Swarm Robots for Warehouse Applications

ED5215 : Motion Planning Course Project Report

Potluri Bhargava Siva Naga Sai - ME19B144
Winston Doss - EE23E008
C R Aswin Raj - ME19B091

Date: 19 May, 2024

Abstract

The motivation behind this project is to optimize the operation of multiple robots working together to move packages in a warehouse, aiming to minimize the overall task completion time. Specifically, our problem involves three robots operating in a static environment with three packages and their corresponding delivery points. Task allocation has been predetermined, and the map of the environment is known. The objective is to plan the robots' paths to pick up and deliver the packages in the shortest possible time while avoiding collisions with static obstacles and other robots. In this project, we implemented and compared three motion planning algorithms: Serial A*, Conflict-Based Search A*, and Serial RRT. The planned paths were executed in CoppeliaSim, and the three algorithms were evaluated based on planning time, number of steps to complete the task, and execution time.

Figure 1: Multiple Robots working in a Warehouse together

Methods

Serial A* Algorithm

We are working on a centralised multi-robot system in an environment with static obstacles and also the each robot is obstacle to other robot if they collide at the same time. The Implementation of the Serial A* algorithm is shown below.

Implementation of Serial A* Algorithm

The Flow Chart showing the implementation/ pseudo code of the serial A* algorithm on multi-robot system is attached below.

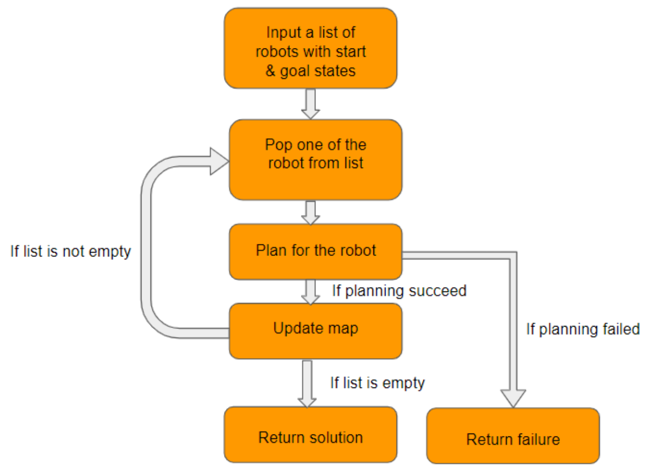


Figure 2: Flow Chart showing Serial A* Algorithm Implementation on Multi-Robot system

The above Flow Chart shows the systematic steps (Pseudo code) for the implementation of Serial A* algorithm. Initially, we assign each robot to a particular package and then to a particular final delivery point. Add all of them to a list. So, now the list contains all the robots assigned to their respective package locations and delivery points. All the multiple robots present in the system are added to the list.

One of the Robots is popped from the list randomly. There is no particular order/strategy for popping the robot from the list. The path is planned for the popped robot from the start location to package location and then to the delivery point using the A* planning algorithm. If the planning is failed and there is no feasible path obtained, failure is returned. If planning is successful, then the map is updated with the path of the first robot which is planned.

In this algorithm, we are keeping the track of time as a path index. For the final path obtained, we are assigning index to each grid in the path and storing it with the map as an updated map. Now, if the list is empty then the algorithm is complete and we have planned the paths for all the robots in the system. If the list is not empty, then the loop is continued and the algorithm returns to step 2 with a updated map. So, we once again pop the robot randomly from the remaining robots in the list. We plan the path for the new robot considering the path for previous robots as a static obstacle only at that particular path index (time).

For example, while planning the path for the second robot, the path planned for the first robot will be considered and each grid in the path will be considered as a static obstacle only at that particular time (index). Similarly, while planning for the third robot, the paths for the first two robots are considered along with the map. This process goes until the path is planned for all the robots in the list. If the list is empty, then the Serial A* algorithm is complete and the path is planned for all the robots in the system. This is the short description about the implementation of Serial A* algorithm. More detailed discussion about results and disadvantages of this algorithm are explained in the later part of the report.

Conflict Based Search (CBS) using A* Algorithm

Static environment

As discussed previously, the total time required for robots to reach their goals using the serial A* algorithm is contingent upon the sequence in which the paths for different robots are planned. This dependency can result in significantly increased travel times, especially when the number of robots is large. To address this issue, we propose a solution based on the Conflict-Based Search (CBS) algorithm in conjunction with the A* algorithm.

Initially, we plan the paths for all robots individually, from their start states to their goal states via pickup points, using the A* algorithm. This planning phase is conducted without considering potential collisions between robots. The complete solution, which includes the list of paths for all robots (e.g., [{Path of Robot 1}, {Path of Robot 2}, ..., {Path of Robot n}]) and the total time taken, is added to a priority queue. In this queue, the element with the minimum 'total time' is assigned the highest priority.

The algorithm then enters a loop that continues until the priority queue is empty. In each iteration of the loop, the highest-priority solution (i.e., the one with the minimum total time) is removed from the queue. The first conflict within this solution is then detected. The two types of conflicts considered are depicted in Figure 3. Upon detecting a conflict, two alternative solutions are generated by re planning the paths of the conflicting robots individually, aiming to avoid the conflict. Both of these new solutions are subsequently inserted into the priority queue.

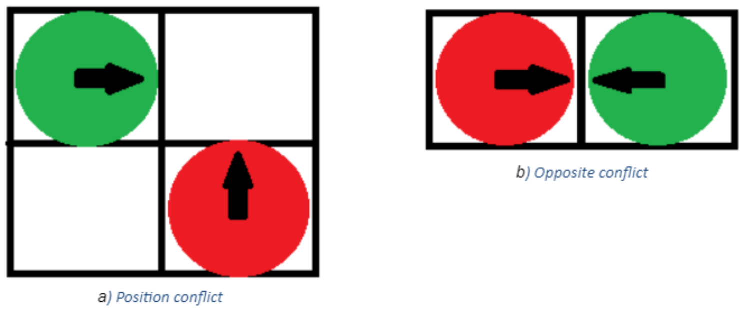


Figure 3: Conflicts in CBS Algorithm

This process repeats until a solution without conflicts is extracted from the queue or the priority queue is depleted. If a conflict-free solution is found, it is returned as the result. Otherwise, the algorithm returns 'Failure'. The flowchart illustrating this algorithm is shown in Figure 4 (shown below). By employing the CBS algorithm alongside A*, we aim to optimize the path planning process in multi-robot systems, thereby minimizing total travel time and effectively resolving potential conflicts.

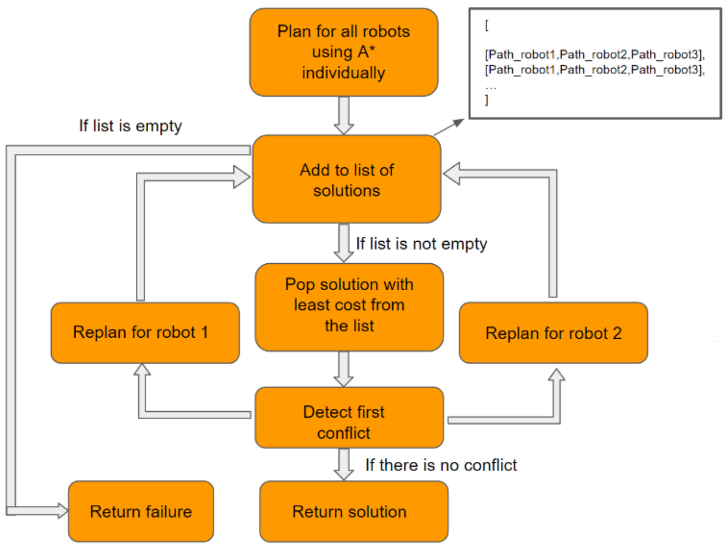


Figure 4: Flow Chart showing CBS A* Algorithm Implementation on Multi-Robot system

Dynamic environment

The algorithm is designed to adapt to real-time environmental changes. Whenever there is a modification in the grid, the algorithm assesses whether any of the planned paths are disrupted by the new obstacles. If a disruption is detected, it replans the affected robots' paths using the A* algorithm, disregarding potential collisions with other robots. The newly generated list is added to an empty priority queue, and then the CBS algorithm loop is executed to produce a conflict-free solution.

Serial RRT Algorithm

The Serial RRT algorithm is the RRT counterpart to Serial A*. Similar to Serial A*, we plan the path for each robot sequentially, treating the paths of previously planned robots as obstacles for subsequent ones.

Algorithm Overview

The flowchart below illustrates the Serial RRT algorithm:

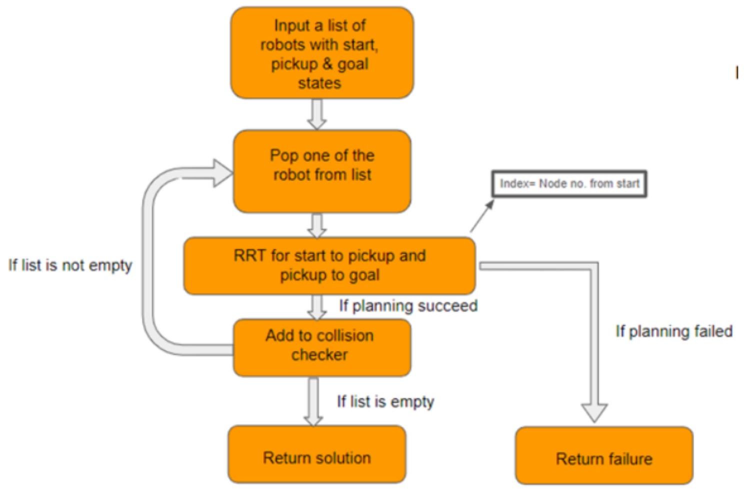

Input a list of robots with start, pickup & goal states
Pop one of the robot from list
Index= Node no. from start
If list is not empty
RRT for start to pickup and pickup to goal
If planning succeed
If planning failed
Add to collision checker
If list is empty
Return solution
Return failure

Figure 5: Flow Chart showing Serial RRT Algorithm Implementation on Multi-Robot system

Implementation

Our implementation takes a list of start, pickup, and goal positions as input. The path planning process for each robot proceeds sequentially:

1. For each robot, plan the path from the start to the pickup point and then from the pickup point to the goal using the RRT algorithm.

2. Combine these two paths and pass them to a collision checker, which will be used for the path planning of subsequent robots.

In the RRT, each node also stores its distance from the start, this will play the same role as index as in Serial A*.

Collision Checking

For subsequent robots, the collision checker ensures that:

- A sample, which is the nth node from the start, must not be within a threshold distance of the line connecting the (n-1)th and nth nodes of the previous robot's path.

Consider the image below:

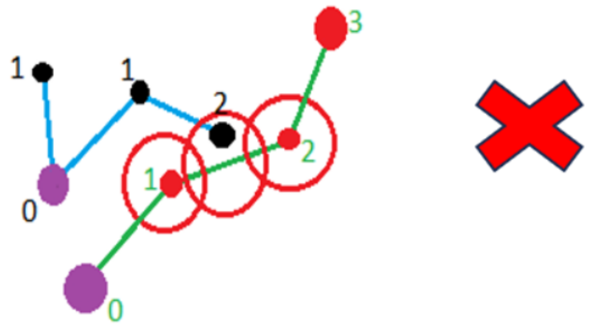


Figure 6: Collision Check in Serial RRT

In the image, the green path represents the path of the first robot, and the blue lines represent the RRT samples of the second robot. The second sample is invalid as it lies within the threshold distance of the line connecting the second and first nodes of the first robot's path. This condition ensures there is no collision between the robots.

Results

Serial A* Algorithm

The Serial A* algorithm was implemented on various multi robot systems (1 robot, 2 robot and 3 robot systems). The Robots go their assigned package locations and then to their assigned final delivery points.

Video Showing the implementation of Serial A* Algorithm

The video showing the implementation of Serial A* algorithm on multi robot system is attached below:
Serial A* Algorithm on Multiple Robots (Click Here)
The above shows the algorithm implementation on 1 robot, 2 robot and 3 robot systems. The Green grids represent the start positions of the robot. The purple grids represent the package locations and the red grids represents the final delivery locations in the video. The screenshot of the Serial A* algorithm implementation video is attached below:

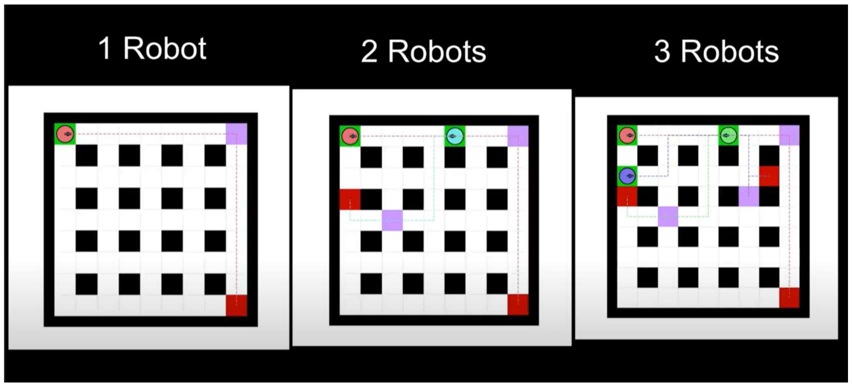


Figure 7: Serial A* Algorithm Implementation (Screenshot of Video)

Video Showing the implementation of Picking and Delivering the Package in Coppeliasim

The video showing the implementation of Picking and Delivering the packages by the robots in coppeliasim is attached below:
Picking and Delivering Packages in Coppeliasim (Click Here)

Conflict Based Search (CBS) A* Algorithm

The CBS A* algorithm was implemented on multi robot system with both static and dynamic obstacles.

Video showing different types of Collision in CBS A*

The videos showing two different types of collision (head on, side on) in CBS A* algorithm are attached below:
Head-on Collision Video (Click Here)
Side-on Collision Video (Click Here)

Video showing CBS A* Implementation during 4-robot Collision Case (Deadlock Situation)

The video that is attached below shows the implementation of CBS A* algorithm during deadlock situation (when the 4 robots are colliding with each other).
4 Robot Collision Video (Click Here)

Video comparing Serial A* and CBS A* Implementation

The video that is attached below compares the implementation of Serial A* and CBS A* algorithms.
Serial A* vs CBS A* Video (Click Here)

Video showing the implementation of CBS A* algorithm in CoppeliaSim

The video that is attached below shows the implementation of CBS A* algorithm on 3 robot system in coppeliasim.
CBS A* in Coppeliasim (Click Here)

Video showing the implementation of CBS A* algorithm in Dynamic Environments

The video that is attached below shows the implementation of CBS A* algorithm on 3 robot system when dynamic obstacles are present (the map changes with time).
CBS A* in Dynamic Environments (Click Here)

Serial RRT Algorithm

The steps involved in the implementation of Serial RRT algorithm on 3 robot system are shown in the picture below.

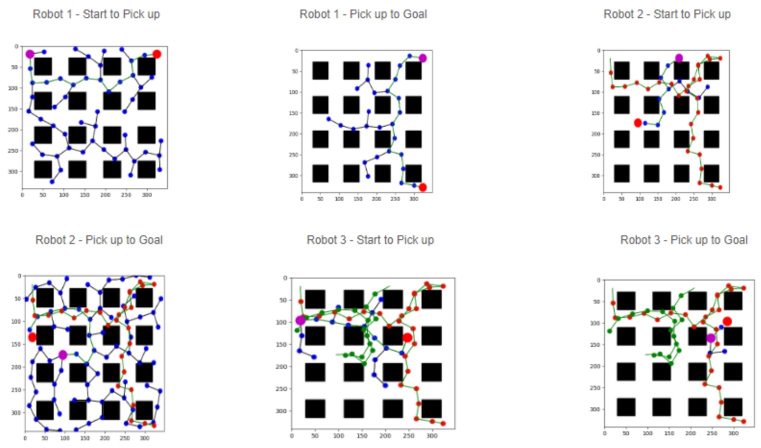


Figure 8: Steps showing implementation of Serial RRT Algorithm

The Final Result obtained after the implementation of Serial RRT algorithm on 3 robot system is attcahed as a picture below.

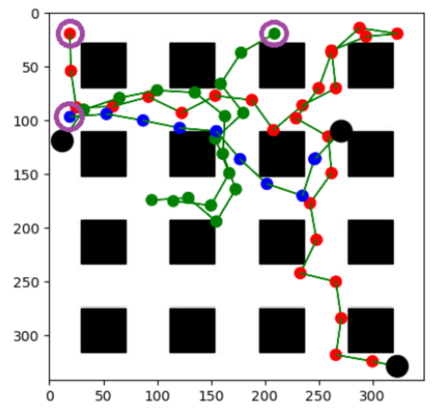
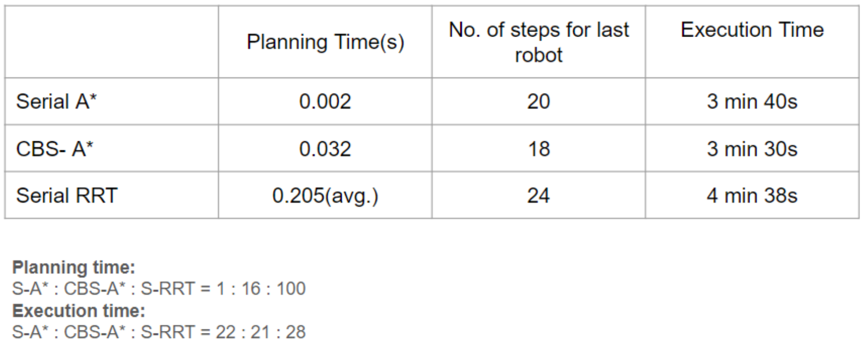


Figure 9: Result showing Serial RRT Implementation

In the above image, the Purple circled points are the starting points of each of the 3 robots. The Red, Blue and Green paths shows the planned paths for 3 different robots using Serial RRT algorithm. The 3 black circles represent the goal points of the 3 robots respectively.

Comparing the Results of the 3 Implemented Algorithms

We have implemented Serial A8, Conflict Based Search (CBS) A* and Serial RRT algorithms on multiple robot systems (3 robot system in this case). Now, we are comparing the final results obtained in the three cases. The Comparison table is attached below.



	Planning Time(s)	No. of steps for last robot	Execution Time
Serial A*	0.002	20	3 min 40s
CBS- A*	0.032	18	3 min 30s
Serial RRT	0.205(avg.)	24	4 min 38s

Planning time:
S-A* : CBS-A* : S-RRT = 1 : 16 : 100
Execution time:
S-A* : CBS-A* : S-RRT = 22 : 21 : 28

Figure 10: Comparison Table of the 3 Algorithms Implemented

As we can observe from the table, Serial A* algorithm has the least planning time among the three algorithms followed by CBS A* and then Serial RRT. But, CBS A* algorithm takes the least number of steps to complete the task. Therefore, it gives more optimal path when compared to the other two. Serial A* algorithm is almost 100 times faster than Serial RRT in this case. CBS A* Algorithm take the least amount of execution time followed by Serial A* and Serial RRT algorithms. These are some of the observations that can be made by the comparing the results of the 3 different path planning algorithms implemented by us on a multi robot system (3 robot system).

Discussion

Serial A* Algorithm

We have implemented the Serial A* algorithm on multi robot system as shown in the results. There are a couple of disadvantages if we are using the Serial A* algorithm for path planning in centralised multi-robot systems. The First disadvantage is that the order in which the robots are popped from the list (the order in which the path is planned for robots) is random. There is no particular strategy in popping the robots from the list. So, this ensures that the total time taken for robots to reach their goals is not always optimised as it depends on the order of planning which is random. The second disadvantage of this algorithm is that some robots may not obtain a feasible path due to restrictions created by pre-planned paths. For example, if the first robot is popped first from the list and the path is planned for it and this path prevents providing a feasible path for the second robot whereas if the second robot is popped first, then we may have feasible paths for both robots and a feasible final solution can be obtained. So, sometimes we may not be able to achieve a feasible solution using this algorithm even if it exists because of the order of planning. Because of these limitations, we are moving towards implementation of CBS A* and Serial RRT algorithms on multi robot system.

Conflict Based Search A* Algorithm

Since the paths of all robots are planned initially and conflicts are resolved to minimize the total time taken for all robots to reach their goals, there is no priority order for planning. This approach results in a solution with lower cost compared to the serial-A* algorithm and guarantees a feasible solution if one exists. However, due to multiple iterations of planning and re-planning, this algorithm requires significantly more time and computational effort than the A* algorithm.

Serial RRT Algorithm

The Serial RRT method can achieve higher precision compared to search-based algorithms. While search-based algorithms operate on an 18x18 grid with each grid cell representing 1m x 1m in CoppeliaSim, RRT can achieve finer precision by simply reducing the step size. In contrast, reducing the grid size in search-based algorithms significantly increases planning time. A significant drawback in our implementation is that we did not reuse the RRT tree for each planning cycle from start to goal. This oversight considerably increases the computational cost. Reusing the RRT tree could substantially improve planning time and potentially outperform Serial A*. However, due to time constraints, this optimization was not implemented. Similar to Serial A*, planning for robots sequentially means that the order of planning impacts the solution quality. Optimizing the planning order could yield better results. Additionally, the paths obtained using RRT can be very close to obstacles, necessitating obstacle inflation for safety. Therefore, some preprocessing of the map is required to ensure adequate clearance from obstacles.

Contributions

Bhargava Siva Naga Sai: Ideating and implementing code for Serial A* algorithm, Edited and recorded the final result videos.
Aswin Raj: Literature review for CBS, code implementation of CBS A* algorithm with static and dynamic obstacles.
Winston Doss: Ideating and implementing Serial RRT algorithm and integrating the path planner with the provided driver code to implement in CoppeliaSim.
The whole work was highly shared as we had occasional brainstorming and code together sessions.

Full Video Showing all the Results obtained by us

The Below attached link gives the final video (merging of all the videos in the results section) which shows all the results obtained by implementing the algorithms written by us. Please click the below link to get the video:
Final Video showing all Results (Click Here)