) where most of the agents other than the purple agent have reached their target. WHCA* cannot resolve this situation further as even when the purple agent has higher priority, the blue agent cannot move in a way that accommodates the path of the purple agent to its goal. If the obstacle at the northeast corner is removed, the blue agent will be able to move back to accommodate the higher priority purple agent to reach its target, as shown in Fig. 3.0(c). Creating random disturbance or a much higher planning window at

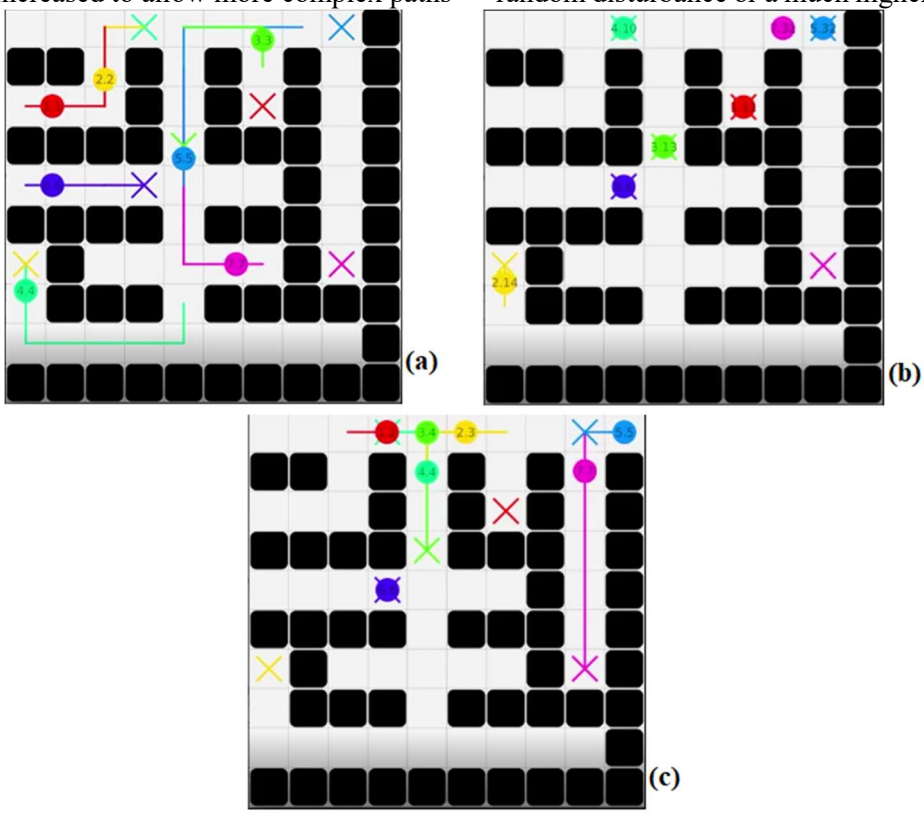


**Fig 3.0 (a)**WHCA* initialized with 7 bots. **(b)** WHCA* fails due to too much congestion at the north-east corner where the blue (5) agent has no space to traverse to make way for purple (7) agent. **(c)** WHCA* successfully reaches the desired positions. The congestion on the north-east corner was removed by eliminating one block from the first row.

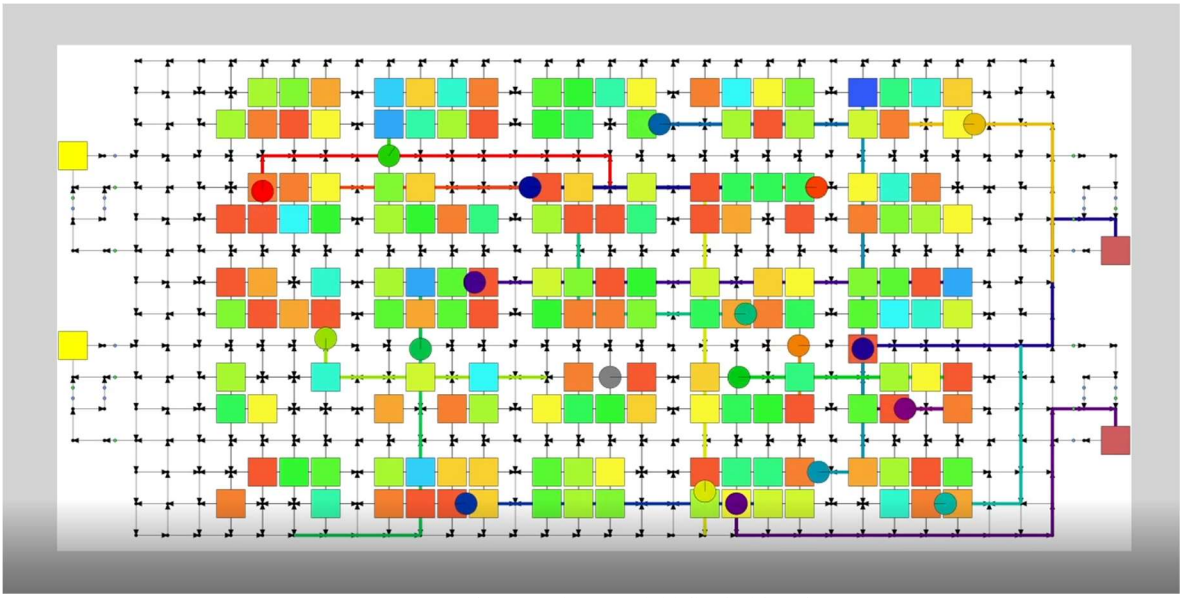

**Fig 4.0** 2D warehouse environment initialized with 12 bots.

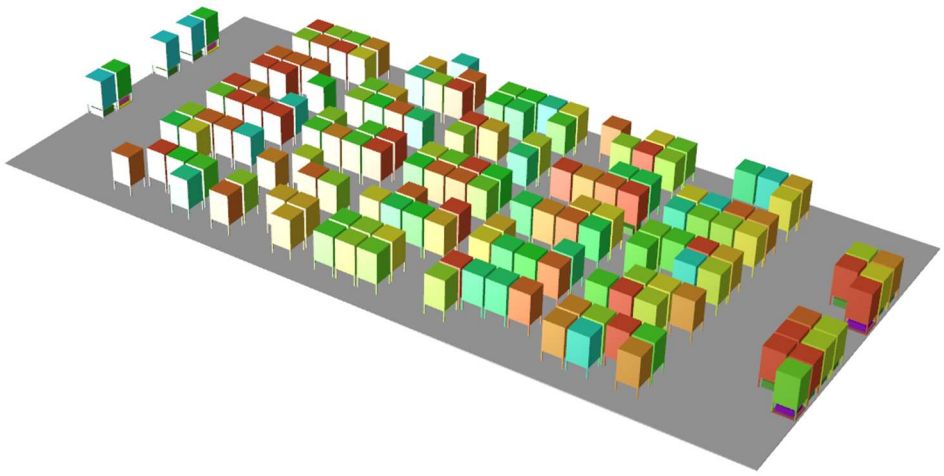

**Fig 5.0** 3D warehouse environment initialized with 12 bots.

the start can enable WHCA* to overcome such obstacles. This can be an interesting research topic and we will try to explore this during this project. We used WHCA* with a very large window to successfully overcome the previous maze problem in which the path followed was similar to HCA*.

*4.4 Implementation in a Warehouse Setting*

We execute the planning algorithms on Marius Merschformann's automated warehouse environment. Fig.4.0 and Fig.5.0 show the 2D and 3D representation of our test environment. The different colors of the inventory show the amount that specific unit is filled, red being almost full and blue being almost empty. We initialize the simulator by keeping the number of horizontal and vertical aisles equal to five. Furthermore, we define the number of inventory refilling and order drop-off stations. In the simulation our warehouse robots pick up an inventory stack and take them to the desired station and vice versa. The East/Red tiles on the end of the hallway in Fig.4.0 is the drop-off station whereas the West/Yellow

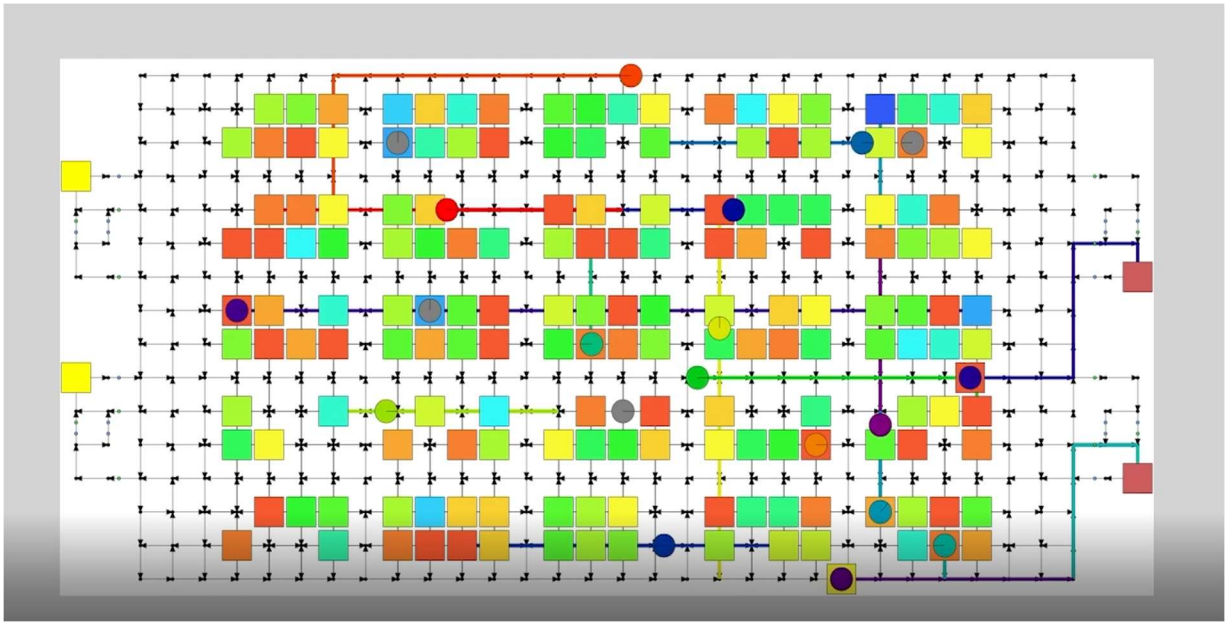

**Fig 6.0** Partial Planned Paths by WHCA* (can be seen in bottom right)

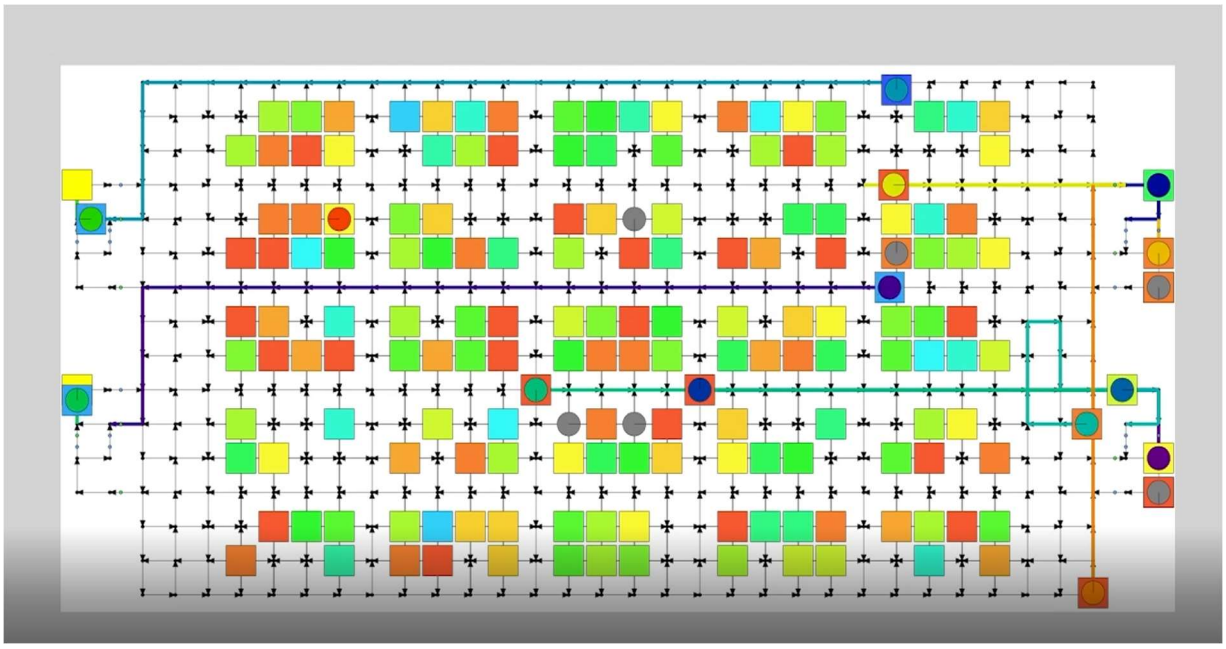

**Fig 7.0** Non-Optimal Planned Paths by CBS (can be seen in bottom right)

tile station on the other end is the Input Station used for filling up the inventory. Finally, inventory is allowed to fill to only 85% of its full capacity. The warehouse bots rest if there are no stacks to be picked up from the input stations and there is no additional order to be fulfilled.

Based on our previous simulations, we select WHCA* for implementation in our 3D environment as we feel that it most practical for implementation in a warehouse setting among the cooperative A* algorithms due to its robustness and relatively high computational efficiency. We also implement CBS because we feel that it should be able to perform well in the constrained and structured environment that exists in large, automated warehouses. An implementation of WHCA* can be seen in Fig.6.0. The partial paths created by each robot spanning the length of its planning window can be clearly seen. Fig.7.0 shows the implementation of CBS on our environment. Here, some non-optimal paths can be seen on the right end of the plot which are the result of additional constraints imposed on an agent in the CBS path planning algorithm.

## V. Results

Both the algorithms were simulated on the environment for a simulation period of 24 hours for 12 robots. Fig.8.0

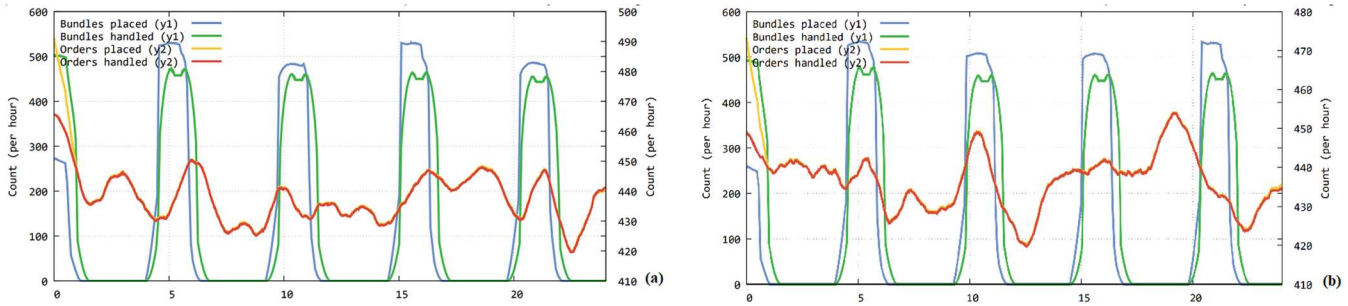


**Fig 8.0** Amounts of Orders Handled for 24 hours by **(a) CBS (b) WHCA***

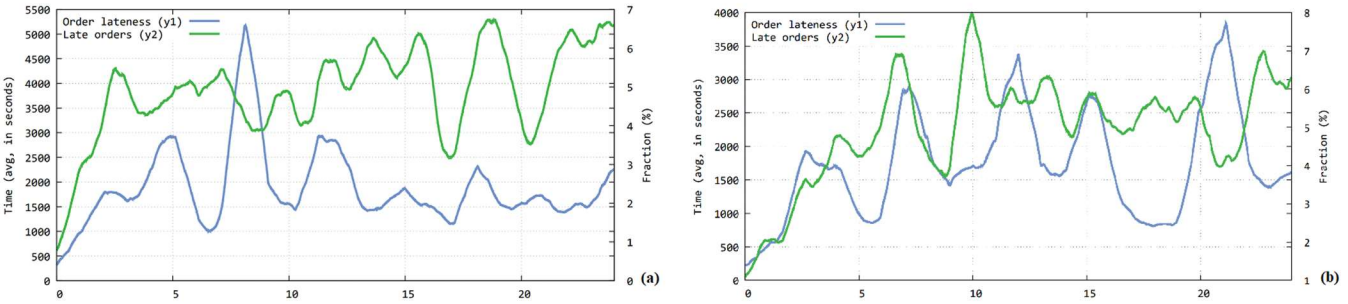


**Fig 9.0** Percent of Order Late and Lateness over a period of 24 hours by **(a) CBS (b) WHCA***

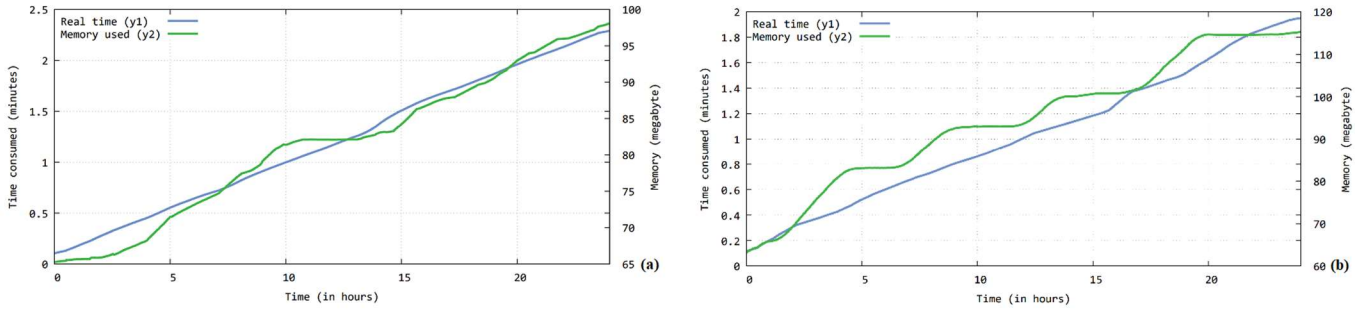


**Fig 10.0** Memory consumed by **(a) CBS (b) WHCA***

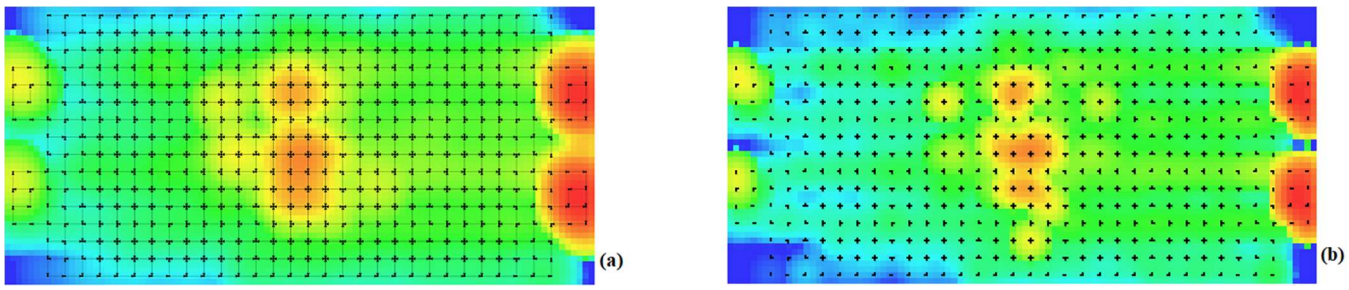


**Fig 11.0** Low Resolution Heat Maps for **(a) CBS (b) WHCA***

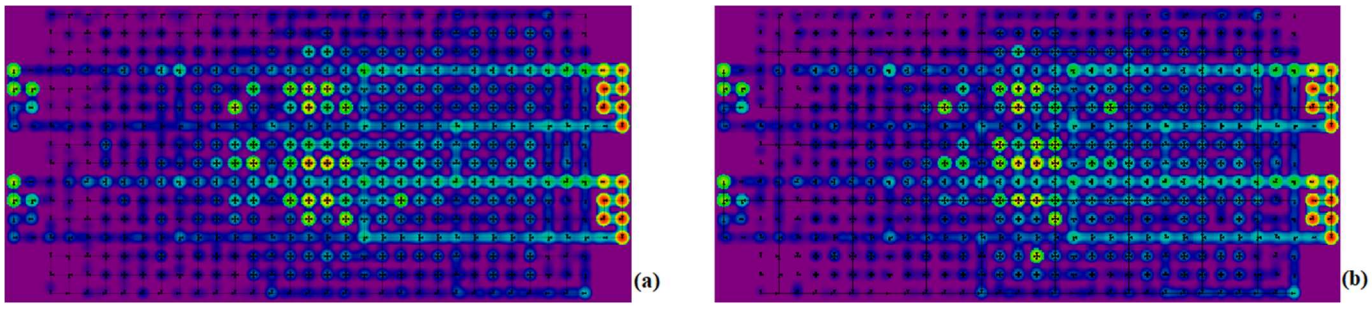


**Fig 12.0** High Resolution Heat Maps for **(a) CBS (b) WHCA***

shows the amount of input and output orders handled in unit time. CBS is able to marginally outperform WHCA* by handling slightly larger number of orders. Fig.9.0 plots the percentage of orders that are delivered late by each planner and the amount by which late orders are delayed. WHCA* has a lower percentage of late orders than CBS. However, the amount of delay in late orders is lower in CBS. Changing the planning window size in WHCA* may affect these results. Fig.10.0 shows the computational cost of the two algorithms. CBS can be seen to be approximately 20% lower than WHCA* both in terms of time and space complexity.

Fig.11.0 and Fig.12.0 show the low and high resolution heat maps for the robot location for the entire simulation. As we can see the heat map of CBS is more spread over the warehouse than WHCA*. This will be easier to see this in the low resolution plot Fig.11.0. WHCA* calculates optimal path based on minimizing cost but this leads to the cluttering of some routes. CBS, instead, follows slightly more convoluted paths which leads to less crowding and marginally more orders handled on average.

## VI. Discussions

A host of multi agent cooperative path planning algorithms were studied and analyzed as part of this project. LRA*, CA*, HCA* and WHCA* were considered during the first part of the project. WHCA* was selected as the best algorithm based on simulations which proved its robustness despite its advantages in computational efficiency. We also decided to consider CBS at a later stage due to its performance in constrained spaces with a lot of bottlenecks. Both the algorithms were successfully executed in a 3D warehouse simulation environment. CBS outperformed WHCA* but only by a slight margin. WHCA* can prove to be better than CBS if the warehouse is more unstructured and cannot be modelled as an environment with tight spaces. Both the algorithms have the potential to be successfully implemented in a real automated warehouse.

## References


[1] Zhang, Han-ye, Wei-ming Lin, and Ai-xia Chen. "Path planning for the mobile robot: A review." *Symmetry* 10, no. 10 (2018): 450.

[2] Silver, David. "Cooperative Pathfinding." *Aiide* 1 (2005): 117-122.

[3] Liu, Zhe, Hesheng Wang, Huanshu Wei, Ming Liu, and Yun-Hui Liu. "Prediction, planning, and coordination of thousand-warehousing-robot networks with motion and communication uncertainties." *IEEE Transactions on Automation Science and Engineering* (2020).

[4] Rawat, Ankit Singh, Arya Mazumdar, and Sriram Vishwanath. "Cooperative local repair in distributed storage." *EURASIP Journal on Advances in Signal Processing* 2015, no. 1 (2015): 1-17.

[5] Sharon, Guni, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. "Conflict-based search for optimal multi-agent pathfinding." *Artificial Intelligence* 219 (2015): 40-66.


## Appendix

**Sayan Das**:
I initially worked on the Literature Review for the project and shortlisted some multi agent cooperative path planning algorithms for use in our project. I then spent time understanding the cooperative algorithms presented in this report from the original paper as well as various other sources. I took part in adapting the various cooperative algorithms for our use in the 2D environment for project. Then, I analyzed the performance of the different algorithms in various mazes to test their strength and weaknesses and estimate their suitability for our application. For the 3D section, I again took part in the selection, comparison and analysis of the algorithms implemented. Finally, I took part in making this report and the presentation for our project.

**Parthsarthi Rawat**:
I started off by reviewing literature related to multi agent path planning. Furthermore, I understood the cooperative algorithms we picked for our project and implemented them in the 2D world demonstrated in the project report and presentation. I also performed performance benchmarking, finding and resolving limitations of the algorithms used by us in this project. For the 3D part, I worked again on the implementation, benchmarking and analytic study of both the algorithms. I also took part in making the report, documentation and presentation of our project.

**Kishor Sabarish G.**:
I looked in AWS Robomaker's Warehouse package. This warehouse package is an opensource package which allows us to spawn Environment which is very similar to actual warehouses. Various objects like shelves, desks, chairs, lamps, boxes, buckets as pre-built callable functions. I spawned a warehouse environment with shelves arranged parallely into many rows which acts like the workspace for the mobile robots. The interfacing with ROS and Gazebo by this package helped us to go with AWS Robomaker.

Similar to, AWS Robomaker, Unity has its own assets to simulate various worlds. One such asset is the Warehouse Environment. The Warehouse asset environment is generally used for Game Development, basically like car showroom. We finally implemented new 3D and 2D warehouse simulations on a C# simulator.