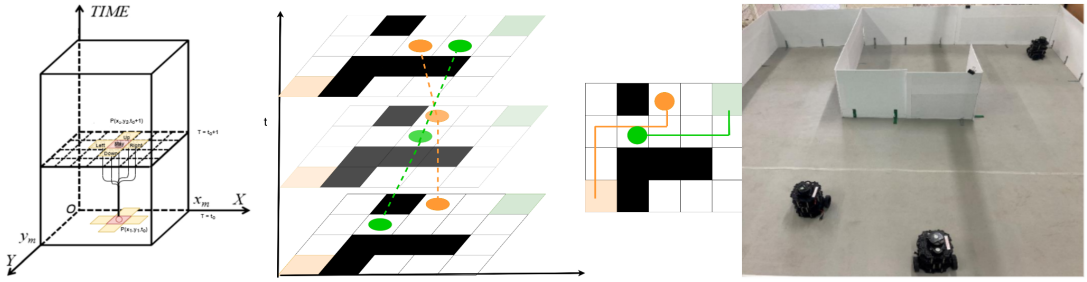
Path Planning for Multiple Mobile Robots using Spacetime Grids

[1]Hruthik V S, [2]Sai Kumar Reddy,[3]Nayani Sri Harsh

[1]Department of Engineering Design , Indian Institute of Technology Madras, Chennai, India

Keywords: *Spacetime planning, A*, Deadlock, Edge Clash, Time Complexity, Collision Check.*



ABSTRACT: Multi-Robot path planning has been an important topic for research because of rapid growth in robot automated processes, in warehouses and other domains. The main focus of our research work is to compute efficient trajectories in a decentralised manner, relying on localisation sensors and inter-robot wireless communication, by performing computations on the local robot. The robots should take decisions on the fly, and resolve conflicts beforehand quickly given the constraints of a simple processor with limited compute capability. Our proposal uses the efficient A* search algorithm in coherence with the 3D space-time path planning to resolve path conflicts. That is, when multiple robots reach within communication distance, they replan trajectories in spacetime to generate collision free paths. The result is a robust versatile planning algorithm that generates routes in real-time, efficiently. The solution can be easily extrapolated to multiple robots with a re-planning time complexity growing as $O(n)$, for *n* robots.

INTRODUCTION

Multi-robot planning is one of the topics in mobile robotics that is receiving much attention lately due to being actively used in many real life applications like autonomous intersection management, warehouse management, video games and other multi-robot systems. There is an increase in usage of multi-robot systems because of advantages in terms of sensing range, processing power and computing power over limited performance of a single-robot system. Above scenarios require robots to plan collision-free paths to different locations to perform different tasks. However, obtaining an optimal path for a system involving a large number of robots is computationally intensive and time consuming. Therefore, solving a multi-robot path planning problem with real-time planning times even with simple settings plays an important role in many real world automated systems. The proposed algorithm aims at finding paths for all the involved robots which are obstacle-free and inter-robot collision-free. The contributions of this article include:

– The formulation of the multi-robot planning in terms of simple search based algorithm

– A novel framework that utilises space time path planning to iteratively solve for collision free paths, which is the main contribution of this article

– Implementation of the framework in the CoppeliaSim simulation environment for a three robot system

– Systematic analysis of the framework and algorithms in terms of parameters like nodes expanded by the underlying A* algorithm, replanning threshold distance.

Most studies related to multi robot motion planning decouple the problem into two parts, finding obstacle-free paths and avoiding inter-robot collisions on these paths. There is also an Iterative Inter-Dependent Planning framework that coordinates single robot plans by making each robot compute a conditionally optimal path given other robots' current plans iteratively. Then each robot exchanges plans with others and replan accordingly. Centralised mission planning for multi-robot-multi-mission problems is another approach that uses rewards of missions for each robot that are calculated according to the purpose of the user. Then , a plan is generated such that the total reward of the system is maximised, if cost is used then it is minimised.

Firstly, we define the problem statement as a spacetime A* search problem, and then we discuss the implementation with a block diagram. Finally, we test our algorithm using 2 and 3 robots for multiple maps and situations. We discuss the results and metrics obtained by testing for various scenarios. The working simulation for this can be found in the github link: https://github.com/hruthikvs/MutliRobotMotionPLanning

METHODOLOGY

Problem Definition:

The multi-robot path planning problem is essentially an extension of the single-robot path planning problem, where a robot finds a trajectory from a defined start point to a goal point using various established search techniques. We create multiple robots having different properties, each having a defined start and goal. all robots run the search algorithm on their local processor, and communicate and simultaneously solve trajectory conflicts on the fly. For our problem, we will be creating an occupancy gridmap with required resolution and using A* search algorithm for path finding. The problem statement generally is restricted to a warehouse or a factory of fixed dimension with fixed obstacles, which allows us to simplify our solution by using a grid map based approach rather than the more sophisticated sampling approaches, which will greatly increase complexity and computational load for running the algorithms. We will be using a 3D spacetime grid map for our planning process which essentially provides a complete solution framework for multi-robot scenarios. This kind of an approach is inspired by how humans resolve conflicts while driving. When two cars approach in a narrow alleyway, each of the car drivers come to a mutual agreement as to what to do to resolve the conflict in their paths. That is to say, they plan a future trajectory together to approach a common and possible solution. In this case, one driver decides to reverse his car until an intersection is reached, while the other car just proceeds forward. The caveat to notice here is that, one driver (robot1) has to give in to the other driver (robot2), in order to resolve the situation. Which robot gets the priority will be a topic discussed further.

The Spacetime A* multi-robot search motion planning problem is defined as follows:

- *State Space:* (x,y,t) , where x,y are cartesian coordinates and t is timestep
- *Action Space:* stay, up, down, left, right
- *State expansion Function:*
  - stay: $(x_0, y_0, t_0) \rightarrow (x_0, y_0, t_0 + 1)$
  - up: $(x_0, y_0, t_0) \rightarrow (x_0, y_0 + 1, t_0 + 1)$
  - down: $(x_0, y_0, t_0) \rightarrow (x_0, y_0 - 1, t_0 + 1)$
  - right: $(x_0, y_0, t_0) \rightarrow (x_0 + 1, y_0, t_0 + 1)$
  - left: $(x_0, y_0, t_0) \rightarrow (x_0 - 1, y_0, t_0 + 1)$

  This expansion function is clearly visualised in Figure 1

- *Cost Function:* 1 (for all actions)
- Expansion Constraints: The graph will add a new node only if the node does not fall in :
  - Map boundaries
  - Obstacles
  - Other Robot's Path

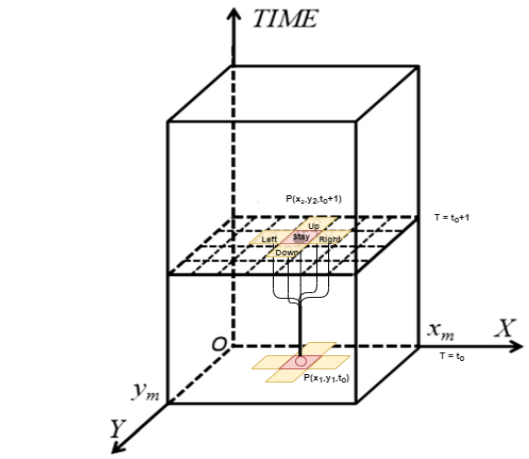


Figure 1: Visualises the node expansion in spacetime for the five actions: stay, up, down, left and right.

MULTI-ROBOT SPACETIME A* SEARCH:

Figure 2 shows a simple two robot planning problem. The Image on the right shows how the robots would move if they plan without considering the other robot's presence, using a simple 2D space grid planning. As we can see, if the robots execute such naively generated trajectories using a constant velocity, they will eventually collide with each other. Therefore, introducing an additional parameter time *t* into the problem solving framework provides us an extra dimension to resolve such conflicts and arrive at solutions.

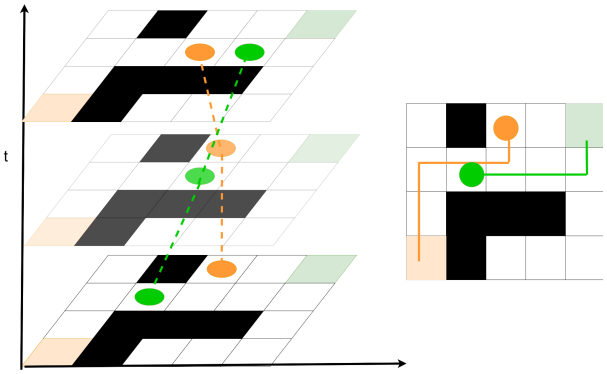


Figure 2. Visualises how path clashes in space can be resolved using the additional dimension of time. Here, the green robot plans first and the orange robot replans avoiding the green robot's trajectory

Before attempting to solve the problem, it is important to conduct a feasibility check, to determine if the solution being developed can be realisable in the real world. We start by laying down some assumptions and requirements for this solution procedure.

Assumptions:

1. *Communication:* All robots are able to communicate with each other reliably anytime from anywhere. The

most important and primary assumption that our solution framework depends on. Robots are required to pass on their current spacetime paths to other robots as it is a necessary resource for generating their paths.

2. *Homogenous Robots:* It is also assumed that all the robots are homogenous, which means that all the robots share the same action space and action outcomes are the same when a particular action is executed on different robots(same state expansion function). However, this constraint can be easily relaxed. We will only discuss the homogenous case
3. *Omni-directional robot* : A holonomic 3 DOF robot to allow direct movement across adjacent grid cells
4. *Stepped Movement:* To solve the problem using the spacetime grid topology, we also discretise our time space at 1 second interval. Hence, it is important for each robot to move to their adjacent positions in one timestep. To achieve this, we need to move all robots and wait for them to reach their respective grid cells defined in the next timestep.
5. *Perfect Localisation*: All robots have the map in their memory and are able to perfectly localise and update its map if any new obstacles are detected.

Robot Requirements:

1. *Sensors:* Sensors for SLAM
2. *Microcontroller:* microcontroller with enough memory to store paths of multiple robots(atleast 5). The microcontroller should also have enough speed to run A* quickly during path clash scenarios to avoid delays in trajectory execution
3. *Wireless Communication :* Bluetooth or zigbee wireless communication protocols to allow for inter robot communication

Solution Procedure:

The idea for the solution is as follows. Robots are considered as independent entities with their own computer, having memory and processing power to plan paths for itself. Each robot is given a specific task, which it can achieve by moving from a predefined start location to a goal location.

1. Firstly, the robots start from their start locations, plan their trajectories assuming absence of other robots in the environment. These trajectories are computed in spacetime, taking 1 second for one timestep. For example, the spacetime trajectory computed will look like this : $[(0,0,0), (1,0,1), (1,1,3), (1,2,3)]$ Here each point is (x_t, y_t, t). The start is (0,0) and the goal is (1,2), which is reached in t=3 seconds.
2. Once they plan their trajectories, they start executing their spacetime trajectories.
3. When the robots reach close to each other,i.e. within a set threshold distance when inter-robot communication is possible, the robots collaboratively plan the trajectories according to a priority order (may be predefined or can be defined based on a heuristic). That is to say, if 3 robots reach closeby, the first robot re-plans its trajectory from the current position. This trajectory is shared with the other two robots. Now the second robot computes its spacetime path, considering the first robot's spacetime path as an obstacle node. Now, this spacetime path is shared with the 3rd robot, which replans its path considering both the first and second robot trajectories as obstacles. This process can be easily extrapolated to multiple robots that reach the vicinity.

Implementation:

The block diagram elaborating the code implementation is given in *Figure 4.* To explain briefly, the implementation contains two modules: a motion planner and a robot controller. The motion planner contains the *Robot* class, the search algorithm and the map required.

- The *Robot* class has several attributes and functions required to perform the aforementioned tasks (See *Figure 3b*).
- The *Main* class coordinates multiple robot object creation, re-planning and communicates with the controller module to control the differential drive robot in the simulator.
- *Control.py* file contains controller parameters like PID values for linear and angular velocity
- The *Robot* class interacts with the *Search* class using the *getPath()* function. The *Search* class consists of different search algorithms like AStar, BFS, UCS, DFS, etc. It also consists of several heuristics like manhattan distance, euclidean distance, etc. For the purpose of our implementation, we will be using the A* Search Algorithm with Manhattan distance heuristic.

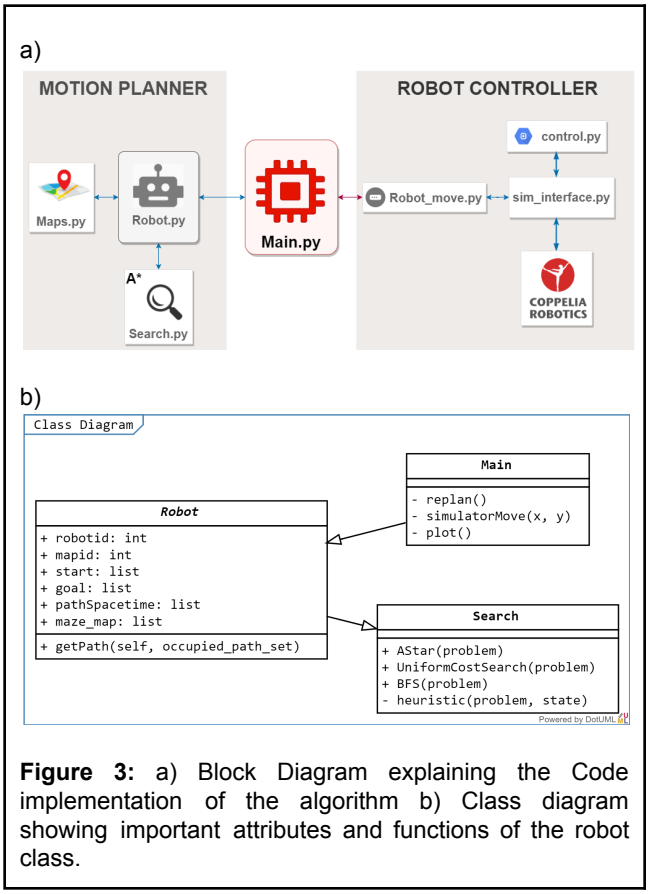


Figure 3: a) Block Diagram explaining the Code implementation of the algorithm b) Class diagram showing important attributes and functions of the robot class.

Using an Object Oriented Programming approach enables easier interpretation of the real world robots, and allows for

efficient code. The Algorithm explaining the complete process happening in the *Main* class, is given below.

Algorithm 1 Spacetime A*

```
# Create Robot Objects
r1 = Robot(robotid = 1,mapid=mapid,start=[14,4],goal=[9,9])
r2 = Robot(robotid = 2,mapid=mapid,start=[14,18],goal=[9,8])
r3 = Robot(robotid = 3,mapid=mapid,start=[10,6],goal=[14,5])

#Get initial path
r1.getPath(), r2.getPath(), r3.getPath()
r1.t=r2.t=r3.t = t=0

while not (r1.goalreached and r2.goalreached and r3.goalreached) do
    # update the time(array index) for robots only if goal not reached
    if not r1.goalreached then
        r1.t = t
    end if
    ...
    ...

    #Move Robot to specified position
    print('Robot 1(x1) :',path1[r1.t][:2], 'Robot 2 (x2):',....)
    robotmove.RobotMove(x1,x2,x3)

    if Robot reached Closeby then
        #update start of robots
        r1.start = list(path1[r1.t][:2]) , r2.start = ....

        #Plan for three robots one after the other
        r1.getPath()
        r2.getPath(r1.pathSpacetime)
        r3.getPath(r1.pathSpacetime+r2.pathSpacetime )

        # Update the respective spacetime trajectories
        newPath1 = r1.pathSpacetime, .., ..
        path1[r1.t+1:] = newPath1[1:], .., ..
    end if
    if Any robot Reaches Goal then
        -Update map with Goal Robot position as obstacle
        -Replan for all robots considering goal reached robot
        as obstacle
    end if
    t = t+1
end while
```

Challenges:

Edge Clash Error:

One of the significant challenges we had to overcome for the success of this project was the condition classed spacetime edge clash. *Figure 4* clearly visualises this error. In the situation given, suppose the orange robot plans first and then the green robot plans around the orange robot's spacetime trajectory, we observe that there is no clash in spacetime for the generated paths, but in reality there will be a physical collision if the robots execute these trajectories.

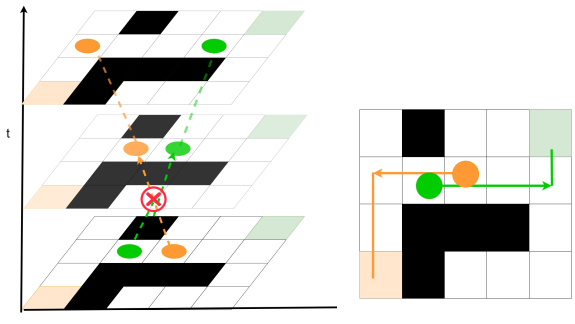


Figure 4. Shows a scenario where there is no clash in spacetime paths, but there is a physical robot clash. We categorise this case as edge clash and resolve it during collision check

To resolve this issue, while expanding a node in the *getSuccesors()* function of the robot class, we not only do a collision check for the successor nodes, but also do an edge collision check. We achieve such an edge collision check by creating a midpoint array for the occupied path, and while expanding to a successor node, we check if the midpoint of the current position and successor position collides with any of the midpoints of the occupiedPath. By using this kind of and edge collision check, we eliminated the occurrence of such conditions in future test cases.

RESULTS AND CONCLUSION

The validity of the proposed algorithm is tested in multi-robot systems consisting of 2, 3 robots and a known environment. First, we showcase the algorithm working to solve a generic scenario with two random start and goal states of two robots and then, present the algorithm resolving the deadlock situation between two mobile robots.

Figure 5 shows a situation with two robots at random start states and corresponding random goal states. As per the algorithm, both robots plan their path without considering the other robot as shown. The purple path is planned by the blue robot, whereas the green path corresponds to the orange robot. When both robots are in each other's vicinity,and the distance between them is less than the replanning threshold distance, replanning is done. And the resultant path of the robots after replanning is shown in *Figure 5b* . This process happens unless there are no other robots in a robot's vicinity.

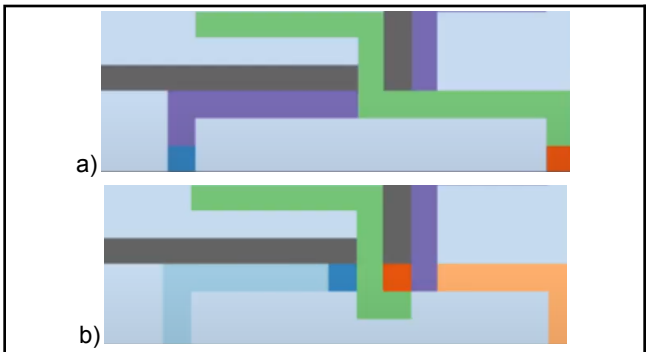


Figure 5. Visualisation of the replanning procedure

As discussed previously, the spacetime planning algorithm is a versatile algorithm and handles different complex situations very effectively. *Figure 6* shows such a deadlock situation. There are two robots, robot 1 (left blue) and robot 2(right blue) with corresponding goals, the right green goal of robot 1 and left green goal of robot 2. We see in the figure that both robots follow the replanned trajectory without any collisions with walls and other robots. From above observations, it is clear that the proposed algorithm can yield paths that satisfy our constraints, even in deadlock situations. Due to simplifying the problem into wall collision and inter-robot collisions into cell collision and edge collision , there are no edge cases that are required to be resolved using hard coding. Hence, it is easy to generalise the spacetime planning algorithm to multiple robots exploiting the advantages of Object Oriented Programming(OOPs). *Figure 7* shows a 3 Robot extension of the algorithm. Now, the same algorithm is extended to 3 mobile robots to analyse its efficiency with respect to some parameters.

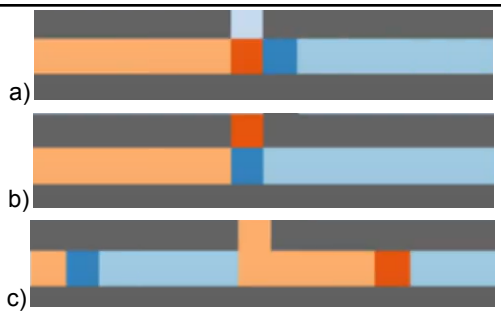


Figure 6. Deadlock Handling : a) Reach a deadlock b) Robot orange decides to give in c) Robot *blue* moves and clears the path for robot *orange*.

The efficiency of the algorithm is considered to be inversely proportional to the number of nodes expanded by the underlying A* search algorithm. First, we examine this by changing the cost of the stay action from zero to thirty. The plot in *Figure 8a* shows an initial rise in the number of nodes expanded and then falls to remain constant for the rest of the costs. When the cost is lower for the stay action, it is preferred over moving. The reduced nodes expanded shows that it is better to stay than to always move around. However at higher costs, stay action would not be preferred and the algorithm biases towards choosing only between the four other actions. Therefore, the number of nodes sees a decrease and after a point saturates. We know that replanning step commences whenever two or more robots are nearby and the frequency of replanning increases with replanning threshold distance. *Figure 8b* verifies the above statement by displaying the increase in number of nodes expanded with increase in replan distance. *Figure 8c* shows that the number of nodes expanded are more for euclidean heuristic than Manhattan heuristic as Manhattan heuristic is a better approximation of the actual cost to goal, for a 4 way moving robot..

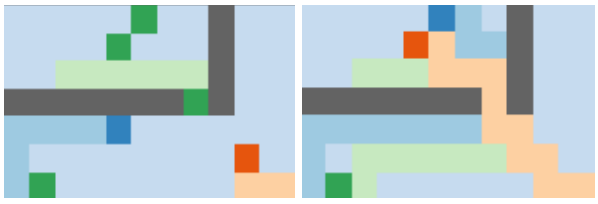


Figure 7. Path Generated for 3 Robot case

DISCUSSION

It is observed that the time taken for a single instance of replanning increases linearly with the number of robots in the considered vicinity. Generally, planning is followed by time parameterization before the controller executes the plan. The advantage of the space time planning approach is that the final path itself contains the time parameter and hence, there is no need for time parameterization. We are aware that the chosen scenario is a very simple one and there may be several controller or hardware constraints in a real world system. It would be interesting to see the behaviour of the proposed approach with the constraints enforced. Time parameterization algorithms find optimal time parameterizations, i.e. which minimise traversal time while maintaining kinodynamic constraints. This might be possible using space time planning using a cost function based on both space and time. Then, the weight of time in the cost function can be manipulated based on the application.

The proposed algorithm also comes with its share of disadvantages. Replanning part is done in a sequential manner according to the order of robots which leads to the robot in the latter part of the order waiting idly. In the current implementation, planning for the robot is stopped as soon as it reaches its goal. This might exclude better paths for other robots that haven't reached their goals till then. We also added stay action to already existing four actions which increases the number of successors to explore therefore increasing the time complexity.

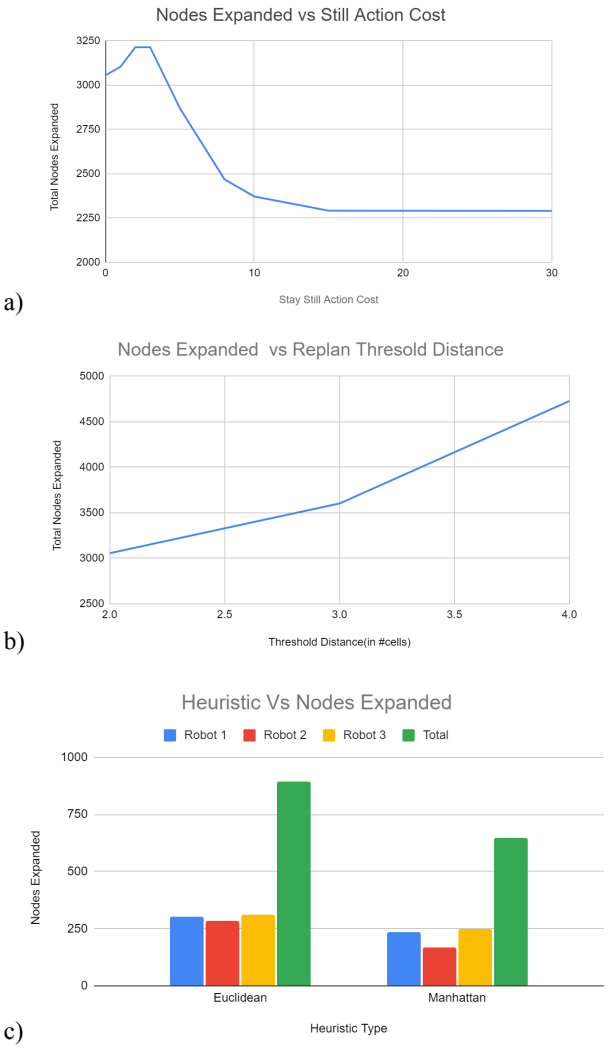


Figure 8. Graphs showing Nodes expanded with varying model parameters

ACKNOWLEDGMENT

The authors acknowledge Bijo Sebastian and Nirav Patel from the Department of Engineering Design., IIT Madras for motivating research on the above-mentioned technologies. We would like to acknowledge the Indian Institute of Technology, Madras for providing this opportunity.

REFERENCES

YouTube. (2021, February 18). *Lecture 9: Multi-robot Path Planning*. YouTube. https://www.youtube.com/watch?v=VJkFHIUHHXw&t=1423s&ab_channel=ProrokLab

García, E., Villar, J. R., Tan, Q., Sedano, J., & Chira, C. (2022). An efficient multi-robot path planning solution using A* and coevolutionary algorithms. *Integrated Computer-Aided Engineering*, *30*(1), 41–52. https://doi.org/10.3233/ica-220695

Jiang, Y., Yedidsion, H., Zhang, S. et al. Multi-robot planning with conflicts and synergies. Auton Robot 43, 2011–2032 (2019). https://doi.org/10.1007/s10514-019-09848-1

Hwang, N. E., Kim, H. J., & Kim, J. G. (2023). Centralised Mission Planning for Multiple Robots Minimising Total Mission Completion Time. Applied Sciences, 13(6), 3737. MDPI AG. Retrieved from http://dx.doi.org/10.3390/app13063737