ITC 2/48
Journal of Information Technology and Control
Vol. 48 / No. 2 / 2019
pp. 179-194
DOI 10.5755/j01.itc.48.2.2139

Optimized RRT-A* Path Planning Method for Mobile Robots in Partially Known Environment

Received 2018/08/05 | Accepted after revision 2019/04/06

http://dx.doi.org/10.5755/j01.itc.48.2.21390

Optimized RRT-A* Path Planning Method for Mobile Robots in Partially Known Environment

Ben Beklisi Kwame Ayawli
College of Electrical Engineering and Control Science, Nanjing Tech University, China.
Computer Science Department, Sunyani Technical University, Sunyani, Ghana, email: bbkayawli@yahoo.com

Xue Mei, Mouquan Shen
College of Electrical Engineering and Control Science, Nanjing Tech University, China

Albert Yaw Appiah, Frimpong Kyeremeh
College of Electrical Engineering and Control Science, Nanjing Tech University, China.
Electrical and Electronic Engineering Department, Sunyani Technical University, Sunyani, Ghana

Corresponding author: seraph_mx@163.com

This paper presents an optimized rapidly exploring random tree A* (ORRT-A*) method to improve the performance of RRT-A* method to compute safe and optimal path with low time complexity for mobile robots in partially known complex environments. ORRT-A* method combines morphological dilation, goal-biased RRT, A* and cubic spline algorithms. Goal-biased RRT is modified by introducing additional step-size to speed up the generation of the tree towards the goal after which A* is applied to obtain the shortest path. Morphological dilation technique is used to provide safety for the robots while cubic spline interpolation is used to smoothen the path for easy navigation. Results indicate that ORRT-A* method demonstrates improved path quality compared to goal-biased RRT and RRT-A* methods. ORRT-A* is, therefore, a promising method in achieving autonomous ground vehicle navigation in partially known environments.

KEYWORDS: Rapidly exploring random tree (RRT); mobile robots; path planning; morphological dilation; autonomous ground vehicles.

1. Introduction

In recent years, mobile robot path planning research has been an active research area gaining considerable attention [6, 26, 35]. Despite the immense research in the area, achieving complete autonomous navigation in complex environment remains a challenge [2, 17]. Roadmap path planning methods are gaining popular-

ity in addressing mobile robot path planning problems [20]. Notable among these methods include probabilistic roadmap (PRM) [33], voronoi diagram (VD) [4, 5] and rapidly exploring random tree (RRT) path planning methods [1, 7, 9, 13, 14, 24, 25, 34, 37, 39]. Consideration is given to RRT path planning in this paper.

RRT is a single query incremental algorithm [22] that generates a tree in a free configuration space until the defined target is reached [7, 18]. RRT is described as a fast path planning method that performs well in complex and high dimensional workspace [14, 39]. RRT is a good method for motion planning for mobile robots because of its strength in controlling inputs computation [13]. It seeks to find a path between an initial state and target in a free configuration space without the representation of the entire environment. Recently, mobile robot path planning research using RRT and its variants has been widely considered. RRT trajectory planning method for car-like robots was proposed in [13], where uniform and goal-biased sampling technique was employed to generate the tree. Consideration was given to controlling input selection for the trajectory planning in urban, office and landscape environments. Generally, the major challenge of the RRT method is the inability to control path quality [19, 21, 37]. However, no consideration was given by the authors to address the path quality problem. Although RRT is probabilistic complete, generated path is described to be far from optimal, because it tries to ensure computational efficiency at the expense of optimality [7, 37].

Intense study to improve the performance of RRT path planning method has been carried out over the years. RRT* algorithm, an extension of RRT was proposed in [19] which tried to improve path quality by computing asymptotically sub-optimal path. The tree structure of RRT* is flexible making it easy to improve path quality. Even though the RRT* ensures asymptotic optimality [20] and performs better than RRT in terms of path optimality, it has high time complexity due to the continuous execution of the local planner during the generation of the tree [7]. It is reported to be difficult to perform dynamic path replanning using RRT* [39]. Islam et al. [16] took advantage of the asymptotic optimality provision by RRT* to present RRT*-smart approach to increase the convergence rate of RRT method to improve path quality with low time complexity. To address slow convergence rate and large sampling space requirement of RRT*, RRT*-adjustable bounds approach was presented in [27] for off-line path planning in cluttered environment. In [31], a triangular geometrized RRT* (TG-RRT*) method was presented to address the high processing time and iterations requirement of RRT* to obtain optimal result. The geometrical center of the initial state, the target position and the sample for the RRT were considered in generating the tree. Compared to RRT*, results were described to be better in terms of efficiency. The TG-RRT* method was meant for static environment, and the validation of the method was done in a less complex environment. Human-guided RRT* motion planning approach was proposed in [11] with the aim of reducing path length and the cost of planning the path of a quadrotor. The approach requires a user to guide the motion planning instead of it performing planning autonomously. Park and Kuipers [30] considered handling non-holonomic constraints of mobile robot's motion planning and presented a method that used a non-holonomic distance function to extend RRT* for unicycle-based robots. The purpose was to obtain feasible and smooth path. The method was validated using simulation in static environment with sparse obstacles. No comparison was made to any existing method to determine its efficiency. In [36], parallel RRT* motion planning architecture design was proposed. Nodes for the tree were kept in distinct pipelines to prevent traversing of all nodes involved in the iteration to generate the tree. Implementation was done on field programmable gate arrays (FPGA), and results were described to be better compared to software RRT*. Instead of obtaining a path between two defined positions, Correia et al. [9] presented RRT-Edge method aimed at obtaining a compact representation of the environment using dispersion of RRT nodes where only the goal position is defined. The main purpose of the RRT-Edge method was to reduce memory requirement. To enhance the expanding speed of the tree generation, Li et al. [25] proposed Liveness-based RRT method for autonomous underwater vehicles (AUVs) motion planning. Although splined-based RRT* method proposed in [37] could find a path according to the authors, it requires the generation of the tree to cover the entire configuration space. This results in generation of large number of nodes; hence, increasing the time and space complexity of the method. Recently, Otte and Frazzoli [28] proposed RRT^x algorithm

which performs continuous path update during the navigation of the robot. Despite research progress to minimize the probability of RRT path generation and to enhance path construction, difficulty of controlling path quality still exists [28, 37].

In recent years, RRT method has been combined with other path planning methods to help improve path quality. In [38], Gaussian process occupancy map was combined with RRT to plan secure path in cluttered environment. Tusi and Chung [34] combined RRT and artificial bee colony (ABC) methods. RRT algorithm was used in this method to generate nodes in the free configuration space and the best nodes were considered by applying ABC method to move the bees. The method was described to have performed better compared to particle swarm optimization (PSO) method. Any-angle search algorithm was combined with RRT to present theta*-RRT motion planning method for non-holonomic wheeled robots [29]. Theta*-RRT method was described to have generated shorter paths in shorter time compared to other RRT methods. To obtain near-optimal path of nonlinear dynamic mobile robots, Li et al. [23] combined neural network and RRT to present NoD-RRT method. The authors considered the nonlinear kinodynamic constraints of the vehicles and revised RRT to perform reconstruction to deal with the kinodynamic constraints problem. Results showed that NoD-RRT performed better compared to RRT and RRT*. Another recent hybrid method involving RRT is RRT-A* motion planning method presented in [24]. The method focused on optimizing RRT path generation for non-holonomic mobile robots in known environment. A* heuristic algorithm was used to decide the selection of the nearest node during the generation of the tree. The authors acknowledged the fact that the method suffered local minima problem. The authors indicated that the path length obtained using RRT-A* method was more optimal compared to goal-biased RRT method. This analysis, however, compared Manhattan based RRT-A* to Euclidean based goal-biased RRT which are of different metric functions as demonstrated in the paper. Comparing Euclidean-based RRT-A* to Euclidean-based goal-biased RRT from their results, the proposed method utilized 849% of the time to achieve 6% path length improvement in sparse environment. In dense environment, 222% time was used to achieve 4.27% path length improvement over Euclidean-based goal biased RRT. But the rate of time used compared to achieved results indicates the method lacks some efficiency. The path quality problem of RRT and challenges identified in [24] motivated this research to optimize and extend RRT-A* method to produce better results.

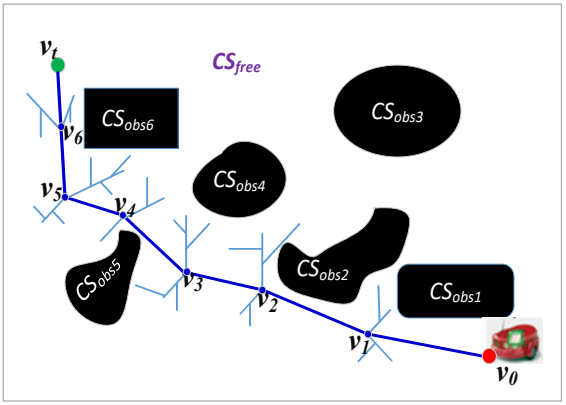
Therefore, this paper presents an optimized RRT-A* path planning method based on morphological dilation (MD), goal-biased RRT, A* heuristic algorithm and cubic spline interpolation to compute safe and optimal path for autonomous mobile robots in partially known complex environment. A step size is usually used in goal-biased RRT for the generation of the RRT. In this paper, additional step-size is introduced to speed up the generation of the tree towards the goal based on the random sample value. In [24], A* heuristic function was applied at every iteration to select the nearest node during the generation of the tree which had high time cost in path computation. In this paper, the A* heuristic algorithm is used to optimize and obtain the shortest path after the path is generated using the modified goal-biased RRT. To provide safe and smooth path for feasible navigation, MD technique is used to inflate the obstacles before generating the path, and cubic spline interpolation (CSI) is used to smoothen the path. Path replanning is provided by generating new path from a current position of the robot when a random obstacle obstructs its navigation path. ORRT-A* approach addresses the local minima problem reported with RRT-A* [24] and generates safe and optimal path with low time cost in partially known environment.

The rest of this paper is organized as follows. Problem formulation is given in Section 2. Section 3 describes the proposed methods: obtaining and processing map, computing the roadmap, path query and optimization, path smoothening, and replanning. The simulation results and discussion are presented in Sections 4 and 5, respectively. Finally, Section 6 concludes the paper.

2. Problem Formulation

The configuration space CS of the mobile robot is represented in Figure 1. The CS is made up of regions with obstacles, CS_{obs} and free configuration space CS_{free}. In this paper, environment of 2D where $CS \subseteq \mathbb{R}^2$ is considered. Hence, $CS_{obs} \subseteq CS$ and each vertex of CS_{obs} can be represented as $v_{obs(i)} \in C_{obs}$. The CS_{free} can be obtained using $CS_{free} = CS \setminus CS_{obs}$

Figure 1
The configuration space of a robot with sample obstacles and RRT path from the initial position v_0 to the target v_t



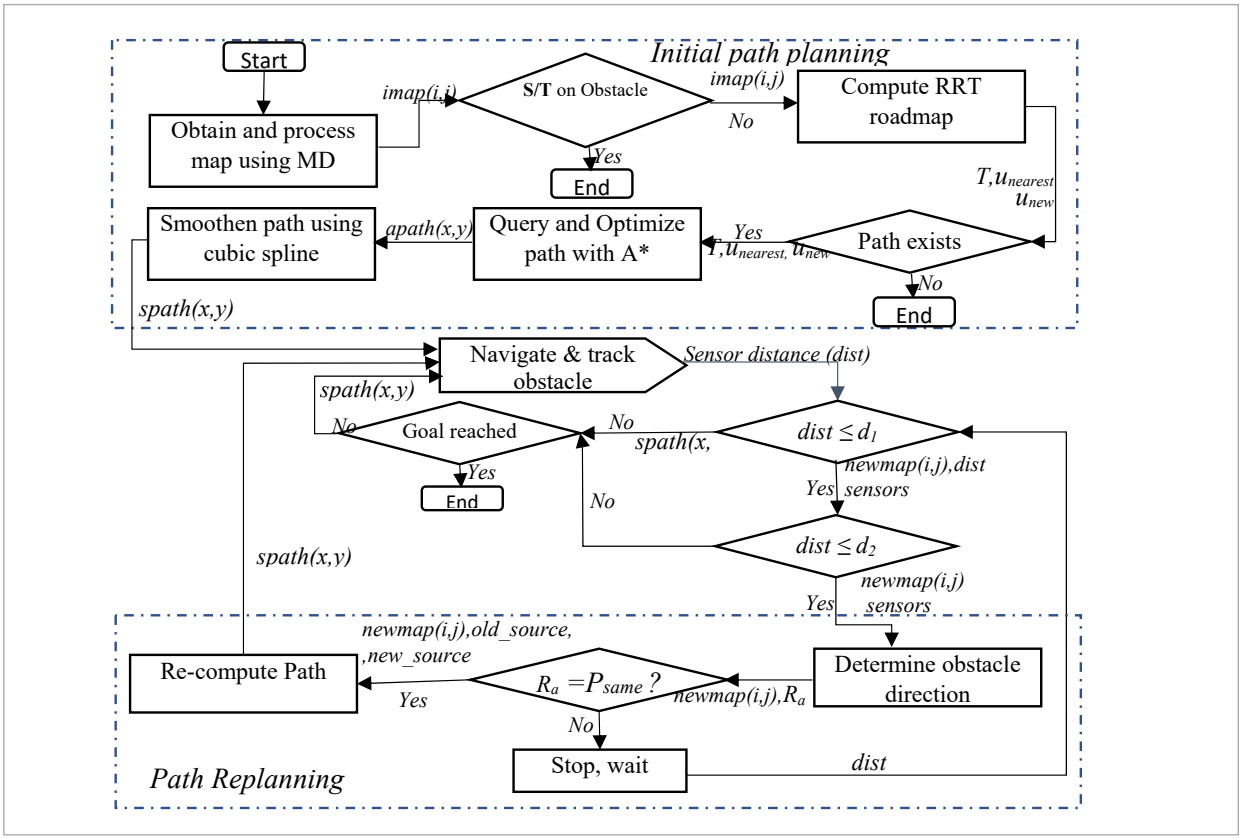
and each vertex of CS_{free} is given as $v_{free(i)} \in CS_{free}$. The RRT is generated in the CS_{free} such that collision with CS_{obs} is avoided. The generation of the tree starts at the initial position of the vehicle and expands towards the target. Given the vertices of the tree from the initial state to the target as $V_i = (v_0, v_1, v_2, v_3, ..., v_t)$, V_i represents positions in the CS_{free}. v_0 is the initial position of the tree while v_t represents the target or the iteration limit of the tree generation.

This paper focuses on computing safe, shortest and smooth path in CS_{free} from v_0 to v_t using less time in partially known environment. At the initial state, the environment of the vehicle is assumed known during path computation. As the vehicle navigates to its target based on the computed path, random obstacles can obstruct the path leading to collision. This paper provides a reactive path replanning to avoid obstacles that obstruct the path of the vehicle during navigation in real-time.

3. Proposed Method

The general algorithms of the method presented in this paper are demonstrated in a workflow diagram in Figure 2. The path planning method presented in this

Figure 2
Workflow diagram of the proposed method



paper has been divided into five steps: obtaining and processing map, computing the RRT roadmap, path query and optimization, path smoothening and path replanning.

3.1. Step 1: Obtaining and Processing Map

The map of the workspace could be obtained using camera or scanning laser range finder. The captured map of the environment is processed to give binary representations of the CS_{obs} and the CS_{free} of the *CS*. To ensure safety of the mobile robots during navigation such that the generated path is not too close to obstacles in the workspace to cause collisions, obstacles in the map are inflated before constructing the RRT roadmap using MD technique given in [12]. The dimension of the vehicle and other safety requirements are considered in inflating the obstacles. The MD equation denoted by $P \oplus Q$ is given as:

$$P \oplus Q = \{r \mid (Q_i)_r \cap P \neq \varnothing\}, \quad (1)$$

where *P* represents the map, *Q* represents the structure element for the dilation, *r* is the set of all nodes of the map and Q_i denotes the reflection of the set *Q*. The MD of a given binary map with the matrix, $bmap(x, y)$ and a structuring element $q(x, y)$ can therefore, be calculated using Equation (2):

$$(bmap \oplus q)R = \max\{bmap(R-x) + q(x) \mid (R-x), \in S_{bmap} \,\&\, \in S_q\}, \quad (2)$$

where *S* and *R* denote the dimension of the map and the translated radius representing the robot's safety space required, respectively. Equation (2) is used to inflate the obstacles on the map before generating the tree to ensure safe navigation.

3.2. Step 2: Computing the RRT Roadmap

Details of basic RRT algorithm are given in [1, 7, 13, 24, 25, 34, 37, 39]. This paper modifies and uses goal-biased RRT to generate the roadmap ensuring that the tree is rapidly generated towards the goal. The generation of the tree is biased towards the target when the random value *rand* to determine the computation of the sample is less than the random threshold *rt*. Two different step sizes, *sz1* and *sz2* are used with *sz2* assigned higher value than *sz1*. *sz2* is used for the growth of the tree when the growth is biased to the target. This is aimed at reducing the number of nodes to be generated before reaching the target, thereby reducing the time and space complexity of the algorithm. The algorithm used in this paper to generate the roadmap is given in Algorithm 1.

Algorithm 1
Computing the roadmap

Input: Step sizes *sz1, sz2,* Distance threshold *dth*
Attempts allowed *faa,* Attempts $f_{attempts,}$
Initial state *source*, rand threshold *rt, map, target*
Output: Tree *T,* $u_{nearest}$, u_{new}

```
1.  Begin
2.   Initialize: 0 < rt < 1, sz1, sz2, dth, faa, source, target, f_attempts = 0
3.   imap ← (map(i, j) ⊕ map(i, j)_obs)R //Equation (2)
4.   T ← (source, ∅);
5.   while f_attempts < faa
6.     sample(rand, imap, target); // Equation (3)
7.     u_rand ← sample;
8.     u_nearest ← Nearest(T, u_rand); // Equation (4)
9.     u_new(u_nearest, sz1, sz2, θ); // Equation (6)
10.    if ~CS_obs(u_new, imap)
11        if || u_new, target ||< dth
12          return pathfound = TRUE; break;
13        end if
14        dist_min ← min(|| T, u_new ||, ∅);
15        if || u_new, T(dist_min) ||< dth
16          f_attempts ← f_attempts + 1; continue;
17        end if
18        f_attempts ← 0;
19        T ← add _ node(T, u_new)
20     end if
21     else
22       f_attempts ← f_attempts + 1; continue;
23     end else
24     if f_attempts ≥ f_aa
25       return pathfound = FALSE; break;
26     end if
27   end while
```

The initial position of the robot represents the root of the tree, $T \leftarrow (source, \varnothing)$. The sample for generating the tree is computed as:

$$sample(rand, imap, target) = \begin{cases} rand(0,1) * size(imap) & if\ rand < rt. \\ target & otherwise \end{cases} \quad (3)$$

The random value *rand* generated is compared to the random threshold *rt*. If *rand* < *rt*, a random sample is computed as given in Equation (3). The target is taken as the sample when *rand* ≥ *rt* to bias the growth

of the tree towards the target. The nearest node, $Nearest(T, u_{rand})$ is computed using k-nearest neighbor (KNN) given as:

$$Nearest(T, u_{rand}) = \min\left(\sqrt{\sum_{i=1}^{k}(T_{(i)} - u_{rand(i)})^2}\right), \quad (4)$$

where $u_{rand} = sample$ as given in Equation (3). The direction θ to extend the sample to create new node is computed using Equation (5):

$$\theta = atan2(u_{rand} - u_{nearest}), \quad (5)$$

where $u_{nearest} = Nearest(T, u_{rand})$ as given in Equation (4). The position of the new node u_{new} is calculated using Equation (6):

$$u_{new}(u_{nearest}, sz1, sz2, \theta) = \begin{cases} u_{nearest} + sz2 * [\sin\theta \quad \cos\theta], & if\ u_{rand} = target \\ u_{nearest} + sz1 * [\sin\theta \quad \cos\theta], & otherwise \end{cases}, \quad (6)$$

where *sz1* and *sz2* denote the step size 1 and 2, respectively. If $u_{rand} = target$, *sz2* is used to compute the position of the new node u_{new} towards the target with a higher step size. This is to facilitate the growth of the tree towards the target at a faster rate with reduced number of nodes. Collision checks are performed to ensure that the tree is generated within the CS_{free}. The growth of the tree ends once the target point is reached, or the iteration attempts $f_{attempts}$ exceed the failed attempts allowed f_{aa}.

3.3. Step 3: Path Query and Optimization

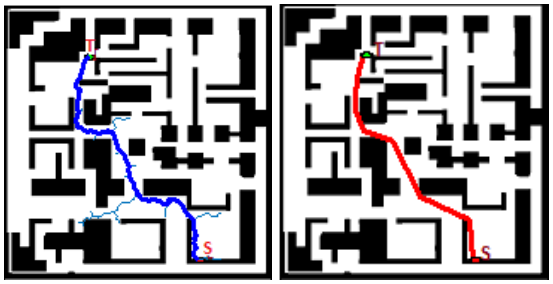
A path is generated from the initial position of the RRT roadmap to the target using the set of $u_{nearest}$ and the T nodes obtained during the generation of the tree. The algorithm used in generating the path is given in Algorithm 2. A sample path generated using Algorithm 2 is demonstrated in Figure 3a. A* algorithm applied in [24] was used to optimize the path in Figure 3a to obtain the path in Figure 3b. In [24], the A* heuristic function was used at every iteration to select the nearest node during the generation of the tree. In this paper, the A* heuristic algorithm is used to optimize and obtain the shortest path after the roadmap is generated using the modified goal-biased RRT. A* heuristic function is good at finding a path on a graph with the least cost provided a path

Algorithm 2
Path query and optimization algorithm

Input: $u_{nearest}$, T, *target* from Algorithm 1
Output: *apath*

```
Begin
  p ← target
  for j ← 1 to size(u_nearest)
      previous ← u_nearest(j)
      while j > 0
          p ← (T(previous); p)
      end while
  end for
  apath ← f(p)   // Equation (7)
  spath ← P(apath)  // Equation (11)
End
```

Figure 3
Generated path on a map with inflated obstacles:(a) A generated path using modified goal-biased RRT; (b) An optimized path using A* algorithm



(a) (b)

exists [32]. The A* heuristic cost function employed is represented as:

$$f(p) = g(p) + h(p), \quad (7)$$

where $g(p) = c(s, p)$ is the path cost c from the initial position s to node p, $h(p)$ is the heuristic component of the cost function which estimates the cheapest cost from node p to the target. $h(p)$ is computed using Euclidean distance metric as:

$$h(p) = \sqrt{(x_p - x_{target})^2 + (y_p - y_{target})^2}, \quad (8)$$

where $p = (p_0, p_1, p_2, ..., p_n)$ represents points on the path.

A* is considered because it is both complete and optimal. It is complete as once a path exists in the CS_{free}, A* can find the path. A* is admissible and consistent. Admissibility and consistency are properties of optimality. With *g(p)* being the actual cost to get to point *p*, *f(p)* would therefore not overestimate the cost to reach the target. This makes A* admissible. Considering *c(p, p')* as the cost from point *p* to *p'*, consistency is achieved since $h(p) \le c(p, p') + h(p')$ and admissibility is achieved since for all arcs of the path, $c(p, p') \ge \varepsilon > 0$.

3.4. Step 4: Path Smoothening

The path obtained after applying A* algorithm (see Figure 3b) is not smooth enough to enable easy navigation of the autonomous vehicle. CSI is employed to enhance the smoothness of the optimized path. With cubic spline, the function of a curve is represented using different cubic functions for each of the data points intervals [3]. Considering m data points, the function of the spline $S(x)$ can be defined as in Equation (9):

$$S(x) = \begin{cases} P_1(x) & x_0 \le x \le x_1 \\ P_i(x) & x_{i-1} \le x \le x_i \\ P_m(x) & x_{m-1} \le x \le x_m \end{cases}, \quad (9)$$

where P_i represents the cubic function. Generally, cubic spline is defined as:

$$P_i(x) = a_i + b_i x + c_i x^2 + d_i x^3, \quad (10)$$

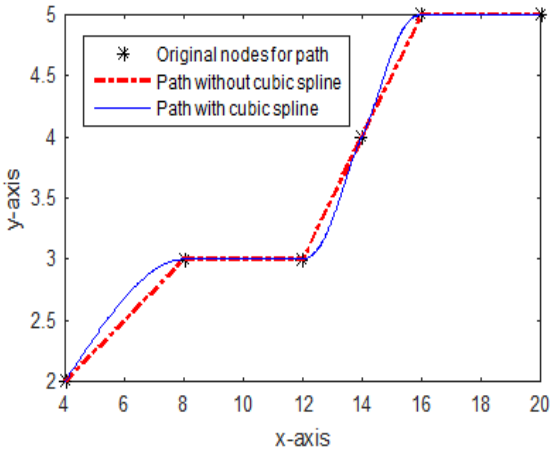
where a_i, b_i, c_i and d_i are the coefficients to be determined for each i. Hence, with m node intervals, we require $4m$ coefficients to be computed. If $x = (x_0, x_1, x_2, ..., x_n)$ represents a set of points on a computed path using A* algorithm, then $a_0 = f(x_0)$, $a_1 = f(x_1), ..., \ a_n = f(x_n)$. The cubic spline $P_{(i)}(x)$ is computed as:

$$P_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3 \quad for \ x_i \le x \le x_{i+1}. \quad (11)$$

Hence, given a set of data points on a path, cubic spline interpolation smoothens the path as demonstrated in Figure 4 by generating more data points between each interval.

Contrary to the claim that CSI poses the challenge of causing collisions [15], the introduction of MD technique to inflate obstacles on the map before computing the roadmap and the path in this paper helps to address the possibility of collision occurrence. Figure 5 presents the result of applying cubic spline to smoothen the generated path in Figure 3b.

Figure 4
Curve smoothening using cubic spline

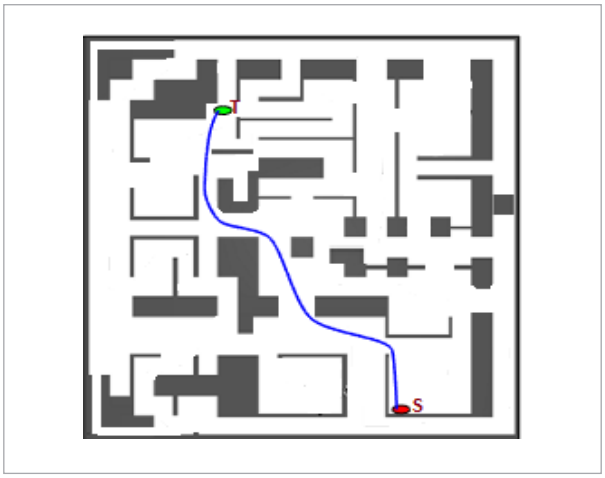


Algorithm 3
Robot Navigation algorithm

Input: *spath, source, target,* distance thresholds d_1, d_2
Output: $navigate_{path}$

```
1.  Begin
2.    old_source ← source;
3.    for j ← 1 to size(spath);
4.      navigate_path ← [spath(j,1), spath(j,2)];
5.      if dist ≤ d_i
6.        Determine obstacle direction, R_a
7.        if R_a == P_cross //Equation (15)
8.          Stop and wait
9.        end if
10.       if R_a == P_same
11.         new_source ← spath(j,:);
12.         old_source ← [old_source, source];
13.         spath = Replan(new_source, old_source, newmap)
14.         Go to 3
15.       end if
16.     end if
17.     if target reached;
18.       return success and stop
19.     end if
20.   end for
21. End
```

Figure 5
Path smoothening applied to the path in Figure 3b on the original map



3.5. Step 5: Path Replanning

During the mobile robot navigation, a random obstacle may obstruct the optimized generated path. The obstacle may be in motion or stationary. If in motion, its direction is required to determine the path replanning action. In [7], the direction of the robot and the obstacles were computed using the angular positions of the robot and that of the obstacle. The difference of these angles is compared to an angle threshold to determine the direction of the obstacle followed by a replanning action. The direction vector of the obstacle θ_{obs}, and the velocity vector of the robot θ_r, were obtained using Equations 12 and 13:

$$\theta_{obs} = atan2((y_{obs} - y_r),(x_{obs} - x_r)) \quad (12)$$

$$\theta_r = atan2(v_y, v_x), \quad (13)$$

where (x_{obs}, y_{obs}) is the position of the obstacle in the environment while (x_r, y_r) is the position of the robot. (v_x, v_y) are the coordinate components of the velocity vector of the robot. The angle difference is compared to an angle threshold using Equation (14):

$$\left|\theta_{obs} - \theta_r\right| < \theta_{threshold}, \quad (14)$$

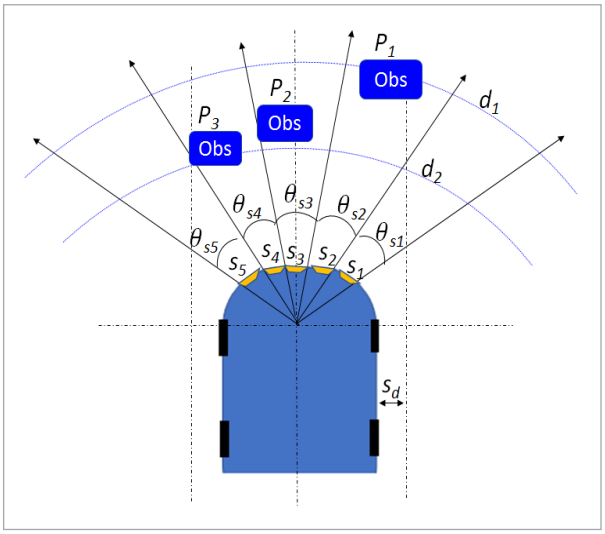
where $\theta_{threshold}$ represents the angle threshold to determine the replanning action. While it is easy to obtain the position of the robot for the computations, it is difficult to obtain the (x, y) position of the obstacle in the real environment for the computations in Equations (12) and (13).

In this paper, a simple sensor-based reactive technique is proposed to determine the direction of an obstacle that obstructs the path of the robot during navigation. Two main conditions of the obstacle are required to take a decision: (1) whether the obstacle in motion is crossing the path of the robot, P_{cross}; (2) whether the obstacle is stationary, in the same direction towards the robot or moving forward in the same direction of the robot, P_{same}. The expected replanning action of the robot, R_a is given as:

$$R_a = \begin{cases} replan & P_{same} \\ wait & P_{cross} \\ navigate & otherwise \end{cases}. \quad (15)$$

The robot waits for the obstacle to cross and continue with its navigation if the condition is P_{cross}. If the condition is P_{same} a path replanning is done. The technique is demonstrated in Figure 6.

Figure 6
Determining the direction of an obstacle using sensors



From Figure 6, $(s_1, s_2, s_3, s_4, s_5)$ represent the sensors of the robot, $(\theta_{s1}, \theta_{s2}, \theta_{s3}, \theta_{s4}, \theta_{s5})$ are the angles of the sensors on the robot, (p_1, p_2, p_3) represent the different positions of an obstacle, d_1 and d_2 are the defined sensor distance thresholds from the robot to the obstacle, and s_d is the required safe distance from

the robot to the obstacle. The sensors track obstacles during navigation. When the minimum of the sensors' distances, *dist*, between the robot and an obstacle is less than d_1, it suggests an obstacle has been detected by the robot at a distance that is of less collision threat to the robot. The sensors that detect the obstacle are recorded. To check the direction of the obstacle, sensor readings are recorded till $dist \le d_2$ and the sensors with the readings are noted. If the same sensors detect the obstacle at $dist \le d_2$, the P_{same} condition is satisfied, and a call is made to re-plan and compute a new feasible path for navigation. The path replanning task involves re-generating the roadmap, path computation and path smoothening as described in steps 2, 3 and 4, respectively. The current position of the robot is used as the new source for the replanning. However, if the obstacle is detected by other sensors (right or left of the first sensor) at $dist \le d_2$, P_{cross} condition is satisfied. The robot stops and wait till $dist > d_1$ (see Equation (15)). From Figure 6, an obstacle *obs* at point p_1 was detected by s_2 when $dist \le d_1$. At point p_2, it was detected by s_3 also at $dist \le d_1$. At point p_3 it was detected by s_4 at $dist \le d_2$. This condition of the obstacle suggests that it is in motion across the path of the robot. This satisfies the P_{cross} condition, hence the robot stops and waits for the obstacle to cross till $dist \le d_1$ without path re-replanning. The algorithm used to guide navigation indicating a call for replanning when a random obstacle is detected is given in Algorithm 3.

4. Simulation Results

To evaluate the efficiency of the proposed method, simulation was carried out using MATLAB. Results obtained from experiment with complex and less complex environments, environment with local minimum possibilities, and narrow passages are given. $sz1 = 10$ and $sz2 = 15$ are the step sizes used in computing tree. Results for path replanning are also demonstrated. Additionally, results of experiment using maps in [10] are given.

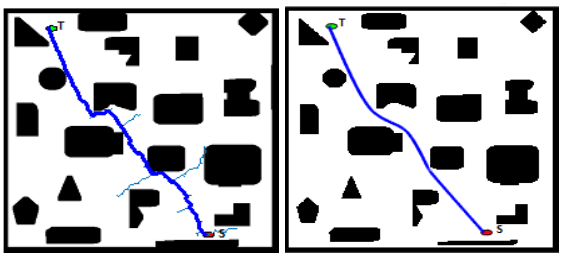
4.1. Experiment with Complex and Less Complex Environment

Figures 7 and 8 present simulation results in environment with less complex and complex environments for the modified goal-biased RRT and the ORRT-A* method proposed, respectively. The complexity of the environment in this paper is based on the occupancy rate of obstacles in the environment, the number and shapes of obstacles. The occupancy rate is computed as:

$$occupancy\_rate = \frac{obspixels}{mpixels} * 100, \tag{16}$$

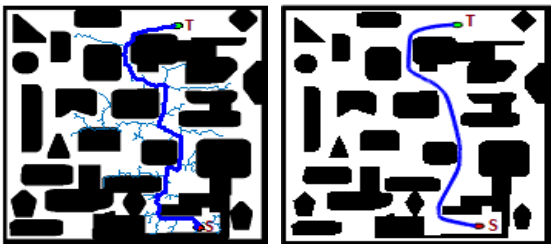
where *mpixels* represents the total number pixels of the map representing the environment and *obspixels* represents the number of pixels of the environment occupied by obstacles.

Figure 7
Path in less complex environment of occupancy rate of 35.18% and 19 different shapes of obstacles represented on a 500 by 500 pixel map: (a) Modified goal-biased RRT with path length 573.63 on the map with inflated obstacles; (b) ORRT-A* with path length 513.48 on the original map



(a) (b)

Figure 8
Path in complex environment of occupancy rate of 54.9% and 23 different shapes of obstacles represented on a 500 by 500 pixel map: (a) Modified goal-biased RRT with path length 668.23 on the map with inflated obstacles; (b) ORRT-A* with path length 572.34 on the original map



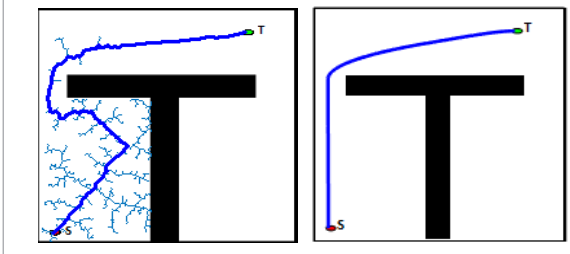
(a) (b)

4.2. Experiment with Environment with Local Minimum Possibilities

Experiment to compare the performance of the proposed method to RRT-A* in [24] was done using "T" shaped obstacle as presented in Figure 8. Li et al. [24] acknowledged the failure of RRT-A* to find path in an environment with T-shaped obstacle (Figure 9) after 5000 iterations. To evaluate the performance of the ORRT-A* algorithm in dealing with local minimum problem beyond T-shaped obstacle, a more complex local minimum environment was used for a simulation as shown in Figure 10.

Figure 9

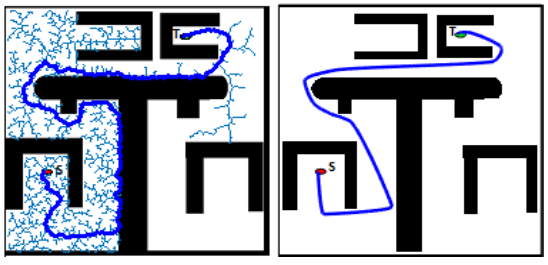
(a) Modified goal-biased RRT in T-shaped obstacle environment of path length 955.69 with 792 iterations; (b) ORRT-A* in T-shaped obstacle environment with path length 703.93 on the original map



(a) (b)

Figure 10

Path computation in complex local minimum environment: (a) Modified goal-biased RRT of path length 1469.8 with 2008 iterations; (b) ORRT-A* path with path length 1008.2 on the original map



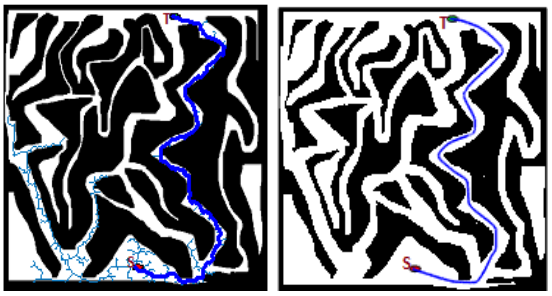
(a) (b)

4.3. Experiment in Narrow Passage Environment

It is known that sampling-based path planning methods have difficulties in finding path in narrow passages. To evaluate the performance of the proposed method, experiment was conducted in an environment with narrow passage to determine its efficiency as presented in Figure 11.

Figure 11

Path computation in a narrow passage environment: (a) Modified goal-biased RRT of path length 885.7 with 644 iterations; (b) ORRT-A* path with path length 775.86 on the original map



a b

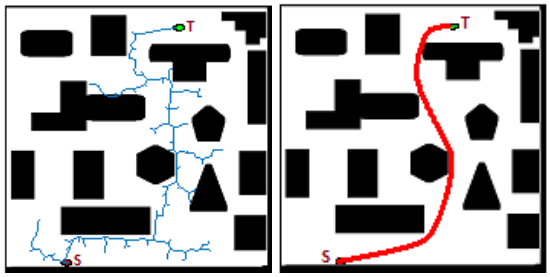
4.4. Results for Path Replanning

To evaluate the ORRT-A* method for path replanning, an obstacle was inserted randomly during the navigation of the robot ensuring that it obstructs the path of navigation. As the random obstacle is detected at a distance d_i, a call is made for replanning and the path is regenerated for navigation to continue. Not until a goal is reached, obstacles are inserted randomly to block the path of navigation. Obstacle detection sensors are, however, required for a real situation.

Figure 12 demonstrates tasks performed to regenerate the path to avoid collision with three random obstacles that obstruct the path during navigation and the final path taken by the robot to reach its target. Paths and roadmaps in Figures 12a to 12g are shown on the map with inflated obstacles while the path in Figure 12h shows the final path on the original map.

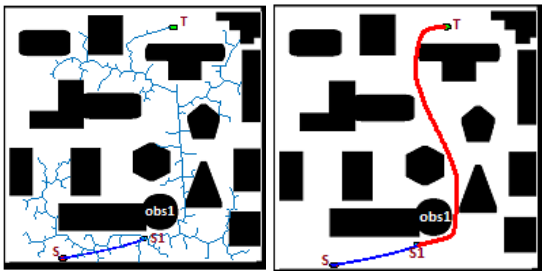
Figure 12
Path planning and replanning to avoid obstacles during navigation:

(a) Initial RRT Roadmap; (b) Initial Path from *S* to *T*.



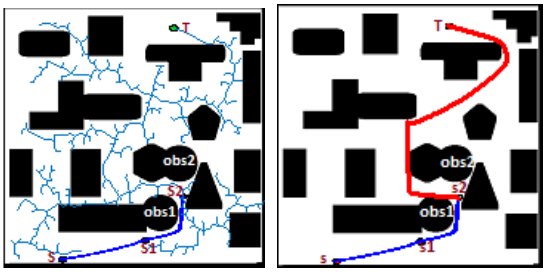
(a) (b)

(c) Roadmap for path replanning from *S1* as the new starting point after encountering the random obstacle *obs1;* (d) New Path generated from *S1* to the target *T;*



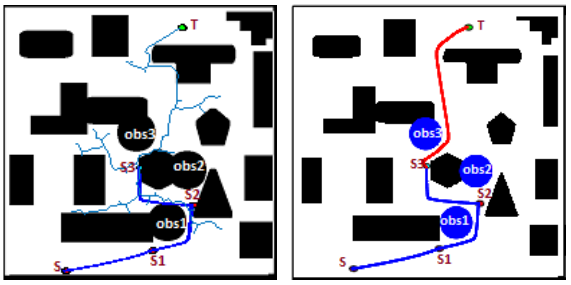
(c) (d)

(e) Roadmap for path replanning from *S2* as the new starting point after encountering the random obstacle *obs2;* (f) New Path generated from *S2* to the target *T*.



(e) (f)

(g) Roadmap for path replanning from *S3* as the new starting point after encountering the random obstacle *obs3* on the map with inflated obstacles; (h) Final Path to the target *T* on the original map.



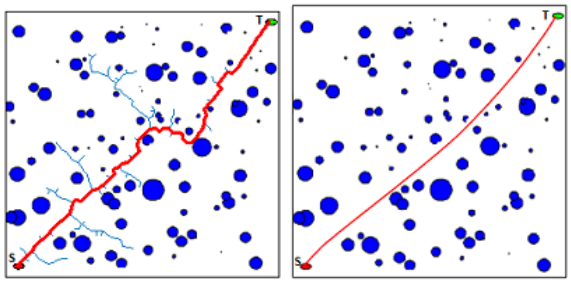
(g) (h)

4.5. Experiment with Maps Used to Evaluate RRT-A*

To aid in comparing the efficiency of the proposed method to RRT-A*, experiments are performed using the maps used in [24].

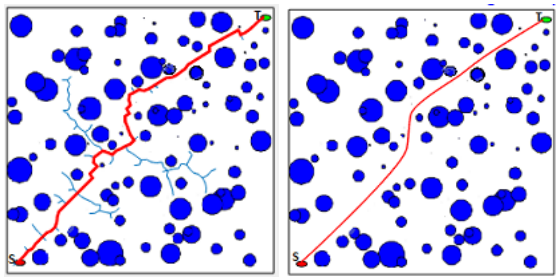
Figure 13 demonstrates the results of goal-biased RRT and ORRT-A* in sparse environment while Figure 14 depicts the results of goal-biased RRT and ORRT-A* in dense environment.

Figure 13
Performance of the proposed method in sparse environment of 50-by-50 with round obstacles used in [10]: (a) Roadmap and path for goal-biased RRT of path length 76.66 with 141 iterations; (b) ORRT-A* path of path length 66.81



(a) (b)

Figure 14
Performance of the proposed method in dense environment of 50-by-50 with round obstacles used in [10]: (a) Roadmap and path for goal-biased RRT of path length 77.69 with 601 iterations; (b) ORRT-A* path of path length 65.4



(a) (b)

5. Discussion

As indicated in Figure 7 and 8, the proposed method performed well in computing safe and shortest path from the initial position to the goal on maps representing less complex and more complex environments. The less complex environment is made up of 19 obstacles of different shapes with 35.18% occupancy rate while the more complex environment comprises of 23 obstacles of different shapes with 54.9% occupancy rate. In environments with T-shaped obstacle and other obstacles that could cause local minima problem (see Figure 9 and 10), the proposed method found path successfully as against the problem associated with RRT-A* in relation to local minima. It is also known that sample-based methods including RRT have problems with narrow passages [8]. The results shown in Figure 11 indicate that the proposed method is effective in dealing with narrow passages. The results of the proposed method as presented in Figure 12 depict the strength of the method in performing reactive path replanning to deal with unforeseen obstacles that may obstruct the path of the robot during navigation.

Moreover, as indicated in Figures 13 and 14, the proposed ORRT-A* method performed very well with sparse and dense environment maps used in [24]. Table 1 presents the performance results of Euclidean based goal-biased RRT and Euclidean based RRT-A* methods as presented in [24]. Table 2 presents the performance results of Euclidean based goal-biased RRT, A* and Euclidean based ORRT-A* methods indicating the average path length and time for each method after 50 runs for each map of sparse and dense environment. Results of RRT-A* method in [24], as presented in Table 1, were compared with goal-biased RRT. In Table 2, the proposed ORRT-A* method is compared to goal-biased RRT and A* methods. Table 3 compares RRT-A*, A* and ORRT-A* using goal biased RRT as the base method to compute the optimality and time-used rates. Information in Tables 1 and 2 is used for the comparison in Table 3. Table 3 indicates the percentage optimality gained and percentage time used by RRT-A*, A* and ORRT-A* methods as against goal-biased RRT using the same Euclidean

Table 1
Results of Euclidean based goal-biased RRT and Euclidean based RRT-A* methods presented in [24]

Environment	Algorithm	Average Length	Average Time
Sparse	Euclidean based goal-biased RRT	73.40	2.86
	Euclidean based RRT-A*	68.96	27.14
Dense	Euclidean based goal-biased RRT	73.367	6.745
	Euclidean based RRT-A*	70.36	21.73

Table 2
Results of Euclidean based goal-biased RRT, A* and Euclidean based ORRT-A* methods

Environment	Algorithm	Average Length	Average Time
Sparse	Euclidean based goal-biased RRT	70.72	5.82
	Euclidean based A*	63.13	484.70
	Euclidean based ORRT-A*	63.49	7.15
Dense	Euclidean based goal-biased RRT	72.63	4.27
	Euclidean based A*	63.98	239.01
	Euclidean based ORRT-A*	64.43	5.71

Table 3
Compared results of RRT-A*, A* and ORRT-A* with goal-biased RRT using Euclidean metric

Environment	Algorithm	% Optimality	% Time used	Average Nodes (n)
Sparse	Euclidean based RRT-A*	6.00	849.00	838
	Euclidean based A*	10.59	8,228.18	2051
	Euclidean based ORRT-A*	10.22	22.85	56
Dense	Euclidean based RRT-A*	4.27	222.00	781
	Euclidean based A*	11.91	5497.42	1715
	Euclidean based ORRT-A*	11.29	33.72	42

metric. The average number of nodes, *n*, used to compute the final path for each method is also indicated. The optimality rate is obtained using $\frac{Gp-Mp}{Gp}*100$, where *Gp* represents path length for the base method (Euclidean goal-based RRT) and *Mp* represents path length for RRT-A*, A* or ORRT-A*. The Time-used rate was computed using $\frac{Mt-Gt}{Gt}*100$, where *Mt* is the time used to obtain *Mp* for the base method (Euclidean goal-based RRT) and *Gt* represents the time used to obtain *Gp*. High optimality rate with low time-used rate indicates good performance. Comparing the results, the proposed method achieved 10.22% path optimality using 22.8% time as against 6% optimality with 849% time used by RRT-A* method in less complex environments. In dense environment, the proposed method achieved 11.29% path optimality using 33.87% time as against 4.2% optimality with 222% time used by RRT-A* method. Regarding path length, A* performs better than both RRT-A* and ORRT-A* methods. However, the average time used (8,228.18% and 5,497.42% for sparse and dense environments, respectively) to attain such optimality is very high and comparatively inefficient for a reactive path planning for obstacle avoidance for robots in motion.

The time and space complexity of RRT* and RRT*-smart is indicated in [16] as $O(n\log n)$ and $O(n)$, respectively. With the same requirements to compute the sample, nearest node, tree extensions and addition of node to the tree as in RRT*-smart, goal-biased RRT uses the same time and space complexities. But with the requirement to bias the tree growth towards the goal, the *n* value in goal-biased RRT is reduced. The proposed method adopted and modified goal-biased RRT for the generation of the roadmap. With the introduction of additional step-size to speed up the growth of the tree to the goal in the proposed method, the *n* value is reduced significantly though the time and space complexities remain $O(n\log n)$ and $O(n)$, respectively. RRT-A* algorithm, on the other hand applied A* during the generation of the tree at the learning phase. The time and space complexities of A* are $O(n^2)$ and $O(n)$, respectively. The use of the A* at the learning phase increases the time complexity of generating the tree to $O(n^2)$. At the query phase of the proposed method, a simple binary search of running time of $O(n\log n)$ is used to obtain the initial path where *n* is the number of nodes on the roadmap. A* algorithm with $O(n^2)$ running time, where *n* represents the number of nodes of the path generated using the binary search is then used to obtain the shortest path from the start position of the robot to the goal position. Hence, the *n* value used in the ORRT-A* method has reduced drastically compared to the *n* value used in RRT-A* though both methods applied A* algorithm with running time of $O(n^2)$. This analysis indicates a better performance of ORRT-A* as against RRT-A* with respect to time as demonstrated in the results in Table 3.

6. Conclusion

In this paper, ORRT-A* path planning method for mobile robots in partially known complex environment is presented based on MD technique, goal-biased

RRT, A* and cubic spline interpolation algorithms. The ORRT-A* algorithm considered the safety of the mobile robots during navigation by introducing MD technique to inflate the obstacles before computing the path. Given initial and target positions, the algorithm generates a roadmap using modified goal-biased RRT on the map with inflated obstacles. Unlike the original goal-biased RRT algorithm, two step-size constraints, where one step-size is assigned higher value than the others are used to facilitate the growth of the tree from the initial position to the target with low time complexity. A* heuristic algorithm was used as the local planner to obtain the shortest path while cubic spline interpolation was used to smoothen the path generated. While RRT-A* method used A* heuristic algorithm at every iteration to select the nearest node during the generation of the roadmap at the learning phase, it is used in ORRT-A* as a local planner at the query phase after generating the roadmap with the modified goal-biased RRT.

Results indicate that ORRT-A* method provides enhanced path quality compared to goal-biased RRT and RRT-A* methods. Compared to RRT-A* method, ORRT-A* method is more efficient in terms of path quality and length, and time complexity using the same distance metric. The algorithm presented performed efficiently in both complex and narrow passages environments. The local minimum problem associated with RRT-A* method has been addressed. Again, ORRT-A* method can avoid random obstacles that obstruct the path of the robot during navigation by replanning its path till the target is reached. These strengths make the method viable to perform efficiently in real partially known complex environment. ORRT-A* is therefore a promising method to achieve autonomous vehicle navigation in partially known and complex environment.

Future work would consider real implementation of ORRT-A* path planning method to evaluate its performance in the real environment. Implementation of the method in 3D environment is under consideration.

Acknowledgements

This work was supported by the Beijing Advanced Innovation Center for Intelligent Robots and Systems under Grand No. 2018IRS20.

References

1. Aguilar, W. G., Morales, S. G. 3D Environment Mapping Using the Kinect V2 and Path Planning Based on RRT Algorithms. Electronics, 5(4), 70, 2016. https://doi.org/10.3390/electronics5040070
2. Ayawli, B. B. K., Chellali, R., Appiah A. Y., Kyeremeh F. An Overview of Nature-Inspired, Conventional, and Hybrid Methods of Autonomous Vehicle Path Planning. Journal of Advanced Transportation, 2018 (2018), 1-27. https://doi.org/10.1155/2018/8269698
3. Burden R. L., Faires J. D. Numerical Analysis, Ninth edition. Boston, USA: Brooks/Cole, Cengage Learning, 2011.
4. Candeloro, M., Lekkas, A. M., Hegde, J., Sørensen, A J. A 3D Dynamic Voronoi Diagram-Based Path-Planning System for UUVs. OCEANS 2016 MTS/IEEE, Monterey, CA, 2016, 1-8. https://doi.org/10.1109/OCEANS.2016.7761427
5. Candeloro, M., Lekkas, A. M., Sørensen, A. J. A Voronoi-Diagram-Based Dynamic Path-Planning System for Underactuated Marine Vessels. Control Engineering Practice, 2017, 61, 41-54. https://doi.org/10.1016/j.conengprac.2017.01.007
6. Chen, Y., Sun, J. Distributed Optimal Control for Multi-Agent Systems with Obstacle Avoidance. Neurocomputing, 2016, 173(3), 2014-2021. https://doi.org/10.1016/j.neucom.2015.08.085
7. Connell, D., La, H. M. Dynamic Path Planning and Replanning for Mobile Robots Using RRT*, arXiv, preprint arXiv: 1704.04585, 2017. https://doi.org/10.1109/SMC.2017.8122814
8. Daee, P., Taheri, K., Moradi, H. A Sampling Algorithm for Reducing the Number of Collision Checking in Probabilistic Roadmaps. 22nd Iranian Conference on Electrical Engineering (ICEE), Tehran, 2014, 1313-1316. https://doi.org/10.1109/IranianCEE.2014.6999737
9. de Santana Correia, A., Kamarry, S., Molina, L., Carvalho, E. Á. N., Freire, E. O. RRT-Edge: A Compact Approach for Path Planning of Mobile Robots. Latin American Robotics Symposium (LARS) and 2017 Brazilian Symposium on Robotics (SBR), Curitiba, 2017, 1-6. https://doi.org/10.1109/SBR-LARS-R.2017.8215282

10. Ferguson, D., Stentz, A. Anytime RRTs, IEEE/RSJ International Conference on Intelligent Robots and Systems, Beijing, 2006, 5369-5375. https://doi.org/10.1109/IROS.2006.282100

11. García, N., Suárez, R., Rosell, J. HG-RRT*: Human-Guided Optimal Random Trees for Motion Planning. Conference on Emerging Technologies & Factory Automation (ETFA), Luxembourg, 2015, 1-7. https://doi.org/10.1109/ETFA.2015.7301536

12. Gonzalez, R. C., Wood, R. E. Digital Image Processing, 2nd edn. Englewood Cliffs: Prentice Hall, 2002.

13. Heß, R., Kempf, F., Schilling, K. Trajectory Planning for Car-Like Robots Using Rapidly Exploring Random Trees. IFAC Proceedings, Seoul, Korea, 2013, 44-49. https://doi.org/10.3182/20131111-3-KR-2043.00018

14. Heβ, R., Lindeholz, T., Eck, D., Schilling, K. RRTCAP* - RRT* Controller and Planner - Simultaneous Motion and Planning. IFAC-PapersOnline, 2015, 48(10), 52-57. https://doi.org/10.1016/j.ifacol.2015.08.107

15. Ho, Y. J., Liu, J. S. Collision-free Curvature-Bounded Smooth Path Planning Using Composite Bezier Curve Based on Voronoi Diagram. IEEE Intl Symp on CIRA, Daejeon, 2009, 463-468. https://doi.org/10.1109/CIRA.2009.5423161

16. Islam, F., Nasir, J., Malik, U., Ayaz, Y., Hasan, O. RRT*-Smart: Rapid Convergence Implementation of RRT* Towards Optimal Solution. IEEE International Conference on Mechatronics and Automation, Chengdu, 2012, 1651-1656. https://doi.org/10.1109/ICMA.2012.6284384

17. Kakillioglu, B., Ozcan, K., Velipasalar, S. Doorway Detection for Autonomous Indoor Navigation of Unmanned Vehicles. IEEE International Conference on Image Processing, Phoenix, AZ, 2016, 3837-3841. https://doi.org/10.1109/ICIP.2016.7533078

18. Karaman, S., Frazzoli, E. Incremental Sampling-Based Algorithms for Optimal Motion Planning. arXiv preprint arXiv: 1005.0416, 2010. https://doi.org/10.15607/RSS.2010.VI.034

19. Karaman, S., Frazzoli, E. Optimal Kinodynamic Motion Planning Using Incremental Sampling-Based Methods. 49th IEEE Conference on Decision and Control (CDC), Atlanta, GA, 2010, 7681-7687. https://doi.org/10.1109/CDC.2010.5717430

20. Karaman, S., Frazzoli, E. Sampling-Based Algorithms for Optimal Motion Planning. The International Journal of Robotics Research, 2011, 30(7), 846-894. https://doi.org/10.1177/0278364911406761

21. Karaman, S., Walter, M. R., Perez, A., Frazzoli, E., Teller, S. Anytime Motion Planning Using the RRT*. IEEE International Conference on Robotics and Automation, Shanghai, 2011, 1478-1483. https://doi.org/10.1109/ICRA.2011.5980479

22. Lavalle, S. M. Rapidly-Exploring Random Trees: a New Tool for Path Planning. Tech. Rep. 98-11, Computer Science Dept, Iowa State University, Iowa, USA, 1998. =20

23. Li Y., Cui R., Li Z., and Xu D. Neural Network Approximation-Based Near-Optimal Motion Planning with Kinodynamic Constraints Using RRT. IEEE Transactions on Industrial Electronics, 2018 65(11) 8718-8729. https://doi.org/10.1109/TIE.2018.2816000

24. Li, J, Liu, S., Zhang, B., Zhao X. RRT-A* Motion Planning Algorithm for Non-Holonomic Mobile Robot. Proceedings of the SICE Annual Conference, Sapporo, 2014, 1833-1838. https://doi.org/10.1109/SICE.2014.6935304

25. Li, Y., Zhang, F., Xu, D., Dai, J. Liveness-Based RRT Algorithm for Autonomous Underwater Vehicles Motion Planning. Journal of Advanced Transportation, 2017, (2017) 1-10. https://doi.org/10.1155/2017/7816263

26. Matveev, A. S., Savkin, A. V., Hoy, M., Wang, C. Safe Robot Navigation Among Moving and Steady Obstacles: Biologically Inspired Algorithm for Safe Navigation of a Wheeled Robot Among Moving Obstacles. Butterworth-Heinemann, 2016, 161-184. https://doi.org/10.1016/B978-0-12-803730-0.00008-1

27. Noreen, I., Khan, A., Ryu, H., Doh, N. L., Habib, Z. Optimal Path Planning in Cluttered Environment Using RRT*-AB. Intelligent Service Robotics, 2018, 11(1), 41-52. https://doi.org/10.1007/s11370-017-0236-7

28. Otte, M., Frazzoli, E. RRTX: Asymptotically Optimal Single-Query Sampling-Based Motion Planning with Quick Replanning. The International Journal of Robotics Research, 2016, 35(7), 797 - 822. https://doi.org/10.1177/0278364915594679

29. Palmieri, L., Koenig, S., Arras, K. O. RRT-Based Nonholonomic Motion Planning Using Any-Angle Path Biasing. International Conference on Robotics and Automation (ICRA), Stockholm, 2016, 2775-2781. https://doi.org/10.1109/ICRA.2016.7487439

30. Park, J. J., Kuipers, B. Feedback Motion Planning via Non-Holonomic RRT* for Mobile Robots. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Hamburg, 2015, 4035-4040. https://doi.org/10.1109/IROS.2015.7353946

31. Qureshi, A. H. et al. Triangular Geometry Based Optimal Motion Planning Using RRT*-Motion Planner.

International Workshop on Advanced Motion Control (AMC), Yokohama, 2014,380-385. https://doi.org/10.1109/AMC.2014.6823312

32. Russell, S., Norvig, P. Artificial Intelligence: A Modern Approach 3rd ed. Upper Saddle River, New Jersey: Prentice Hall Series, Pearson Education Inc, 2010.
33. Santiago, R. M. C., De Ocampo, A. L., Ubando, A. T., Bandala, A. A., Dadios, E. P. Path planning for Mobile Robots Using Genetic Algorithm and Probabilistic Roadmap. IEEE 9th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM), Manila, 2017, 1-5. https://doi.org/10.1109/HNICEM.2017.8269498
34. Tusi, Y., Chung, H. Y. Using ABC and RRT Algorithms to Improve Mobile Robot Path Planning with Danger Degree. 5th International Conference on Future Generation Communication Technologies (FGCT), Luton, 2016, 21-26. https://doi.org/10.1109/FGCT.2016.7605068
35. Wang, M., Luo, J., Walter, U. A Non-Linear Model Predictive Controller with Obstacle Avoidance for a Space Robot. Advances in Space Research, 2016, 57(8) 1737-1746. https://doi.org/10.1016/j.asr.2015.06.012
36. Xiao, S., Bergmann, N., Postula, A. Parallel RRT* Architecture Design for Motion Planning. International Conference on Field Programmable Logic and Applications (FPL), Ghent, 2017, 1-4. https://doi.org/10.23919/FPL.2017.8056773
37. Yang, K., Gan, S. K., Huh, J., Joo, S. Optimal Spline-Based RRT Path Planning Using Probabilistic Map. 14th International Conference on Control, Automation and Systems (ICCAS), Seoul, 2014, 643-646. https://doi.org/10.1109/ICCAS.2014.6987859
38. Yang, K., Gan, S. K., Sukkarieh, S. A Gaussian process-based RRT Planner for the Exploration of an Unknown and Cluttered Environment with a UAV. Advanced Robotics, 2013, 27(6), 431-443. https://doi.org/10.1080/01691864.2013.756386
39. Yang, L., Wei-guo, Z., Jing-ping, S., Guang-wen, L. A Path Planning Method Based on Improved RRT. Proceedings of 2014 IEEE Chinese Guidance, Navigation and Control Conference, Yantai, 2014, 564-567. https://doi.org/10.1109/CGNCC.2014.7007284