International Journal of Advanced Robotic Systems

OPEN ACCESS
ARTICLE

Cellular Automata Based Real-time Path-planning for Mobile Robots

Regular Paper

Usman Ahmed Syed[1,2,*] and Faraz Kunwar[1]

1 Probabilistic Robotics and Intelligent Systems in Motion (PRISM) Lab, Mechatronics Engineering Department, College of Electrical and Mechanical Engineering, National University of Sciences and Technology (NUST), Islamabad, Pakistan
2 Department of Electrical Engineering, COMSATS Institute of Information Technology, Islamabad, Pakistan
* Corresponding author E-mail: s.usman87@yahoo.com

Received 31 Oct 2013; Accepted 27 Mar 2014

DOI: 10.5772/58544

Abstract Intelligent mobile robotic agents demand optimal motion planners with minimal query time. Most contemporary algorithms lack one of these two required aspects. This paper proposes an efficient path-planning scheme based on cellular automata (CA) that generates optimal paths in the minimum time. A Cellular automaton is evolved over the entire search space and subsequently used for the determination of the shortest path. This approach generates a parent-child relationship for each cell in order to minimize the search time. Performance comparisons with A*, Dijkstra's, D* and MPCNN have proven it to be time-efficient. Analysis, simulation and experimental results have proven it to be a robust and complete path-planning scheme. Also it has demonstrated to be time-efficient in both static and dynamic environments.

Keywords Cellular Automata, Path-planning, Obstacle Avoidance

1. Introduction

Path-planning is an essential constituent of the design of an intelligent mobile robot. The field of motion-planning has been explored in detail by researchers over the past two decades with the aim of developing an algorithm that shows convergence upon the optimal path in the minimum possible time. This work aims to develop an efficient path planner that provides the shortest path from robot to goal location in a grid environment.

Visibility graphs [1], [2] and Voronoi diagrams [3] are among the earlier techniques exploring optimal path searches. These algorithms offered promising results but they were mainly developed for static environments. Furthermore, a visibility graph is computationally expensive and unsuitable for use in real-time systems, whereas Voronoi diagrams fail to provide an optimal solution. Another class of algorithms developed for this purpose are cell decomposition methods [4]. However, the computational efficiency of cell decomposition techniques is highly dependent upon the size of the cells, thereby also making it an inefficient algorithm for real-time implementations. Another approach is the use of potential field [5] methods, which provides encouraging results in most cases but which fail in some specific situations where attractive and repulsive fields tend to cancel each other, i.e., local minima [6].

In order to overcome the problems confronting the above-mentioned algorithms, probabilistic approaches, namely PRMs [7] and RRTs [8], [9] have been developed. These algorithms are capable of finding paths in complex environments but, due to their probabilistic nature, they do not provide optimal solutions. Furthermore, their sampling strategies tend to become complex in the case of

narrow passages. It has been proven that RRT does not approach optimality, whereas RRT* [10], [11] requires an infinite number of iterations to converge upon optimality.

Another popular category of algorithms that have been extensively exploited for path-planning problems comprises heuristic-based search algorithms. In this category of algorithms, A* search [12] and Dijkstra [13] have been used for both static and dynamic environments. The D* algorithm [14] is another heuristic-based search algorithm, designed to handle specifically dynamic obstacles. The method presented in [15] uses one such heuristic search method for 3D vision-based maps. The present work carries out a comparative analysis with these search algorithms below.

More recently, machine learning paradigms have been utilized for global obstacle-free path searches. One of the major advances in this paradigm is the use of genetic algorithms [16], [17] and fuzzy-based [18], [19] approaches for the extraction of global collision-free paths. Another promising learning paradigm for this problem is the use of neural networks [20]. The problem with these approaches is that they require a large database to learn and generalize. It is generally difficult and sometimes impossible to provide such data. However, an exception to the rule is the use of a modified pulse coupled neural network (MPCNN) [21], which is capable of path-planning in arbitrarily complex environments and which has been proven to provide the globally shortest path. However, the major problem with this technique is that it is computationally inefficient. A comparison of our method with this technique is presented later in this paper.

Finite automata comprise a class of algorithms with discrete inputs and outputs. Cellular automata [22] are a special class of finite automata which constitute an n-dimensional array of cells wherein each cell can take a set of possible values. Path-planning using cellular automata has been addressed previously and a straight-moving path planner has been proposed in [23] which derives its strategy from a multi-agent path-planning architecture using cellular automata [24]. However, this approach does not provide an optimal solution since it requires an elaborate search of the environment which degrades the efficiency of the method. In [25], CA is applied to polar depth maps to determine a collision-free path. However, this work is focused on the local-planning problem, which may result in a globally non-optimal path. Another similar sub-optimal approach that utilizes CA is given by Behring et al. [26]. A case in which a diamond-shaped robot is considered for navigation in a 2D environment using CA is presented in [27]. With this strategy, CA are used to determine cells that are equidistant from obstacles and which later determine the path of robot. Since the path is required to be equidistant from all obstacles, it therefore results in a non-optimal solution.

This paper presents a CA-based approach to compute the shortest path in a 2D configuration space. Rule-based exploration of the environment is coupled with

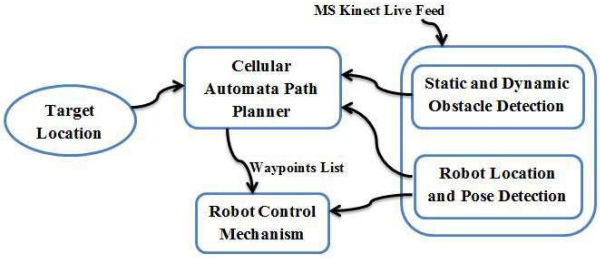


Figure 1. System modules

parent-child relationships for each cell to simplify the search process. This work focuses on path-planning in 2D grid environments. Since, in a grid environment, the robot is constrained to move only in eight possible directions, the optimality of the algorithm presented in this work is also constrained. However, this constraint can be relaxed by taking a sparse map of the environment(at the cost of reduced accuracy). Simulations and ground results have verified it to be an efficient method, both in the presence of static as well as dynamic obstacles. Our proposed method assumes that the robot and obstacle locations are known a priori. For environment perception, we have utilized a RGB-D camera (Microsoft Kinect). Kinect is being used extensively in robotics research. In [28], a quad-rotor uses a Kinect camera for navigation as well as localization. Kinect Monte Carlo localization [29] utilizes RGB-D Kinect for scene simulation. In our experiments, we use Kinect to determine the state of the robot and the various obstacles. Based upon the information from this perception sensor, a workspace is developed. The workspace is converted into the configuration space and path-planning is carried out using the proposed CA.

The main objective of this paper is to present a novel CA-based real-time path-planning algorithm. The effectiveness and completeness of the proposed algorithm is demonstrated by carrying out a detailed comparison with contemporary and well-established techniques, such as A*, Dijkstra, MPCNN and D*. The rest of the paper is organized as follows. Section 2 gives an overview of the complete system. Section 3 highlights the model of the proposed CA-based scheme. Section 4 explains the computational steps of the proposed algorithm using pseudo-code. A comparison of the proposed algorithm with A* search, Dijkstra, D* and MPCNN is carried out in Section 5. The next section 6 presents the simulation results and the analysis of the proposed technique. Section 7 deals with the experimental implementation and the results of the proposed automata planner. Section 8 concludes the paper.

2. System Overview

As given in Figure 1, the complete architecture can be thought of as a combination of three modules: vision-based object identification, CA-based path-planning and robot control using the robot's speed and turn-rate parameters. The step-by-step control flow of the three modules is presented in Figure 2 and each module is discussed independently in the succeeding paragraphs.

2.1. Object Identification (Vision) Module

In this module, the Microsoft Kinect is utilized as an image acquisition source to identify both the static and dynamic obstacles as well as the robot. As shown in Figure 2 (a), the image is acquired in the first step and is then processed in the second step to estimate the position and orientation of the static obstacles, the dynamic obstacles and the robot (details are given in Appendix A). Based on the information provided by the vision module, the workspace of the robot is generated (Figure 2 (b)). Since, in this case, the robot is considered as a point robot in the planning phase, the size of the obstacles needs to be increased. Therefore, the robot's workspace is further processed using the Minkowski sum [30] to generate the configuration space of the robot (as shown in Figure 2 (c)).

2.2. Configuration Space

The space of all possible configurations of a robot is called the configuration space [11]. Consider a robot R navigating in a 2-D Euclidean space, where the set of all possible configurations of the robot is represented by $Q = \{q_1, q_2, ..., q_n\}$ and the set of obstacles is represented by $O = \{O_1, O_2, ..., O_n\}$. Accordingly, the configuration space can be modelled as a continuous mapping represented by $\tau : [0,1] \rightarrow Q$, where $\tau(0) = q_{init}$ and $\tau(1) = q_{goal}$. The path-planning problem is to find a path in the configuration space such that no configuration of the robot collides with the obstacles. In other words, the problem is to find a set of configurations of the robot from q_{init} to q_{goal} in the free configuration space where:

$$Q_{free} = \{q \in Q \mid R(q) \cap (\bigcup_{i=1}^{n} O_i) = \phi\} \tag{1}$$

Since the Robot can be of any arbitrary shape in the workspace, the profile of the robot also needs to be considered in the configuration space. This is done by taking the Minkowski sum of the profile of the robot R with every obstacle O_i. Hence, every obstacle in the configuration space is remodelled such that:

$$R \oplus O_i = \{x + y \mid x \in R, y \in O_i\} \tag{2}$$

Since the shape of the robot has been catered for, it can now be considered as a point robot in the configuration space.

2.3. Path Planning Module

The robot's location identified by the vision module is appended with the target location and the configuration space of the robot (shown in Figure 2 (d)). This information is passed on to the path planner. A CA-based path planner utilizes this data and generates a collision-free path to the goal state within the configuration space, as shown in Figure 2 (e). The path returned by the planner is in the form of a set of way-points that the robot is supposed to pass through in order to reach its destination.

2.4. Robot Control Module

In the last step, we generate control commands for the robot based on the way-points selected above. Based on both the current and the next way-point robot control parameters, the robot's speed and turn-rate are generated using the differential drive kinematics model. These parameters are then issued to the robot through the PlayerStage wireless interface. This complete loop of planning and execution is repeated until the robot reaches its destination.

3. CA Model

CA are decentralized, discrete space-time systems defined as quadruples over a cellular space [24]. CA consist of a large number of locally connected identical entities, whereby each entity is updated based on a set of transition rules.

CA are formally defined as quadruples (d, q, N, f), where:

- d Dimension of the CA.
- q Set of the possible states of the CA.
- N Set of the relative positions of the CA's neighbouring cells.
- f Local function defining the local transition rule.

In the proposed algorithm, the CA architecture consists of a 2D lattice of cells. Each cell constitutes of a six-element tuple, the members of which are used in the evaluation of the local transition function. These are:

$$(S_{state}, S_{cf}, S_{pf}, t_{on}, P_{(i,j)}, \phi_s)$$

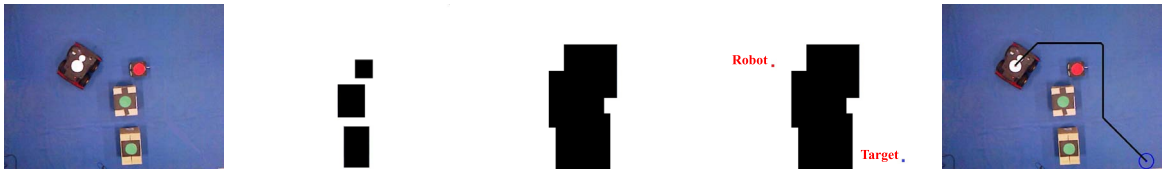
where:

- S_{state} Current state of the cell. 0 for off, 1 for on.
- S_{cf} Child Flag. High if the cell is a child.
- S_{pf} Parent Flag. High if the cell is a parent.
- t_{on} Time at which the cell transitioned to active state.
- $P_{(i,j)}$ Address of the parent cell.
- ϕ_s Maintains the accumulative cost from goal to the current cell.

Cells are interconnected in the local neighbourhood within the constraints of the eight nearest neighbours, which is also called 'type-II neighbourhood' and 'Moore neighbourhood' [22]. For example, if d = 2, this neighbourhood will comprise nine sites (as shown in Figure 3).

The activation of each automaton is governed by the following transition rules:

Rule: 1 - The current cell will become active if, and only if, one of the neighbouring cells is in an active state and the t_{on} of the neighbouring cell causing it to activate is less than t (where t is the time of the current iteration). Otherwise, it will remain quiescent.



(a) Image acquisition (b) Obstacle identification (c) Configuration space (d) Robot and target identified (e) Planned path

Figure 2. Compete system illustration

$$S^t_{state}(i,j) = \begin{cases} 1 & iff \quad \exists\, S^t_{state}(i+m, j+n) = 1 \\ & \quad \& \; t_{on}\ (i+m, j+n) < t \\ & \quad m,n \in \{1,-1\} \\ 0 & otherwise \end{cases} \tag{3}$$

Rule: 2 - A cell will only become a child of another cell if the other cell is already in an active state with a $t_{on} < t$.

$$S^t_{cf}(i,j) = \begin{cases} 1 & iff \quad \exists\, S^t_{state}(i+m, j+n) = 1 \\ & \quad \& \; t_{on}\ (i+m, j+n) < t \\ & \quad m,n \in \{1,-1\} \\ 0 & otherwise \end{cases} \tag{4}$$

Rule: 3 - A cell will only become a parent cell if it is active, if it is already a child and if its $t_{on} < t$.

$$S^t_{pf}(i,j) = \begin{cases} 1 & iff \quad \exists\, S^t_{state}(i,j) = 1 \\ & \quad \& \; S^t_{cf}(i,j) = 1 \\ & \quad \& \; t_{on}\ (i,j) < t \\ 0 & otherwise \end{cases} \tag{5}$$

Rule: 4 - A cell will only become a child of a neighbouring cell with the lowest accumulative cost function.

$$\phi_s(i,j) = \begin{cases} min(\phi_s(i+m, j+n) + \delta) & iff \\ & \quad \exists\, S^t_{state}(i+m, j+n) = 1 \\ & \quad \& \; S^t_{pf}(i+m, j+n) = 0 \\ & \quad m,n \in \{1,-1\} \\ 0 & otherwise \end{cases} \tag{6}$$

where δ is the cost of connecting the current cell $S(i,j)$ with a neighbouring cell. The costs are:

$$\delta = \begin{cases} |1| & \forall S(i+m, j), S(i, j+n) \\ & \quad m,n \in \{-1,1\} \\ |\sqrt{2}| & \forall S(i+m, j+n) \\ & \quad m,n \in \{-1,1\} \end{cases}$$

Rule: 5 - A cell will remember the address of its parent cell as calculated in Rule (4).

$$P_{(i,j)} = \{(r,s) | \phi_s(i,j) = min(\phi_s(r,s) + \delta)\} \tag{7}$$

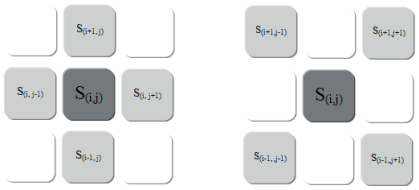


Figure 3. Moore neighbourhood

4. The Algorithm

To plan a path using the proposed technique, all the cells are initialized according to their state of occupancy. Obstacles are initialized as NaN and free space as state zero. The algorithm proceeds with the switching of states in outward fashion from the goal location until the robot cell is reached. In the next step, the path is extracted from the robot to the target using parent child relationships.

The computation steps of the algorithm are as under:

1) Initialization:
   $\forall S \in q_{obstacle}$=NaN, $\forall S \in q_{free}$=0, $S_{target} = 1, S_{cf} = true$,
   $\phi_{target} = 0, t^{target}_{on} = 0$ and $P(i,j) = (i,j)$
2) CA iterations:
   Repeat
   Initialize $fire_{flag} = false$.
   For each cell
   if($S^t_{state} == 0$)
   result=Execute rule 1 using (3).
   if(result==true)
   $fire_{flag} = true$
   Set t_{on} for current cell.
   Set child flag using Rule (2) as given by (4).
   Update the cost of the current cell using (6).
   Configure the parent of the current cell using (7).
   Increment the firing count.
   else
   continue
   else
   if (S_{pf}== false)
   if ($t_{on} < t$)
   Configure S_{pf} using (5)
   if (S_{robot}==1)
   return path
   else if ($fire_{flag} == true$)
   return failure

4.1. Completeness of the Algorithm

The proposed algorithm is a complete algorithm, i.e., if a solution exists, it will find it, otherwise it will declare that no solution exists. At every iteration, the algorithm keeps count of the number of cells which were activated during that iteration. If, for an iteration, none of the cells are activated, this means that there are no more cells which can be activated, and hence no path exists. On the other hand, as soon as the algorithm finds a path from the goal to the robot, it is the optimal path which is declared as output.

To prove this, we suppose that the contrary is true, i.e., that the algorithm is not complete. Therefore, there exists a path from start to goal but the algorithm cannot find it. For this statement to be true, either the algorithm never terminates or else it terminates incorrectly.

Suppose it never terminates. But, at any instant, if a parent-child relationship could be built, a new cell will be activated. According to Rule (4), a situation can arise where there is no other neighbour left with an accumulative cost lower than the current parent. Also, there might be no cell in the neighbourhood which is in a quiescent state. Hence, if there is no possible parent-child relationship left, no cell will be activated. If no cell is activated, the algorithm will declare that no path exists from start to goal and will then terminate.

Suppose it terminates incorrectly. According to Rule (1), a cell can only become active if its neighbouring cell was active. In addition, the obstacle cells are configured to be permanently off. Hence, an obstacle cell will never become a child and, according to Rule (3), it will never become a parent as well. Since an obstacle cell will never be activated, the resultant path will never end inside an obstacle. Also, since, according to Rule (1), no cell can become active if its neighbours are not active, only successive parent-child relationships are created, which only terminate when the start point and goal point are connected, and hence which produce a continuous path from start to goal.

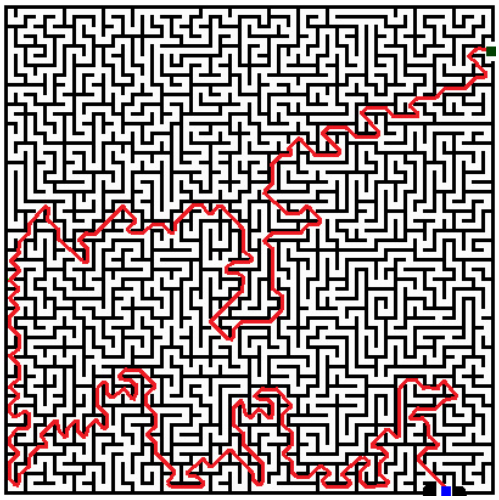


Figure 4. Solution of a complex maze using CA

Since both the above statements have been proved false, the opposite is true (i.e., the algorithm is complete).

5. Performance Comparison

In order to rigorously gauge the performance of the proposed algorithm and its improvements over the existing algorithms, its performance is compared with similar algorithms, namely the A*, Dijkstra, D* and MPCNN algorithms. All these algorithms work on the similar principle of the cost-based exploration of the search space coupled with parent-child-based linking for the retrieval of the shortest path.

*5.1. Comparison with A**

A* [12], a search algorithm which has been extensively exploited for the problems of path-planning and graph traversals, utilizes a best first-search technique and returns the least cost path given a start node and a goal node. A* evaluates the cost of the path based upon the sum (sometimes, a weighted sum is also considered) of the path it has already traversed and an admissible heuristic representing the path that is still to be covered. A* and its variants are directly influenced by their heuristic functions, whereas our proposed method is independent of any such heuristic cost. In order to keep any comparison unbiased, similar settings were created for both A* and our proposed method. In our case, A* uses Euclidean distance as its heuristic function with a unity cost for straight neighbours and $\sqrt{2}$ as the diagonal neighbour cost. The implementation of A* is taken from [31].

Figure 5 summarizes the performance comparison results of the proposed method with the A* search. As shown in Figure 5 (blue and yellow bars), the path lengths returned by both the algorithms in most of cases are the same, with a few exceptions in which A* results in a relatively longer path than CA (however, the differences are negligible). In terms of time efficiency, our proposed algorithm is much better than the A* search, as shown in Figure 5 (green and red bars). The query time increases with the increase in the distance between the start node and the goal node. Here, it is worth mentioning that the search time for A* is directly influenced by the length of the path, whereas in the case of CA the path length has a minimal effect on the search time (the search time is well below 1 s in all the cases tested with CA). Hence, CA not only determines the shortest path but is also a time-efficient algorithm in comparison with A*. Both the algorithms were tested under various obstacle configurations and it was noted that, as the complexity of the environment increases, CA becomes increasingly more efficient than A*.

Figure 6 presents two possible comparison scenarios which were considered. As presented here, in most of the cases tested for comparison, the paths returned by A* and CA were different but the commutative cost for both the paths was the same.

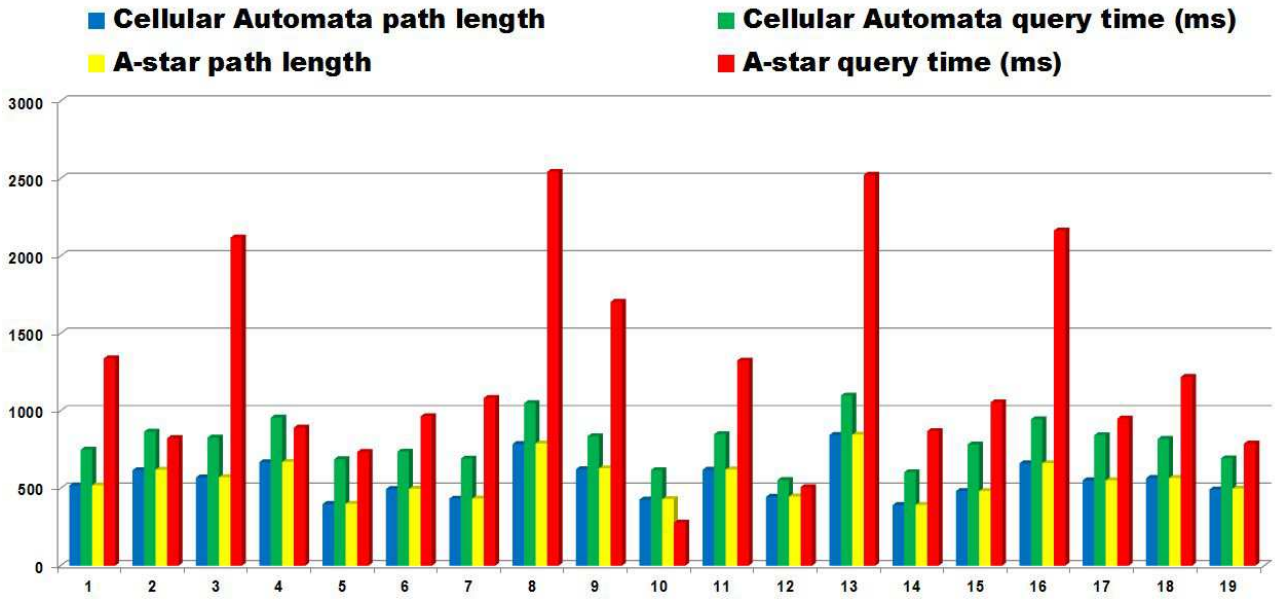
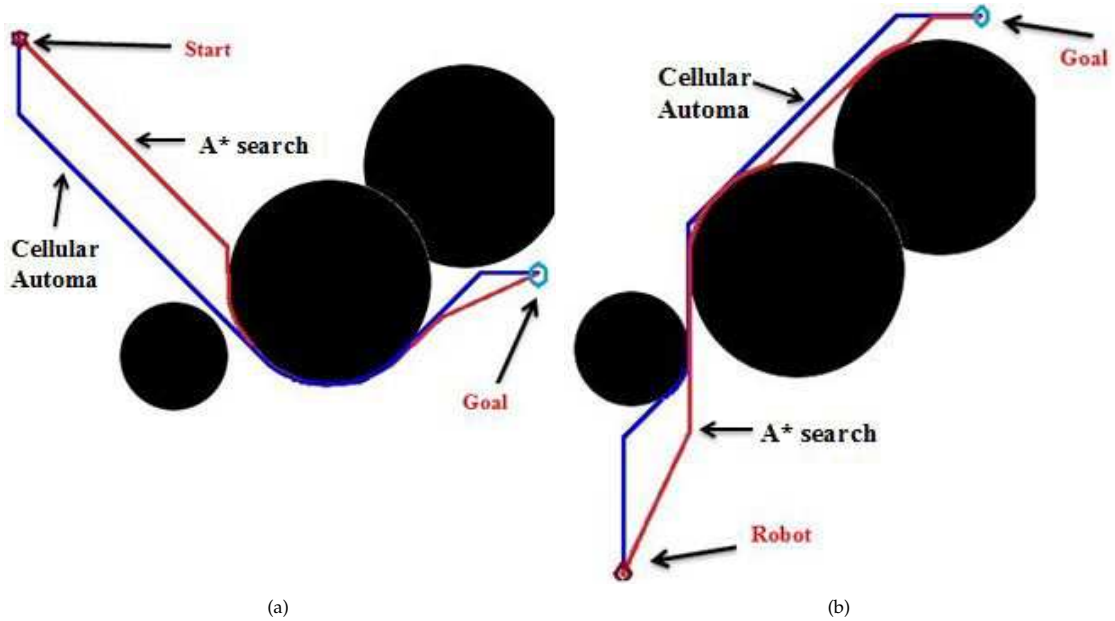


Figure 5. Path length and planning time-based comparison of CA with A*



(a) (b)

Figure 6. Sample images with paths returned by CA and A* searches

5.2. Comparison with Dijkstra's Algorithm

Dijkstra's algorithm [13] is yet another graph search algorithm used for solving single-source shortest path problems. Similar to the approach we presented in this paper, Dijkstra's algorithm utilizes a cost-based approach for exploration of a graph and couples it with the parent-child relationship for finding the resulting path from robot to target node. Dijkstra's algorithm is a generalization of the A* algorithm (A* becomes Dijkstra's algorithm if the heuristic function is taken as zero). Numerous comparisons of A* with Dijkstra's algorithm available in literature [32–34] have reported that A* achieves much better performance through the use of heuristics provided that the heuristic function is admissible. [32] has experimentally shown that A* outperforms Dijkstra's algorithm using a Euclidean heuristic function. Additionally, A* becomes more advantageous as the size of the graph increases [33]. Since we have shown that our algorithm outperforms the A* algorithm, it therefore also outperforms Dijkstra's algorithm.

*5.3. Comparison with D**

Another algorithm that has been extensively utilized for path-planning in dynamic scenarios is D* [14]. This algorithm initially plans a path with limited obstacle information and assumes that the unobservable space is completely traversable. As the robot navigates, it senses obstacles in the environment and performs local modifications to the initially planned path.

Sr.	Initial path length	Avg. CA time (s)	Avg. D* time (s)
1	904.428	1.5293	1.5213
2	767.703	1.3751	1.6436
3	874.201	1.4095	2.0083

Table 1. Comparison results with D*

In order to gauge performance, a comparison of the proposed work with D* was carried out under identical environments and using parameters. It was observed that in all cases the path length computed by each algorithm was the same; however, the computational time involved reveals some important information regarding the working of both algorithms. The analysis of the comparative study reveals that D* takes, on average, more time in computing the total path as compared to the CA approach. Furthermore, in the absence of major modifications required in the path to compensate for dynamic obstacles, the behaviour of the D* algorithm is much better. During simulations/experiments, it was determined that D* is highly dependent upon the number and dynamic behaviour of all obstacles present in the environment. This is primarily due to the fact that, whenever major modification in the initially planned path is required, the behaviour of the D* algorithm degrades significantly as compared to CA. A comparison of computational times as robot moves from start to goal locations is presented in Figure 7 (a). As mentioned earlier, the sudden peaks in D*'s planning time appear due to interaction of dynamic obstacles with the existing path. Table 1 shows a summary of the analysis. It was learned experimentally that the path lengths computed by each algorithm were the same. However, the average computational time for CA comes out better than D* as the robot moves from start to goal. One of the settings in which the comparison was conducted is shown in Figure 7 (b).

5.4. Comparison with MPCNN

Our proposed CA-based method was tested against the recently proposed MPCNN. MPCNN has been shown to be a robust method that can determine an optimal path. MPCNN utilizes a similar approach to our proposed method. It considers all the nodes of the graph as neurons. The firing of each neuron is controlled by its energy. Initially, the goal neuron is intentionally fired, which propagates the chain. The target neuron activates its neighbouring neurons as child neurons, thereby activating them. Once a neuron is activated, its internal energy increases with time and the neuron fires as its energy reaches a particular level. This firing pattern of neurons proceeds in a circular manner (called an 'autowave') with the target as the centre. As more and more neurons fire, the autowave spreads outwards, thereby reaching the robot neuron. Since all the neurons are coupled by parent-child relations, the shortest path is retrieved by back-tracking the neurons' parent-child relationships.

Figure 8 (a) shows a comparison between CA and MPCNN on the basis of computation time, where time axes for both algorithms are shown separately (right axis for CA, left axis for MPCNN). 100 random cases were tested, Figure 8 (a) clearly demonstrates that CA outperforms MPCNN, while the path determined by CA was the same as that for MPCNN. Statistically, the average time taken by CA was 8.19 ms whereas that of MPCNN was 749.94 ms. Figure 9 shows a comparison of the paths returned by CA and MPCNN for a similar setting.

6. Simulation Results

In order to gauge the performance of the proposed algorithm, we tested it in both static and dynamic environments. In all the simulations, our proposed method was proven to be an efficient algorithm in terms of time and path-optimality. Some of these results are discussed here. In all the subsequent figures, red represents the robot, blue shows the target configuration, black shows the obstacles and green shows the path extracted by the planner. The algorithm was tested in a challenging maze as shown in Figure 4, in which it successfully provided an optimal solution.

6.1. Local Minima Evasion

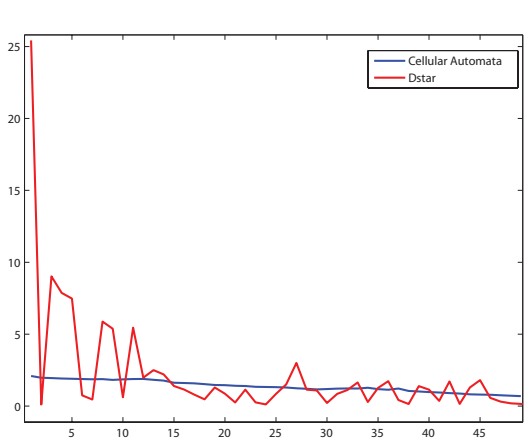
In order to verify the completeness of our algorithm, it was tested to plan a path in the presence of a local minima. Local minima issues arise in the presence of concave/U-shaped obstacles and, generally, path-planning algorithms direct the robot towards the centre of the obstacle instead of encouraging it to move at the boundary of the obstacle in order to avoid it. However, our algorithm considers the geometry of the obstacles and plan manoeuvres accordingly, thereby avoiding local minima problems. Figure 8 (b) shows the resultant path in the presence of a local minima. It is clearly evident that the proposed method evades the local minima efficiently.

6.2. Static Environment

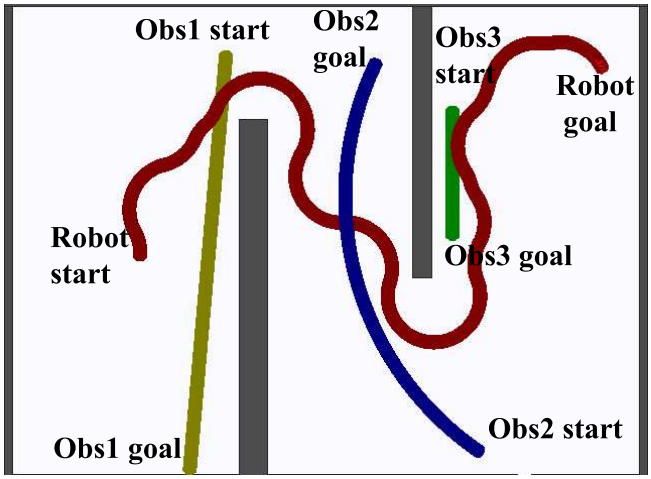
Real-world robots often make use of SLAM-based mapping of the environment. Therefore, we tested our algorithm in real-world SLAM environments as shown in Figure 9 (a). Figure 10 highlights the search pattern taken by the proposed algorithm. In this case, it is evident that there are two possible paths from the start to the goal location, and at first it seems that the "Left" path is lower in cost compared to the "Right" path. However, the search pattern shown in Figure 10 reveals that in fact the path on the "Right" is shorter then the "Left" one.

6.3. Dynamic Environment

To ensure that the proposed scheme works in real-time, it is mandatory for the algorithm to work in the presence of dynamic obstacles. To ascertain this, we simulated real-world scenarios on PlayerStage [35]. Figure 11 to Figure 13 present the results for dynamic obstacle avoidance, where the top row represents the path covered by each robot from its starting location to its current time-step. In these figures, there are a maximum of four robots, and dynamic obstacle 'A' is shown with a blue trajectory, 'B' is shown with a green trajectory and obstacle 'C' is shown with a yellow trajectory, while the intelligent

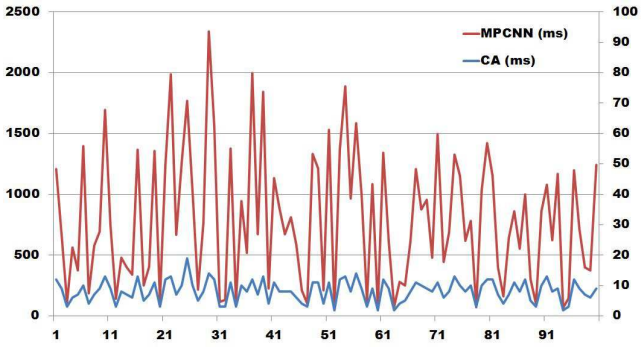


(a) Planning time-based comparison of CA and D*

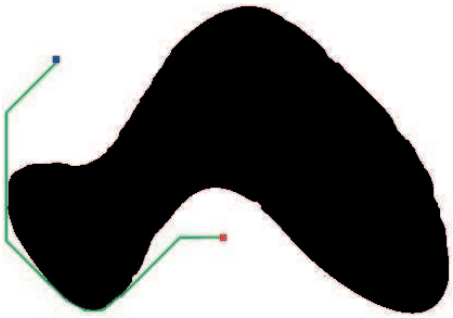


(b) Sample testing scenario used in the comparison

Figure 7. Performance comparison of CA and D*

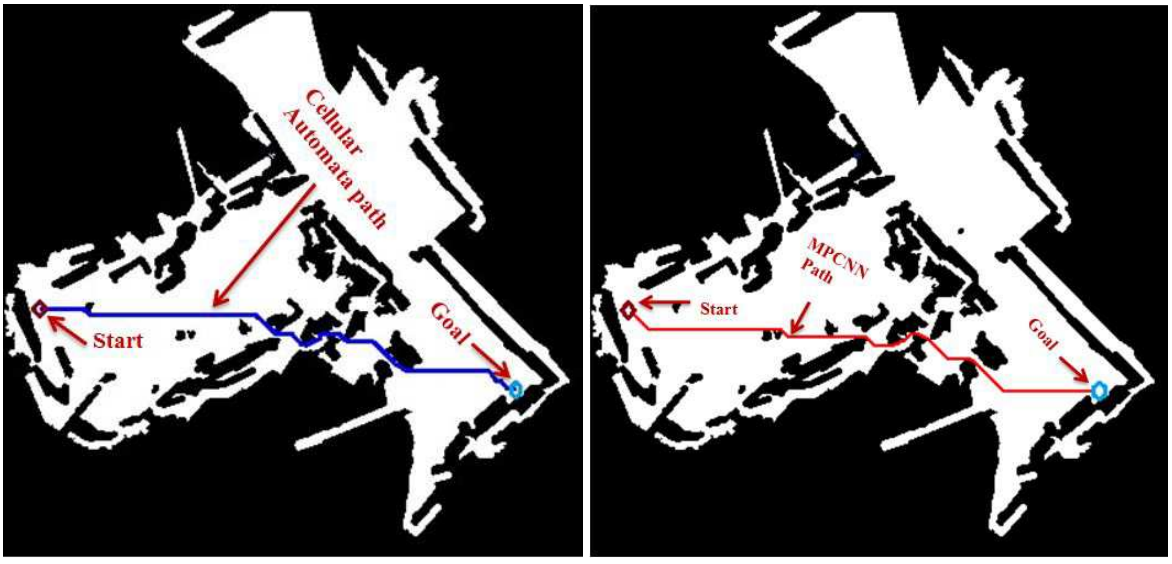


(a) Time comparison of CA and MPCNN.



(b)Path planned by CA in the presence of a local minima

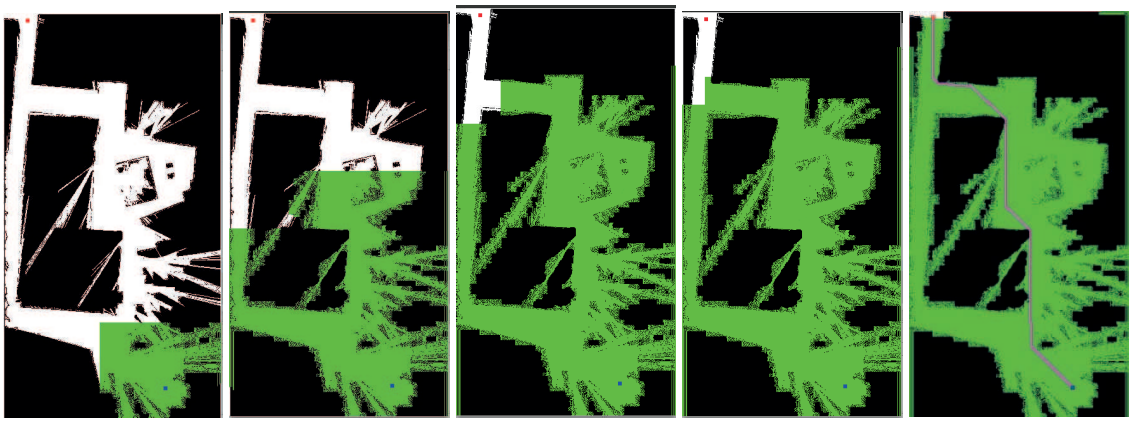
Figure 8. Performance comparison of CA with MPCNN and local minima evasion



(a) Path planned by CA in a SLAM environment.

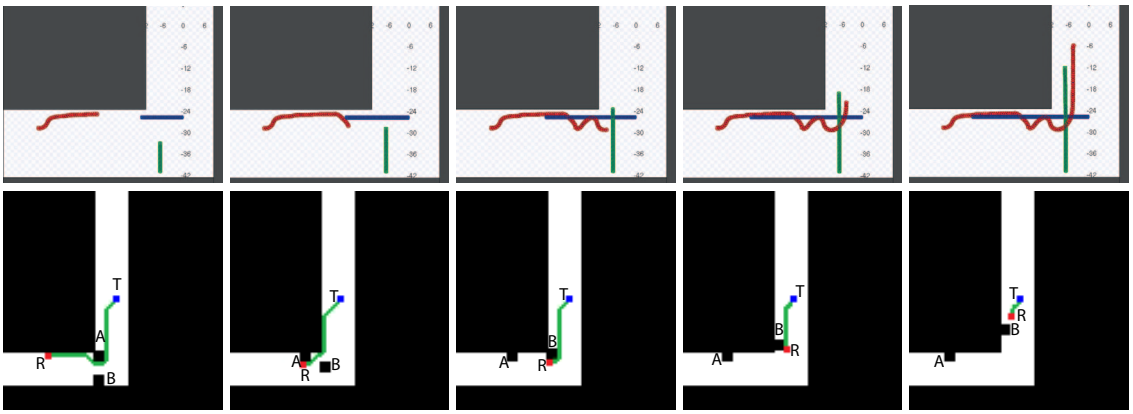
(b)Path planned by MPCNN in a SLAM environment

Figure 9. Comparison of paths in a 2D SLAM environment: CA vs MPCNN



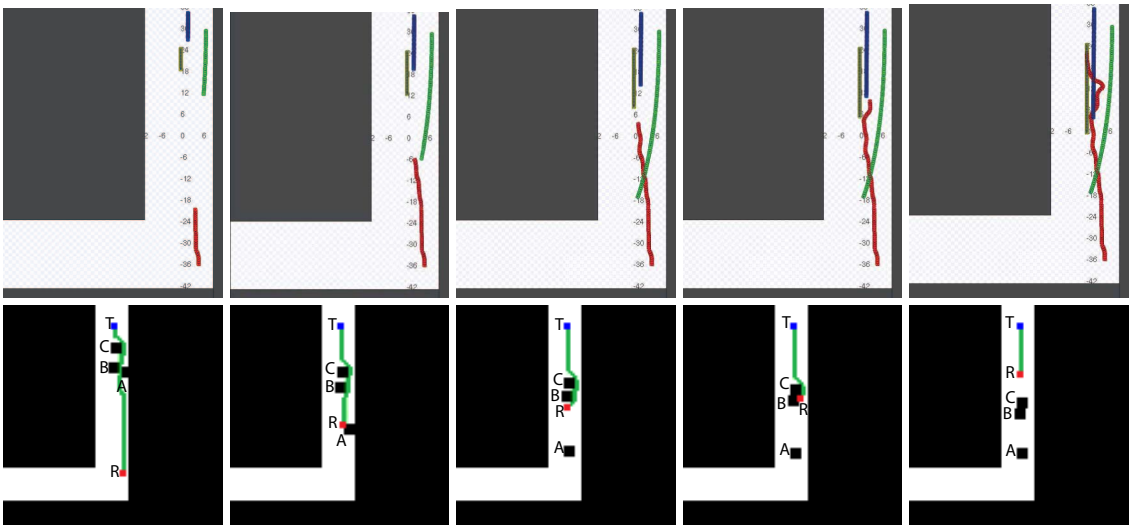
(a) (b) (c) (d) (e)

Figure 10. Cellular automata exploration pattern



(a) (b) (c) (d) (e)

Figure 11. Dynamic environment with two moving obstacles



(a) (b) (c) (d) (e)

Figure 12. Dynamic environment with three dynamic obstacles

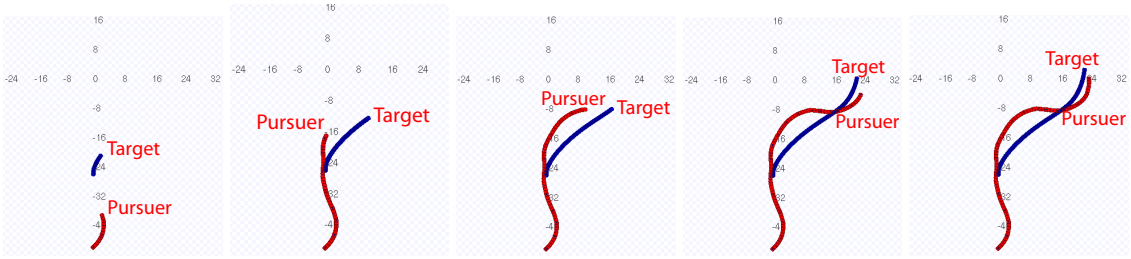


Figure 13. Pursuer robot with a target moving in a sinusoidal manner

path planner robot 'R' is shown with a red trajectory. Here ,'T' represents the target cell for the robot.

Figure 11 represents a test scenario where robot 'R' must intelligently traverse through an L-shaped hallway to reach its destination. Initially, the robot plans a path and begins navigation, as shown in Figure 11 (a); if the path is not updated, the robot will have a head-on collision with obstacle 'A'. Figure 11 (b) shows a successful avoidance manoeuvre where the robot reroutes its path to pass around the dynamic obstacle. Here, the planner initially executes a solution that makes robot 'R' pass in-front of obstacle 'A'. However, in the next time-step the planner realizes that passing in front of obstacle 'A' will cause a collision. Therefore, the path is dynamically updated accordingly. While traversing towards the goal, robot 'R' is next intercepted by obstacle 'B', and in this case the planner initially determines its optimal path by adopting a path to the right of the obstacle, as shown in Figure 11 (c). As shown in Figure 11 (d and e), the robot successfully evades obstacle 'B'.

Figure 12 (a) shows a case where a robot has to reach a goal location while avoiding three closely-moving dynamic obstacles in a narrow hallway. Figure 12 (b) shows how a CA-based path planner avoids obstacle 'A'. After avoiding obstacle 'A', robot 'R' immediately encounters obstacle 'B', which is about to have a head-on collision with robot 'R', as shown in Figure 12 (c). The head-on collision is successfully avoided by moving to the right, as shown in Figure 12 (d). However, as a result of this avoidance manoeuvre robot 'R' comes directly into the path of obstacle 'C'. Therefore, robot 'R' initiates another avoidance manoeuvre to reach its goal, as shown in Figure 12 (e).

Another real-world scenario is where the target is non static. Figure 13 shows an example in which the target is moving in a sinusoidal manner. Figure 13 (a and b) show how the pursuer robot 'R' initially tries to catch the target 'T'. As long as the target is moving towards the right, the pursuer also traverses to the right (Figure 13 (c) and Figure 13 (d)). As soon as the target changes direction, robot 'R' also changes direction, closing the distance and ultimately catching the target, as shown in Figure 13 (e).

7. Experiments

To validate the performance of the proposed planner, it was tested in both static and dynamic environments. As presented in the experimental results here, the CA-based planner proved to be a real-time algorithm in both static and dynamic environments. Numerous experiments were conducted to verify the efficacy of the proposed method, both for static and dynamic environments. Two of these experiments are presented here for discussion.

Components	characteristics
Two mobile robots	one wirelessly-controlled P3AT. The dynamic obstacle is indigenously developed and preprogrammed
PC	Host computer with a wireless module
Microsoft Kinect	resolution: 640 x 480 and distance from floor: 3 m

Table 2. Experimental hardware

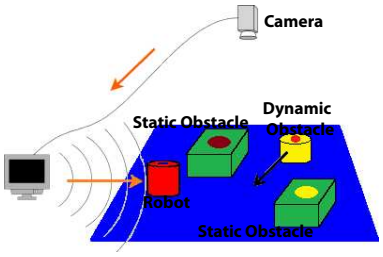


Figure 14. Experimental setup.

The hardware utilized in the experimental work is provided in Table 2. The software, running on an i3 2.26 GHz processor PC, is constitutive of three processes: image acquisition and processing, path-planning and robot control. Microsoft Kinect captures the workspace activity and transfers it to PC. The vision algorithm determines objects of interest and, based upon this information, CA plans a path. Control parameters are generated based upon the path way points.

A number of experiments were conducted for both static and dynamic obstacles. However, only a small subset of these experiments (one for each case) is discussed in the following paragraphs. The complexity of the experiments discussed here is given in Table 3. The experiments were conducted using a P3AT mobile robot autonomously controlled via wireless using PlayerStage (as shown in Figure 14). Each experiment was repeated three times under identical conditions and also compared with the simulated robot behaviour for that particular scenario.

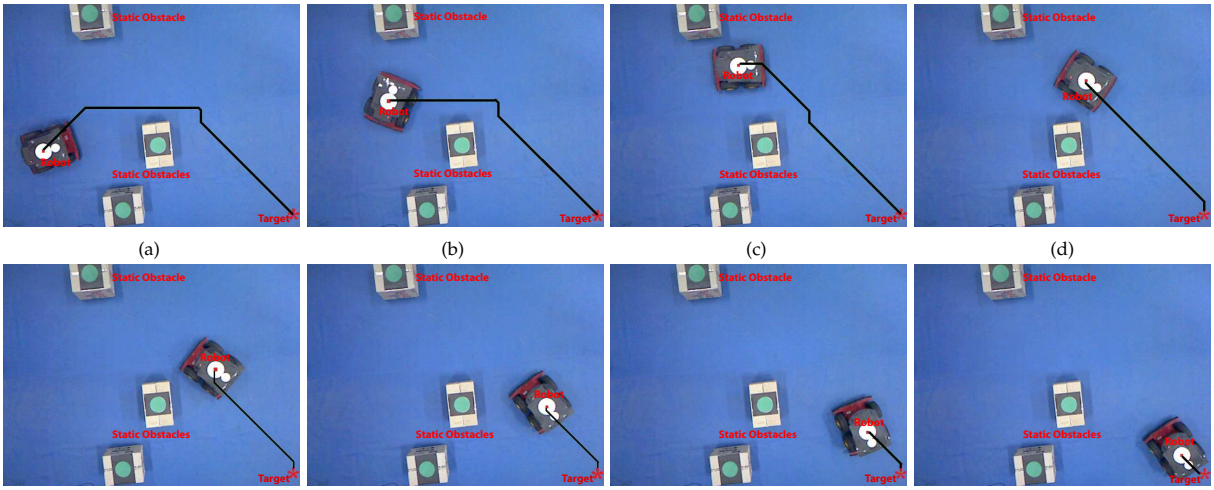


Figure 15. Experimental results of path-planning with CA in a static environment

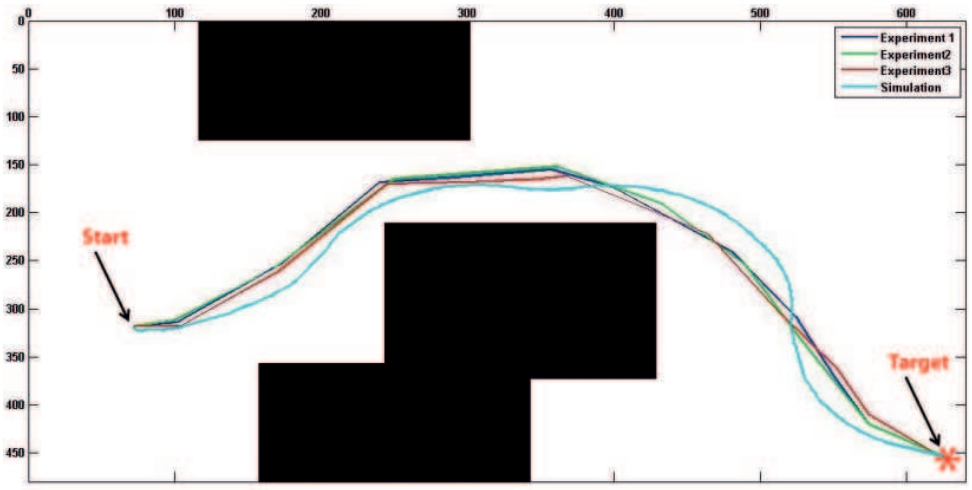


Figure 16. Path followed by the robot in a static environmental setting

Sr.	complexity	max. speed (m/s)	max. turn-rate (degrees)	obstacle speed (m/s)
1	Three static obstacles	0.3	25	-
2	Two static obstacles and one dynamic obstacle	0.25	20	0.05

Table 3. Path-planning experiments

7.1. Static Environment

In the static case, the robot encounters three obstacles. It has to avoid two obstacles in its path en route to the target while keeping itself sufficiently far away from the third obstacle. Initially, a path is planned taking the current robot and obstacle locations into account. Although the control commands are generated in accordance with the desired trajectory, shortly after the commands are executed the robot deviates from its desired path due to slippage and other inherent uncertainties. In order to keep the robot on the optimal track, sensor measurements are acquired and replanning is carried out. This planning and execution cycle continues until the robot successfully reaches its destination while avoiding the obstacles (Figure 15).

Figure 16 presents a comparison of the paths taken by the robot for three test runs conducted under identical conditions. Simulation studies were also conducted to compare the path obtained in the experiments with the path obtained in simulations. Figure 16 shows that the path taken by the robot, in presence of obstacles, expanded in the workspace using Minkowski's sum, closely matches the simulated path of the robot.

7.2. Dynamic Experiment

The proposed path planner has also been successfully applied to dynamic path-planning scenarios. In each experiment, the planner succeeded in planning collision-free global paths while avoiding every dynamic obstacle present in the environment. Figure 17 shows one of the many experiments that we had conducted in which the robot successfully evaded a dynamic obstacle

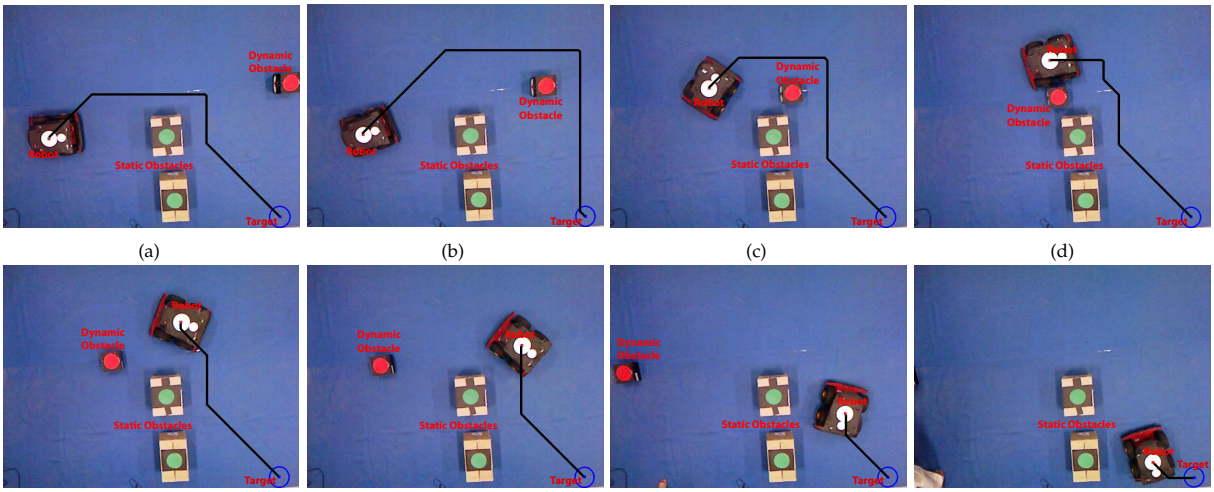


Figure 17. Experimental results of path-planning with CA in a dynamic environment

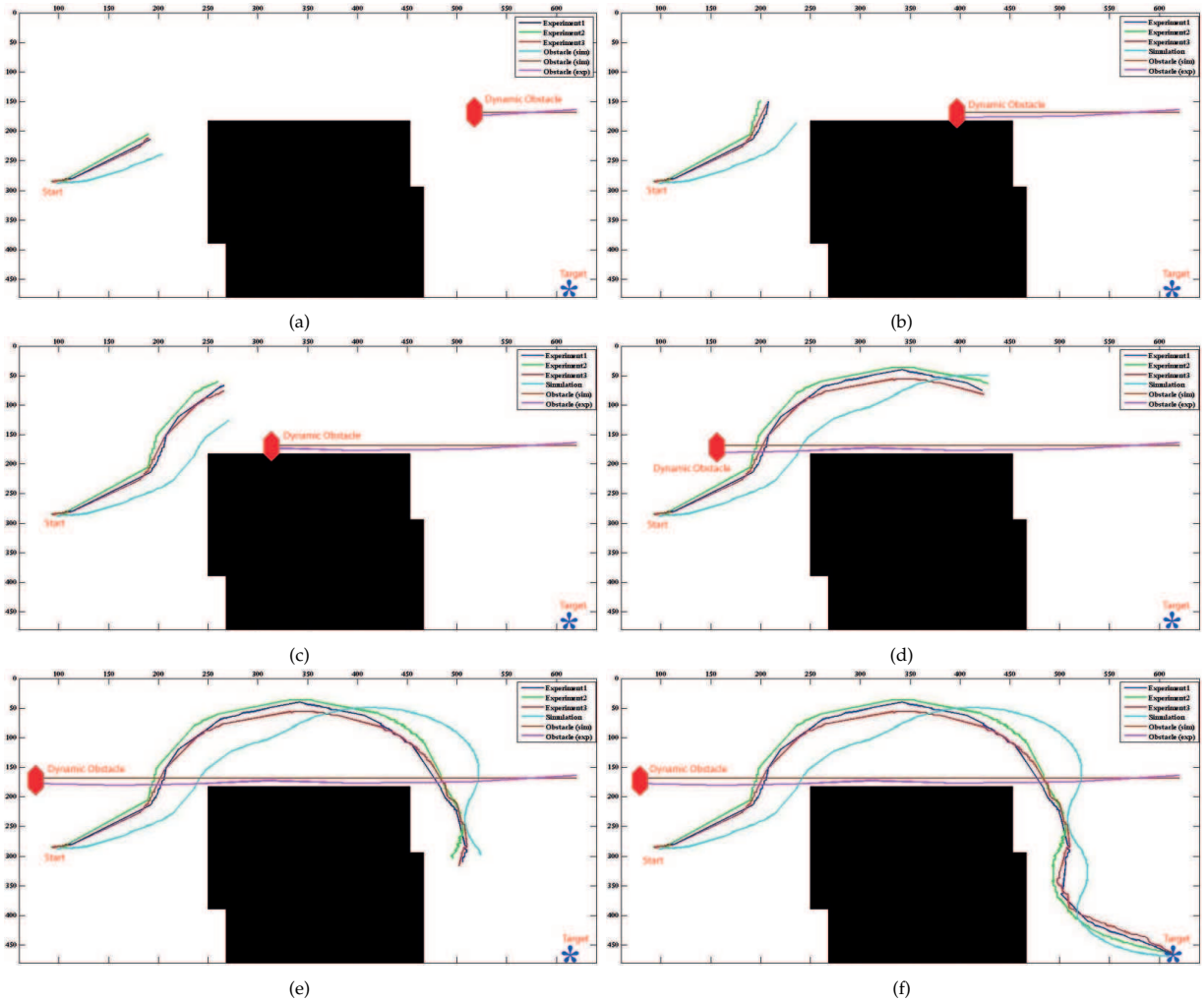


Figure 18. Experimental comparison of the paths adopted by the robot in a dynamic environment

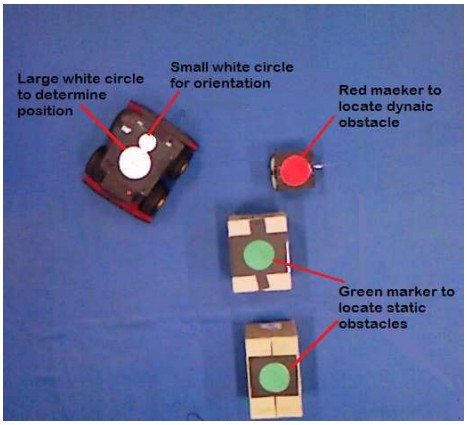


Figure 19. Markers used for object detection

and two static obstacles. Figure 18 presents the path taken by the robot for a dynamic case at different instants of time in reaching its goal. In this case, the obstacle in the workspace has been expanded using Minkowski's sum. Here, it is assumed that the manoeuvrability and speed of the robot is much higher than the obstacle, enabling the robot to execute an effective avoidance manoeuvre before any collision can take place. The experiment shows the ability of the algorithm to dynamically update its path in the next time step in which an obstacle enters the planned path between the robot and the target. Repeated experimentation has proven that our method does not require any special treatment of dynamic obstacles (e.g., the prediction of a moving obstacle's speed or trajectory, etc.) and the repeated dynamic updating of the path is robust enough to deal with dynamic obstacles.

Figure 17 (a) shows the first path returned by the CA planner. Initially, the dynamic obstacle is not in close proximity to the robot and there is free space between the static and dynamic obstacles. Therefore, CA plans a path for evading the two static obstacles that is sufficiently far away from the dynamic obstacle. The robot follows this planned path from its start location until it reaches the position shown in Figure 18 (a). When the robot reaches this location, it recalculates and alters its previous path due to the motion of the dynamic obstacle, as shown in Figure 17 (b). At this point, the dynamic obstacle is very close to the static obstacle. The previously calculated path becomes infeasible, since there is no space between the static and the dynamic obstacle. A new path is calculated that is much longer than the previous path. In the next time-step, the dynamic obstacle proceeds closer towards the robot, causing imminent collision; therefore, a new path is calculated. In the next time-step (Figure 18 (c)), the dynamic obstacle is closest to the robot, and CA again determines the path - this time, a much shorter path is returned since the obstacle has moved from its previous location. In the next interval (Figure 18 (d)), the dynamic obstacle has been avoided and the planner re-plans a path that takes it away from the static obstacle (Figure 17 (e)). The robot executes this path and reaches the location shown in Figure 18(e). While navigating from the location as presented Figure 17 (f) to Figure 17 (g), the robot moves much closer to the static obstacle due to robot momentum and slippage issues. In order to avoid the collision of the robot with the static obstacle (Figure 18 (e)), it re-plans its path (Figure 17 (g)) and finally reaches its target (Figure 18 (f)).

This experiment is repeated three times and compared with the simulation results. In this case, the experimental results vary slightly from the simulation results. This is mainly because the robot's dynamic and update rate is much faster in the case of the simulation as compared with the experimental case.

8. Conclusion

The paper presents a CA-based real-time path planner that always results in an optimal path. A CA is coupled with a parent-child relationship for each cell to achieve improved and real-time performance. PlayerStage simulations and experiments are conducted to validate the real-time behaviour of the proposed scheme. The results prove that it outperforms previous path-planning algorithms in light of optimality and time efficiency, as shown in comparison with A*, Dijkstra, D* and MPCNN.

9. Acknowledgements

The authors would like to thank the Robotics and Artificial Intelligence Department, NUST, for their support in providing us with the facilities for hardware implementation. In particular, the authors would like to mention the help, guidance and support extended by Mr. Muhammad Ali Ramay and Dr. Yasar Ayaz.

10. References

[1] T. Kito, J. Ota, R. Katsuki, T. Mizuta, T. Arai, T. Ueyama, and T. Nishiyama. Smooth path planning by using visibility graph-like method. In *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, volume 3, pages 3770 – 3775 vol.3, sept. 2003.

[2] Han-Pang Huang and Shu-Yun Chung. Dynamic visibility graph for path planning. In *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2813 – 2818 vol.3, sept.-2 oct. 2004.

[3] O. Takahashi and R.J. Schilling. Motion planning in a plane using generalized voronoi diagrams. *Robotics and Automation, IEEE Transactions on*, 5(2):143 –150, apr 1989.

[4] F. Lingelbach. Path planning using probabilistic cell decomposition. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 1, pages 467 – 472 Vol.1, april-1 may 2004.

[5] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, volume 2, pages 500 – 505, mar 1985.

[6] Y. Koren and J. Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 1398 –1404 vol.2, apr 1991.

[7] L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566 –580, aug 1996.

[8] Steven M. Lavalle, James J. Kuffner, and Jr. Rapidly-exploring random trees: Progress and prospects. In *Algorithmic and Computational Robotics: New Directions*, pages 293–308, 2000.

[9] Jr. Kuffner, J.J. and S.M. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, volume 2, pages 995 –1001 vol.2, 2000.

[10] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[11] S. Karaman and E. Frazzoli. Optimal kinodynamic motion planning using incremental sampling-based methods. In *Decision and Control (CDC), 2010 49th IEEE Conference on*, pages 7681 –7687, dec. 2010.

[12] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.

[13] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press and McGraw-Hill, second edition, 2001.

[14] Anthony Stentz. Optimal and efficient path planning for partially-known environments. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3310–3317, 1994.

[15] Konstantinos Charalampous, Ioannis Kostavelis, Dimitrios Chrysostomou, Angelos Amanatiadis, and Antonios Gasteratos. 3d maps registration and path planning for autonomous robot navigation. *CoRR*, abs/1312.2822, 2013.

[16] Jianping Tu and S.X. Yang. Genetic algorithm based path planning for a mobile robot. In *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, volume 1, pages 1221 – 1226 vol.1, sept. 2003.

[17] Woong-Gie Han, Seung-Min Baek, and Tae-Yong Kuc. Genetic algorithm based path planning and dynamic obstacle avoidance of mobile robots. In *Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997 IEEE International Conference on*, volume 3, pages 2747 –2751 vol.3, oct 1997.

[18] Xiaoyu Yang, M. Moallem, and R.V. Patel. A layered goal-oriented fuzzy motion planning strategy for mobile robot navigation. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 35(6):1214 –1224, dec. 2005.

[19] Yanrong Hu and S.X. Yang. A knowledge based genetic algorithm for path planning of a mobile robot. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 5, pages 4350 – 4355 Vol.5, april-1 may 2004.

[20] Yongmin Zhong, B. Shirinzadeh, and Yanling Tian. A new neural network for robot path planning. In *Advanced Intelligent Mechatronics, 2008. AIM 2008. IEEE/ASME International Conference on*, pages 1361 –1366, july 2008.

[21] Hong Qu, S.X. Yang, A.R. Willms, and Zhang Yi. Real-time robot path planning based on a modified pulse-coupled neural network model. *Neural Networks, IEEE Transactions on*, 20(11):1724 –1739, nov. 2009.

[22] Stephen Wolfram. Statistical mechanics of cellular automata. *Reviews of Modern Physics*, 55(3):601–644, July 1983.

[23] F.R. Soofiyani, A.M. Rahmani, and M. Mohsenzadeh. A straight moving path planner for mobile robots in static environments using cellular automata. In *Computational Intelligence, Communication Systems and Networks (CICSyN), 2010 Second International Conference on*, pages 67 –71, july 2010.

[24] Yashar Tavakoli, H. Haj, Seyyed Javadi, and Sepideh Adabi. A Cellular Automata Based Algorithm for Path Planning in Multi-Agent Systems with A Common Goal. 2008.

[25] Ioannis Kostavelis, Evangelos Boukas, Lazaros Nalpantidis, and Antonios Gasteratos. Path tracing on polar depth maps for robot navigation. In GeorgiosCh. Sirakoulis and Stefania Bandini, editors, *Cellular Automata*, volume 7495 of *Lecture Notes in Computer Science*, pages 395–404. Springer Berlin Heidelberg, 2012.

[26] C. Behring, M. Bracho, M. Castro, and J. A. Moreno. An algorithm for robot path planning with cellular automata. In *Proceedings of the Fourth International Conference on Cellular Automata for Research and Industry: Theoretical and Practical Issues on Cellular Automata*, pages 11–19, London, UK, UK, 2000. Springer-Verlag.

[27] P.G. Tzionas, A. Thanailakis, and P.G. Tsalides. Collision-free path planning for a diamond-shaped robot using two-dimensional cellular automata. *Robotics and Automation, IEEE Transactions on*, 13(2):237 –250, apr 1997.

[28] Albert S. Huang, Abraham Bachrach, Peter Henry, Michael Krainin, Daniel Maturana, Dieter Fox, and Nicholas Roy. Visual odometry and mapping for autonomous flight using an rgb-d camera. In *Proceedings of the International Symposium of Robotics Research (ISRR)*, Flagstaff, AZ, 2011.

[29] M. F. Fallon, H. Johannsson, and J. J. Leonard. Efficient scene simulation for robust Monte Carlo localization using an RGB-D camera. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, St. Paul, MN, May 2012. To appear.

[30] Eduard Oks and Micha Sharir. Minkowski sums of monotone and general simple polygons. *Discrete and Computational Geometry*, 35:223–240, 2006.

[31] Subhrajit Bhattacharya. Yagsbpl: A template-based c++ library for large-scale graph search and planning, 2013. Available at http://subhrajit.net/index.php?WPage=yagsbpl Accessed on 10 Dec 2013.

[32] Liang Dai. Fast shortest path algorithm for road network and implementation. Technical report,

Carleton University School of Computer Science, 2005.
[33] Michael Henderson and Valentin Koch. P2p shortest path problem and beyond. Technical report, University of British Columbia, April 2006.
[34] Andre K. Prajogo Leo Willyanto Santoso, Alexander Setiawan. Performance analysis of dijkstra, a* and ant algorithm for finding optimal path case study: Surabaya citymap. International Conference, Medical Image Computing, 2010.
[35] Brian P. Gerkey, Richard T. Vaughan, and Andrew Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *In Proceedings of the 11th International Conference on Advanced Robotics*, pages 317–323, 2003.
[36] L.G. Shapiro and G.C. Stockman. *Computer visión*. Prentice Hall, 2001.

Appendix - Visual Identification of Objects

A vision-based object identification system repeatedly makes use of the Hough circle transform to detect the markers installed on different obstacles and the robot. In our experiments, the robot bears two white circles of different radii - one locates the robot's centre and the other determines its orientation. Red circles are used for dynamic obstacles while green circles denote static obstacles, as shown in Figure 19.

In order to identify the location of different objects, a Hough Circle transform [36] is applied. The result of this operation is the identification of different makers installed in the workspace. In the next step, the centres of all the circles are visited to differentiate between various objects. At the centre of all the objects, a 3x3 region is sampled and its average red, green, blue and average greyscale values are calculated. Based upon the dominating colour, each circle is associated with an object. This information and the obstacles' sizes are then used to generate the workspace of the robot.

A special case holds for the white robot circle. The algorithm initially identifies the larger white circle and subsequently searches for the smaller white circle. This search is performed by sampling at a specified radius with the origin at the centre of the larger circle. A 3x3 path is sampled at increments of 15 degrees in a circular fashion. The average greyscale values at each sample point are determined to identify the white patch. Once this patch is identified we consider its neighbouring region as the "region of interest" for finding the center of the smaller circle. To locate the center of the smaller circle Hough Circle Transform is applied to the region of interest. Once the center of smaller circle is found, the line joining the center of the two circles gives the orientation of the robot in global frame.