Safe Deep Learning-Based Global Path Planning Using a Fast Collision-Free Path Generator

Shirin Chehelgami[a,], Erfan Ashtari[a], Mohammad Amin Basiri[a], Mehdi Tale Masouleh[a], Ahmad Kalhor[a]

[a]*Human and Robot Interaction Laboratory, School of Electrical and Computer Engineering, University of Tehran, Tehran, Iran*

Abstract

In this research, a global path planning method based on recurrent neural networks by means of a new loss function is presented, which regardless of the complexity of the configuration space, generates the path in a relatively constant time. The new loss function is defined in such a way that in addition to learning the input data of the network, it creates an adjustable safety margin around the obstacles and ultimately creates a safe path. Moreover, a new global path planning method is also introduced, which is used to create the dataset needed to train the proposed neural network. The convergence of this method is mathematically proven and it is shown that this method can also produce a suboptimal path in a much shorter time than the common methods of global path planning reported in the literature. In short, the main purpose of this research consists in providing a method which can create a suboptimal, fast and safe path for a mobile robot from any random starting point to any random destination in a known environment. First, the pro-

Email addresses: shirin.chehelgami@ut.ac.ir (Shirin Chehelgami), erfan.ashtari@ut.ac.ir (Erfan Ashtari), ma.basiri@ut.ac.ir (Mohammad Amin Basiri), m.t.masouleh@ut.ac.ir (Mehdi Tale Masouleh), a.kalhor@ut.ac.ir (Ahmad Kalhor)

posed methods will be implemented for different two-dimensional environments consisting of convex and non-convex obstacles, considering the robot as a point-mass, and then it will be implemented in a 3D simulation environment, AI2THOR. Compared to classical global path planning algorithms, such as RRT and A*, the proposed approach demonstrates better performance in complex and challenging environments.
Keywords:
Mobile Robots, Deep Learning in Robotics and Automation, Recurrent Neural Network, Fast Global Path Planner, Safe Path Generator

1. Introduction

One of the most important features of mobile robots is path planning, which reduces their dependence on human intervention [1]. In robotics, path planning pertains to determining a destination and finding a sequence of actions which should be taken to make the robot reach safely a prescribed goal point from its current state, without encountering obstacles in the environment [2].

There are several methods for generating a collision-free path from the starting point to the goal point, while the environment around the robot is known from the outset, the most common of these methods are grid-based approaches and sampling-based approaches. In grid-based approaches such as A* [3], the number of grids on the network grows exponentially in the configuration space dimension and making them unsuitable for high-dimensional environments [4]. However, the A* algorithm has been applied in several areas, such as the areas of automation, robotics [5], medicine [6] and games [7]. Sampling-based approaches such as RRT [8] are not highly dependent on the dimensions of the environment, but as the

complexity of the environment increases, their execution speed decreases sharply and they are computationally unusable for precise motion planning in environments with complex constraints [9]. However, the RRT algorithm has already been applied in various areas, such as automation, robotics [10], path planning for mobile robots [11] and so forth. Since an ideal path planning algorithm for solving real-world problems should provide key features such as completeness and optimality of the path, insensitivity to the dimensions and complexity of the environment, computational efficiency, and so on, there is a fundamental trade-off between the mentioned features among the existing algorithms, and few motion planners have been proposed to solve the latter problems [12].

There is a lot of research which has used deep neural networks to solve the problem of path planning [13] [14] [15], and, due to the hand-in-hand advent made on the computer hardware and artificial intelligence algorithms, recently a lot of progress has been made in this area. From the studies propounded in the literature, it follows that the use of deep neural networks for path planning has great potential for creating accurate, fast and optimal motion planners [12].

In [16], an interesting method, called OracleNet, which, based on deep neural networks, produces fast and optimal collision-free paths for static environments is presented. In this study, an LSTM network is used to create suboptimal paths by reducing the loss function. This simple algorithm creates these suboptimal paths in an almost constant time, and so, this method is more popular than the path planning algorithms in complex environments and higher dimensions which their calculations increase and so the production speed of the path decreases.

In this paper, the training data set includes a number of paths created by the RRT and A* path planners, and as aforementioned, as the size or complexity of the

state space increases, the speed of these algorithms decreases and the production of training data using these methods will be time-consuming. In addition, the loss function used to train this network is MSE, which has no sense of the obstacles and their location and it causes errors in predicting the paths, and in many cases, the predicted paths will pass through the corners of obstacles and as a result, the network will not be able to predict collision-free paths. Therefore, despite the fact that the proposed model leads to acceptable performance and produces suboptimal paths in a short time, it contains some drawbacks which upon solving them result in a safe and confident path planning approach.

In this paper, as the first step, a fast and efficient global path planning method based on the Bug2 local path planning algorithm is presented, which is called the Fast Global Bug (FGB). From the best knowledge of the authors, the FGB method is faster than other existing global path planning methods and does not depend too much on the complexity and dimensions of the environment. It is shown that the proposed algorithm can be used to create the data needed for neural network training due to its high speed. Then a new loss function is proposed for mobile robot path planning. The presented loss function, in addition to learning the training data, has a repulsive term which moves the predicted points of the path away from the nearest point of the nearest obstacle to it, thus creating a safe adjustable margin around the obstacles. This will increase the success rate of path planning, especially in complex environments.

The results of the proposed method on a point robot is shown in different environments and a virtual robot on the AI2THOR [17] platform. Given that the proposed network in any new environment requires training, this algorithm is suitable for robots which work in static environments (such as warehouses,

shopping centers, airport terminals, etc.). However, by combining this method with the local path planning algorithm, Artificial Potential Field (APF) [18], a hybrid path planning algorithm which can be used in dynamic environments is also proposed in [19].

The rest of this paper is organized as follows. Section 2 provides an overview of the studies conducted on path planning for mobile robots in the literature. Section 3 provides the details of the proposed model and the alternative approaches which are used for comparison. Section 4 and Section 5 give the experimental results, and the final section takes everything into a conclusion.

2. Related works

In this paper, for proposing a new path planning method, several former methods for path planning are used in different parts of this project. Therefore, in this section, prior to proposing the new path planning method, some path planning methods such as A* [3], RRT [8], Potential field [18], BUG [20] [21] and Oracle Net [16] will be reviewed in what follows.

*2.1. A**

This method is a graph search method in which the environment is divided into discrete points or nodes and the shortest path to the goal can be obtained, only by taking into account these nodes. One of the methods used to select proper nodes is the A* method. In this method, the nodes are added to the graph with a regular pattern, and this is done by prioritizing the nodes which are more likely to produce the optimal path [22]. To do so, some exploration methods, such as the straight line distance from one node to the target, are used in addition to the cost of the node, and the sum of the two functions indicates the cost of the path:

$$f(n) = g(n) + h(n) \tag{1}$$

As stated, the above cost function has two parameters: $g(n)$: the cost of transferring from the primary cell to the current cell, which is essentially the sum of all the cells visited by the first cell. $h(n)$: also known as the exploration value, is the estimated cost of moving from the current cell to the final cell. The actual cost cannot be calculated until the final cell reaches. Therefore, $h(n)$ is the estimated cost [23]. However, in algorithms such as A* which search in a connected chart or network, are fast and optimized in small and simple environments, their calculations are expanded as the size of the environment and its complexity increases which leads to slowing down the production speed of the path [24].

2.2. Rapidly exploring random tree (RRT)

The Rapidly-exploring Random Tree path planning method, the so-called RRT, has been developed by Lavalle and Kuffner [8] to find a route in a configuration space. In the RRT method, the process starts from the first node and a new node will be added to the tree, but not too far away, because in this case the probability of passing through the obstacles or excessive movement in the wrong direction increases; thus, a maximum distance specified so that the new node can be close to the last node. In this case, a few situations happen; if the random node is closer than the maximum distance, then the new node will be placed right there. If there is an obstacle between the node and where the new node is supposed to be placed there, the new node will be ignored and will not be added to the tree and continues to add the new node, which prevents the tree from passing through the obstacles [25].

So, at each stage, a new node is selected and then the closest node to it will be found. By increasing in the number of new nodes, the paths entered the unknown areas therefore the environment is quickly explored, which is why this algorithm is called Rapidly-exploring Random Tree. This process continues until the path reaches a certain distance from the target. Although the paths produced by this method are zigzag and are not optimized, the generated paths use fewer nodes than A* because the nodes can have more distance from each other. RRT can work very well in situations where a valid path is only quested also the start and target nodes are available [11].

2.3. Artificial Potenial field

The Artificial Potential Field method (APF) was initially proposed by Khatib in the year 1986 for path planning of mobile robots [18]. A robot in the APF method is considered as a mass point in the robot configuration space and, it is affected by an artificial potential field, like a particle. The APF algorithm uses an artificial potential field to adjust the movement of a robot in a particular space. In this method areas far from the target, have high potential energy, and as the robot approaches the target, the potential energy decreases and ultimately at the target point, the potential energy will be equal to zero.

The main idea of this method is to imagine that all obstacles can create a repulsive force for the robot, while the target point creates an attraction force for the robot. The robot applies a force which is related to the gradient of the combination of attractive and repulsive forces. So, the movement direction vector of the robot can be characterized by combining attractive and repulsive forces. The final force which specifies the robot's movement is as follows [26]:

$$F = F_{\text{att}} + F_{\text{rep}} \tag{2}$$

In [26], the APF method is used for motion planning of a moving robot in a dynamic environment where targets and obstacles are moving. The speed and path of the robot are determined by the relative velocity and the direction of the obstacles and targets. Simulation results confirm that this method can effectively track the moving target while avoiding obstacles in its path.

The APF method offers a simple yet effective way for robot path planning, but it also has disadvantages. The main problem with this method is that the robot often traps in a local minimum before reaching the target. This problem occurs when all artificial forces (attractive and repulsive) cancel each other, such as in situations where an obstacle is exactly between the robot and the target or when the obstacles are close to each other. To overcome this issue, several approaches have been proposed [27]. For instance, a simple way to fix this problem is to insert a random noise into the robot's movement which leads the robot to avoid the local minimum.

2.4. Bug

The Bug algorithms are ideal for indoor robot path planning with limited sources because these algorithms require little memory and processing, so they are expected to occupy little space in the internal computer [28]. The basic principle of this algorithm is that the robot does not know the obstacle's locations in its environment and only knows the relative position of its purpose. In other words, the robot reacted locally, only in contact with obstacles and walls, allowing the robot to move towards its target by following the obstacles' boundaries (wall chase) [29].

Lumelsky and Stepanov [20] are the pioneers of the development of this technique. Initially, they introduced a very simple Bug algorithm called the "common sense algorithm" also known as COM. In this method, the robot moves toward the target whenever it can. The point where a bug algorithm first hit the obstacle is called the hit point and the point where the path leaves the obstacle to reach the target point is called the leaving point. It seems that the COM can solve the path planning problem in many situations. However, Lumelsky and Stepanov [20] pointed out that there are scenarios in which the robot cannot reach the target point. Furthermore, they introduced the Bug1 algorithm with a different strategy to overcome the problems that the COM was facing. In this strategy, every time that the path encounters an obstacle, it first explores the obstacle by following its entire border, while simultaneously finding the closest position on the border to the target point and storing it in memory. After colliding with the first hit point, the path moves toward the closest position to the target and leaves the obstacle. It is also shown in this article that the length of the path never exceeds the following limit:

$$P = d(S;T) + 1.5 \times \sum p_i \tag{3}$$

where P is the length of the entire path, $d(S;T)$ is the distance between the starting point and the target point and p_i is the length of the border of i^{th} obstacle.

However, this algorithm is a less intuitive approach and as it needs to explore the entire border of the obstacles, so it produces unnecessary long paths. So, a year later, Lumelsky and Stepanov [21] recognized the non-optimality in the length of paths created by Bug1 algorithm and thus they presented Bug2 as an alternative method. In this algorithm, between the start and target point, a hypothetical line

is considered which is called the m-line, then the path explores the border of obstacles until it hits the m-line on another side; If the point is closer to the target than the initial hit point, the path will leave the obstacle from that point. In [21], it is shown that the latter method reduces the maximum length of the path to:

$$P = d(S;T) + \sum p_i \tag{4}$$

2.5. Oracle Net

In [16], a new method called OracleNet have introduced, which uses a neural network-based path generator and produces the desired paths at a fixed time for static environments. Furthermore, a Long-Short Term Memory (LSTM) [30] model as a Recurrent Neural Network (RNN) is used. For generating the training dataset, a number of path planners such as RRT and A* are used and also the article used Mean Squared Error [31] as the loss function. The network learns paths in the training dataset in a predefined environment by decreasing the loss function and eventually predicts a relatively smooth path for each random start and target point in the environment. In practice, OracleNet generates paths in a fixed and short amount of time, regardless of the complexity of the configuration space, and also performs better than common path planning algorithms in complex environments and higher dimensions.

3. Training-data Generation

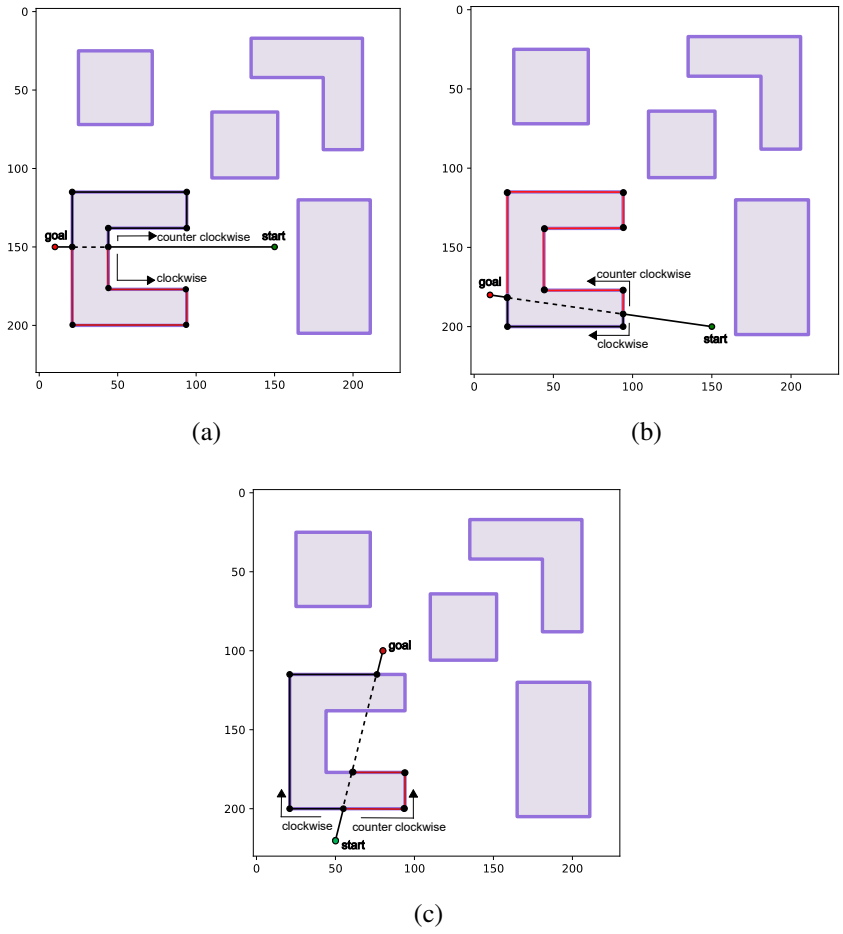
In this paper, for the sake of data creation, a new method is introduced, which is inspired by the Bug2 algorithm upon applying some modifications. The Bug2 algorithm uses sensors of a robot and a local knowledge of the environment to

perform path planning and obstacle avoidance in a real environment. But this new method, which is called Fast Global Bug (FGB), is for obtaining paths as the training data in a static environment. In this method, it is assumed that global knowledge about the environment including the coordinates of the corner points of obstacles, the start point and the goal point are available.

In the FGB method, a straight line is considered from the start point to the goal point, the aforementioned m-line, and the points of collision with obstacles, which are called hit-points, are determined. By arranging the hit-points from the start to the goal and then the path starts from the start point and goes along the m-line till reaches the first hit-point. After reaching the hit-point, the path is virtually divided into two branches, clockwise and counterclockwise, and continues in both directions on the obstacle's perimeter, the corner points of the obstacles which have passed along the paths are stored which leads to two general situations:

1. One of the two paths reaches one of the next hit-points with passing of fewer obstacle corner points, in which the mentioned path is selected.
2. Both paths reach the next hit-points by passing of equal number of obstacle corner points which in this condition, two situations occur:
   (a) The two paths have reached a hit-point, in which the distance till this hit-point is calculated from both directions and the path with the smallest distance is selected.
   (b) The two paths have reached different hit-points, in which the path which it hit-point is closer to the goal is selected.

Then the path continues on the m-line and the same process is repeated until the path ends at the goal. Fig. 1 shows schematically the 3 situations explained



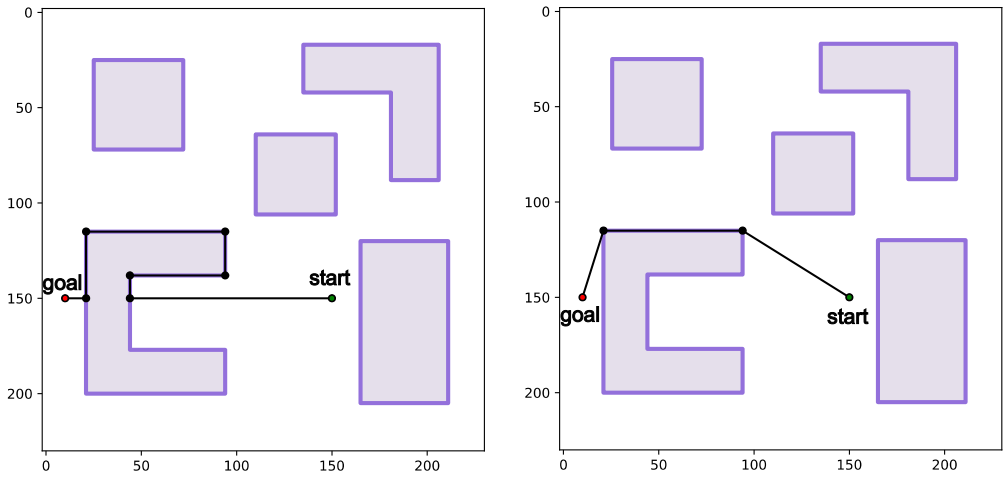
(a) (b)

(c)

Figure 1: 3 situations of the FGB method.

above for the FGB method, and it has to be mentioned that the unit of measurement of the figures in this paper are decimeter. After reaching the goal, a collision-free path from start to goal has been obtained, but because it goes along the direct line from start to goal and moves around obstacles it is not an optimal path and it has some redundant points which make the path longer.

For solving this problem, a rewiring process which has been introduced in [16] is applied; during this process, as is shown in Fig. 2, the unnecessary nodes in the



(a) The path (b) The rewired Path

Figure 2: Rewiring process in the FGB method.

paths will be removed if a straight line which connects two non-consecutive nodes in the path is collision-free. The algorithm for creating the training data is shown in Algorithm 1.

Algorithm 1 Creating the training data

$q_0^H \leftarrow q_{start}, i = 0$

while True **do**

from q_i^H move toward q_{goal} along the m-line until goal is reached or obstacle encountered at q_{i+1}^H

if Goal is reached **then** Exit

else

follow boundary in two branches, clockwise and counterclockwise, and save corner points of the obstacles (N_c) in $path_R$ and $path_L$

if q_{i+1}^H is encountered & $d\left(q_{i+1}^H, q_{goal}\right) < d\left(q_i^H, q_{goal}\right)$ **then**

if $N_c\left(path_L\right) < N_c\left(path_R\right)$ **then** Delete $path_R$

else if $N_c\left(path_L\right) > N_c\left(path_R\right)$ **then** Delete $path_L$

else if $N_c\left(path_L\right) = N_c\left(path_R\right)$ **then**

if $q_{H_L} = q_{H_R}$ **then**

if $d\left(path_L\right) < d\left(path_R\right)$ **then** Delete $path_R$

else Delete $path_L$

end if

end if

if $q_{H_L} \neq q_{H_R}$ **then**

if $d\left(q_{H_L}, q_{goal}\right) < d\left(q_{H_R}, q_{goal}\right)$ **then** Delete $path_R$

else Delete $path_L$

end if

end if

end if

end if

if Goal is reached **then**

Exit

else if $path_R$ is selected **then** $q_j = q_{H_R}$

else if $path_L$ is selected **then** $q_j = q_{H_L}$

end if

$i \leftarrow j$

end while

3.1. Proof of convergence

The proof of convergence depends on the type of obstacles, namely, convex and non-convex. To the end of achieving a proof for convergence of paths gener-

ated by FGB method, the proof for both types of obstacles is investigated in what follows.

3.1.1. For non-Convex obstacles

Theorem 3.1. *The length of the path generated by FGB method is always lower than a limit.*

Lemma 3.2. *The m-line has n hit-points with a non-convex obstacle which $n > 2$ and n is an even number $(n = 2k)$.*

Lemma 3.3. *If the set of hit-points is numbered, the hit- points that have the odd number, are the points where m-line hit the obstacles; In other words, m-line enters the obstacles in these points of the free space, and the hit-points that are even, are the points where m-line leaves the obstacles and continues in the free space.*

Lemma 3.4. *Under the procedure of FGB method, on the perimeter of an obstacle path always goes from a leaving point to one of the following hits and a hit to one of the next leaves.*

Lemma 3.5. *Under the procedure of the proposed method, because of not selecting previous hit-points, the path will not get stuck in a local cycle and then the worst-case scenario on the perimeter of an obstacle is to pass from all hit-points consecutively.*

Proof. According to Lemma 3.5, in the worst case, it is assumed that the path passes through all the hits and leaves. The path goes in two directions, left and right, then:

$$p^i_{L_j} + p^i_{R_j} \leq P^i \tag{5}$$

where p is for noticing perimeter, i is for i^{th} obstacle, L and R are for determining path direction and j is for number of hit on the i^{th} obstacle. It is obvious that $p^i_{L_j}$ and $p^i_{R_j}$ are positive and lower than p^i because actually, they are parts of the perimeter of i^{th} obstacle. Under the procedure of the FGB method, it needs to select one of the $p^i_{L_j}$ and $p^i_{R_j}$ directions according to the pseudo code and then call the selected path $p^i_{j_{\text{selected}}}$ therefore $p^i_{j_{\text{selected}}}$ is lower than p^i. $d_{\text{tot}_{\text{non-convex}}}$, the maximum length of the generated paths by FGB method with the presence of non-convex obstacles, is obtained as follows:

$$d_{\text{tot}_{\text{non-convex}}} = d_s + \sum_{i=1}^{O} \left(\Big(\sum_{j=1,j=2k+1}^{n_i} P^i_{j_{\text{selected}}} + \sum_{j=2,j=2k}^{n_i} d_{i_j} \Big) + d_{i,i+1} \right) + d_g \tag{6}$$

where d_{i_j} are segments of the m-line which are in free space and between j^{th} hit-point and $(j+1)^{th}$ hit-point of i^{th} obstacle, $d_{i,i+1}$ is a segment of the m-line which is in free space and between the last hit-point of i-th obstacle and first hit-point of $(i+1)^{th}$ obstacle and finally d_s and d_g are first segment and last segment of the m-line which are in free space. So, from the definition, it is obvious that:

$$d_s + \sum_{i=1}^{O} [(\sum_{j=2k,j=2}^{n_i} d_{i_j}) + d_{i,i+1}] + d_g \leq D \tag{7}$$

in the above, D is the sum of all segments of the m-line which are in free space; and also:

$$\sum_{i=1}^{O} \sum_{j=1,j=2k+1}^{n_i} P^i_{j_{\text{selected}}} \leq \sum_{i=1}^{O} \frac{n_i}{2}.P^i \tag{8}$$

Therefore:

$$P_{\text{tot}_{\text{non-convex}}} \leq D + \sum_{i=1}^{O} \frac{n_i}{2}.P^i \tag{9}$$

since P^i and D are limited and also n^i is a finite number $P_{\text{tot}_{\text{non-convex}}}$ is also limited, paths generated by FGB method have an upper bound for non-convex shape obstacles. □

3.1.2. For convex obstacles

In convex shape obstacles, there are always two hit-points, one hit and one leave and therefore both directions reach a same hit-point, so:

$$p_L^i + p_R^i = P^i \tag{10}$$

therefore, p_L^i and p_R^i have lower values than P^i. Under the procedure of FGB method, one of the p_L^i and p_R^i directions should be selected according to their lengths. Thus, the selected direction needs to be shorter than the other direction, therefore:

$$P_{i_{\text{selected}}} = \min(p_{i_L}, p_{i_R}) \leq \frac{P_i}{2} \tag{11}$$

$$d_{\text{tot}_{\text{convex}}} = d_s + \sum_{i=1}^{O} [P_{i_{\text{selected}}} + d_{i,i+1}] + d_g \tag{12}$$

$$d_{\text{tot}_{\text{convex}}} \leq D + \sum_{i=1}^{O} P_{i_{\text{selected}}} = D + \sum_{i=1}^{O} \frac{P_i}{2} \tag{13}$$

$$d_{\text{tot}_{\text{convex}}} \leq D + \sum_{i=1}^{O} \frac{P_i}{2} \leq \sum_{i=1}^{O} P^i \tag{14}$$

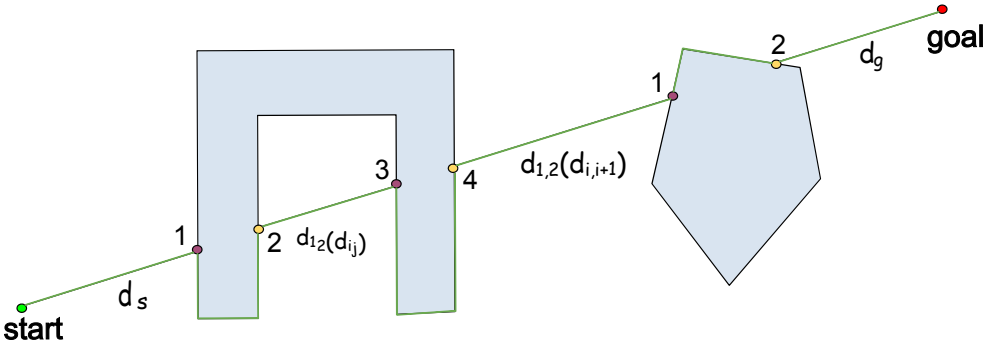


Figure 3: A sample of the path generated by FGB method.

Since P^i and D are limited, and $P_{\text{tot}_{\text{non-convex}}}$ is also limited, paths generated by the proposed method have an upper bound for convex shape obstacles. So, it's obvious that by putting $n_i = 2$ in Eq. (9), Eq. (14) can be reached; and the upper bound for non-convex objects with perimeter P^i is also an upper bound for convex objects with perimeter P^i. Therefore, as it is shown in Fig. 3, an upper bound for paths created by the FGB method, from the starting point to the target point, considering convex and non-convex obstacles, will be as follows:

$$d_{\text{tot}} = d_s + \sum_{i=1}^{O} \left(\left(\sum_{j=1, j=2k+1}^{n_i} P^i_{j_{\text{selected}}} + \sum_{j=2, j=2k}^{n_i} d_{i_j} \right) + d_{i,i+1} \right) + d_g \qquad (15)$$

Therefore, $D + \sum_{i=1}^{O} \frac{n_i}{2}.P_i$ is an upper bound for an environment with all of convex and non-convex shape obstacles and therefore convergence of paths generated by FGB method has been proved for all environments.

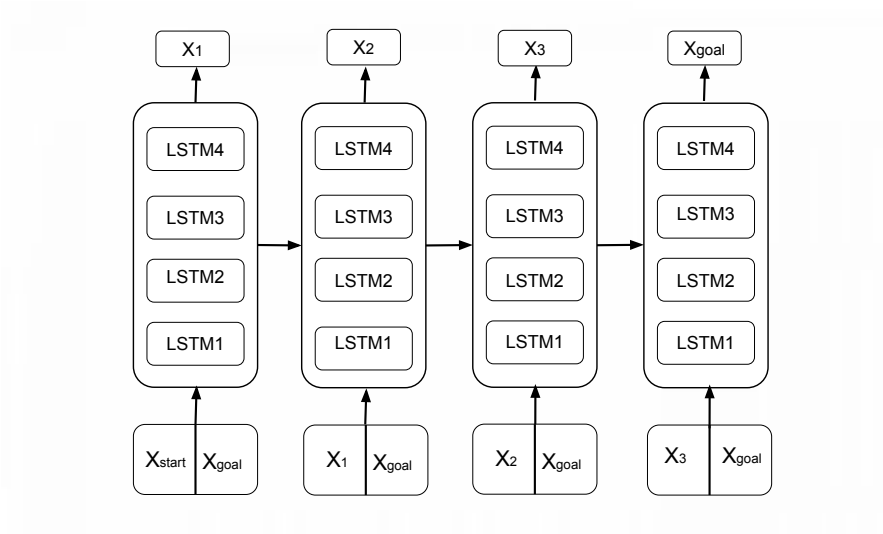


Figure 4: The LSTM network used for path planning [16].

4. Path Planning Network

The model used in this section is taken from OracleNet [16]. Each path generated in Section 3 is divided into their composing waypoints to form the training data. As shown in Fig. 4, the network consists of a number of stacked LSTM layers, each point of each path in the data, to find out where it should eventually converge, is connected to the goal point of that path and as the input vector in each layer of prediction is given to the network. The next point in the path is considered as the output of that layer and this process continues until it reaches the goal. The outputs are finally captured by the fully connected layers that are connected to the last hidden layer of LSTM.

The loss function in this network will be described in Section 4.2. Given that the data in question includes paths from all free space, the network can learn this sequence of points from a random start point to a random goal point, and

ultimately predict a near-optimal path.

4.1. Evaluation metrics

The proposed method is evaluated in different environments with different metrics such as Success Rate, Reach Goal, Safety, Generation Time, Smoothness and Path Length. These metrics are defined as follows. Success Rate is the average percentage of success of the model in generating obstacle-free paths. Reach Goal is the average percentage of success of the generated paths by the model in reaching the goal point. Safety shows the average distance of path points from the nearest obstacle. The higher this metric, the greater the distance of the generated path from the obstacles and the safer the path. Smoothness indicates the average angles of rotation of the robot along the path, the lower value of this metric indicates greater smoothness. Path Length is the average Euclidean distance of the path points generated by the model.

4.2. MSE-NER Loss Function

In order to train the proposed model, existing functions such as MSE can be used. However, using MSE causes an increase in the error of the predicted paths, and in some cases, the predicted path passes through obstacles. This issue occurs because of two reasons. First of all, in all available methods for creating data (RRT, FGB, A*), the probability of crossing the paths from the corners and the side of obstacles is high. Furthermore, MSE has no sense of the location of obstacles and it only tries to learn the input dataset by reducing the error, and due to the fact that the MSE error in the best case of network training is not zero and will still be a small value.

In order to solve the aforementioned problem, a new loss function is proposed which will be explained below. As mentioned, MSE Loss Function has no information about the location of the obstacles, and this issue along with the items mentioned above, will increase the network predicated error. In order to solve this problem, in the loss function, in addition to learning the input data, the network is trained to avoid obstacles. In this way, the nearest point on the nearest side of the nearest obstacle will be extracted then the network will be trained in such a way that in addition to learning the input dataset, it moves the predicted points away from the nearest obtained point on the nearest edge of the nearest obstacle, so this proposed method is called Mean Squared Error-Nearest Edge Repulsive (MSE-NER). After calculating the nearest point on all sides of the obstacles from the desired point, the distances of these points to the mentioned point are calculated and the shortest distance is considered as the smallest distance to the obstacles, min_{dist}, and the corresponding point as the closest point, y_{obs}, is considered among all obstacles.

As aforementioned, in this loss function, in addition to the attractive term which brings the predicted points by the network closer to the points on the path in the data, there is also a repulsive term which moves the predicated point away from the nearest point on the obstacles. However, in cases where the predicted point is inside the obstacle, the repulsive term directs it further into the obstacle, causing more error. In order to fix this error, another term is added to the repulsive term, which by examining the predicted points, brings the predicted points inside the obstacles closer to the points on the path in the data. As a result, the repulsive term is composed of two separate terms, one of which moves the predicted points out of the obstacles from the nearest point to the nearest obstacle, which is called

the external repulsive term, and the other term that brings the predicted points inside the obstacles closer to the points in the input data is called the internal repulsive term.

For this purpose, the desired function of the attractive term is defined as attractive $= ||y_{\text{pred}} - y_{\text{true}}||^2$, which is in fact, the Euclidean distance between the predicted point by the network and the input data, and so is the external repulsive term is defined as external-repulsive $= ||y_{\text{pred}} - y_{\text{obs}}||^2$, which is in fact, the Euclidean distance between the predicted point by the network and the point closest to it on the obstacles and the internal repulsive term is defined as internal-repulsive $= ||y_{\text{pred}} - y_{\text{true}}||^2$, which is basically the same term as the attractive term with a different coefficient. The coefficients of these terms will be explained below. Finally, for each batch of data, the loss function is defined as follows:

$$L_{\text{MSE-NER}} = \frac{1}{N}\sum_{i=0}^{N}[||y_{\text{pred}} - y_{\text{true}}||^2 - L_{\text{out}} \times ||y_{\text{pred}} - y_{\text{obs}}||^2 + L_{\text{in}} \times ||y_{\text{pred}} - y_{\text{true}}||^2 + \gamma] \tag{16}$$

where N is the number of data in each batch and γ is a coefficient which adjusts the margin around the obstacles. Also, L_{out} and L_{in} are the external and the internal term coefficients, respectively.

For the external term coefficient, after calculating the nearest point of the nearest obstacle (y_{obs}), its distance from the point on the path in the dataset (y_{true}) is calculated, which is called $dist_{\text{out}}$ and the desired coefficient is defined as follows:

$$L_{\text{out}} = p_{\text{out}} \times \frac{(\alpha^{(-dist_{\text{out}}+1)})^2}{\alpha} \tag{17}$$

where P_{out} indicates points that are predicted outside the obstacles. The closer data points are to the obstacles, the coefficient is higher, and the farther the data points

are from the obstacles, L_{out} is lower. For the internal term coefficient, the distance of the points predicted by the network to the nearest point of the nearest obstacle is obtained, which is called $dist_{\text{in}}$ and the coefficient is defined as follows:

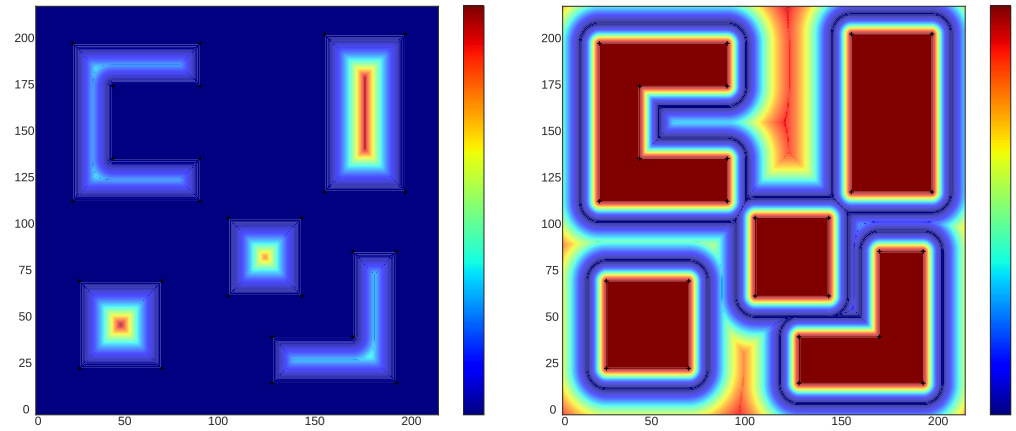
$$L_{\text{in}} = p_{\text{in}} \times (\alpha - \alpha^{(-dist_{\text{in}}+1)}) \tag{18}$$

in the above, P_{in} indicates points that are predicted inside the obstacles. The more network predicted point goes into the obstacle, the coefficient is higher and the less the predicted point goes into the obstacle, L_{in} value is lower.

As illustrated in Fig. 5a, if the predicted points are place into the obstacles, the internal term value of the loss function will increase. Also, Fig. 5b represents that as far as the predicted points are from the edges of the obstacle, the external term value decreases until it reaches its minimum. This value represents the margin, and then the value of the term increases by moving farther from the margin. The mentioned Loss Function will be compared with MSE Loss in the following sections.

5. Results and Discussion

The main purpose of navigation with the proposed method is to find a safe and obstacle-free path in a short time, from a random starting point to a random goal point. In this section, a comparison between different methods of data creation and also network training in various environments has been done. In other words, the results of the evaluation of the proposed method are presented and it is compared with the classical motion planners, namely, RRT*, A* and the classical loss function: MSE. A major advantage of the proposed method is the high speed of generating the data required for network training and in addition generating a



(a) The internal term value of the loss function. (b) The external term value of the loss function.

Figure 5: Heat-maps of external and internal terms of the loss function in a sample environment. As the color of the area in the heat-map goes from blue to red, the value of the loss function increases at that area. Also, the corners of the obstacles are indicated by a plus sign.

safe path. First, the performance of the data creation method is evaluated and the ability of the proposed model to generate safe paths is shown in Fig. 6, an environment with convex obstacles. In the following, two additional tests are exerted to evaluate the ability of the proposed model to create safe paths in environments with non-convex obstacles shown in Fig. 7. Finally, the proposed method is implemented in one of the indoor 3D scenes of the AI2-THOR framework shown in Fig. 8, which shows major improvements in the success rate and safety of navigation in scenes with both convex and non-convex obstacles.

5.1. Environment with convex obstacles

As mentioned, in this section, the desired environment is considered as a simple 2D environment with convex obstacles, and the desired robot is also considered as a 2D point-mass. Then by using RRT and FGB, 40000 data is generated

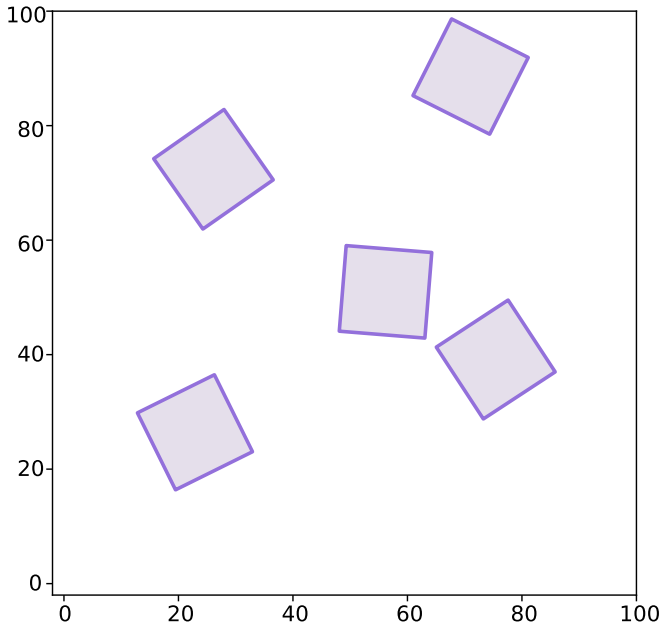
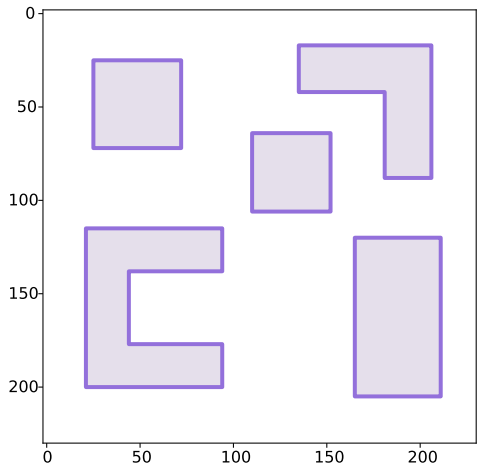


Figure 6: An environment with convex obstacles.

for network training. Some examples of the data are represented in Fig. 9, and the average path generation time and the average length of paths created with these two methods can be seen in Table 1. As it can be seen, the creation time of paths by FGB method is at least 200 times faster than the RRT method. In addition, the length of paths produced by this method is less than the RRT method, and as a result, the paths produced by this method will be closer to the optimal path.

The mentioned model is trained with the MSE and MSE-NER loss function with $\alpha = 20$ and $\gamma = 500$, which an example of a generated path by the four trained models is shown in Fig. 10. The results reveal that the model trained with MSE loss function only tries to learn the input data, and because this network has no sense of the location of the obstacles, the paths generated by it passed through the corners of the obstacles, which reduces the Success Rate of the model. But the



(a) A simple environment with both convex and non-convex obstacles.

(b) Top view of the Human-Robot Interaction Laboratory, University of Tehran.

Figure 7: Environments with non-convex obstacles.

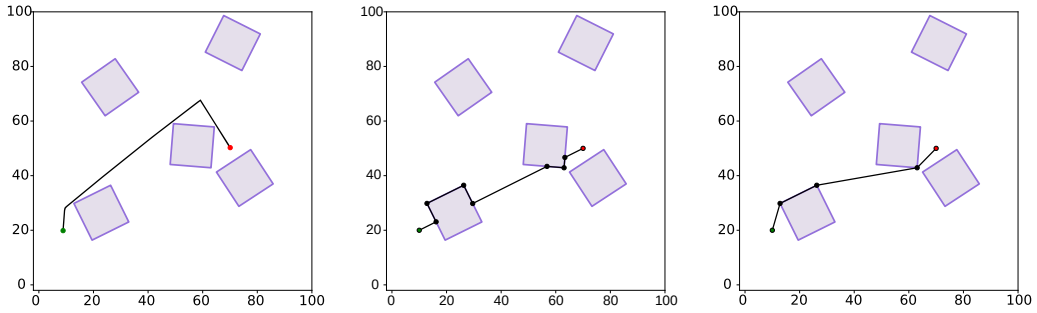
models trained with MSE-NER loss function, in addition to being able to learn the input data, have also moved away from the corner of obstacles, and although the input data has passed through the corner of obstacles, the predicted paths have created a good margin around the obstacles.

Then Success Rate, Reach Goal and Path Generation Time of 100 random paths generated with 4 trained models are calculated and compared in Table 2. As it is shown, the model trained with FGB data and MSE-NER loss function performs very well and compared to the other three models, can well create a safe margin around obstacles. This has increased the success rate of the model. From the results, it can be concluded that by minimizing MSE-NER loss function,

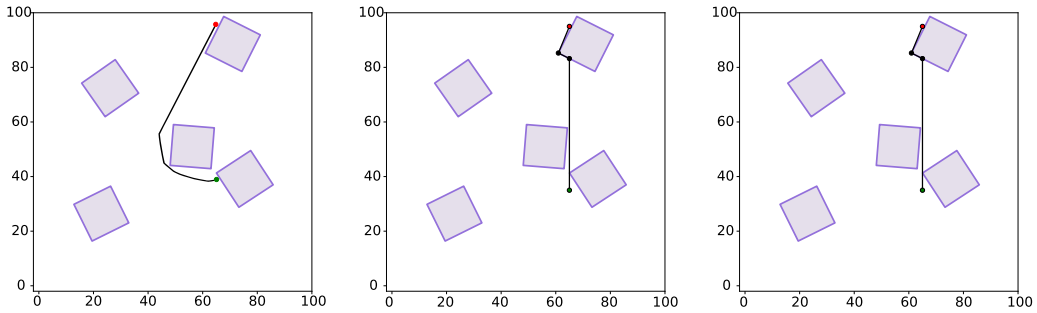
Figure 8: AI2-THOR Framework.

the network was able to learn the input data well and in addition create a margin around the obstacles. In other words, the network has been able to produce safe paths without any obstacles for any random starting point and target, in a short and fixed period of time. However, there is a problem of not reaching the exact goal point for some paths. As mentioned, for testing the trained network, two random points from free space are selected and given to the network as the start and goal points. Then the network starts from the starting point and predicts the next points of the path in succession. This continues until the last predicted point is in a small neighborhood of the goal point. In this case, a number of paths do not reach exactly the target point and the algorithm may stop near the target point, thus reducing the Reach Goal. To make the path generating process more robust, bi-directional path generation is used.

To solve this problem, the production of the route is done in two directions. In other words, in addition to generating a path from the starting point to the



(a) RRT (b) FGB before rewiring (c) Rewired FGB



(d) RRT (e) FGB before rewiring (f) Rewired FGB

Figure 9: Samples of paths created using RRT and FGB in the environment map of Fig. 6.

target point, a path will also be generated in parallel from the target point to the starting point. This process continues until the two branches meet, and then the branches are combined to form a complete path. In this way, instead of forcing the network to generate a path from the start point toward a fixed goal point, a constantly changing target point can be used as a new goal point, in other words, the current point of another branch as a new target point is considered, and at each stage, the path is produced to a new goal. In this way, without increasing the time of the path creation, the desired path will be produced from the starting point to the exact target point, and the Reach Goal will increase.

Table 1: Comparison of generating 40000 paths for the same starting and target points in the environment map of Fig. 6.

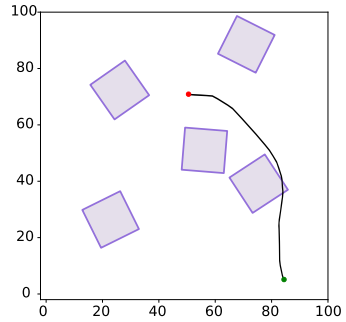
Method	**Average length of generated paths**	**Average time of generating paths**
FGB	53.85	0.015 (s)
RRT	57.06	3.635 (s)

Table 2: Performance comparison of the models trained with RRT and FGB data and using MSE and MSE-NER loss functions for the same starting and target points in the environment map of Fig. 6.

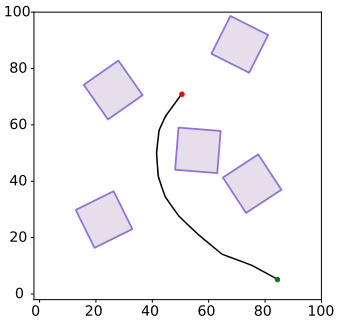
Data Creation Method	**Loss function**	**Success Rate**	**Reach Goal**	**Time of path generating**
RRT	MSE	78%	98%	0.8(s)
RRT	MSE-NER	90%	98%	0.52(s)
FGB	MSE	72%	98%	0.6(s)
FGB	MSE-NER	99%	98%	0.74(s)

5.2. Environment with Non-convex obstacles

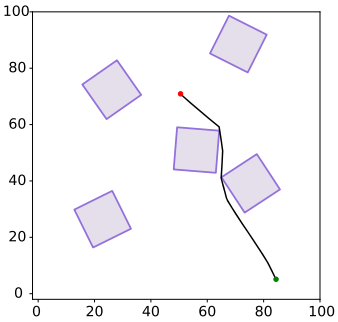
In this section, three non-convex environments are used to evaluate the proposed methods. First, in a simple non-convex environment, the proposed data creation method is compared with existing methods, and then the model trained with MSE-NER loss function is evaluated in this environment and compared with the model trained with MSE loss function and RRT global path planning method. Then, in the Human-Robot Interaction Laboratory at the University of Tehran, the



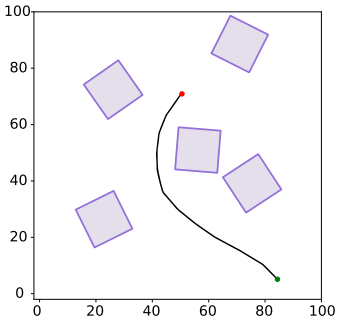
(a) Trained with RRT data and the MSE loss function.



(b) Trained with RRT data and the MSE-NER loss function.



(c) Trained with FGB data and the MSE loss function.



(d) Trained with FGB data and MSE-NER loss functions.

Figure 10: Paths generated by model trained with RRT and FGB data and using MSE and MSE-NER loss function for the same starting and target points in the environment map of Fig. 6.

proposed methods are evaluated. Finally, the proposed methods are evaluated in a three-dimensional environment.

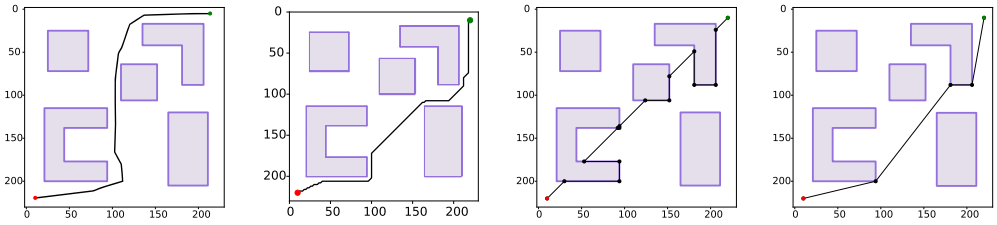
5.2.1. Simple Non-convex environment

The comparative experiments are expanded to evaluate MSE-NER loss function performance against MSE in a more complex environment. Therefore, an environment is considered with convex and non-convex U-shaped and L-shaped

obstacles and after preparing the dataset, the network is trained with MSE-NER and MSE loss functions.

Due to the fact that classical data creation methods such as RRT and A* have a long time to generate this amount of data, and in addition, as the obstacles become more complex and the dimensions of the environment increase, the data creation speed of these algorithms decreases, resulting in complex environments it is not possible to generate all data with classical methods. For this reason, in this section, for the environment consisting of convex and non-convex obstacles, shown in Fig. 7a, 100,000 random paths have been created using FGB methods, which will be used to train the desired neural network and for example 100 random path created by FGB, RRT and A* methods which some examples are illustrated in Fig. 11. The average Path Generation Time and the average length of paths created with these 3 methods can be seen in Table. 3. As it can be deduced from the latter table, despite the larger dimensions of the environment than the first environment and as a result of increasing the average length of the created paths, the paths produced by the FGB method still have a shorter average length than the A* and RRT methods. In addition, the path creation time with the FGB method, in this environment, is at least 100 times less than the A* method and at least 500 times less than the RRT method.

Then, the introduced model is trained with FGB data and the MSE and MSE-NER loss functions with $\alpha = 20$ and $\gamma = 1000$. An example of the paths generated by these two networks and RRT method is shown in Fig. 12 and the Success Rate, Reach Goal, Path Generation Time, Safety, Smoothness and Path Length of these two models are compared with each other and with the classical RRT methods in Table. 4.

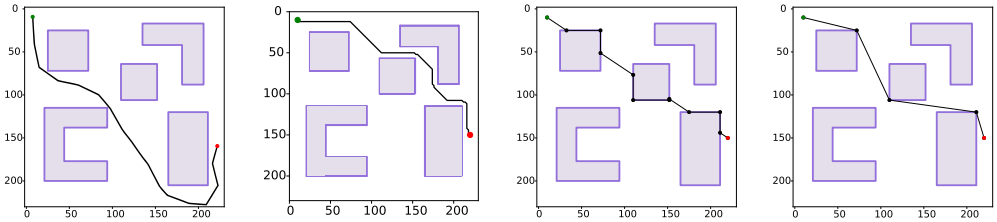


(a) path created by RRT

(b) path created by A*

(c) path created by FGB before rewiring

(d) rewired path created by FGB



(e) path created by RRT

(f) path created by A*

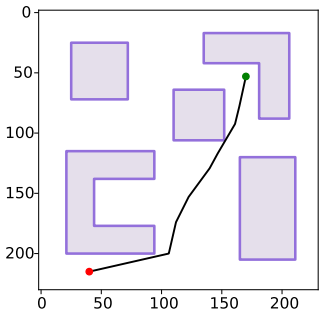
(g) path created by FGB before rewiring

(h) rewired path created by FGB

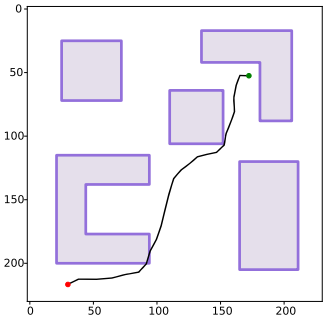
Figure 11: Sample of paths created by A*, RRT and FGB methods in the environment map of Fig. 7a.

Table 3: Comparison of generating 100 paths for the same starting and goal points in the environment map of Fig. 7a.

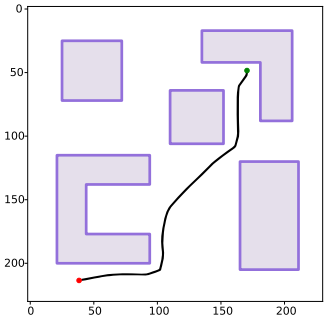
Method	**Average length of generated paths**	**Average time of generating paths**
FGB	125.38	0.0344 (s)
RRT	172.2	18.43 (s)
A*	135.43	5.065 (s)



(a) RRT method



(b) trained with FGB data and the MSE loss function



(c) trained with FGB data and MSE-NER loss functions

Figure 12: Comparison of paths generated by model trained with FGB data and using MSE and MSE-NER loss function and also RRT method for the same starting and target points in the environment map of Fig. 7a.

It can be seen that the trained network with MSE loss function does not work well in more complex environments because the training data passes through the corners of the obstacles and also the MSE loss function does not know the location of the obstacles. Even a small error in the desired loss function causes the generated paths to cross the corner of the obstacles and the network success rate is low. However, the training model with MSE-NER loss function creates a safe margin around the obstacles and therefore the success rate of the network is high.

Table 4: Performance comparison of the models trained with FGB data and using MSE and MSE-NER loss functions and also RRT method for the same starting and target points in the environment map of Fig. 7a.

Method	**Loss function**	**Success Rate**	**One-directional Reach Goal**	**Bi-directional Reach Goal**	**Average time of generating paths**	**Average length of generated paths**	**Smoothness**	**Safety**
FGB	MSE	49%	98%	100%	0.569(s)	130.86	11.51	11.62
FGB	MSE-NER	99%	98%	100%	0.571(s)	131.2	8.88	13.23
RRT	-	100%	100%	-	17.4(s)	172.7	12.11	12.91

The RRT method also works well for creating safe paths, which is confirmed by the Success Rate shown in Table 4.

Also, in the paths predicted with the network trained with MSE, although the Reach Goal and Path Generation Time are almost similar to the network trained with MSE-NER, the length of the generated paths was slightly longer since the paths are farther from the obstacles. However, in the RRT method, unlike the high Success Rate and the Reach Goal, the average Path Generation Time and the length of the generated paths are higher than in the other two methods, which proves the non-optimization of the RRT in terms of path generation time and path lengths. The safety of the paths generated by the model using the MSE-NER loss function is also determined according to Table 4, and as illustrated, the paths

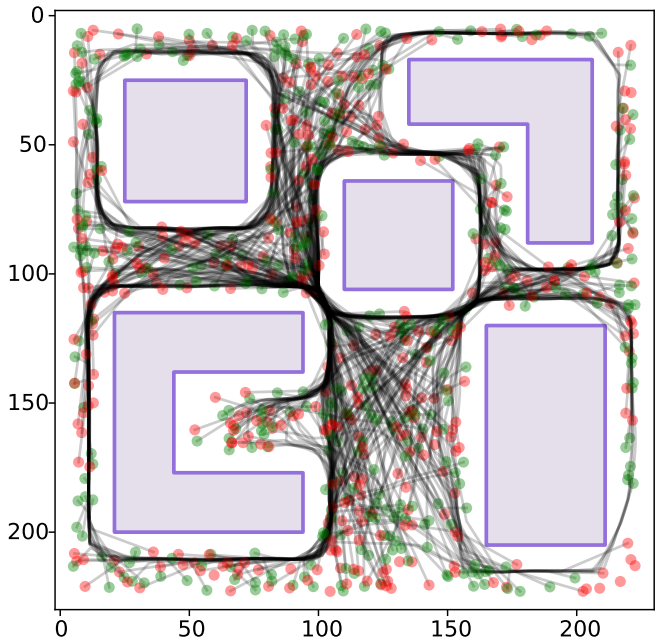


Figure 13: 400 paths created by the trained model with FGB data and the MSE-NER loss function in the environment map of Fig. 7a.

generated by the RRT method are safer than the paths generated by the model trained with the MSE loss function. In addition, the paths generated by the model using the MSE-NER loss function are smoother than the paths generated by the other two methods. Finally, 400 examples of paths created by the trained model with the MSE-NER loss function are given in Fig. 13 which demonstrates the safety of the proposed loss function compared to the classical loss function.

5.2.2. Real Environment

In order to obtain a map of a real environment, using the LiDAR sensor, the environment of the Human-robot Interaction Laboratory at the University of Tehran was scanned and its three-dimensional map was obtained. Then, to obtain the

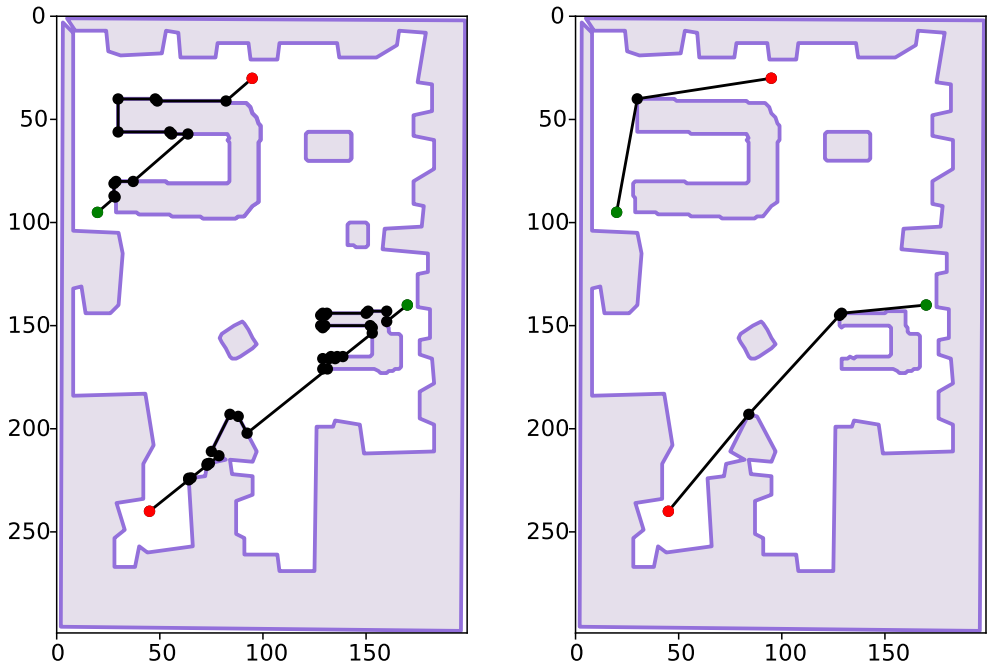
Table 5: Performance of FGB method in generating 100000 in the environment map of Fig. 7b.

Method	**Average length of generated paths**	**Average time of generating paths**
FGB	142.1	0.0271 (s)

required map, the top view is used, which is also shown in Fig. 7b. Then, using edge detection algorithms, the edges of obstacles are determined and estimated with polygons. Finally, the coordinates of the corners of the polygonal obstacles will be stored in an array.

After obtaining the environment map of the Fig. 7b, the desired training data for this environment will be created. As shown in the previous section, classical methods have weak performance than the FGB method and their speed decreases exponentially as the complexity and dimensions of the environment increase, and they are not suitable for creating large amounts of data in complex environments. For this reason, in this section, the required data is created only with the FGB method which examples of this are demonstrated in Fig. 14. The average length of the generated paths and the time of data creation with the FGB method are stated in Table 5.

In this section, the introduced model with FGB data and MSE-NER loss function is trained. Examples of paths generated by the trained model are shown in Fig. 15. Also, the Success Rate, Reach Goal and the average Time of the Path Generation for this model are reported in Table 6. It can be observed that the trained model has a good performance and can produce safe, smooth and collision-free paths in a short time and create a safe margin around obstacles.



(a) 2 paths created by FGB method before rewiring.

(b) 2 rewired paths created by FGB method

Figure 14: Samples of paths created by FGB method before and after rewiring in the environment map of Fig. 7b.

Table 6: Performance of the model trained with FGB data and MSE-NER loss function in the environment map of Fig. 7b.

Method	**Loss function**	**Success Rate**	**One-directional Reach Goal**	**Bidirectional Reach Goal**	**Average time of generating paths**
FGB	MSE-NER	98%	97%	100%	0.81(s)

5.2.3. Simulation Environment

Finally, in order to evaluate the model a framework for path planning in a 3D environment is needed. One of the newest platforms is the AI2THOR platform,

(a) (b) (c)

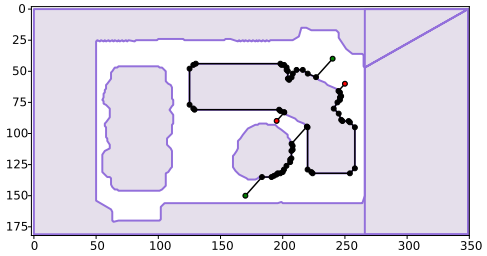
Figure 15: Samples of paths generated by model trained with FGB data and MSE-NER loss function in the environment map of Fig. 7b.

which has been able to simulate the real environment well. Therefore the last environment in which the developed methods have been tested is the AI2THOR simulation environment. As mentioned, this environment is an interactive platform simulated from real environments designed using deep learning. In this simulation environment, an example of the most complex environments of this platform is selected, which is shown in Fig. 8. By moving in the environment, the robot takes all the feasible points, and thus the environment map shown in Fig. 16 is created.

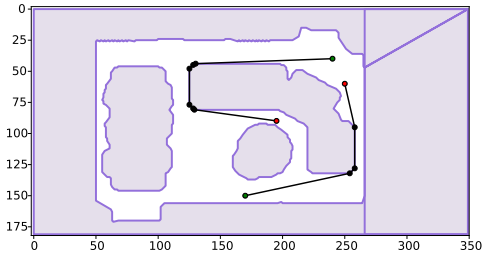
In this section, the required 100,000 data are generated using only the FGB algorithm. Examples of these paths are shown in Fig. 17. Also, the average length of the generated paths and the average creation time of this number of data with the FGB method are shown in Table 7.

In the following, the final model with FGB data and MSE-NER loss function

Figure 16: The environment map of Fig. 8.



(a) 2 paths created by FGB method before rewiring



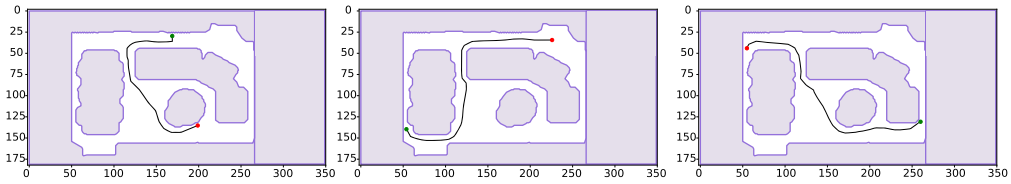
(b) 2 rewired paths created by FGB method

Figure 17: Samples of paths created by FGB method before and after rewiring in the environment map of Fig. 8.

for this environment is trained. After saving the trained model, its performance will get evaluated for random start and goal points. Examples of paths generated by this method are shown in Fig. 18. Finally, the Success Rate, Reaching Goal and the average Data Generation Time for this model are stated in Table 8. It can be seen that a well-trained network can create a safe margin around obstacles and also the generated paths are smooth and sub-optimal and the paths are generated in the shortest possible time.

Table 7: Performance of FGB method in generating 100000 in the environment map of Fig. 8.

Method	**Average length of generated paths**	**Average time of generating paths**
FGB	117.29	0.057 (s)



(a) First Sample (b) Second Sample (c) Third Sample

Figure 18: Samples of paths generated by model trained with FGB data and MSE-NER loss function in the environment map of Fig. 8.

Table 8: Performance of the model trained with FGB data and MSE-NER loss function in the environment map of Fig. 8.

Method	**Loss function**	**Success Rate**	**One-directional Reach Goal**	**Bidirectional Reach Goal**	**Average time of generating paths**
FGB	MSE-NER	97%	95%	100%	1.2(s)

6. Conclusion

This study introduced an approach to generate paths from any random starting point to any random target in the environment. These paths can be used as the data needed for training the neural networks in a short time, contrary to conventional methods. In this paper, a global path planning method, which is called FGB, was

proposed for providing the training data. Finally, in order to evaluate the performance of the proposed path planning method, for the various environments under study, a number of routes with the proposed method as well as common methods such as RRT and A* were produced. The results illustrated that the average time of path creation with FGB method is at least 100 times faster than conventional path planning methods. In addition, in this study, in order to solve the issues of MSE loss function for training a recurrent neural network for path planning, a new loss function, MSE-NER, was proposed. This loss function tries to move away from the boundaries of obstacles while learning the input paths. Thus, besides creating an adjustable safe margin around obstacles, it led to real-time execution. This ease the implementation of path planning for robots in real environments. The performance of the proposed model was evaluated in a simple environment with convex obstacles and two more complex environments with non-convex obstacles. It was observed that the recurrent neural network trained with MSE-NER loss function performed better than the model trained with MSE loss function and also the RRT method. The results showed that the proposed algorithm is able to produce safer routes, in a shorter time with a higher success rate than the other two methods. Finally, the proposed algorithm was implemented in the AI2THOR simulator environment and it was shown that this algorithm is completely operational and can be implemented in real environments. The performance of the project can be increased by adapting to dynamic environments and unseen environments which can obtain a complete path planning system. There are some new studies which by utilizing an Autoencoder model and also using images of environment map as input can generate paths for each environment without training the model especially for that environment. By combining the mentioned studies

and the approach proposed in this paper a new method can be obtained which has both methods' advantages.

References

[1] J. R. Sánchez-Ibáñez, C. J. Pérez-del Pulgar, A. García-Cerezo, Path planning for autonomous mobile robots: A review, Sensors 21 (23) (2021) 7898.

[2] R. S. Nair, P. Supriya, Robotic path planning using recurrent neural networks, in: 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT), IEEE, 2020, pp. 1–5.

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

[4] G. E. Mathew, Direction based heuristic for pathfinding in video games, Procedia Computer Science 47 (2015) 262–271.

[5] Z. Zhang, Z. Zhao, A multiple mobile robots path planning algorithm based on a-star and dijkstra algorithm, International Journal of Smart Home 8 (3) (2014) 75–86.

[6] A. Mohammadi, M. Rahimi, A. A. Suratgar, A new path planning and obstacle avoidance algorithm in dynamic environment, in: 2014 22nd Iranian Conference on Electrical Engineering (ICEE), IEEE, 2014, pp. 1301–1306.

[7] E. R. Firmansyah, S. U. Masruroh, F. Fahrianto, Comparative analysis of a and basic theta algorithm in android-based pathfinding games, in: 2016 6th

International Conference on Information and Communication Technology for The Muslim World (ICT4M), IEEE, 2016, pp. 275–280.

[8] S. M. LaValle, et al., Rapidly-exploring random trees: A new tool for path planning (1998).

[9] S. K. Gan, K. J. Yang, S. Sukkarieh, 3d online path planning in a continuous gaussian process occupancy map, in: Proc. Australian Conf. Field Robotics, 2009.

[10] A. Shkolnik, M. Walter, R. Tedrake, Reachability-guided sampling for planning under differential constraints, in: 2009 IEEE International Conference on Robotics and Automation, IEEE, 2009, pp. 2859–2865.

[11] K. Yang, S. Moon, S. Yoo, J. Kang, N. L. Doh, H. B. Kim, S. Joo, Spline-based rrt path planner for non-holonomic robots, Journal of Intelligent & Robotic Systems 73 (1) (2014) 763–782.

[12] A. H. Qureshi, Y. Miao, A. Simeonov, M. C. Yip, Motion planning networks: Bridging the gap between learning-based and classical motion planners, IEEE Transactions on Robotics 37 (1) (2020) 48–66.

[13] A. Shantia, R. Timmers, Y. Chong, C. Kuiper, F. Bidoia, L. Schomaker, M. Wiering, Two-stage visual navigation by deep neural networks and multi-goal reinforcement learning, Robotics and Autonomous Systems 138 (2021) 103731.

[14] Q.-L. Li, Y. Song, Z.-G. Hou, Neural network based fastslam for autonomous robots in unknown environments, Neurocomputing 165 (2015) 99–110.

[15] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, A. Farhadi, Target-driven visual navigation in indoor scenes using deep reinforcement learning, in: 2017 IEEE international conference on robotics and automation (ICRA), IEEE, 2017, pp. 3357–3364.

[16] M. J. Bency, A. H. Qureshi, M. C. Yip, Neural path planning: Fixed time, near-optimal path generation via oracle imitation, in: 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2019, pp. 3965–3972.

[17] E. Kolve, R. Mottaghi, W. Han, E. VanderBilt, L. Weihs, A. Herrasti, D. Gordon, Y. Zhu, A. Gupta, A. Farhadi, Ai2-thor: An interactive 3d environment for visual ai, arXiv preprint arXiv:1712.05474 (2017).

[18] Y. K. Hwang, N. Ahuja, et al., A potential field approach to path planning., IEEE Transactions on Robotics and Automation 8 (1) (1992) 23–32.

[19] M. A. Basiri, S. Chehelgami, E. Ashtari, M. T. Masouleh, A. Kalhor, Synergy of deep learning and artificial potential field methods for robot path planning in the presence of static and dynamic obstacles, in: 2022 30th Iranian Conference on Electrical Engineering (ICEE), 2022.

[20] V. Lumelsky, A. Stepanov, Dynamic path planning for a mobile automaton with limited information on the environment, IEEE transactions on Automatic control 31 (11) (1986) 1058–1063.

[21] V. J. Lumelsky, A. A. Stepanov, Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape, Algorithmica 2 (1) (1987) 403–430.

[22] C. Hernández, J. Baier, R. Asín, Making a* run faster than d*-lite for path-planning in partially known terrain, in: Proceedings of the International Conference on Automated Planning and Scheduling, Vol. 24, 2014, pp. 504–508.

[23] B. Fu, L. Chen, Y. Zhou, D. Zheng, Z. Wei, J. Dai, H. Pan, An improved a* algorithm for the industrial robot path planning with high success rate and short length, Robotics and Autonomous Systems 106 (2018) 26–37.

[24] L. Cheng, C. Liu, B. Yan, Improved hierarchical a-star algorithm for optimal parking path planning of the large parking lot, in: 2014 IEEE International Conference on Information and Automation (ICIA), IEEE, 2014, pp. 695–698.

[25] C. Yuan, G. Liu, W. Zhang, X. Pan, An efficient rrt cache method in dynamic environments for path planning, Robotics and Autonomous Systems 131 (2020) 103595.

[26] L. Huang, Velocity planning for a mobile robot to track a moving target—a potential field approach, Robotics and Autonomous Systems 57 (1) (2009) 55–63.

[27] N. Ahuja, J.-H. Chuang, Shape representation using a generalized potential field model, IEEE Transactions on Pattern Analysis and Machine Intelligence 19 (2) (1997) 169–176.

[28] K. N. McGuire, G. C. de Croon, K. Tuyls, A comparative study of bug algorithms for robot navigation, Robotics and Autonomous Systems 121 (2019) 103261.

[29] S.-K. Kim, J. S. Russell, K.-J. Koo, Construction robot path-planning for earthwork operations, Journal of computing in civil engineering 17 (2) (2003) 97–104.

[30] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural computation 9 (8) (1997) 1735–1780.

[31] T. Hastie, R. Tibshirani, J. H. Friedman, J. H. Friedman, The elements of statistical learning: data mining, inference, and prediction, Vol. 2, Springer, 2009.