sciendo

VOLUME 16, N° 1 2022
Journal of Automation, Mobile Robotics and Intelligent Systems

A Scalable Tree Based Path Planning For A Service Robot

Submitted: 23rd November 2021; accepted: 8th February 2022

A. A. Nippun Kumaar, Sreeja Kochuvila, S. R. Nagaraja

DOI: 10.14313/JAMRIS/1-2022/4

Abstract:
Path planning plays a vital role in a mobile robot navigation system. It essentially generates the shortest traversable path between two given points. There are many path planning algorithms that have been proposed by researchers all over the world; however, there is very little work focussing on path planning for a service environment. The general assumption is that either the environment is fully known or unknown. Both cases would not be suitable for a service environment. A fully known environment will restrict further expansion in terms of the number of navigation points and an unknown environment would give an inefficient path. Unlike other environments, service environments have certain factors to be considered, like user-friendliness, repeatability, scalability, and portability, which are very essential for a service robot. In this paper, a simple, efficient, robust, and environment-independent path planning algorithm for an indoor mobile service robot is presented. Initially, the robot is trained to navigate to all the possible destinations sequentially with a minimal user interface, which will ensure that the robot knows partial paths in the environment. With the trained data, the path planning algorithm maps all the logical paths between all the destinations, which helps in autonomous navigation. The algorithm is implemented and tested using a 2D simulator Player/Stage. The proposed system is tested with two different service environment layouts and proved to have features like scalability, trainability, accuracy, and repeatability. The algorithm is compared with various classical path planning algorithms and the results show that the proposed path planning algorithm is on par with the other algorithms in terms of accuracy and efficient path generation.

Keywords: *Learning from Demonstration, path mapping, path planning, navigation system, mobile robot, service robot*

1. Introduction

Mobile service robots have been getting popular in the last decade in the consumer electronics field. They are used in service environments like homes, offices, hospitals, hotels, museums, etc. The major reason for this popularity is that they are operated autonomously and help users with day-to-day tasks. An inherent part of any kind (wheeled, legged, airborne, etc.) of a mobile service robot is the navigation system. A navigation system is a key element in making a service robot autonomous. It helps the robot to navigate from a given source to a destination autonomously. A navigation system is composed of subsystems like localization, which locates the robot in the environment, path planning, used to plan the path between two points, obstacle avoidance, to navigate in a dynamic environment, etc. Overall, autonomous service robots can be classified into two types: environment-dependent, and environment independent. Environment-dependent robots use onboard sensors (sensors in the robot's chassis) and off-board sensors (sensors placed in the environment) for navigation. Environment independent robots use only onboard sensors for navigation and thus environment portability is possible.

The service environment demands that the robot traverse a path repeatedly, increasing the number of locations drastically and that even a novice user should be able to handle the robot and move from one service environment to another without any expert help. In this work, a path-planning architecture for a wheeled mobile robot deployed in a service environment with all the aforementioned key factors is proposed. The proposed algorithm is generic in nature, which caters to the needs of various service environments. Service environments targeted are those with large carpeted area, devoid of any kind of environment augmentation. Overall work is divided into two phases: the Learning from Demonstration (LfD) phase and the path planning phase. In the LfD phase, the robot is taught all the possible destinations sequentially with a minimal user interface, which gives a partial understanding of the paths available. In the path planning phase, a complete path tree is generated by mapping all the possible logical paths among possible destinations and navigating in the shortest path possible autonomously. These logical paths are totally based on the learnt paths in the LfD phase.

The remainder of the article is organized as follows. Section 2 highlights the state of art in the LfD and path-planning field. The proposed LfD and path-planning algorithm are described in section 3. Section 4 presents the result and analysis, and a comparison with various classical approaches. Section 5 concludes the work.

2. Related Work

Research on LfD in robotics started in the early 1970's and has grown significantly during the past decade [19]. The main aim for LfD is to train a robot to perform a task rather than programming it. Main advantages are user-friendliness and environmental

portability. To program a robot, an expert is required, whereas to teach a robot, a novice user will be able to do it. The robot can be ported to any environment with an overhead of just training for the new environment; this would reduce both time and cost. Initially, the robot is manually trained by a human to perform a task. Training can be done using various tools like teaching pendant [3] camera-based learning, or imitation learning [14]. Most of the initial work in robotics LfD is for industrial manipulators [13, 25]. For the past decade, LfD has been used in mobile robots to perform specific navigation tasks like traversing a corridor or passing a doorway [32, 40].

The robot learns by storing the on-board and off-board sensors' data, and it is used to perform the learnt task autonomously. There have been a lot of research advancements on processing the learnt data and using it efficiently in autonomous mode. In [8], a nonlinear regression model to identify the relationship between onboard sensors and actuators while training is proposed. A set of library tasks is taught, and these library tasks are used in the autonomous phase in [13]. In article, [6], feedback in terms of the advice operator is given to the robot by the user based on task learned. This allows the trainer to correct the robot in real time. While in all the aforementioned literature, the trainer and the robot are in the same place, in [36], a remote training algorithm using the Web is proposed. A tour guide robot in [2] is trained to all possible paths by following a trainer and stores the poses as route parameters. In [17], a different approach of training a robot in a static environment and execution in a dynamic environment is proposed.

The LfD field has developed rapidly since its inception, but there is no one general technique that can be used in different domains of robotics. Most approaches are tested using only a single domain with one or a few tasks on a single robotic platform. This is the primary reason for the lack of comparisons between algorithms. Most of the work done in LfD is on industrial manipulators. It is not possible to use the same technique in mobile robots, as the learning parameters and teaching approach differ for both platforms. Existing LfD systems heavily depend upon offboard sensors like the camera, wearable sensors for trainer, etc., which is not suitable for a service environment.

Path planning is an active research area with a lot of scope for advancements. Earlier path planning algorithms have been used in the fields of computer networks, circuit board design, etc. Lately they have been extensively used in the field of mobile robotics. Path planning is identifying the shortest path between a given source and destination. Overall path planning techniques can be categorized based on the environmental awareness of the robot, with known, unknown, and partially known environments [31]. The main prerequisite for known-environment path planning is that the environment map has to be preloaded, no dynamic changes in the environment are possible, and that it will plan the shortest path available. In unknown-environment path planning, the robot perceives the environment using sensors and decides upon the shortest path, and this can work in dynamic environments. Partially known environment path planning is a combination of both the aforementioned types; this works in a dynamic environment, and at the same time the shortest possible path is identified.

Classical heuristic search path planning algorithms like Dijkstra's [11] and A* [21] are based on path planning with known environments. A* is based on Dijkstra's algorithm, with a more focussed search towards optimal states. These algorithms can identify an optimal path between the start and endpoint. While these algorithms work well in a static known environment, they fall short in dynamic unknown environments where the path is planned based on the onboard sensors in real time. Algorithms like the motor schema-based approach [4,23], potential field method [24], improved potential field method [27], artificial potential field method [43], modified flexible vector field method [22] are used in some of the earlier literature for unknown environments. Techniques like D* [41] and Field D* [15] belong to this category. The D* algorithm is based on the A* algorithm but with dynamic replanning capability. Algorithms like conjugate gradient descent, improved potential field, or D* Lite [12, 46, 47] use known path planning algorithms like A* deliberative planning technique and use replanning locally if required.

Some techniques can't be categorized into one of the aforementioned classes. Visibility binary tree [39] is one such technique that can be used in known or unknown environments. This method creates a binary tree of free space and obstacles to plan the path towards the goal. Some other popular techniques are coverage-based path planning [16] and rapidly exploring random trees (RRT) [29]. RRT generates paths with local constraints using probability. Another work [18] uses probability for its navigation function, along with Gaussian probability distribution for locations of obstacles and the robot in the environment. In [42] the problem of convergence time in a cluttered environment by bi-directional RRT* and intelligent bi-directional RRT* is discussed. The authors propose a potentially guided bidirectional tree, which improve the convergence rate. A predictive algorithm to avoid dynamic and static obstacles using the range sensors in the robot is discussed in [26]. The predictive algorithms predict the velocity vectors of the dynamic obstacles and plan a collision-free path towards the goal. Article [7] discusses a nonparametric motion controller using Gaussian process regression for trajectory prediction. Predicted trajectories, along with the robot's limited perception, are used for robot motion control. In [45], a SLAM algorithm based on particle filter optimization is discussed. Image processing-based navigation techniques are also explored extensively, and in [44], a dynamic landmark identification system using the visual servo method is proposed.

With the latest advancements in soft computing techniques, extensive research on applying it to robotic path planning is being carried out. In [20], path

planning in a partially known environment is presented. Initially, an offline path planner will plan the paths within the given environment. During actual execution, the online path planner uses reactive path planning for unknown obstacles. A fuzzy inference system is used for trajectory tracking. Adaptive particle swarm optimization techniques are used for path planning in [9]. Objective functions are based on the distance from robot to goal and from robot to obstacle. Article [38] uses the firefly algorithm for mobile robot navigation in unknown dynamic environments. Hybrid soft computing approaches have been getting more traction lately. [37] uses an adaptive neuro-fuzzy inference system controller to get the robot steering angle dynamically based on the robot's forward obstacles in the environment. Path planning using a genetic algorithm and adaptive fuzzy-logic control is proposed in [5]. A genetic algorithm is used to generate the collision-free initial path. A piecewise cubic termite interpolating polynomial is used to smooth the generated optimal path. Later, an adaptive fuzzy-logic controller is used to maintain the robot in the desired path. In [1] a fuzzy logic controller is used for navigation in an unknown environment. Later the controller is optimized using genetic algorithms, neural networks, and particle swarm optimization. All these optimization techniques with a manually constructed fuzzy logic controller are compared. A survey paper [30] discuss the efficiency of path planning techniques using fuzzy, neural networks, genetic algorithms, nature-inspired algorithms, and hybrid algorithms, and concludes that hybrid algorithms are better. A cloud-based map storage approach is discussed in [28]. In this paper, the entire environment's information is stored in the cloud. This is helpful if the robot work area consists of multiple buildings/floors. The robot can access the cloud data by scanning specific environmental tags like ARTags and QR codes.

This work aims at developing a generalized LfD technique to train a mobile service robot to navigate to certain locations in an indoor service environment. Major features of the algorithm are its portable, generic, and user-friendly nature. Portability can be achieved by training only with the onboard sensors available on the robot and not augmenting the environment. This makes the system independent of the environment. The system will be generic if it can be used in any mobile robot platform in any indoor service environment with minimal modifications. A simple, intuitive user interface without any wearable device for training makes it user-friendly so that even a novice user can train the robot.

3. Proposed System

The overall work is divided into two stages: learning from demonstration stage and path-planning stage. In the LfD stage, the robot is trained to navigate to all possible destinations sequentially. Training is started from the robot's home position to destination 1, then destination 1 to destination 2, and so on. "Home" is the robot's starting position in the autonomous execution stage. The proposed LfD algorithm is the Enhanced Encoder Based LfD algorithm (EEBL), which is based on author's previous work on the Encoder Based LfD algorithm (EBL) [33–35]. Some of the key features of EEBL that are not in EBL include training to all the destinations sequentially, dynamic orientation and position error correction mechanism, and identification of reference points in the environment. EEBL has a training phase and autonomous phase; during the training phase paths are learned and path variables are stored as a path matrix. As destinations are sequentially taught, not all the possible routes are learned. With the learned paths as seed paths, the proposed Tree Based Planner (TBP) maps, the possible logical paths and a tree is formed with the home position as the root node and all the destinations as child nodes. TBP algorithm generates Virtual Land-Mark (VLM) based on the taught paths, which will act as waypoint for navigation. VLMs are the intermediate nodes in the tree. Given a source and destination, TBP uses the tree to plan the shortest route to navigate autonomously. Fig. 1 shows the flow diagram of the proposed system.

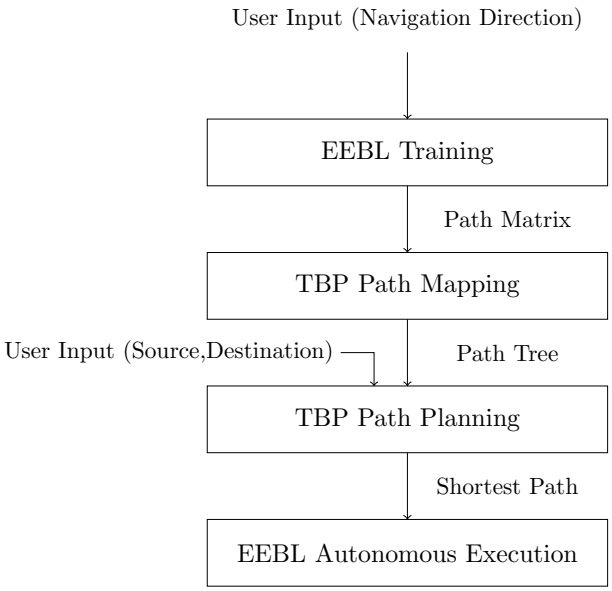


Fig. 1. Flow Diagram of the Proposed System

A hexagon-shaped differential drive mobile robot, as shown in Fig. 2, is used for testing. The robot has three onboard sensors: wheel encoders, proximity sensors, and pose sensors. Four proximity sensors are placed strategically, covering the robot without any blind spots.

3.1. Enhanced Encoder Based LfD Algorithm (EEBL)

This algorithm uses onboard sensors to learn paths taught by the trainer. The trainer uses a simple interface to move the robot front, left, right, and back. Wheel encoder data is used to learn the linear and angular distance for a path and is stored in a matrix called the path matrix. When the robot is navigating autonomously just with the distance information, it is prone to having errors like position error

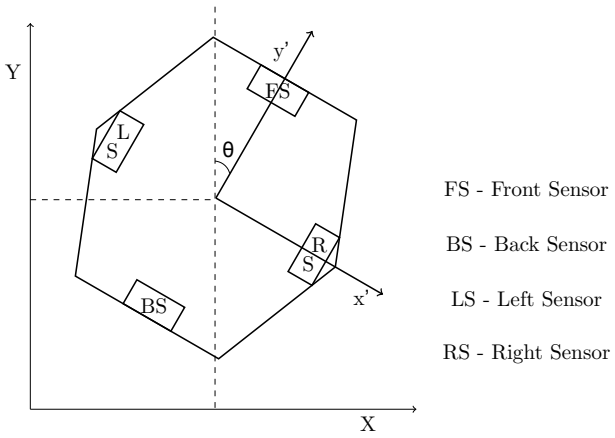


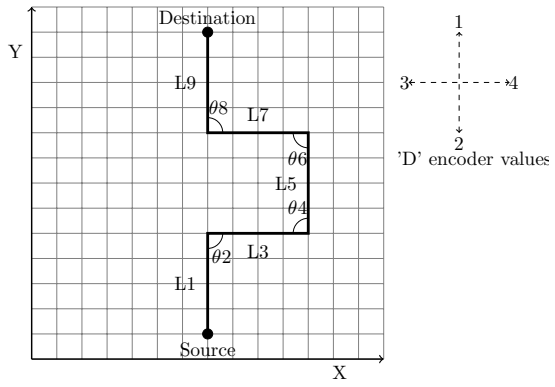
Fig. 2. Robot's Configuration and Sensor Location

and orientation error. In order to overcome the aforementioned errors, proximity sensors and pose sensors are used. Proximity sensors are used to identify the reference points while learning a path. Reference points are fixed points in the environment like walls, corridors, etc., and the proximity to them is used to adjust the robot's position during autonomous navigation. When a reference point is identified, proximity values are stored in a matrix called the reference matrix. Pose sensors give the orientation angle θ with reference to Y-axis, as shown in Fig. 2 of the robot. While training, the pose of the robot is updated in a matrix called the orientation matrix after every forward movement. Both the reference matrix and orientation matrix are used to correct the position and orientation errors, respectively, during autonomous navigation.

The EEBL algorithm is divided into two phases: the training phase and autonomous phase. In the training phase, the robot is trained to navigate a path, using an interface to direct it towards the goal point. Training is initiated from the home position to the first destination and continued sequentially until the path to the last destination is taught. Two path variables are learned during this phase: heading direction and linear or angular distance. These two variables are stored in a path matrix as on when the robot is stopped or heading direction is changed during training. Every row in the path matrix is the path variables of a single path. Even though the training is carried out sequentially, the path matrix is built based on the home position as a source. This is achieved by taking a partial path from the previous row in the matrix as per user input and appending the new path variables learned for a new destination to it. This is to maintain uniformity and ease in building a path tree with the home position as the root node.

$$Path\_Array = [D_j, L_j/\theta_j, D_{(j+1)}, L_j/\theta_{(j+1)}, \ldots, D_{(j+n)}, L_j/\theta_{(j+n)}] \quad (1)$$

Equation 1 is a path array for a single path with path variables D and L/ θ. D is the heading direction, which is encoded as 1,2,3 and 4 for forward movement, backward movement, left turn, and right turn respectively. L/θ is the linear distance encoder value if the heading direction is front/back or angular distance encoder value. 'D' and 'L/θ' are called a pose pair, which will determine the pose of the robot's state. Here, 'n' is the total number of states, the robot will transit through, before getting to the corresponding destination. If a path has fewer pose pairs than 'n', then zeros will be padded to the remaining pose pairs to keep the matrix not skewed. All the paths from home to different destinations are rows of the formed path matrix. An example path with its array is depicted in Fig. 3 with 9 pose pairs.



$example\_path = [1, L_1, 4, \theta_2, 1, L_3, 3, \theta_4, 1, L_5, 3, \theta_6, 1, L_7, 4, \theta_8, 1, L_9]$

Fig. 3. Example path and array with heading direction encoded values

The path matrix formed in the training phase is used to navigate autonomously. Unfortunately, this data is not enough for a robot to reach a destination accurately and repeatedly. Mobile robots are prone to errors due to sensor errors, motor speed mismatch and wheel misalignment, which lead to lower accuracy and repeatability. Two errors are identified: position error and orientation error. If position or orientation changes in the source or along the path, the destination point will vary relative to the error in position and orientation. Fig. 4 and Fig. 5 depict a scenario where the robot's starting position/orientation is changed; the effect is the same if an error occurs along the path. As the path distance increases, the error also increases relatively.

Position Error Position error occurs if there is a displacement in the robot's position. It is corrected by identifying the reference points in the environment. Reference points are fixed points in the environment like walls and corridors. These points are identified using the onboard proximity sensor of the robot. In the training phase, when the robot is moving forward, it scans for reference points using its left and right proximity sensors. A location in the environment is selected as a reference point if the proximity value (for either left or right sensor) is more than the given threshold value and is constant for a given sampling distance. When a reference point is identified, proximity values

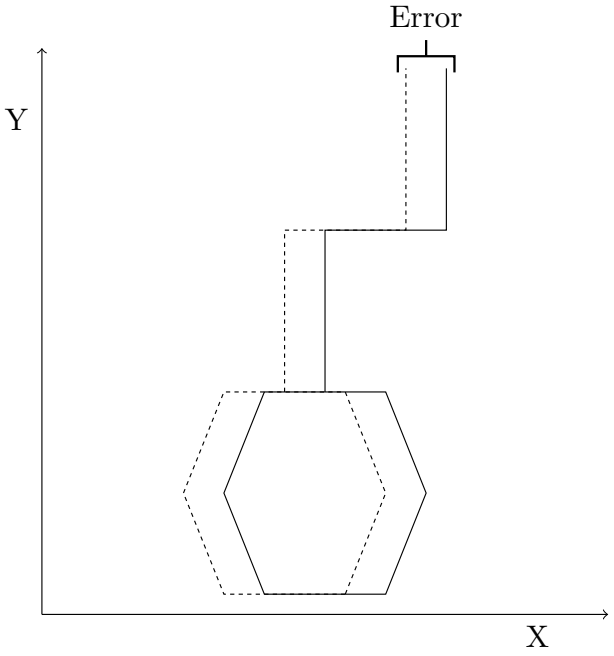


Fig. 4. Position Error

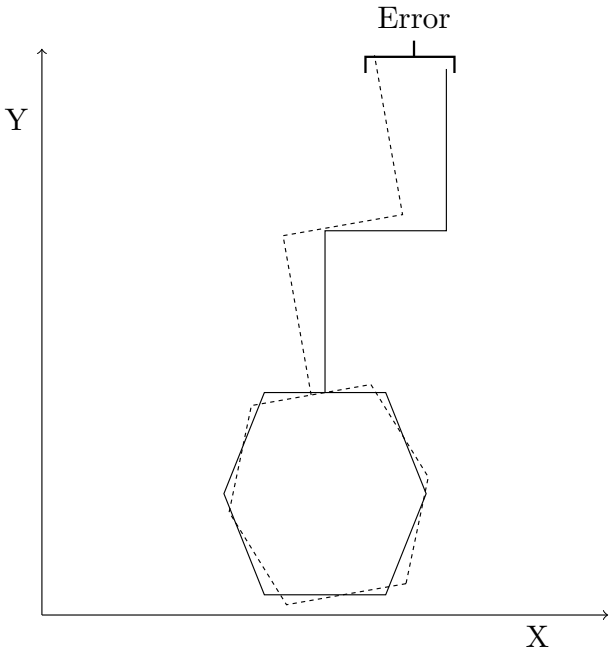
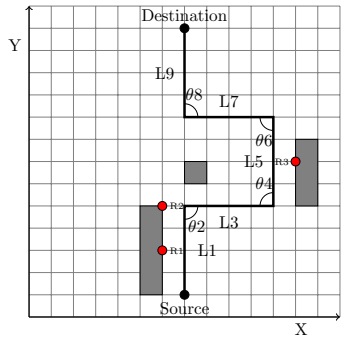


Fig. 5. Orientation Error

(for the left and right sensor) and its distance from the source location are stored in a matrix called the reference matrix. Similar to the path matrix, each row of the reference matrix is for a path, and there can be multiple reference points in a path. The reference matrix is updated in sync with the path matrix. The path matrix is updated to reflect a reference point as soon as it is identified. Equation 2 is a reference array for a path with the reference point 'm'.



$$Reference\_array = [LSV_1, Nil, LSV_2, Nil, Nil, RSV_3]$$

$$Orientaiton\_array = [\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5]$$

$$Example\_path = [1, L_1/2, R1, 1, L_1, R2, 4, \theta_2, 1, L_3, 3, \theta_4, 1, L_5/2, R3, 1, L_5, 3, \theta_6, 1, L_7, 4, \theta_8, 1, L_9]$$

Fig. 6. Example path with reference landmarks and its corresponding arrays

$$Reference\_array = [LSV_1, RSV_1, LSV_2, LSV_2, \ldots, LSV_m, RSV_m] \tag{2}$$

where,
LSV = Left Sensor Value
RSV = Right Sensor Value

Orientation Error The orientation of a robot can change, mainly due to hardware malfunctions like wheel misalignment and wheel encoder error. This causes over-turning or under-turning of the robot, which will cause relative change in the destination coordinates. It can be corrected using the orientation information of the robot, which is obtained using a pose sensor. The pose sensor gives the orientation angle θ with reference to Y-axis, as shown in Fig. 2 of the robot. In the training phase, orientation information is updated as pose angle α in a matrix called the orientation matrix at the beginning of each forward movement. This matrix is updated along with the path matrix and reference matrix. Equation 3 shows the orientation array for a single path.

$$Orientation\_array = [\alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_p] \tag{3}$$

Error Correction Let us consider the example path shown in Fig. 3, but with walls along the path as shown in Fig. 6. The gray blocks are walls and the red dots along the walls are the points identified as reference points by the algorithm. In order to find the reference point two factors have to be fixed: proximity threshold and sampling distance. Let us consider the proximity threshold to be 0.5 units (unit can be any standard unit of length) and sampling distance as 1 unit. Assume the grids in the figure are 0.5 x 0.5 unit. Based on these

two parameters, the algorithm has identified three reference points (R1, R2, and R3) along the forward movement path of the robot. As soon as the reference point is identified, the path matrix is updated with the forward distance value until that point and the corresponding sensor value is stored in the reference matrix. If only one sensor value is satisfying the condition, then another sensor value is stored as Nil. Orientation array is updated with pose angles α_1, α_2, α_3, α_4 and α_5 at the beginning of the forward movements L1, L3, L5, L7 and L9, respectively.

In the autonomous phase, a PI control system is used to auto-correct the robot's position and orientation error. The identified reference point proximity values and pose angles are used as the reference values for position and orientation correction, respectively. The control equation is given in equation 4. Kp value is assumed to be 1 and Ki is assumed to be 0.0001, which is obtained by the trial and error method.

$$Y(t) = K_p e(t) + K_i \sum e(t) \tag{4}$$

where,
e(t) = (Current_value - Reference_value)
e(t) - Error
Y(t) - Controlled variable
Kp - Proportional constant
Ki - Integral constant

In the position correction system, the robot's position is corrected by controlling the sliding velocity using Y(t), and in the orientation correction system, the robot's angular velocity is controlled using Y(t) to correct the pose of the robot. Algorithm 1 shows the EEBL algorithm with all possible robot movements and its matrices updating on different conditions for a path.

3.2. Tree Based Planner (TBP)

The tree based planner is a path mapping and planning algorithm that uses the learned path matrix as input. The path matrix formed by the EEBL algorithm has one possible route to all destinations from home; this is called a partially known path. But this information is not enough to plan efficient paths. Initially using the TBP algorithm, all the other possible logical paths to all the destinations are mapped. As an end result of path mapping, a path tree is formed with the home position as the root node and all the destinations as child nodes. Path mapping is carried out by identifying VLMs, which act as waypoints in the logical paths created and essentially maps the shortest possible paths. VLMs are identified based on the taught paths. An intuitive strategy is used to identify the VLMs and all the points where the robot takes a turn (left or right) are considered to be VLMs. VLMs are the intermediate nodes in the tree. Later, given a source and destination, TBP's path planner uses the path tree to plan efficient path to navigate autonomously.

Fig. 7 depicts a scenario with a home position and three destinations D1, D2 and D3. All the gray blocks are obstacles, and the line between all the destinations are the paths it is taught to navigate. The trai-

Algorithm 1 Enhanced Encoder Based LfD Algorithm

```
Initialize on-board sensors
Initialize path_matrix, reference_matrix, orientaiton_matrix
Initialize path count P
Check for user input D
if D == Forward then
    while D == Forward do
        Robot ← Move forward
        if Reference Point then
            Reference_Matrix[P] ← (LSV, RSV)
            Path_Matrix[P] ← (1, Encoder_Value)
        end if
    end while
    Path_Matrix[P] ← (1, Encoder_Value)
    Orientation_Matrix[P] ← α
else if D == Backward then
    while D == Backward do
        Robot ← Move backward
    end while
    Path_Matrix[P] ← (2, Encoder_Value)
else if D == Left then
    while D == Left do
        Robot ← Steer Left
    end while
    Path_Matrix[P] ← (3, Encoder_Value)
else if D == Right then
    while D == Right do
        Robot ← Steer Right
    end while
    Path_Matrix[P] ← (4, Encoder_Value)
else
    P ← P++
end if
```

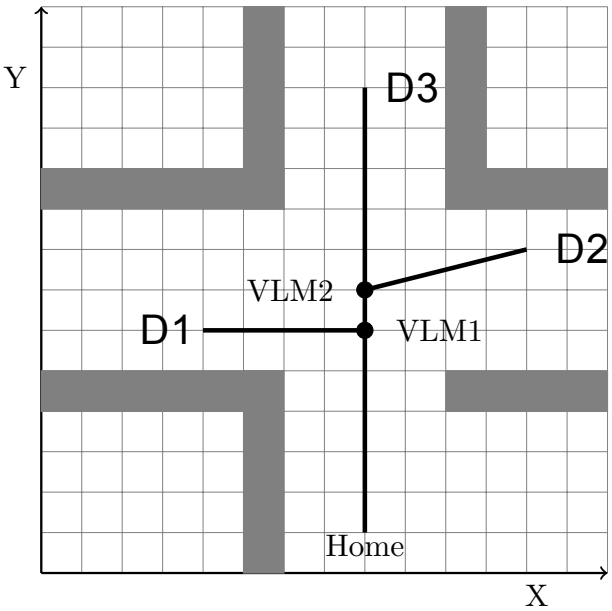


Fig. 7. Sample multi destination paths with identified VLM's (black circles)

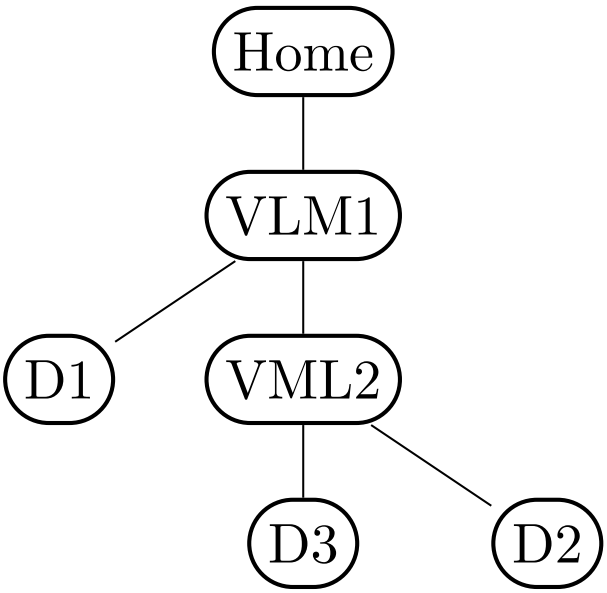


Fig. 8. Tree generated by TBP Algorithm with VLM's

Algorithm 2 TBP Algorithm- Path Mapping

```
Preprocessing Phase - Sorting Stage
for ri = 0 ; i < max_r ; ri++ do
    for rj = ri + 1 ; rj < max_r ; rj++ do
        if Path_Matrix[ri][0]>Path_Matrix[rj][0] then
            swap(ri,rj);
        else
            continue;
        end if
    end for
end for
```

Algorithm 3 Preprocessing Phase - Sub Path Matrix Stage

```
int fcount, rcount, lcount;
for r = 0 ; r < max_r ; r++ do
    if Path_Matrix[r][0] = = 1 then
        for c = 0 ; c < mac_c ; c++ do
            Path_Matrixf[fcount][c]=
Path_Matrix[r][c];
        end for
    end if
    Similarly for Left and Right direction creating
Path_Matrixl & Path_Matrixr matrix
end for
```

Algorithm 4 State Formation Phase

```
fcount
if (fcount != 0) then
    for r = 0;r < max_r;r++ do
        for c = 0;c < max_c & Path_Matrixf[r][c]! = 0; do
            if Path_Matrixf[r][c] = = 1 then
                create-forward-VLM;
            end if
            Similarly ffor Left-VLM & Right-VLM
            c += 2;
        end for
        create-destination;
    end for
else
    Exit;
end if
```

ning sequence followed was Home→D1, D1→D2, and D2→D3, respectively. With this trained data, the TBP algorithm will generate VLMs and form the path tree. Two VLMs are identified (black dots along the path in the figure), namely VLM1 and VLM2. VLM1 is chosen because there is a left turn in the path to reach D1. Likewise, VLM2 is chosen because there is a right turn to reach D2. The final path tree with VLMs and Destination is shown in Fig. 8. The home position is the root node, all the VLMs are intermediate nodes, and all the destinations are child nodes. This is a ternary graph, where every node can have a maximum of three child nodes, namely center child, left child, and right child. Center, left and right nodes are the symbolic representations of navigation direction for the parent to reach the child node. For a parent node to reach a center child node, the robot has to move forward, and left and right child nodes can be reached similarly. Every node contains information like forward distance, turning angle, reference point details, pose angle, and child nodes details.

The TBL algorithm has two phases: the path mapping and the path planning phase. The path mapping phase has two stages: the preprocessing stage and the node formation stage. The preprocessing stage splits the path matrix into three matrices based on the reachable destination by moving forward from home, moving left from home, and moving right from home. This would be the input for the node formation stage. In this stage, VLMs and their connectivity to home or any destination is identified. Finally, a tree with the home as root node is formed. Algorithms 2,3 and 4 show the path mapping phase.

In the path planning phase, the formed tree is used to navigate from a given source to a destination. In this phase, two paths will be identified from home→source and home→destination. These two paths are merged on a common VLM node as a waypoint; this would give the shortest route possible. If there is no common VLM, then home is the common node that will act as the waypoint, and this is the longest route possible. Algorithm 5 shows the path planning phase.

Consider the tree in Fig. 8. If a path has to be planned between D3→D1, paths from home to both the nodes have to be identified. They are home→VLM1→D1 and home→VLM1→VLM2→D3. These two paths have VLM1 as a common node.

Algorithm 5 TBP Algorithm- Path Planning

```
path1[m] = find_path(home to source)
path2[m] = find_path(home to destination)
for r = 0;r < max_r;r++ do
    for c = 0;c < max_c;c++ do
        if path1[r] == path2[c] then
            break;
        else
            continue;
        end if
    end for
end for
merge_routes(r,c)
```

Merging the paths as VLM1 as the waypoint we get D3→VLM2→VLM1→D1, and this is the shortest logical path possible. The longest path without using this technique would be D3→VLM2→VLM1→home→D1, and this goes through the home, which is unnecessary.

The path generated by the path planning phase is passed to the EEBL algorithm autonomous phase. This phase navigates the robot in the desired path to the destination autonomously, while navigation, position, and orientation correction are carried out if necessary. This reduces the error in reaching the destination accurately.

4. Results and Analysis

The proposed algorithms are implemented in the player/stage 2D robotic simulation platform. The system is tested in two different environment layouts, house and hospital, as shown in Fig. 9. The house layout is a smaller environment, with 8 destinations, and the hospital is a comparatively bigger and more complex environment, with 28 destinations.

Initially, the robot is assigned a home position in the environment and sequential training to navigate toward all the destinations is carried out using the EEBL algorithm. While the paths are being learned, the robot also identifies virtual landmarks (VLM) and reference points in the environment. After the training phase, the TBP algorithm generates a tree with the home as the root node, VLMs as intermediate nodes, and all the destinations as child nodes.

Fig. 10 depicts the paths in the house layout with VLMs and reference points, and Fig. 11 shows the tree generated for the paths learned with 4 VLMs and 14 nodes. Fig. 12 shows the paths learned in hospital layout with VLMs and reference points. Fig. 13 shows the tree formed using the hospital layout with 24 VLMs and a total of 52 nodes. Compared to the house layout, this layout is 3.5 times bigger in terms of size, number of destinations, and number of nodes in the tree. This proves that the algorithm is scalable. Given a destination, a path can be planned using the tree and navigated to reach the destination. For example, consider the robot's current position to be in D1 of the hospital layout and the given destination to be D21. In this case, the robot plans a path by traversing the tree and finding the shortest possible route from source to destination, which will be D1→VLM8→VLM7→VLM6→VLM5→VLM4→VLM2→VLM1→VLM3→VLM9→VLM10→VLM11→VLM12→VLM13→D21.

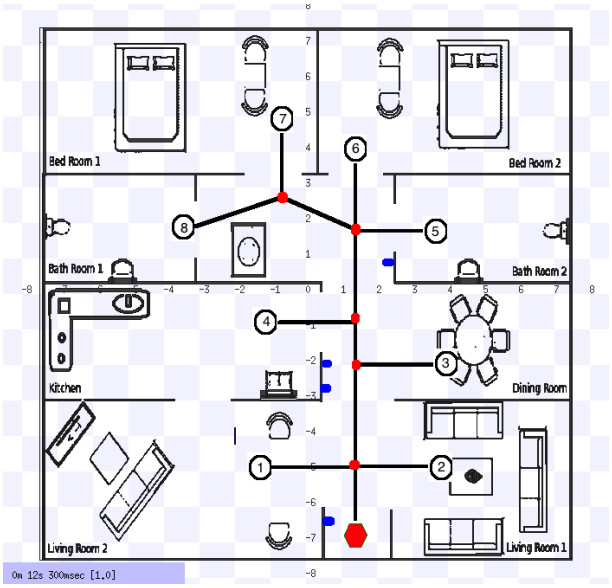


Fig. 10. House layout with paths learned (black lines), identified VLM's (red circles) and reference points (blue circles)

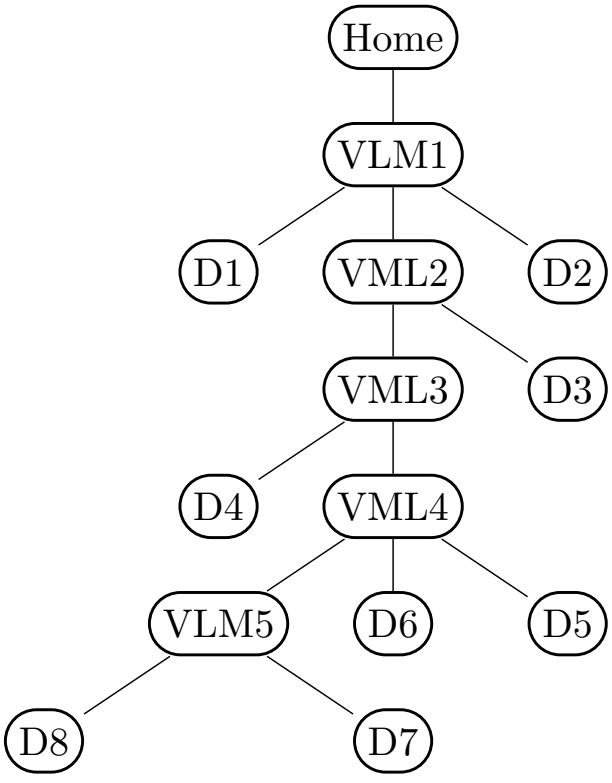


Fig. 11. Tree formed based on house layout paths

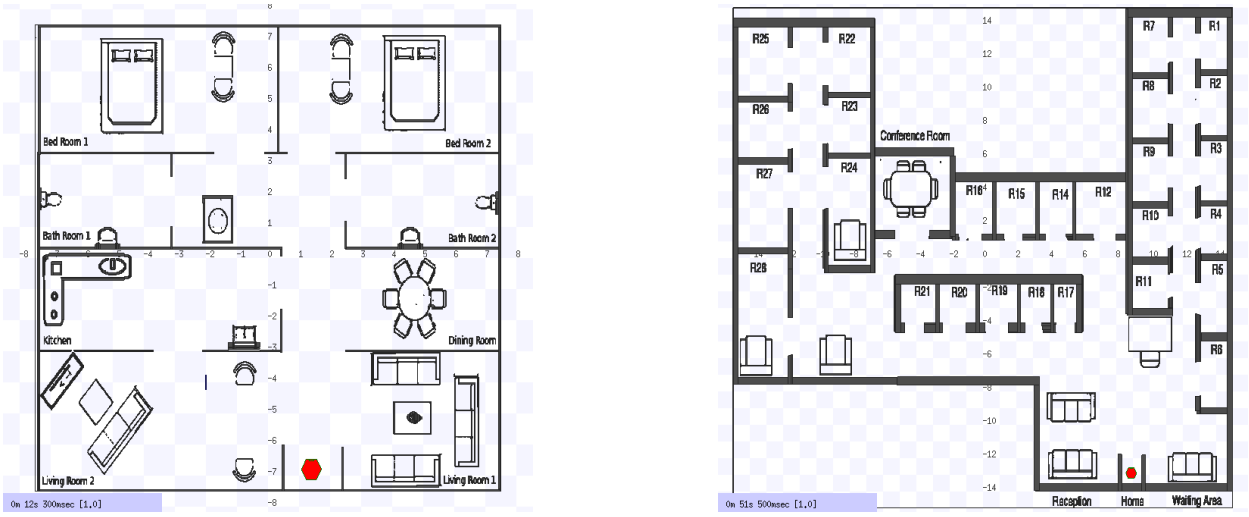


Fig. 9. Stage environment with robot (hexagon shaped red object) in home position, house layout (left) and hospital layout (right)

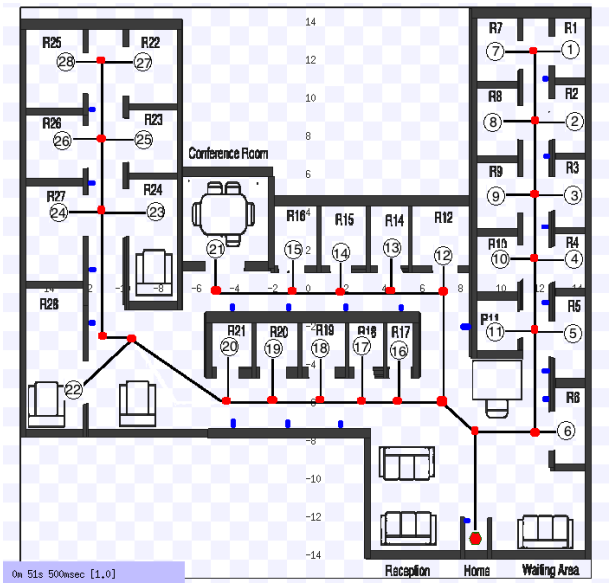


Fig. 12. Hospital layout with paths learned (black lines), identified VLMs (red circles) and reference points (blue circles)

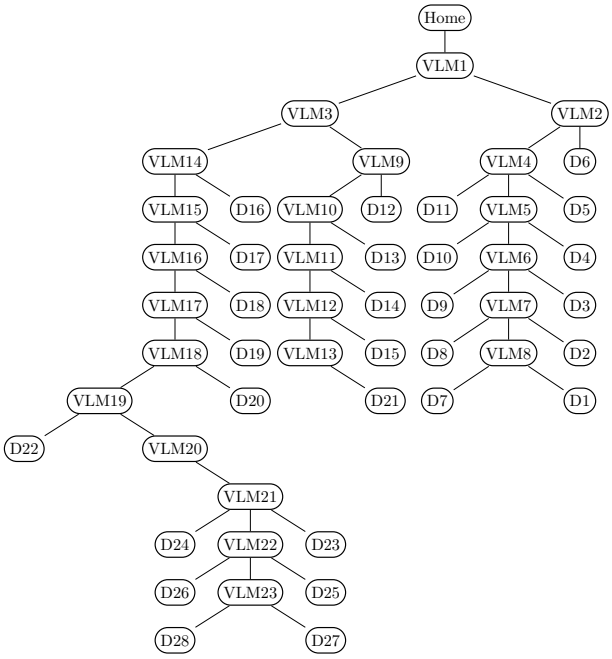


Fig. 13. Tree formed based on hospital layout paths

4.1. Position and Orientation Error Correction

The most common errors in a mobile robot navigation system are position error and orientation error. Orientation error is corrected by storing the pose value of the robot along with path variables during path learning. Position errors can be corrected by identifying reference points along the paths. These references will be used in the autonomous phase to correct position error along the x-axis of the robot's coordinates. A closed-loop PI control system is used to correct the errors.

A test was conducted to prove the efficiency of the error correction control system, and the repeatability of the proposed system. In order to test the error correction algorithm efficiency, a Gaussian error is added in the forward movement of the robot (to add a deviation in the heading direction) and the wheel encoder of the robot (to introduce error in the turning angle). Repeatability was verified by running a robot to and from a fixed source and destination several times and calculating the relative average error of its position. A shadow robot is added in the simulation to run with the actual robot but without going through any error correction algorithm, to show the effectiveness of the proposed error correction algorithm. Equation 5 shows the robot position representation in the simulation environment.

$$Robot\_Position = (X, Y, A) \tag{5}$$

where,
X - Robot's X-coordinate in world frame
Y - Robot's Y-coordinate in world frame
A - Robot's pose angle in world frame

Case 1: Home to D3 Consider the house layout with the source as home and destination as D3. This path was traversed 100 times back and forth continuously with a Gaussian error of μ=0 and σ=1, which was chosen by trial and error method. Essentially, the robot reaches home and D3, 50 times each. Position data for all the trials are collected. Reference position (recorded in the training phase), actual position with error correction and actual position without error correction are plotted in Fig. 14. Fig. 15 shows the plot of the robot's D3 position.

Case 2: D2 to D5 This path is mapped by merging two paths from home→D2 and home→D5. This path is traversed back and forth 100 times with Gaussian error of μ=0 and σ=1. The robot's position plots in D2 and D5 after 100 runs are shown in Fig. 16 and Fig. 17, respectively.

Case 3: Home to D3 with Gauss error (0, 2) To validate the efficiency of the error correction algorithm the same path as case 1 is executed with a different Gaussian error parameter. The corresponding plots are shown in Fig. 18 and Fig. 19.

Relative average position error is calculated for all the cases. Equation 6 is used to calculate the error % with respect to the reference position. Table 1 shows the relative average position error of the cases discussed. It is very evident that the X position and pose average with error correction are much less than without error correction average. Even though the main objective of our error correction algorithm is to reduce X position and pose angle errors, to an extent it is indirectly correcting the Y position error, too. This can be observed in case1:D3 and case3:D3 scenarios. The overall average positional error based on all the three cases is given in equation 7.

$$(Avg\_error)_i\% = \frac{|Reference_i - Actual_i|}{|Reference_i|} * 100 \tag{6}$$

where,
i = X or Y or A
Reference_i = Reference coordinate
Actual_i = Actual coordinate

$$\begin{gathered} Avg\_Pos\_Error\_With\_Correction = \\ (4.29\%, 3.72\%, 0\%) \\ Avg\_Pos\_Error\_Without\_Correction = \\ (258.11\%, 9.06\%, 14.33\%) \end{gathered} \tag{7}$$

4.2. Comparison Study

The proposed algorithm is compared with breadth first search (BFS), Dijkstra's, greedy best first search (BBF), A* and rapidly exploring random trees (RRT) algorithms. To compare with the proposed algorithm, a Java-based path planning simulator [10] is used. An exact replica (with the same number of cells) of the hospital layout is created in the Java simulator and is compared with the resultant path. Two paths are considered: Home→D28 and D7→D28. Fig. 20 and 21 show the path generated by the proposed and other algorithms for two different paths. The path mapped by the proposed algorithm is given in equation 8. A quantitative comparison based on both the paths is depicted in table 2. The path length (No. of cells) mapped by the proposed algorithm is on par with the other algorithms, where as the visited cell count is very small compared to other algorithms. This considerably reduces the path planning time in the proposed algorithm. Hence the proposed algorithm is faster with efficient path mapping.

$$\begin{aligned} Path1 : Home \rightarrow VLM1 \rightarrow VLM3 \rightarrow VLM14 \rightarrow \\ through \rightarrow VLM22 \rightarrow D28 \end{aligned} \tag{8a}$$

$$\begin{aligned} Path2 : D7 \rightarrow VLM8 \rightarrow VLM7 \rightarrow VLM6 \rightarrow VLM5 \rightarrow \\ VLM4 \rightarrow VLM2 \rightarrow VLM1 \rightarrow VLM3 \rightarrow VLM14 \rightarrow \\ through \rightarrow VLM22 \rightarrow D28 \end{aligned} \tag{8b}$$

5. Conclusion

In this paper, a novel path planning algorithm based on Learning from Demonstration and Tree Based Path planner algorithms is proposed. The main focus is on indoor mobile service robots, which can be used in commercial and domestic places without any environment augmentation. Overall, two algorithms, the Enhanced Encoder Based LfD (EEBL) algorithm and Tree Based Path Planner (TBP) algorithm, are proposed. Both these algorithms work in cohesion to achieve the desired path given a source and destination. Initially, EEBL is used to learn the partial paths in the environment. Later, TBP maps all the logical paths with virtual landmarks (VLM) and reference points. The system is also capable of correcting positional/orientation errors that occur due to hardware anomalies in the robot. A simple PI control system is used to correct the errors. The proposed system is implemented and tested in Player/Stage, a 2D robotic simulator. The algorithm is tested in two different environments without any changes in the algorithm. This makes the proposed algorithm environment-independent i.e. portable. One environment is 3.5 times bigger than the other, which proves the algorithm is scalable. In order to test the accuracy and repeatability, a single path is executed repeatedly by adding gaussian error in the robot movement. The overall average destination position accuracy was calculated to be $\pm 5\%$ (both X-position and pose). Finally, the path planned from a fixed source and destination by the proposed algorithm is compared with some of the classical and state-of-the-art path planning techniques. The result shows that the proposed algorithm has fewer visited cells

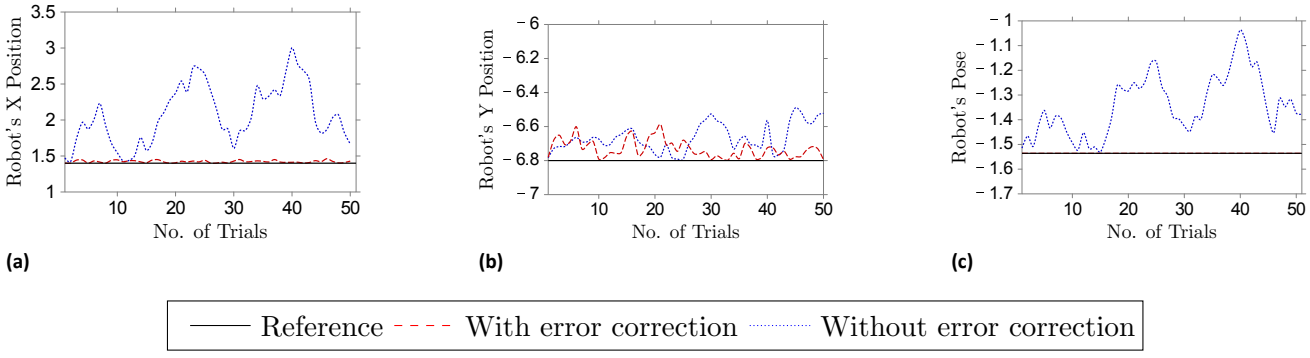


Fig. 14. Case1: Robot's Home Position with Gaussian error (0,1)

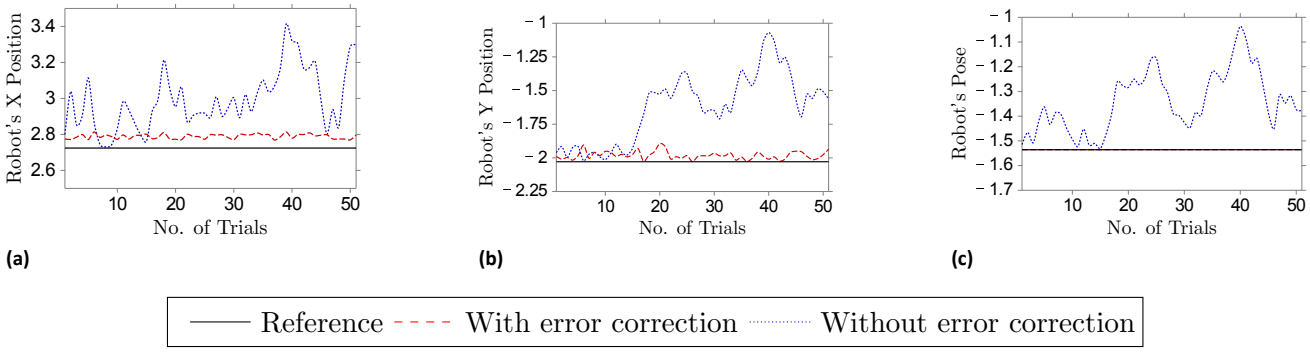


Fig. 15. Case1: Robot's D3 Position with Gaussian error (0,1)

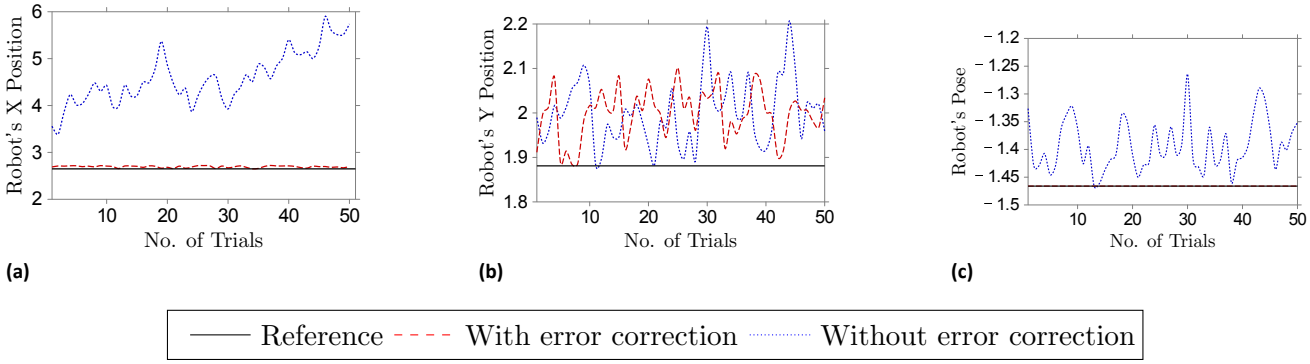


Fig. 16. Case2: Robot's D2 Position with Gaussian error (0,1)

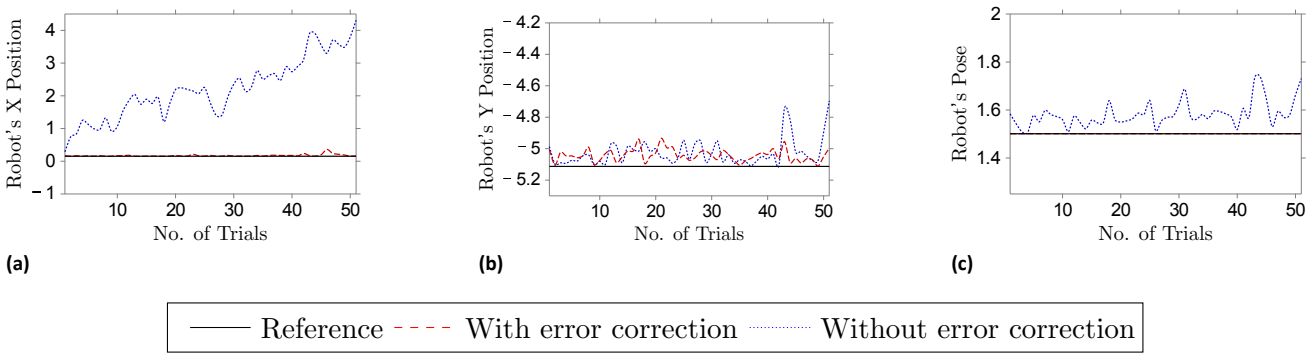


Fig. 17. Case2: Robot's D5 Position with Gaussian error (0,1)

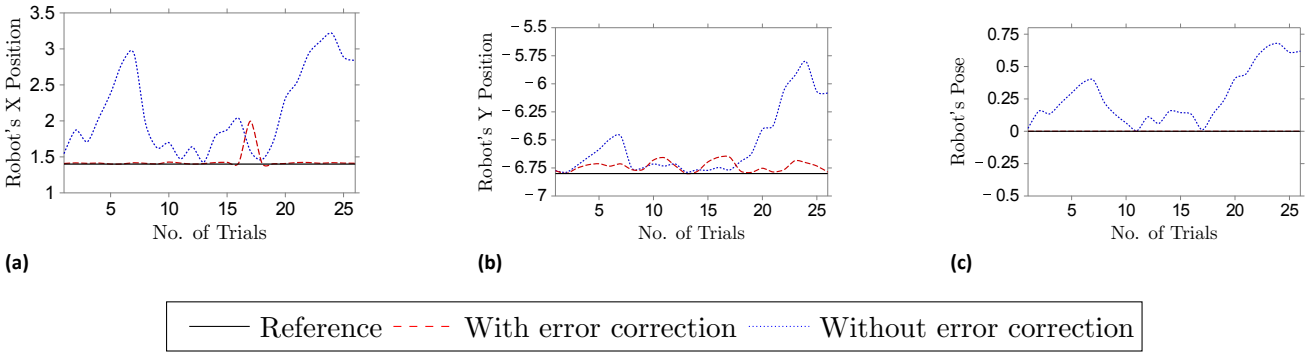


Fig. 18. Case3: Robot's Home Position with Gaussian error (0,2)

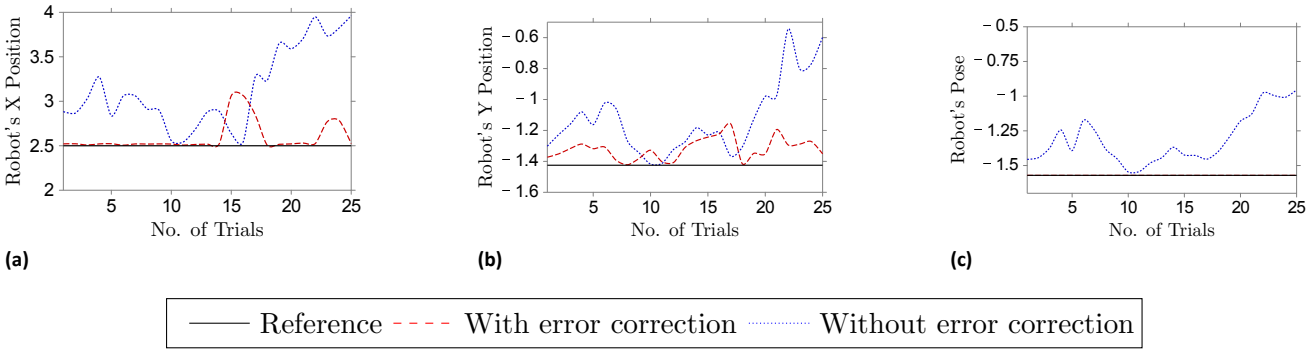


Fig. 19. Case3: Robot's D3 Position with Gaussian error (0,2)

Tab. 1. Relative Average Error

Location	With Correction (%)			Without Correction (%)		
	X Position	Y Position	Pose Angle	X Position	Y Position	Pose Angle
Case1: Home	1.57	1.01	0	47.8	1.9	19.7
Case1: D3	2.37	2.32	0	9.8	19.8	12.6
Case2: D2	1.7	6.2	0	73.2	6.3	4.9
Case2: D5	13.8	1.7	0	1340	1.71	5.3
Case3: Home	2.5	3.8	0	52.3	3.8	27
Case3: D3	3.8	7.3	0	25.6	20.9	16.5

Tab. 2. Quantitative path analysis

Paths	Algorithm	Length (cell count)	Visited Cells
Path 1	Proposed	41	41
	BFS	48	274
	Dijkstra	47	274
	GBF	48	76
	A*	46	188
	RRT	42	101
Path 2	Proposed	59	59
	BFS	63	205
	Dijkstra	62	205
	GBF	64	165
	A*	62	236
	RRT	66	215

and is on par in terms of path length. Hence the proposed algorithm proves to have properties like portability, scalability, repeatability, accuracy, shorter path planning time, and efficient path generation.

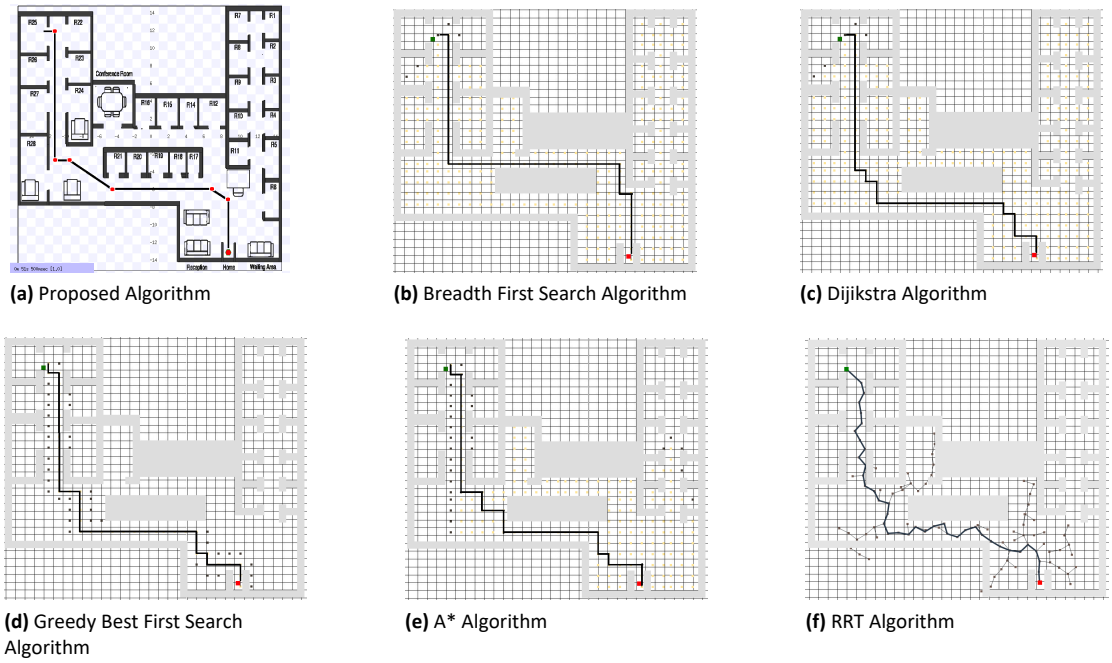
AUTHORS

A. A. Nippun Kumaar* – Department of Computer Science and Engineering, Amrita School of Engineering, Bengaluru, Amrita Vishwa Vidyapeetham, India, e-mail: nippun05@gmail.com, www: www.nippunkumaar.in.

Sreeja Kochuvila – Department of Electronics and Communication Engineering, Amrita School of Engineering, Bengaluru, Amrita Vishwa Vidyapeetham, India, e-mail: k_sreeja@blr.amrita.edu, www: https://amrita.edu/faculty/k-sreeja/.

S. R. Nagaraja – Department of Mechanical Engineering, Amrita School of Engineering, Bengaluru, Amrita Vishwa Vidyapeetham, India, e-mail: sr_nagaraja@blr.amrita.edu, www: https://amrita.edu/faculty/sr-nagaraja/.

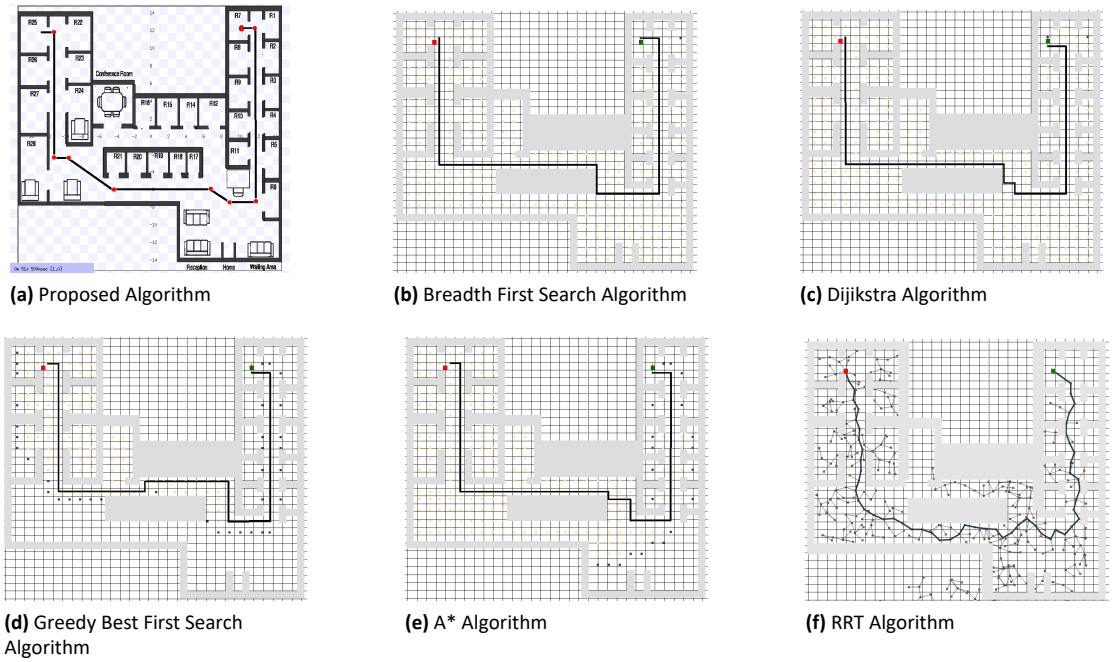
*Corresponding author



(a) Proposed Algorithm **(b)** Breadth First Search Algorithm **(c)** Dijikstra Algorithm

(d) Greedy Best First Search Algorithm **(e)** A* Algorithm **(f)** RRT Algorithm

Fig. 20. Path 1: Hospital layout from Home to D28



(a) Proposed Algorithm **(b)** Breadth First Search Algorithm **(c)** Dijikstra Algorithm

(d) Greedy Best First Search Algorithm **(e)** A* Algorithm **(f)** RRT Algorithm

Fig. 21. Path 2: Hospital layout from D7 to D28

REFERENCES

[1] M. Algabri, H. Mathkour, H. Ramdane, and M. Alsulaiman, "Comparative study of soft computing techniques for mobile robot navigation in an unknown environment", *Computers in Human Behavior*, vol. 50, 2015, pp. 42 – 56. https://doi.org/10.1016/j.chb.2015.03.062

[2] V. Alvarez-Santos, A. Canedo-Rodriguez, R. Iglesias, X. Pardo, C. Regueiro, and M. Fernandez-Delgado, "Route learning and reproduction in a tour-guide robot", *Robotics and Autonomous Systems*, vol. 63, Part 2, 2015, pp. 206 – 213. https://doi.org/10.1016/j.robot.2014.07.013

[3] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration", *Robotics and Autonomous Systems*, vol. 57, no. 5, 2009, pp. 469 – 483. https://doi.org/10.1016/j.robot.2008.10.024

[4] R. Arkin, "Motor schema-based mobile robot navigation", *International Journal of Robotics Research*, vol. 8, 1989, pp. 92–112.

[5] A. Bakdi, A. Hentout, H. Boutami, A. Maoudj, O. Hachour, and B. Bouzouia, "Optimal path planning and execution for mobile ro-

bots using genetic algorithm and adaptive fuzzy-logic control", *Robotics and Autonomous Systems*, vol. 89, 2017, pp. 95 – 109. https://doi.org/10.1016/j.robot.2016.12.008

[6] D. A. Brenna, B. Brett, and M. V. Manuela, "Policy feedback for the refinement of learned motion control on a mobile robot", *International Journal of Social Robotics*, vol. 4, no. 4, 2012, pp. 383–395. 10.1007/s12369-012-0156-9

[7] S. Choi, E. Kim, K. Lee, and S. Oh, "Real-time nonparametric reactive navigation of mobile robots in dynamic environments", *Robotics and Autonomous Systems*, vol. 91, 2017, pp. 11 – 24. https://doi.org/10.1016/j.robot.2016.12.003

[8] K. Dermot, U. Nehmzow, and A. S. Billings. "Towards automated code generation for autonomous mobile robots", June 2010.

[9] H. S. Dewang, P. K. Mohanty, and S. Kundu, "A robust path planning for mobile robot using smart particle swarm optimization", *Procedia Computer Science*, International Conference on Robotics and Smart Manufacturing (RoSMa2018), vol. 133, 2018, pp. 290 – 297. https://doi.org/10.1016/j.procs.2018.07.036

[10] S. S. Dhanjal. *Path Planning in Single and Multi-robot Systems*. PhD thesis, BITS, Pilani, May 2016.

[11] E. W. Dijkstra. "A note on two problems in connexion with graphs", December 1959. 10.1145/3544585.3544600

[12] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel. "Practical search techniques in path planning for autonomous driving", Jan 2008.

[13] S. Dong and B. Williams, "Learning and recognition of hybrid manipulation motions in variable environments using probabilistic flow tubes", *International Journal of Social Robotics*, vol. 4, no. 4, 2012, pp. 357–368. 10.1007/s12369-012-0155-x

[14] M. Ehrenmann, O. Rogalla, R. Zöllner, and R. Dillmann. "Teaching service robots complex tasks: Programming by demostration for workshop and household environments", 2002.

[15] D. Ferguson and A. T. Stentz. "The field d* algorithm for improved path planning and replanning in uniform and non-uniform cost environments", June 2005.

[16] E. Galceran and M. Carreras, "A survey on coverage path planning for robotics", *Robotics and Autonomous Systems*, vol. 61, no. 12, 2013, pp. 1258 – 1276. https://doi.org/10.1016/j.robot.2013.09.004

[17] A. M. E. Ghalamzan and M. Ragaglia, "Robot learning from demonstrations: Emulation learning in environments with moving obstacles", *Robotics and Autonomous Systems*, vol. 101, 2018, pp. 45 – 56. https://doi.org/10.1016/j.robot.2017.12.001

[18] S. Hacohen, S. Shoval, and N. Shvalb, "Applying probability navigation function in dynamic uncertain environments", *Robotics and Autonomous Systems*, vol. 87, 2017, pp. 237 – 246. https://doi.org/10.1016/j.robot.2016.10.010

[19] D. Halbert. "Programming by example". PhD thesis, University of California, Berkeley, November 1984.

[20] M. Hank and M. Haddad, "A hybrid approach for autonomous navigation of mobile robots in partially-known environments", *Robotics and Autonomous Systems*, vol. 86, 2016, pp. 113 – 127. https://doi.org/10.1016/j.robot.2016.09.009

[21] P. Hart, N. Nilsson, and B. Rafael, "A formal basis for the heuristic determination of minimum cost paths", *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, 1968, pp. 100–107.

[22] J. Hong, Y. Choi, and K. Park. "Mobile robot navigation using modified flexible vector field approach with laser range finder and IR sensor", 2007 International Conference on Control, Automation and Systems, 2007, pp. 721–726. 10.1109/ICCAS.2007.4406993

[23] R. Huq, G. K. Mann, and R. G. Gosine, "Mobile robot navigation using motor schema and fuzzy context dependent behavior modulation", *Applied Soft Computing*, vol. 8, no. 1, 2008, pp. 422 – 436. https://doi.org/10.1016/j.asoc.2007.02.006

[24] Y. K. Hwang and N. Ahuja, "A potential field approach to path planning". In: *IEEE Transaction on Robotics and Automation*, vol. 8, 1992, pp. 23–32.

[25] R. Jäkel, S. R. Schmidt-Rohr, S. W. Rühl, A. Kasper, Z. Xue, and R. Dillmann, "Learning of planning models for dexterous manipulation based on human demonstrations", *International Journal of Social Robotics*, vol. 4, no. 4, 2012, pp. 437–448, 10.1007/s12369-012-0162-y

[26] F. Kamil, T. S. Hong, W. Khaksar, M. Y. Moghrabiah, N. Zulkifli, and S. A. Ahmad, "New robot navigation algorithm for arbitrary unknown dynamic environments based on future prediction and priority behavior", *Expert Systems with Applications*, vol. 86, 2017, pp. 274 – 291. https://doi.org/10.1016/j.eswa.2017.05.059

[27] J. Lee, Y. Nam, S. Hong, and W. Cho, "New potential functions with random force algorithms using potential field method", *Journal of Intelligent & Robotic Systems*, vol. 66, no. 3, 2012, pp. 303–319. 10.1007/s10846-011-9595-z

[28] R. Limosani, A. Manzi, L. Fiorini, F. Cavallo, and P. Dario, "Enabling global robot navigation based on a cloud robotics approach", *International Journal of Social Robotics*, vol. 8, no. 3, 2016, pp. 371–380. 10.1007/s12369-016-0349-8

[29] S. M. Lavalle. "Rapidly-exploring random trees: A new tool for path planning", May 1999.

[30] T. T. Mac, C. Copot, D. T. Tran, and R. D. Keyser, "Heuristic approaches in robot path planning: A survey", *Robotics and Autonomous Systems*, vol. 86, 2016, pp. 13 – 28. https://doi.org/10.1016/j.robot.2016.08.001

[31] D. Nakhaeinia, S. Tang, S. Noor, and O. Motlagh, "A review of control architectures for autonomous navigation of mobile robots", *International Journal of Physical Sciences*, vol. 6, 2011, pp. 169–174.

[32] U. Nehmzow, O. Akanyeti, C. Weinrich, T. Kyriacou, and S. A. Billings. "Programming mobile robots by demonstration through system identification", European Conference on Mobile Robots, Freiburg, Germany, 19 Sep 2007 - 21 Sep 2007.

[33] A. A. NippunKumaar and T. S. B. Sudarshan, "Mobile robot programming by demonstration". In: *2011 Fourth International Conference on Emerging Trends in Engineering Technology*, vol. 394, 2011, pp. 206–209.

[34] A. A. NippunKumaar and T. S. B. Sudarshan, "Learning from demonstration with state based obstacle avoidance for mobile service robots", *Applied Mechanics and Materials*, vol. 394, 2013, pp. 448–4556.

[35] A. NippunKumaar and T. Sudarshan, "Sensor counter approach for a mobile robot to navigate a path using programming by demonstration", *Procedia Engineering*, International Conference on Communication Technology and System Design, vol. 30, 2012, pp. 554 – 561. https://doi.org/10.1016/j.proeng.2012.01.898

[36] S. Osentoski, B. Pitzer, C. Crick, G. Jay, S. Dong, D. Grollman, H. B. Suay, and O. C. Jenkins, "Remote robotic laboratories for learning from demonstration", *International Journal of Social Robotics*, vol. 4, no. 4, 2012, pp. 449–461. 10.1007/s12369-012-0157-8

[37] A. Pandey, S. Kumar, K. K. Pandey, and D. R. Parhi, "Mobile robot navigation in unknown static environments using anfis controller", *Perspectives in Science*, Recent Trends in Engineering and Material Sciences, vol. 8, 2016, pp. 421 – 423. https://doi.org/10.1016/j.pisc.2016.04.094

[38] B. Patle, A. Pandey, A. Jagadeesh, and D. Parhi, "Path planning in uncertain environment by using firefly algorithm", *Defence Technology*, vol. 14, no. 6, 2018, pp. 691 – 701. https://doi.org/10.1016/j.dt.2018.06.004

[39] A. T. Rashid, A. A. Ali, M. Frasca, and L. Fortuna, "Path planning with obstacle avoidance based on visibility binary tree algorithm", *Robotics and Autonomous Systems*, vol. 61, no. 12, 2013, pp. 1440 – 1449. https://doi.org/10.1016/j.robot.2013.07.010

[40] I. R. Roberto, N. Ulrich, K. Theocharis, and B. Steve. "Modelling and characterisation of a mobile robot's operation", September 2006.

[41] A. Stentz. "Optimal and efficient path planning for partially-known environments", May 1994.

[42] Z. Tahir, A. H. Qureshi, Y. Ayaz, and R. Nawaz, "Potentially guided bidirectionalized rrt* for fast optimal path planning in cluttered environments", *Robotics and Autonomous Systems*, vol. 108, 2018, pp. 13 – 27. https://doi.org/10.1016/j.robot.2018.06.013

[43] P. Vadakkepat, T. H. Lee, and L. Xin. "Application of evolutionary artificial potential field in robot soccer system", Annual Conference of the North American Fuzzy Information Processing Society, July 2001.

[44] J. Wang, H. Kimura, and M. Sugisaka, "Intelligent control for the vision-based indoor navigation of an a-life mobile robot", *Artificial Life and Robotics*, vol. 8, no. 1, 2004, pp. 29–33. 10.1007/s10015-004-0283-y

[45] L. Wang, "Automatic control of mobile robot based on autonomous navigation algorithm", *Artificial Life and Robotics*, vol. 24, no. 4, 2019, pp. 494–498. 10.1007/s10015-019-00542-0

[46] L. Xu, L. G. Zhang, D. G. Chen, and Y. Z. Chen, "The mobile robot navigation in dynamic environment", *2007 International Conference on Machine Learning and Cybernetics*, vol. 1, 2007, pp. 566–571. 10.1109/ICMLC.2007.4370209

[47] D. Zhenjun, Q. Daokui, X. Fang, and X. Dianguo. "A hybrid approach for mobile robot path planning in dynamic environments", 2007 IEEE International Conference on Robotics and Biomimetics (ROBIO), 2007, pp. 1058-1063. 10.1109/ROBIO.2007.4522310