J Intell Robot Syst (2018) 92:545–564
https://doi.org/10.1007/s10846-017-0739-7

Experience Learning From Basic Patterns for Efficient Robot Navigation in Indoor Environments

Olimpiya Saha[1] · **Prithviraj Dasgupta**[1]

Received: 21 November 2016 / Accepted: 2 November 2017 / Published online: 20 November 2017

Abstract In this paper we propose a machine learning technique for real-time robot path planning for an autonomous robot in a planar environment with obstacles where the robot possess no *a priori* map of its environment. Our main insight in this paper is that a robot's path planning times can be significantly reduced if it can refer to previous maneuvers it used to avoid obstacles during earlier missions, and adapt that information to avoid obstacles during its current navigation. We propose an online path planning algorithm called LearnerRRT that utilizes a pattern matching technique called Sample Consensus Initial Alignment (SAC-IA) in combination with an experience-based learning technique to adapt obstacle boundary patterns encountered in previous environments to the current scenario followed by corresponding adaptations in the obstacle-avoidance paths. Our proposed algorithm LearnerRRT works as a learning-based reactive path planning technique which enables robots to improve their overall path planning performance by locally improving maneuvers around commonly encountered obstacle patterns by accessing previously accumulated environmental information. We have conducted several experiments in simulations and hardware to verify the performance of the LearnerRRT algorithm and compared it with a state-of-the-art sampling-based planner. LearnerRRT on average takes approximately 10% of the planning time and 14% of the total time taken by the sampling-based planner to solve the same navigation task based on simulation results and takes only 33% of the planning time, 46% of total time and 95% of total distance compared to the sampling-based planner based on our hardware results.

Keywords Experience learning · Real-time robot path planning · Obstacle feature matching

✉ Olimpiya Saha
osaha@unomaha.edu

1 Computer Science Department, University of Nebraska at Omaha, Omaha, USA

1 Introduction

Autonomous navigation is a central issue in many robotics problems that have real-life applications. For example, autonomous vehicles used in civilian and military applications [27], assistive robots used for manipulating and grasping household objects [25], warehouse management robots [12], and extra-terrestrial rovers [34], employ navigation algorithms for controlling their movement. In autonomous navigation, a robot has to determine a path between a start and a goal location and navigate along the path while avoiding collisions with obstacles. Autonomous navigation is challenging because, in most real-life situations, the robot might not have an *a priori*, accurate map of its environment, and is also constrained by the limited range of its on-board proximity sensors. To successfully navigate, the robot has to find a collision-free path in real-time by determining and dynamically updating a sequence of waypoints that connect the robot's initial position to its goal position. In many existing robot navigation algorithms [7], a motion planner generates a route for the robot to follow and reach its goal. However, in situations where the robot does not have prior information about the location of obstacles in its environment, it has to modify its path whenever it detects a previously unknown obstacle that occludes its path towards its goal. This is a computationally expensive operation that can consume considerable time (order of a few seconds to minutes) especially in a large and complicated environment that is cluttered with multiple complex obstacles. Excessively expended path planning time also attenuates the robot's energy (battery) which in turn deteriorates the overall performance of the robot in achieving its mission.

To address this problem, in this paper, we make the insight that, although obstacles could be geometrically dissimilar in nature, they can share some generic features that are prevalent in many commonly encountered obstacles. Our main hypothesis is that if a robot can be initially demonstrated manually (e.g., through tele-operation) to perform navigation maneuvers around a set of simple obstacle boundaries, it can reuse these maneuvers, with suitable adaptations, to navigate efficiently in different, future environments with more complex obstacle boundaries, without having to resort to time intensive motion planning calculations. In this manner, navigation can be learned incrementally by a robot, without having to be trained independently for each new environment that the robot is working in. To validate our hypothesis we describe an algorithm called LearnerRRT, where, we first train a robot by navigating it in an environment where obstacles share some distinctive geometric features. The features of the obstacles perceived by the robot's sensors along with the corresponding movements or actions which enabled the robot to avoid the encountered obstacles are recorded and preserved by the robot in its repository for future reference. Subsequently when the robot needs to achieve a navigation task in a new environment, which requires it to navigate successfully around obstacles, it retrieves the best obstacle feature-action pair from its repository. For retrieving the best obstacle feature-action pair, the robot determines the obstacle feature having the highest similarity to its currently perceived obstacle feature and retrieves the action sequence corresponding to the best matched obstacle feature from its repository. The robot then adapts the action sequence by considering the geometric details of the current obstacle. We have tested our algorithm on a simulated Corobot robot using the Webots simulator as well as a physical robot within different environments having different obstacle geometries and spatial distributions while varying the start and goal locations of the navigation task given to the robot. Our results show that our proposed algorithm LearnerRRT can perform path planning in real time in a more time effective manner compared to sampling based techniques like Informed RRT*. LearnerRRT on average takes approximately 10% of the planning time and 14% of the total time taken by Informed RRT* to solve the same navigation task based on simulation results and takes only 33% of the planning time, 46% of total time compared to Informed RRT* based on our hardware results. With respect to navigation times and distances traveled LearnerRRT on average takes approximately only 23 percent more navigation time and 20 percent more distance and 80% of the navigation time and 95% of the total distance to solve a given task based on simulations and hardware results respectively. However, the difference in the simulation results is trivial when compared to the significant improvements observed in planning time and total time taken. We also performed experiments that assessed the learning performance of LearnerRRT. Our results show that LearnerRRT improves its performance with increase in the size of the library like any conventional learning-based algorithm. It was finally able to achieve a learning rate of 88 percent with the given maximum size of the library. A preliminary version of this work appeared as an extended abstract [33] and a non-archival technical report [32]. In this paper, we have completely rewritten the introduction, motivation and related work sections of the paper and added several new references. We have added analytical results about the performance of our algorithm in Section 4. In Section 5.1, we have added several new experimental results including comparisons of our approach with a state-of-the-art robot motion planning algorithm called Informed RRT* for different metrics including the planning and navigation times, and distances traveled by robots. We have also added new experiments to validate the learning performance of our algorithm in Fig. 9. Finally, the entire Section 5.2 describing hardware results of our proposed algorithm and Section 6 discussing the findings and results of our research, are new in this paper.

2 Related Work

Researchers over the decades have investigated techniques to enable autonomous navigation in robots by proposing different path planners designed to enable robots to plan a suitable path to be followed for the timely completion of a well-defined task. Among the earlier works in this area are graph-based planners like A*, sampling-based planners like Probabilistic Roadmap Planners(PRMs) [21] and Rapidly Exploring Random Trees(RRTs) [23]. Conventional A* algorithm works with a grid embedded in the robot's workspace and find the optimal paths based on heuristics. D^*, Lifelong-A^*, D^*-Lite are advanced graph replanning algorithms based on the original A^* algorithm which are constructed to correct a solution generated by A^* after the edge weights in the graph have changed.

PRMs are graph-based planners which are more geared towards multi-query planning. They construct roadmaps in the free configuration space by randomly sampling valid configurations and constructing simple paths satisfying collision constraints. The constructed roadmap is then queried for generating a path from the start to the goal. On the other hand, RRTs are tree-based planners which are more suited for single-query problems. RRTs explore C-space from some start configuration by extending nodes and joining the nearest neighbor in the tree to the new node only if the edge between the two nodes are valid. These algorithms have grown into a family of advanced path planning algorithms over the recent years like PRM^* [20], LazyPRM [5], OBPRM [2], UOBPRM [47], Spark-PRM [35], RRT-Connect [22], RRT^* [20], ERRT [6], DRRT [15], MP-RRT [48], LRF [18], $RRT^\#$ [3], RRTLocTrees [39] and RRT^x [26] where researchers have looked into several aspects to improve the original algorithm like more efficient sampling strategy, adaptability to dynamic environments and usability in environments with narrow passages. [13] presents a comprehensive review of different sampling-based motion planning methods with special emphasis on kinodynamic planning and motion planning in dynamic and/or uncertain environments. Sampling-based planners and their advanced variants have been proposed to solve more complicated problems of motion planning like planning in higher dimensional spaces and in multi-robot and networked environments. In [46], the authors propose four path planning algorithmic families which can be applied on metric occupancy grid maps (OGMs)- Probabilistic RoadMaps (PRMs), Visibility Graphs (VGs), Rapidly Exploring Random Trees (RRTs) and Space Skeletonization. In this work the authors defined metrics for path planning benchmarks as well as the actual benchmarks of the most common global path planning algorithms and an educated algorithm parameterization based on a global obstacle density coefficient. In [9], the authors investigate the coordination of multiple robots with pre-specified paths on the basis of two criteria- motion safety and travel time minimization. In this work, the authors proposed a search method which was applied on PRM to achieve the Pareto-optimal coordination solution for multiple robots. [10] proposes an adaptive path planning technique for multiple autonomous underwater vehicles (AUVs) which estimates the scalar field over a region of special interest. In this work, the authors utilize a measurable model consisting of multiple basis functions which represent the scalar field along with a selective basis function Kalman filter which estimates the model by exploiting the information accumulated by multiple AUVs. Additionally, the authors proposed the multi-dimensional rapidly exploring random trees star algorithm which was used as a path planning technique for the multi-AUV system under consideration. In [36], the authors consider a networked system composed of unmanned aerial vehicles (UAVs), automated logistics service stations (LSSs), customer interface software, system orchestration algorithms and UAV control software and aim to utilize this networked system to provide uninterrupted service to its customers in real-time. However, majority of these works assume that the robot has access to an initial map of the environment and the environment can change in the future thus introducing dynamicity to the same environment. In contrast to this, our problem considers that the robot is deployed in a completely unknown environment with no initial map of the environment. The robot can only perceive its local environment with the help of its onboard sensors. Another challenge encountered by these sampling-based planners is the higher sampling time required by them in environments with complex obstacle geometries. Our problem has some similarities with Spark-PRM where the authors addressed the problem encountered by these planners when they encounter narrow passages owing to the deficiency of valid samples within such regions. However, in our work we generalize the problem beyond narrow passages to more different varieties of complicated patterns by utilizing concepts from machine learning.

Machine learning has been a topic of interest in the robotics community over the recent years. In one of the earliest works in this direction, Fernandez and Veloso [16] proposed a policy reuse technique where an agent calculates its expected rewards from possible actions within a reinforcement learning framework to select whether to use the action prescribed by its learned policies to explore new actions. da Silva and Mackworth [11] extended this work by adding spatial hints in the form of expert advice about the world states. Recently, transfer learning has been proposed to leverage knowledge gained in one task to a related but different task [43] by transferring appropriate knowledge from related source tasks. Most transfer learning techniques rely on a source task selection strategy, where the most suitable source task is selected from the source task pool and applied to the target task in order to solve it. A related problem with such an approach is that if the source task is selected incorrectly, the target task can be peformed poorly suffer owing to irrelevant or 'negative' transfer of knowledge from the source task. In [41], the authors have addressed this problem using a transfer hierarchy. Approaches such as a human-provided mapping [45] and a statistical relational model [44] to assess similarities between a source and a target task have also been proposed to mitigate negative transfer. Other techniques to learn source to target task mappings efficiently include an experience-based learning framework called MASTER [42] and an experts algorithm which is used to select a candidate policy for solving an unknown Markov Decision Process task [40]. Our paper is along this direction of work, and our proposed LearnerRRT algorithm uses a feature matching algorithm called Sample Consensus Initial Alignment (SAC-IA) [30] from computer vision to mitigate the problem of negative transfer.

In contrast to using machine learning techniques to learn suitable actions, researchers have proposed techniques to reuse robots' paths, represented as a set of waypoints, learned from previous navigation experiences. Lien et al. [24] proposed a robot motion planner called ReUse-based PRM(RU-PRM) which constructs local roadmaps around geometric models of obstacles and stores them in a database; the roadmaps are later used to build a global roadmap for the robot to follow. Researchers have also proposed the concept of experience graphs [19, 28, 29] where the robot learns a network of paths from past experiences and uses it to accelerate planning whenever possible. The technique reduces to planning from scratch when no past experiences can be reused. In [37], the authors proposed a technique to create a global policy based on some sample trajectories collected during the training that eventually enables robots to perform navigation effectively from different parts of the same environments. In the Lightning and Thunder frameworks [4, 8], a robot learns paths from past experiences in a high-dimensional space and reuses them effectively in the future to solve a given manipulation task. In most of these approaches, the metric for calculating similarity to previous navigation tasks is based on similarity between the robot's start and goal locations between a previous task and the navigation task at hand. Similar to these approaches, our LearnerRRT algorithm exploits past knowledge if a considerable amount of relevance between the current obstacle and a previously encountered obstacle is identified. Otherwise, to avoid negative transfer, it reverts to planning from scratch using a state-of-the-art motion planner called Informed RRT* [17] to navigate in the target environment. However, in contrast to these techniques, our approach considers a higher level of granularity by learning and reusing actions for avoiding obstacles instead of actions between start and goal locations of previous navigation paths, to make the navigation apply across environments that can vary in size and obstacle nature and distributions. It is important to note here that in order to be fully successful and autonomous, the robots in these approaches need an optimal strategy to decide when to add trajectories, how many trajectories to add and from which starting points trajectories need to be added which are currently open challenges in this area of research. Moreover, these techniques use trajectory samples accumulated during training in a specific environment and use them to improve navigation during later runs in the same environment. In contrast to this approach, our technique is reusable across domains and can be useful in any environment assuming that the environments possess some inherent structures commonly observed in most of the real-world environments. In [38], the authors suggested an approach to transfer policy libraries to new environments by representing libraries in the feature space. However, their approach stores trajectories as longer paths from start to goal which presents higher challenges to transfer across environments with significant differences. In contrast our approach is based on transferring only local policies or sub policies around common obstacle patterns which help the robots to successfully avoid obstacles and then switch back to direct paths to the goal once line of sight to the goal is achieved. Hence, our approach improves navigation by improving obstacle avoidance capabilities in robots in one-shot manner. The adopted path can definitely be improved once the robot has access to few sample trajectories in its environment following similar strategies.

3 Problem Formulation

The symbols used in the following problem formulation and their descriptions are summarized in Table 1 above. Consider a wheeled robot situated within a bounded environment $Q \subseteq \Re^2$. We use $q \in Q$ to denote a configuration of the robot, $Q_{free} \subset Q$ to denote the free space, and $Q_{obs} = Q - Q_{free}$ to denote the space occupied by obstacles in the environment respectively. The action set for the robot's motion is given by $\mathcal{A} \subset \{[-\pi, \pi] \times \Re\}$; an action is denoted as $a = (\theta, d) \in \mathcal{A}$, where $\theta \in [-\pi, \pi]$ and $d \in \Re$ are the angle (in radians) the robot needs to turn and the distance it needs to move respectively. Performing an action a in configuration q takes the robot to a new configuration q', which is denoted mathematically as $a(q) = q'$. A path is an ordered sequence of actions, $P = (a_1, a_2, a_3, ...)$. Let T denote a navigation task for the robot, $T = (q_{start}, q_{goal})$, where $q_{start}, q_{goal} \in Q_{free}$ denote the start and goal locations of T respectively. The objective of the robot is to find a sequence of actions that guarantees a collision free navigation path connecting q_{start} and q_{goal}. In other words, no action along the navigation path should take the robot to a configuration that is in collision with an obstacle. Using the mathematical notations above, the navigation problem facing the robot can be formally stated as the following: Given navigation task $T = (q_{start}, q_{goal})$, find navigation path for task T, $P_T = (a_1, a_2, a_3, ...) \subset \mathcal{A}$ such that $\nexists a_i \in P_T$, where $a_i(q) = q' \in Q_{obs}$. To solve the navigation problem, we propose the LearnerRRT algorithm that first creates a library of obstacle patterns and robot actions and reuses the learned actions to navigate around obstacles. The LearnerRRT algorithm proceeds in two steps that are described in the subsequent subsections.

3.1 Library Creation

The robot is first trained in an offline manner to find a collision-free path for navigating around obstacles that have different but well-defined geometries. Each navigation task used for training is called a source task. We assume that the environments in which the robot will perform navigation tasks later on will have obstacles with significant similarities in their boundary patterns with respect to the source tasks, although the orientation and dimensions of individual obstacles in the later environments might vary.

Table 1 Symbols used in Problem Formulation and their descriptions

Symbols	Description
Q	Bounded environment where the robot operates.
Q_{free}	Free space in the environment.
Q_{obs}	Space occupied by obstacles in the environment.
q	A configuration of the robot.
A	Action set of the robot.
a	An action of the robot.
q_{start}	Configuration of the robot at the start.
q_{goal}	Configuration of the robot at the gaol.
T	Navigation task of the robot.
P_T	Navigation pat for task T.
LAB	The set of labels for obstacles learned during the training phase.
L	Library containing the set of actions learned during the training.
Lab	The label of an obstacle in the library L created during training.
G_{lab}	Set of gaol locations spatially distributed around an obstacle with label lab in the library L.
g_j	Each gaol location in G_{lab}
LS_{lab}	The set of 2D coordinates which describes the obstacle with label lab created during the training.
$path_{lab_j}$	Ordered set of distance and orientations required to follow the path to gaol location g_j around the obstacle with label lab
LS_{obs}	The set of 2D coordinates which describes the obstacle obs perceived during the testing phase while the robot navigates in the environment.
SF	Scaling factor expressed as the ratio of the size of the perceived obstacle LS_{obs} and the best matched obstacle pattern LS_{lab} obtained form L
$\theta_{obd,Lab}$	The transformation angle between LS_{obs} and LS_{lab}
JI	Jaccard Index calculated to determine the extent of overlap between LS_{obs} and LS_{lab} after applying scaling and transformation.
JI_{Thr}	Threshold value for calculated jaccard index to determine the extent of overlap between the two obstacle.
lab_{match}	The obstacle label from L with which the perceived obstacle has the highest match.
$path_{lab_{match}}$	The set of paths retrieved from the library corresponding to the obstacle pattern lab_{match}.
$path_{lab_{match.J}}$	j^{th} path obtained from $path_{lab_{match}}$
g_j^{scale}	The last waypoint in a path obtained by scaling and transforming

We consider four well-defined obstacle geometry patterns as source tasks - cave, column or passage, corner and block, as shown in Fig. 1a-d. Each pattern is identified by a label corresponding to its name. The set of labels is denoted by LAB. The actions learned by the robot during training are stored in an action library, L, as described below. We would like to mention here that these four patterns specifically evolved from our observations of different real-world environments as patterns that are most frequently present in majority of these environments and regions where it gets difficult for existing path planners to plan paths circumnavigating them. To the best of our knowledge there is no formal study in this area aiming to determine a succinct set of real-world obstacle patterns.

To construct the action library, L, for a source task corresponding to an obstacle with label lab, the robot is initially placed in front of the obstacle in such a way that the robot's range sensor can perceive the obstacle's inner boundary. A set of goal locations, G_{lab}, corresponding to positions where the robot will have avoided the obstacle are specified. Locations in G_{lab} are spatially distributed uniformly around the outer obstacle boundary, such that the straight line path from the robot's initial position to each $g_j \in G_{lab}$ is in collision with the obstacle, as shown in Fig. 1a-d. The range or proximity data obtained from the robot's sensor while perceiving the obstacle from its initial position is stored as a set of coordinates in the 2D plane denoted by $LS_{lab} = \{(\hat{x}, \hat{y})\}$. The robot internally constructs the obstacle boundary from the range data and uses the Informed RRT* path planner [17] to plan a path to each of the goal locations. The path returned by Informed RRT* consists of an ordered set of waypoints. It is post-smoothed to reduce any unnecessary turns in the path. The smoothed path of goal $g_{lab,j} \in G_{lab}$, $path_{lab,j} = \{(\hat{d}, \hat{\theta})\}$, is an ordered set of distances $\hat{d}$ and orientations $\hat{\theta}$ required to follow the path. This path is stored in the action library. Each path stored in the action library is indexed by its obstacle label, and for each label by the different goal locations for that label. The action library after completing all the tasks in the source task set is given by Eq. 1.

$$L = \cup_{lab \in LAB}(LS_{lab}, Path_{lab}) \tag{1}$$

where $Path_{lab} = \cup_{g_{lab} \in G_{lab}} path_{g_{lab}}$.

The robot records both the obstacle contours in the form of streams of 2D contours and the corresponding planned paths generated by InformedRRT* while solving the different test cases during the training phase. The robot accesses

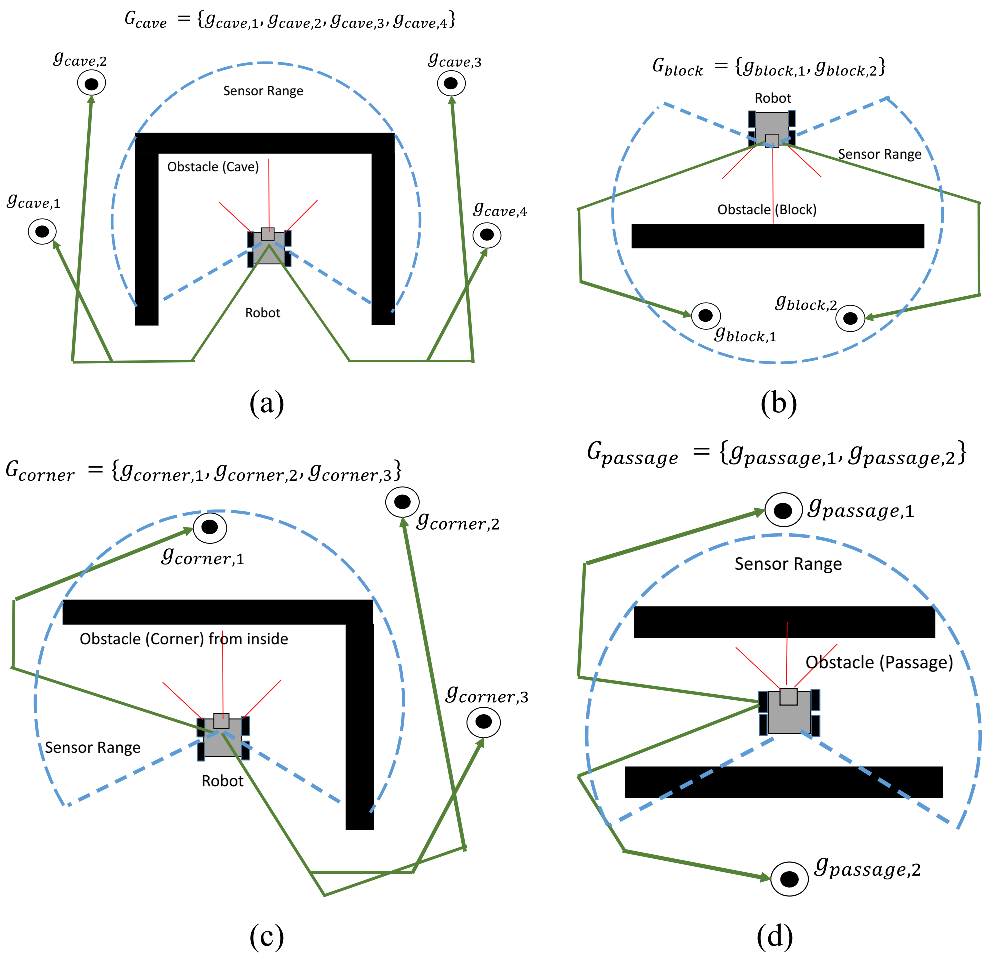

Fig. 1 **a-d** Different obstacle patterns used as source task to construct the action library. The robot's initial position corresponds to a location from which its sensor can perceive the entire obstacle's inner boundary. Different goal locations for each obstacle are denoted by the set G_{cave}, G_{block}, G_{corner} and $G_{passage}$ respectively

this information later on while solving future navigation tasks. It is important to mention here that our approach in its core combines experience-based learning technique with the concept of transfer learning where information related to environmental obstacle patterns and the corresponding obstacle avoidance maneuvers learned in one environment are transferred across other environments to facilitate faster navigation.

3.2 Obstacle Avoidance Using Learned Navigation Actions

After learning actions for avoiding common obstacle patterns, the robot (Fig. 2) is given a navigation task $T = (q_{free}, q_{goal})$. Note that the new navigation task can be given in a different environment than the one in which the action library was created. The general scheme that the robot uses after constructing its action library is to reuse actions from its action library, after suitable adaptations, when it encounters an obstacle while navigating towards the goal. The pseudo-code for our proposed LearnerRRT algorithm based navigation is given in Algorithm 1. While moving towards the goal, when the robot encounters an obstacle obs, its laser sensor scans the environment and records the proximity data from the obstacle, $LS_{obs} = \{(\hat{x}, \hat{y})\}$, as a set of coordinates in the 2D plane. The laser actually returns a sequence of distances where it gets obstructed owing to the presence of the obstacle. It then converts the laser distances into a sequence of 2D coordinates based on its current location and the laser bearing at that instant. In order to ensure safe navigation, the robot always has to maintain a minimum distance from the obstacle(s) present in the environments which is determined from the robot's sensor readings. The robot records an obstacle as soon as the laser sensor returns a value lower than the minimum safety distance. It then uses a state-of-the-art algorithm for aligning object features called Sample Consensus Initial Alignment(SAC-IA) algorithm [30], to match LS_{obs} with the obstacle proximity data for the different obstacles recorded in the action library L.

For each $lab \in LAB$, the LS_{obs} data is first preprocessed to scale it approximately to match LS_{lab}. For determining the approximate scaling factor, we perform a principal component analysis(PCA) on each of LS_{obs} and

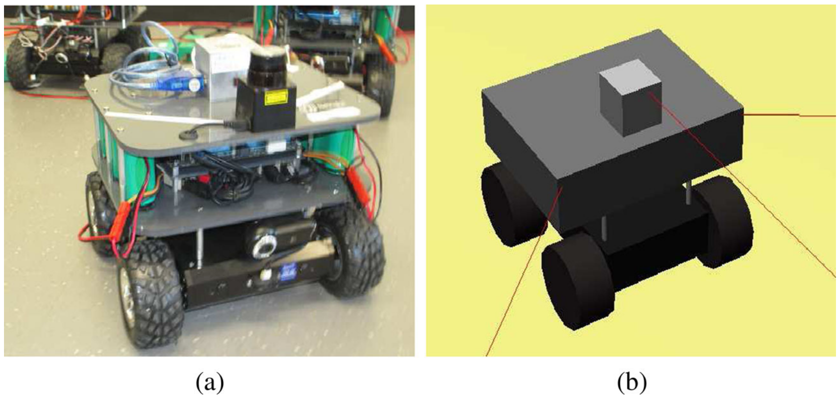
Fig. 2 **a** Photograph of a Corobot robot, **b** simulated Corobot robot with visible distance sensor rays used for simulations within Webots

LS_{lab} and retrieve the largest eigenvalues from each of the computed PCA, which reflects the maximal variance of each data set. The scaling factor SF is given by Eq. 2.

$$SF = \sqrt{\frac{\max eig(LS_{obs})}{\max eig(LS_{lab})}} \tag{2}$$

The SAC-IA algorithm takes two point clouds, corresponding to scaled LS_{obs} and LS_{lab} respectively, as inputs and aligns the first point cloud with respect to the second to get the best matching between the two. The algorithm returns the corresponding transformation $\theta_{obs,lab}$ between the two point clouds. The extent of match between calculated by SAC-IA between LS_{obs} and LS_{lab} is measured by analyzing their Jaccard Index, which reflects the extent of overlap between two geometric shapes. To limit negative transfer between the two patterns, we admit only those obstacle patterns whose Jaccard Index, JI, is greater than a certain threshold JI_{Thr}, so that two patterns with very low similarity are not considered in the match calculation. The obstacle label with which the currently perceived obstacle has highest match is given by Eq. 3.[1]

$$lab_{match} = \arg \max_{lab \in LAB} JI(LS_{lab}, LS_{obs}) \tag{3}$$

subject to $JI(LS_{lab}, LS_{obs}) > JI_{Thr}$.

Once the best obstacle match in the action library lab_{match} has been determined, the robot retrieves the set of paths $Path_{lab_{match}}$ from the library. It applies the transformation angle $\theta_{obs,lab_{match}}$ returned by SAC-IA to each of the orientations in the $Path_{lab_{match}}$, followed by applying the same scaling factor calculated during preprocessing the two point clouds before applying SAC-IA, to each of the distances in $Path_{lab_{match}}$. It then selects the path $path_{lab,j} \in Path_{lab_{match}}$ as the path whose scaled goal location, g_j^{scale} minimizes the distance to q_{goal}, given by $j = \arg\min d(g_j^{scale}, q_{goal})$, where, $g_j^{scale} = scale(g_{lab,j})$ and $\nexists a = (\hat{d}, \hat{\theta}) \in path_{lab_{match},j}$ s.t. $q' = a(q) \notin Q_{free}$. The last constraint ensures that the robot does not collide with an obstacle while following the scaled path computed by our algorithm, as described below.

Finally, the robot does two post-processing steps to correct and optimize the selected path. The selected path is first checked for collision with the obstacle at individual segments consisting the path. If any of the component segments gets within very close proximity of the obstacle or intersects with the obstacle, we apply our path correction strategy which can be classified in two ways depending on the position of the unsafe segment in relation to the whole path. If the unsafe segment has both predecessor and successor segments, then new waypoints are sampled along the direction of the predecessor and successor segment and the unsafe segment in the path is replaced by the new segment formed by the newly sampled waypoints. On the other hand, if the unsafe segment has only a predecessor or a successor in the path, new points are generated by sampling points in the direction of the predecessor/successor segment and by extrapolating the affected segment. The node at the juncture of the affected segment and predecessor/successor is replaced by the segment generated by the newly sampled points. In the above procedures sampling continues till the collision constraints are satisfies by the newly generated segments. For segments that are not colliding with the obstacle, the path is interpolated between the extremities of two successive segments to reduce the length of the path. Hence, LearnerRRT can handle minor variations in the learned obstacle patterns by adapting paths suitably following the discussed path correction/optimization approach. It is important to note that the post processing step discussed in this section will be able to rectify paths for robots as long as there are no occlusions and the training and testing environments significantly share similarities in obstacle patterns with minor variations. In case the post processing procedure is unable to correct the planned path (owing to the complexity, occlusions or uniqueness of a particular obstacle discovered in the testing phase), the

[1]If LS_{obs} does not match any LS_{lab} from the action library, the robot uses the Informed RRT* planner to plan its path around obstacle *obs* instead of attempting to learn from past navigation data.

robot uses a local path planner which plans a path from scratch.

The robot follows the mapped path retrieved from the action library until is reaches g_j^{scale}. At g_j^{scale}, the robot might still perceive an obstacle, e.g., if the obstacle extended beyond the robot's sensor range perceived from its initial location. In such a case, it reapplies the above steps of reusing a path from its action library based on the perceived obstacle proximity data. To avoid retrieved paths from forming a cycle inside the same obstacle, the robot remembers the direction in which it had turned when it first encountered this obstacle, and gives a higher preference to successive retrieved paths from the action library that are in the same direction as the first turn.

We would like to emphasize here that although *LearnerRRT* works with 2D point clouds in planar environments, the algorithm can be easily extended to 3D or higher dimensional spaces as the components used by LeanerRRT for pattern matching like FPFH, SAC-IA and PCA have been successfully applied in higher dimensional spaces for object registration and manipulation by a humanoid robot [30, 31]. As the action library is preserved as sequences of angles and distances, it will be applicable in 3D and higher dimensional spaces as well.

4 Analysis

Lemma 1 *LearnerRRT will generate a collision free path which will enable the robot to avoid the encountered obstacle and reach a point as close as possible to the goal*

For any real-time path planning problem, the robot has to determine a collision-free path from $start$ to $goal$. Our planner at first optimistically assumes that there exists no obstacle in the environment and follows the minimum cost path or the direct path from the $start$ to $goal$. While the robot continues in its path, two cases can arise. (1.) The robot encounters an obstacle in its path and (2.) The robot does not encounter an obstacle in which case it continues its motion. For the first case, our LearnerRRT planner will be able to determine the best matched obstacle pattern lab_{match} and return the best mapped path from its library L. As we use Informed-RRT* to build our library by setting goal positions around the obstacles, it can be claimed that by following the Informed-RRT* generated path retrieved and mapped from the library, the robot will be able to effectively avoid the obstacle in its path. We select the path from the library corresponding to the best matched pattern by minimizing the total distance of the path to the goal. This ensures that by following the path, the robot will definitely reach a point which is as close as possible to the goal. Hence, the capability of Informed RRT* to generate collision free paths integrated with our minimal path distance heuristic guarantees that LearnerRRT will generate a collision free path which will enable the robot to avoid the encountered obstacle and reach a point as close as possible to the goal.

Algorithm 1 LearnerRRT algorithm

```
   Input: $currX, currY, goalX, goalY$
   Output: A collision-free $path$ from start to goal
1  $path \leftarrow \emptyset$
2  repeat
3      $adjustToGoal(goalX, goalY)$
4      repeat
5          Move towards goal
6      until obstacle is detected;
7      $ls \leftarrow getLaserScan(currX, currY)$
8      $ls^{scaled} \leftarrow scaleLaserScan(ls)$
9      $pattern, \theta_{match} \leftarrow findBestMatch(ls^{scaled}, L)$
       $path^{best} \leftarrow findBestPath(L, pattern)$
       $path^{transf} \leftarrow transformPath(path^{best}, \theta_{match})$
       foreach $node \in path^{transf}$ do
10         if node collides with obstacle then
11             $path^{mapped} \leftarrow correctPath(path^{best})$
12     foreach $edge \in path^{transf}$ do
13         if edge collides with obstacle then
14             $path^{mapped} \leftarrow correctPath(path^{transf})$
15     if no node collided with obstacle and no edge
       collided with obstacle then
16         $path^{mapped} \leftarrow optimizePath(path^{transf})$
17     repeat
18         if sensor detects obstacle then
19             go to Line 7 and repeat rest of the steps
20         else
21             $followPath(path^{mapped})$
22     until Line of Sight $LOS$ is achieved or path gets
       exhausted;
23 until $(goalX, goalY)$ is reached;
```

Theorem 1 *For a well-defined and relatively similar environment E, LearnerRRT terminates and the solution it returns is guaranteed to be no worse than Nk times the optimal solution cost in environment E where N is the number of obstacles and k is the cumulative error in path cost owing to scaling error and transformation error.*

Recall that LearnerRRT returns lab_{match}, the best-matched obstacle pattern from the library $\theta_{obs,lab_{match}}$ and the transformation angle between currently perceived obstacle pattern and the best match. Let D denote the total distance of the optimal path between $start$ and $goal$ while avoiding obstacles, d_{last} denote the length of the last segment in the path and θ denote the angular error in the transformation angle returned by SAC-IA.

Two main factors which contribute to the error in the predicted path by LearnerRRT are errors from approximate scaling and the same from the transformation angle returned by Informed-RRT*. If we consider the error in the determined scaling factor is SF_{err}, then the error in the path cost owing to inappropriate scaling can be expressed by Eq. 4.

$$Error_s = SF_{err} D \tag{4}$$

Hence, inappropriate scaling will result in greater distance traversed which is expressed as the error in the above equation. The error due to erroneous scaling is illustrated in Fig. 3a. On the other hand, it is important to note that error in the transformation angle will result in the deviation of the path from the appropriate mapped path in the library which will result in a path that is not as close to the goal as the perfect mapped path. The distance of both the paths remain constant though. The error due to approximate transformation can be expressed by Eq. 5.

$$Error_{tf} = \theta d_{last} \tag{5}$$

Considering the maximum angular error as θ_{max}, Eq. 5 gets replaced by Eq. 6.

$$Error_{tf} = \theta_{max} d_{last} \tag{6}$$

The error due to error in transformation is illustrated in Fig. 3b. Considering that there can be N obstacles in the robot's path, the total error is expressed by Eqs. 7 and 8.

$$\Delta C = \frac{C_{LearnerRRT}}{C_{best}} = \sum_{i=1}^{N} (Error_s + Error_{tf}) \tag{7}$$

$$\Delta C = N(SF_{err} D + \theta_{max} d_{last}) = Nk \tag{8}$$

where ΔC is the error in total path cost, C_{best} is the cost of the optimal path predicted by Informed RRT* and $C_{LearnerRRT}$ is the cost of the path generated by LearnerRRT.
This error propagates along the path predicted by the planner which deviates it from the optimal path. Hence, it can be concluded that LearnerRRT terminates and the solution it returns is guaranteed to be no worse than Nk times the optimal solution cost in the environment. It is worth mentioning here that as the error in the scaling factor will result in the difference of distance of the entire path, the latter is considered in the calculation of $Error_s$. In contrast due to the error in transformation angle, the distance covered by the robot to avoid the obstacle still remains the same. The excess distance covered by the robot is a consequence of covering a higher distance to the goal after avoiding the obstacle. Hence, only the length of the last segment is considered for the calculation of $Error_{tf}$ in the worst case scenario.

Theorem 2 *LearnerRRT algorithm has a time complexity of $\mathcal{O}(Nn_1n_2)$ where N is the number of obstacles encountered by the robot, and, n_1 and n_2 are respectively the number of maximum points in an obstacle pattern present in the library and the number of points in the currently perceived obstacle pattern.*

From Algorithm 1, it can be observed that LearnerRRT mainly consists of five components- scaling of the obstacle data in the form of point clouds to be compatible with each other (scaleLaserScan function), pattern matching (findBestMatch function) and fitness score calculation, determining the best path for the selected candidate obstacle (findBestPath function), transforming and scaling the retrieved path to be applicable for the current obstacle and finally correcting/optimizing the path if required. We evaluate the time required for each of the components separately

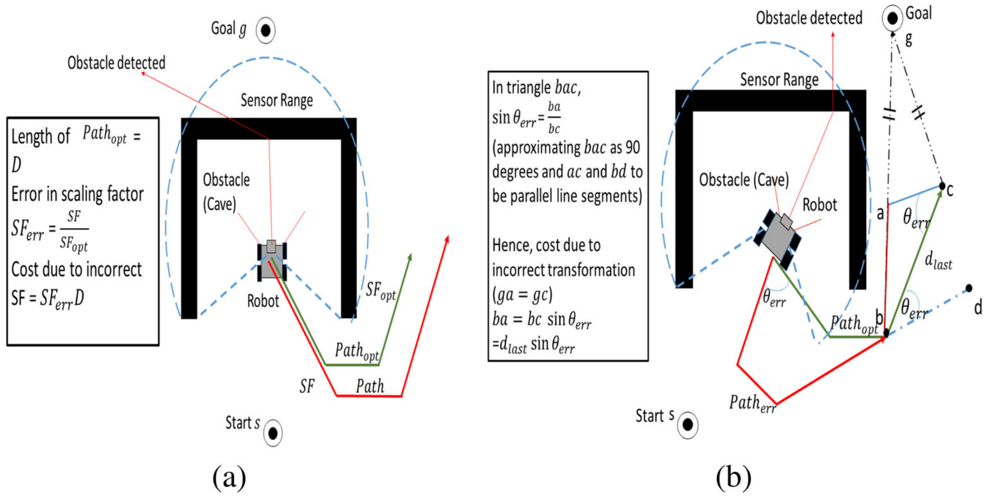
Goal g
Obstacle detected
Sensor Range
Length of $Path_{opt}$ = D
Error in scaling factor
$SF_{err} = \frac{SF}{SF_{opt}}$
Cost due to incorrect SF = $SF_{err}D$
Obstacle (Cave)
Robot
SF_{opt}
$Path_{opt}$
SF
Path
Start s
(a)
Obstacle detected
Goal g
In triangle bac,
$\sin\theta_{err} = \frac{ba}{bc}$
(approximating bac as 90 degrees and ac and bd to be parallel line segments)
Hence, cost due to incorrect transformation
(ga = gc)
$ba = bc \sin\theta_{err}$
$= d_{last} \sin\theta_{err}$
Sensor Range
Obstacle (Cave)
Robot
a
c
θ_{err}
d_{last}
θ_{err}
$Path_{opt}$
b
θ_{err}
d
$Path_{err}$
Start s
(b)

Fig. 3 **a** Illustration of optimal path (green lines) and path covering more distance under scaling error (red lines) (b) Illustration of optimal path (green lines) and path under transformation error (red lines)resulting the robot to travel more distance to reach the goal (segment *bc*) to analyze the time complexity of the *LearnerRRT* algorithm. In order to scale the current obstacle point cloud with the obstacle patterns from the library *LearnerRRT* performs principal component analysis (PCA) on each of the obstacle point cloud. PCA primarily consists of covariance matrix calculation and its eigenvalue decomposition which results in a time complexity $T_1 = \mathcal{O}(|L+1|(nD^2 + D^3))$ [14] where $n = \rho(obs)$ is the number of points in the corresponding obstacle obs, D is the dimensionality of the point clouds and $|L|$ indicates the size of obstacles in the library L. As *LearnerRRT* works with 2D point clouds and the size of library is constant, the above expression can be reduced to $T_1 = \mathcal{O}(n)$. Let us consider that $n_1 = max_{obs \in L}\rho(obs)$ refers to the number of points in the largest obstacle present in the library. Hence, the worst case time complexity for the scaling component evaluates to $T_1 = \mathcal{O}(n_1)$. Next, we evaluate the time complexity from the pattern matching and fitness score calculation component. SAC-IA computes the Fast Point Feature Histograms (FPFH) of each of the point clouds in the library along with the current point cloud which results in $T_{2_1} = \mathcal{O}(|L+1|rn)$ where $|L|$ refers to the number of obstacle patterns in the library, r is the size of the neighborhood radius considered by SAC-IA and $n = \rho(obs)$ is the number of input points in the corresponding obstacle. As the size of the library is constant, the above expression can be simplified as $T_{2_1} = \mathcal{O}(rn_1)$ where $n_1 = max_{obs \in L}\rho(obs)$ refers to the number of points in the largest obstacle present in the library. In order to evaluate the best match, *LearnerRRT*calculate the fitness score of the current obstacle with each of the obstacle patterns in the library after applying the transformation and scaling retrieved in the previous step resulting in the time complexity of $T_{2_2} = \mathcal{O}(|L|nn_2)$ where $n_2 = \rho(obs)$ is the number of points in the obstacle encountered by the robot while navigating in the testing phase. Considering the size of library to be constant, the worst case time complexity evaluates to $T_{2_2} = \mathcal{O}(n_1 n_2)$. Hence, the time complexity of the second component described above is equivalent to $T_2 = T_{2_1} + T_{2_2} = \mathcal{O}(rn_1) + \mathcal{O}(n_1 n_2)$. As the size of radius r considered by SAC-IA is generally small, hence, $r << n_1$, resulting in the effective time complexity of $T_2 = \mathcal{O}(n_1 n_2)$. Once the best matched obstacle pattern from the library L is discovered as lab, the best path is obtained from L_{lab} after the path is scaled and transformed using the information from PCA and SAC-IA. Considering the transformation and scaling assumes constant time, the best path search in worst case takes $T_3 = \mathcal{O}(|L_{lab}|)$ time. Considering post-processing of the selected path (correction/optimization) takes constant time, the time complexity can be evaluated as $T = T_1 + T_2 + T_3 + T_4 = \mathcal{O}(n_1) + \mathcal{O}(n_1 n_2) + \mathcal{O}(|L_{lab}|) + O(1) = \mathcal{O}(n_1 n_2)$. This whole process occurs each time a new obstacle is encountered by the robot. Hence, the final time complexity can be expressed as $T = \mathcal{O}(Nn_1 n_2)$, where N is the number of obstacles detected by the robot. Hence, proved.

5 Experimental Setup and Results

We have verified the performance of the LearnerRRT algorithm using both simulated as well as hardware Corobot robots on the Webots version 6.3.2 simulator. The robot model utilized for the simulations is the Coroware Corobot robot - an indoor, four-wheeled, skid-steer robot. The footprint of the robot measures 40 cm ×30 cm. The robot has a Hokuyo laser sensor with a 270° field of view and a range of approximately 5 m. An indoor localization device called a Hagisonic Stargazer, with a localization accuracy of ±2 cm was added to the robot. On the simulated robot, a GPS and a compass node was used to emulate the behavior of the localization device. A laser sensor was also added on the simulated robot to enable the robot perceive its environment. A photograph of the Corobot robot is shown in Fig. 2a and the simulated robot within Webots is shown in Fig. 2b.

5.1 Simulation Results

We have performed experiments in the navigation domains using simulated environments. Our test environments have dimensions of 22 m by 22 m and have different distributions of obstacles with obstacles varying in scales and alignments in the environments. On average, in our environments obstacles cover approximately about $40-50$ percent of the environment spaces. Figures 4a-d above illustrate different test environments used in our experiments; (0, 0) and (22, 22) correspond to the bottom left and top right corners of each environment. Motion planning algorithms that plan from scratch based on RRTs and their variants have been used extensively in recent robot motion planning literature. Therefore, we have selected Informed RRT* as the baseline approach to compare our technique with, as it is the most recent and most improved variant of RRT-based motion planning algorithms. Different test cases have been created by selecting different start-goal pairs in each of the environments, as shown in Table 2. The separation distance between the start and goal locations ranges between $15-20$ meters. We have selected the start-goal pairs for our test cases in such a manner that the direct path connecting the start to the goal contains maximum number of obstacles in it. This means that the robot had to replan its path multiple times while navigating from the start to the goal. In order to make the test cases representative, we have selected start and goal locations from different regions of the environments. For comparing the performance of our algorithm with Informed RRT*, we have primarily used three main

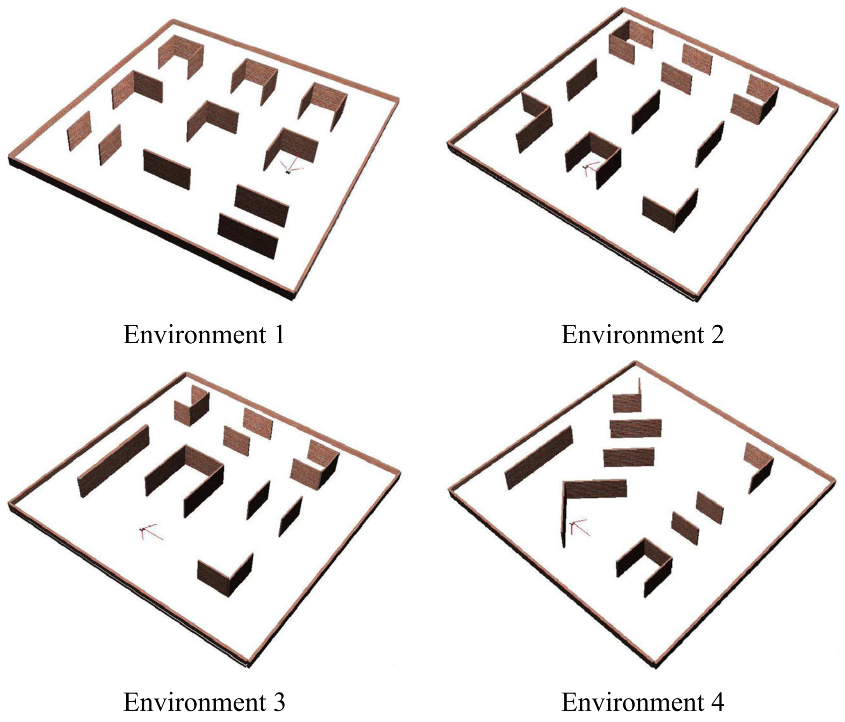

Fig. 4 Environments used for testing proposed LearnerRRT algorithm. (0, 0) and (22, 22) correspond to the leftmost and top rightmost corners of each environment

measures- the planning time to predict the path to be undertaken by the robot, the total time that the robot requires in order to traverse the entire arena and lastly we also compared the total distance that the robot navigated in each of the cases in order to reach the goal. All our tests were conducted on a computer with four, 3.20 GHz cores and 12 GB RAM. The algorithms are implemented in C++. We have used the Point Cloud Library PCL 1.6.0 for the implementation of the SAC-IA module in our algorithm. The collision threshold to avoid getting within close proximity of the obstacle using the mapped path calculated by our algorithm was set to 0.8 m. The threshold for matching using the Jaccard Index, JI_{Thr} was set to 0.3 m. This means we considered two points in the source and the target obstacles have overlapped if they lie within the proximity of 0.3 m to each other.

First, we have created the action library by generating paths for each of the source task obstacle patterns using the Informed RRT* algorithm. For Informed RRT* we have set a sampling distance of 0.3 m which is approximately equivalent to the width of the robot and the total number of iterations N to be 500. Paths are generated for each pattern by setting the start position of the robot at the center of the obstacle and goal positions across multiple locations around the obstacle patterns, as illustrated in Fig. 1. For two of our patterns, corner and block which can be observed both from an internal as well as an external position with respect to the obstacle, the robot's initial position is varied in addition to setting different goal locations. Environments used for building the library are of dimensions 9 meters by 12 meters. The separation distance between the start and goal locations ranges between 3 to 5 meters. Our library consists of 4 patterns - cave, corner, block and passage and a total of 16 paths for navigating different goal locations around each

Table 2 Coordinates of the different start and goal locations used as test cases for the four environments shown in Fig. 4

	Start	Goal
Env. & Test Case		
1	8.61, 11.26	19.62, 9.21
2	2.62, 18.34	19.62, 9.21
3	16.40, 4.18	20.80, 19.88
4	0.47, 10.53	20.80, 19.89
Environment 1		
1	0.72, 18.37	15.93, 16.75
2	0.63, 4.99	20.34, 4.77
3	0.60, 18.68	20.34, 4.77
4	17.42, 4.87	11.26, 17.87
5	0.54, 18.27	11.37, 4.89
Environment 2		
1	0.63, 0.72	20.80, 19.88
2	1.9, 13.9	18.27, 8.16
3	1.9, 13.9	10.22, 12.08
4	17.20, 1.68	20.80, 19.88
5	0.86, 3.99	20.86, 4.51
Environment 3		
1	0.61, 3.91	19.62, 9.21
2	0.60, 18.68	20.34, 4.77
3	1.9, 15.9	19.62, 9.21
4	0.61, 3.91	15.93, 16.75
Environment 4		

pattern. Figures 5 and 6 show the plots of different paths generated by Informed RRT*, which are then stored in our action library.

We performed our experiments on a total of 18 test cases executed across 180 runs and uniformly distributed over the four test environments shown in Fig. 4. As our algorithm uses SAC-IA as a pattern matching algorithm and SAC-IA returns the approximate transformation between the current obstacle pattern and the best matched pattern from the library, there exists a randomness factor in our algorithm which in effect helps the robot to find its way to the goal in spite of the presence of multiple obstacles in its route. In order to account for the randomness factor, we have executed ten runs for each of the test cases and recorded the means and the standard deviations for both time and distance.

Figures 7 and 8 show the comparative planning time, total time and distance traveled by the robot for solving the test cases by following our LearnerRRT algorithm and Informed RRT*. It can be observed that for all the 18 test cases, the planning time and total time taken by LearnerRRT is much lesser when compared to Informed RRT*. For our experiments, we have set the total number of iterations N to be 3000 for Informed RRT* and a step size of 1 meters. Informed RRT* was tested to give the best performance with these parameter values in our environments. When compared with the mean of the planning times and total times for all the test cases across all environments, LearnerRRT was found to beat Informed RRT* approximately by a factor of 10 for planning time and a factor of 7 for total time. This huge improvements of LearnerRRT in terms of planning time and total time stems from the fact that our algorithm reuses knowledge in the form of previously encountered paths from the library instead of planning paths from scratch as done by most contemporary

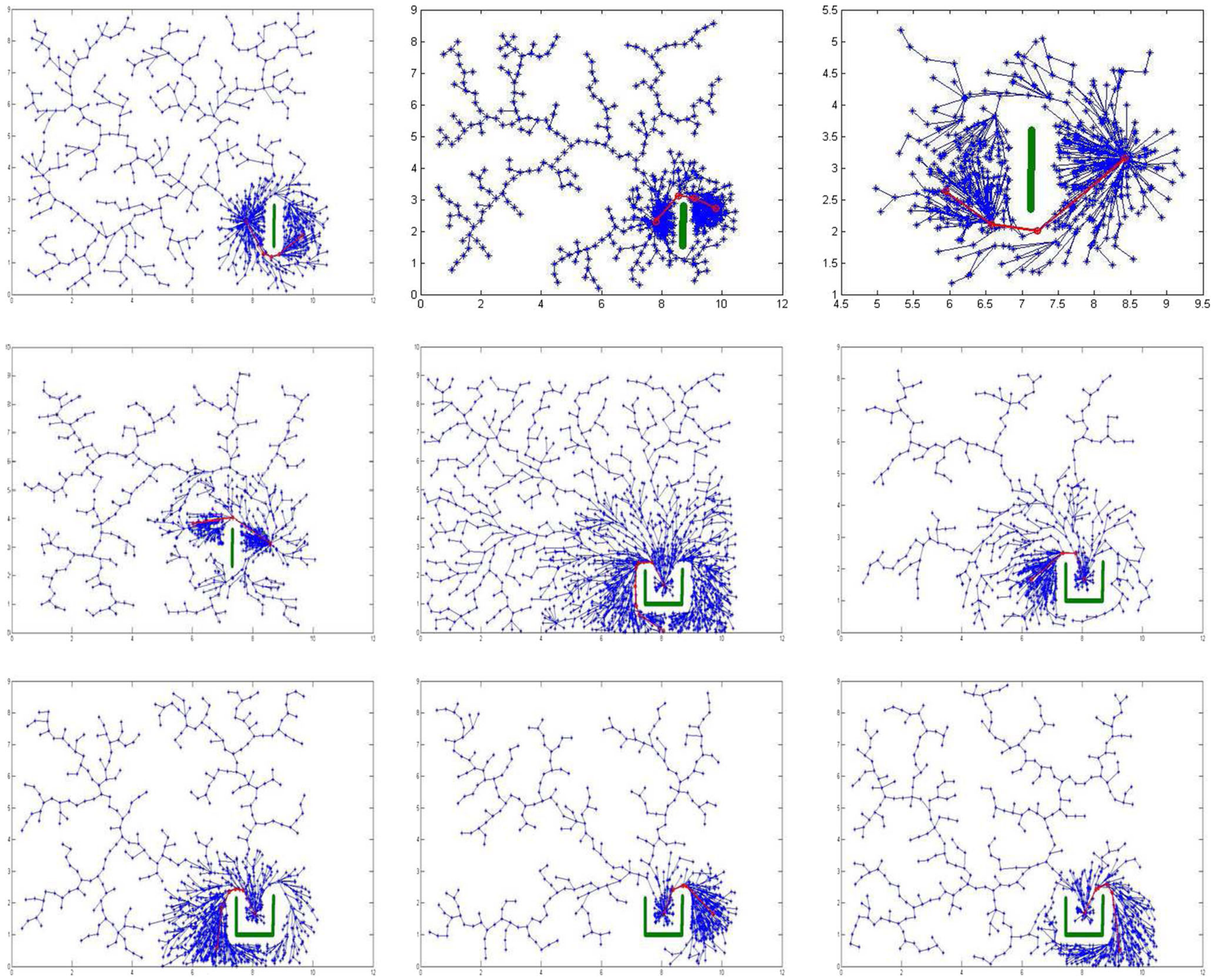

Fig. 5 Paths obtained for different source task obstacle patterns using Informed RRT*. The green contour represents the obstacle. The blue starred edges represents the paths generated by Informed RRT* and the red edges represents the final paths obtained for the specified goal around the obstacles

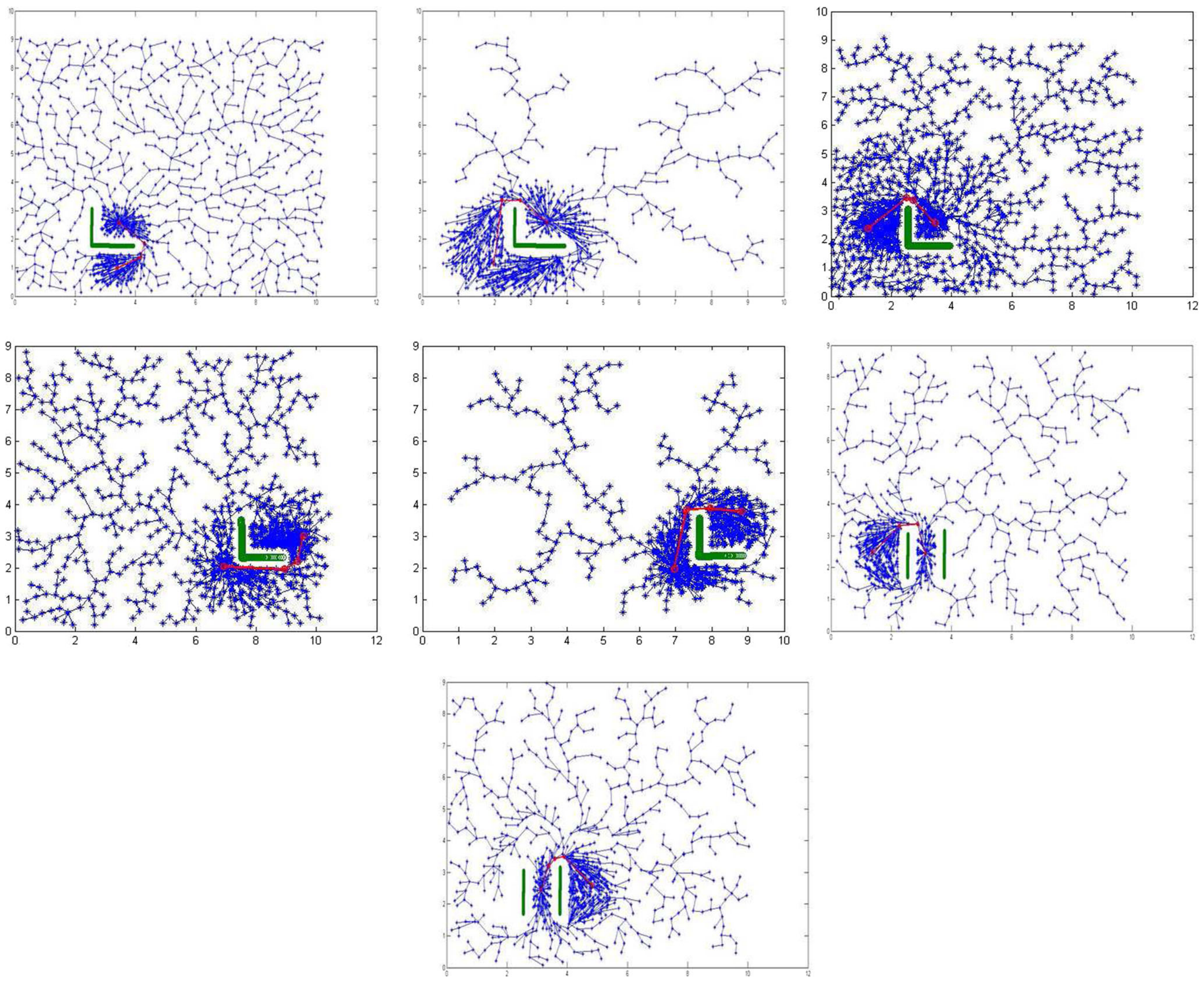

Fig. 6 Paths obtained for different source task obstacle patterns using Informed RRT*. The green contour represents the obstacle. The blue starred edges represents the paths generated by Informed RRT* and the red edges represents the final paths obtained for the specified goal around the obstacles

sampling-based planners like Informed RRT*. The total time covers the time taken by LearnerRRT to iteratively plan the paths for the robot once it encounters an obstacle as well as the navigation time taken by the robot to follow the plan and ultimately reach the goal. This means that both planning time and navigation time approximately contributes for 50 percent of the total time taken by the robot. From our results it can be seen that the maximum planning time and total time taken by LearnerRRT is approximately 6 minutes and 4.65 minutes respectively. Considering the approximate separation distance between the start and goal locations to be $15 - 20$ meters, the environmental dimensions to be 22 by 22 meters, the presence of obstacles for considerable portions of the environment, the laser range of the robot to be 2 meters and the robot not being maintaining any partial map of the environment, we believe that our LearnerRRT algorithm illustrates significant amount of efficiency in enabling the robot to achieve its task in a time effective manner. Both the maximum planning times and total times are recorded while the robot tries to solve test case 4 in Environment 4. In general, it can be observed that LearnerRRT takes relatively more time to solve test cases in Environment 4 which we believe is the most complicated among the 4 of our environments. This is because it involves obstacles with varying scales and irregular alignments as well as multiple cave like patterns which is the most complicated pattern encoded in our library. It is important to note that the approximate planning time and navigation times taken by Informed RRT* to solve the same test cases is approximately 31.60 minutes and 32.71 minutes which is much higher than the corresponding time taken by our LearnerRRT algorithm. These times also turn out to be the

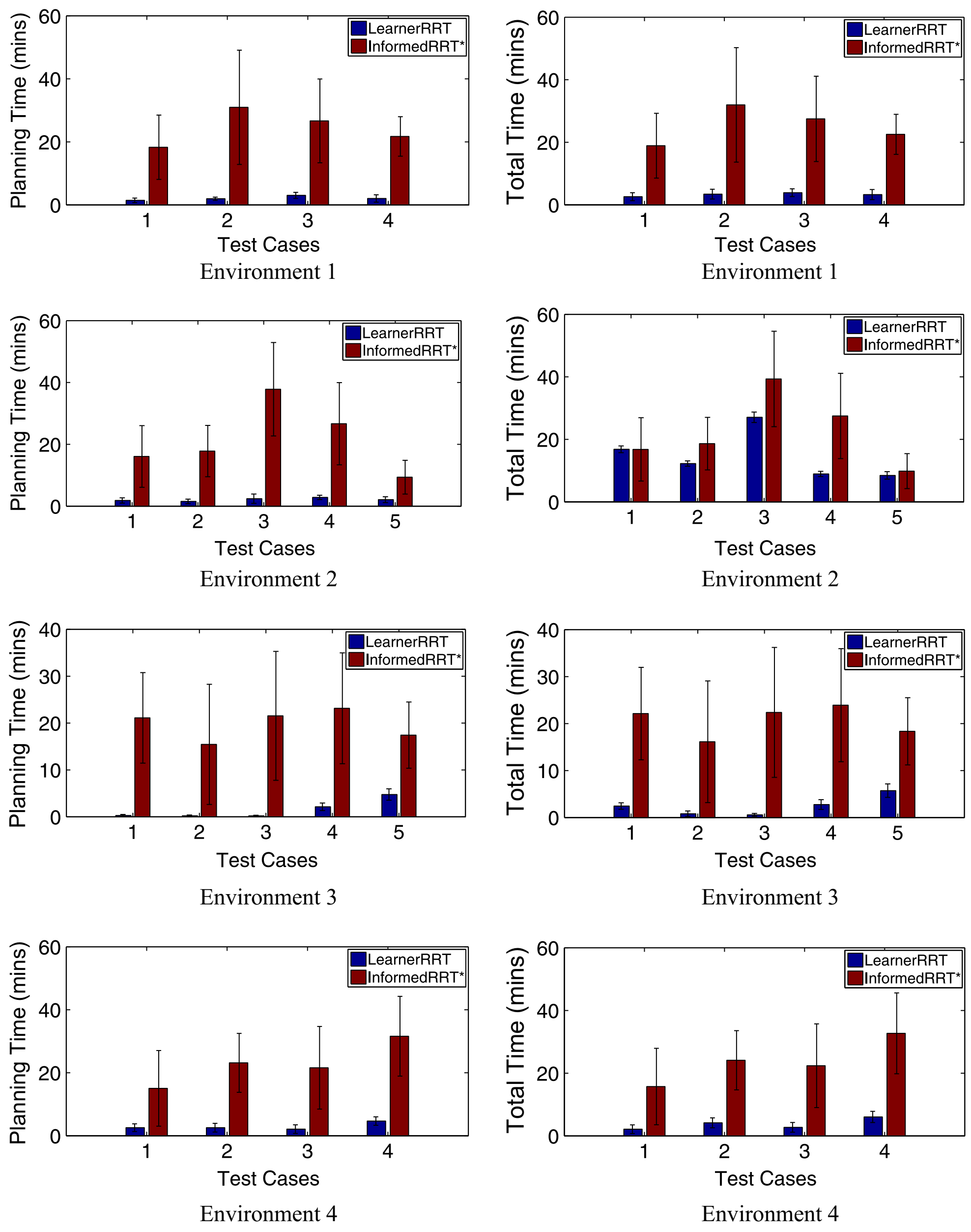

Fig. 7 Planning time and total time for the test cases for environments 1 – 4

maximum times taken by Informed RRT* to solve the test cases.

We would like to mention that the lesser improvement for LearnerRRT in terms of total time taken is because our algorithm cannot always reach the exact optimal path which is achieved by Informed RRT*. Figure 8 illustrates the comparative total distances traveled by the robot to solve the test cases by following both LearnerRRT and Informed RRT*. When compared to the mean of the distances traveled for all the test cases across all environments, LearnerRRT was found to travel an approximate distances of 38 meters and Informed RRT* travels for an approximate distance of 31.65 meters. Hence, LearnerRRT on average covers about 20 percent more distance in comparison to Informed RRT*. From our results, it can also be observed that for the majority of the test cases (16 out of 18), the differences of mean distance between LearnerRRT and Informed RRT* lies in the approximate range of 8 - 10 meters. Considering the significant improvements in terms of planning time and total time, it can be claimed that this difference in total distance traveled is trivial. In effect we can say that the robot is expected to be more energy-efficient if it adopts LearnerRRT as it will lead to lower battery consumption by the robot.

In order to measure the learning performance of our LearnerRRT algorithm, we have allowed the robot to solve a navigation task in a carefully constructed test environment which involves all the patterns included in our library. We have created the test case for assessing the learning

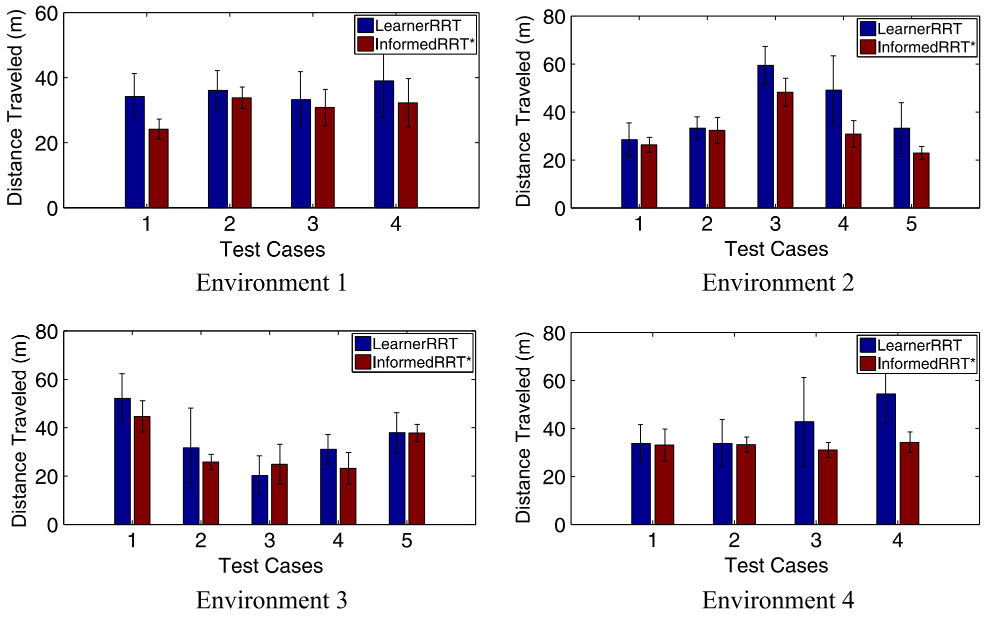

Fig. 8 Distances traveled for the test cases for environments 1 – 4

performance in such a manner that the straight line connecting the start and goal positions intersects through all the obstacles in the environment. This essentially means that to reach the goal, the robot has to avoid each of the obstacles in its direct path. Figure 9a illustrates the environment used for assessing the learning performance of LearnerRRT. Although there is no explicit block like pattern present in this environment, the passage-like pattern serves as both block as well as passage depending on the robot's relative position to the specific obstacle. For solving the test case for learning, the robot starts from near the leftmost corner of the environment and needs to reach its goal which is located diametrically opposite to its starting location near the rightmost corner of the environment. We allowed the robot to solve the same task using Informed RRT* multiple times and adopted the path from the run which attained the minimum total distance as the baseline for measuring the learning performance. We also allowed the robot to solve the test case by incrementally adding patterns to its library and increasing the library size from no pattern to 4 patterns. For each of the library sizes, we have evaluated the discrete fréchet distance FD [1] between the path generated and adopted by LearnerRRT and the baseline path generated by Informed RRT*. The Discrete fréchet distance is a standard metric to compare the distance between polygonal curves. In relation to measuring the learning performance, we have also measured the success ratio SR of LearnerRRT with respect to this test case. The success ratio expresses the number of times a known pattern is matched from the library as a ratio of the total number of obstacles and is given by Eq. 9.

$$SR = \frac{No.\ of\ matched\ obstacle\ patterns}{Total\ no.\ of\ encountered\ obstacles} \tag{9}$$

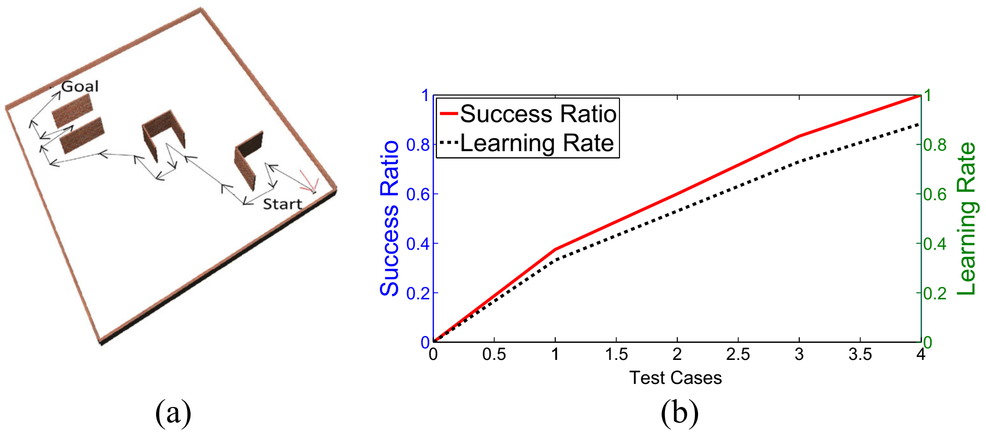

Fig. 9 **a** Environment used for testing learning performance of LearnerRRT algorithm. (0, 0) and (22, 22) correspond to the bottom right and top left corners of each environment. The black line illustrates the path followed by the robot and the start and goal denotes the start and goal locations given to the robot. **b** Success ratio and learning rate varying with library size

We measure the learning rate LR by following Eq. 10.

$$LR = \left(1 - \frac{FD}{Dis_{Best}}\right) SR \tag{10}$$

where Dis_{Best} is the total distance of the path generated by Informed RRT*.
We have interpreted and adapted the idea of learning rate to be suitable in our navigation domain. Learning rate in our case gives a comparative measure of the quality of the path constructed by LearnerRRT in comparison to the perfect path generated by InformedRRT* in relation to success ratio. In other words, learning rate gives the normalized error between the path adopted by LearnerRRT and InformedRRT* as the evaluated Frechet distance. From Fig. 9b, it can be seen that the value of success ratio increases as the size of the library increases and eventually reaches a value of 1. As the environment consists of our considered patterns only, the value eventually reaches 1 when all 4 patterns are propagated to the library. From the graph it can be observed that similar to the success ratio, the learning rate also increases with the increasing size of the library and the ultimate learning rate achieved by LearnerRRT corresponding to the maximum size of the library is approximately 88 percent. This graph thus illustrates that the learning performance of LearnerRRT increases with increased size of its library as can be expected in any learning based algorithm. It can be said that with the limited variation and restricted size of the library, LearnerRRT achieves a reasonably good learning performance.

5.2 Hardware Results

We have performed hardware experiments with a real Coroware Corobot robot. The Corobot is a 4-wheel 30cm × 30cm, skid-steer robot which is controlled in a

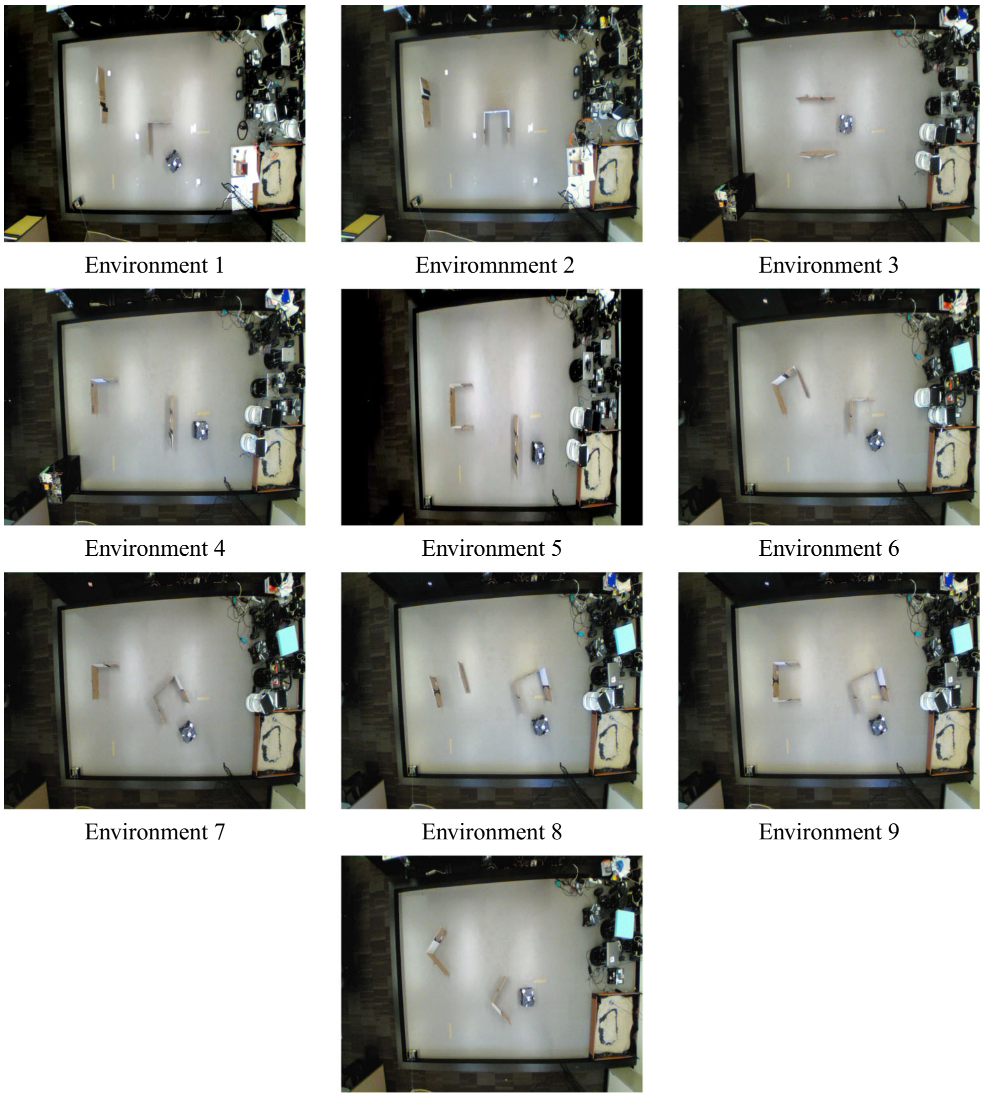

Fig. 10 Test environments used for hardware experiments

differential wheel manner. It contains an indoor Hagisonic Stargazer localization system which provides 2D location and heading, a Hokuyo laser distance sensor which provides a 270° range up to 5m in distance at a resolution of $\frac{1}{3}^{\circ}$, and wireless communications. The on-board computer has an 1.6GHz Intel Atom 330 processor with 4GB of RAM. A photograph of the Corobot robot is shown in Fig. 2a.

The hardware experiments were performed in 10 different test environments which are illustrated in Fig. 10. Figure 10 also shows the initial and the goal positions of the robot for each of the tests. We have varied our test environments by using combinations of the basic obstacles in different orientations and distributions in the hardware environments. The hardware experiments were performed in an arena with dimensions of 3 × 5 sq. meters. Similar to the simulation tests, we have compared the performance of LearnerRRT with Informed RRT* with respect to the three previously discussed metrics- planning time, total time and total distance covered to finish the navigation tests. In order to handle the randomness in both the algorithms, we have performed each of the tests for both the algorithms over 5 runs making a total of 50 runs for each of the algorithms. Figure 11 demonstrates the comparative results of our algorithm. For the purpose of clearer illustration, we have divided the environments into two sets each consisting of 5 environments. The hardware results are similar to our simulation results. It can be observed from the results that LearnerRRT takes much lesser planning time and total time compared to Informed RRT* in majority of the tests. When we compare the performance according to our third metric-total distance covered, LearnerRRT is found to cover more distance compared to Informed RRT* in 50% of the tests. However, compared to significant time benefits that LearnerRRT offers for real-time navigation, the excess distance covered by it is nominal. From our comparative analysis of the hardware performance we found out that LearnerRRT on average takes only 33% of the planning time, 46% of the total time and 95% of the total distance compared to Informed RRT*. Hence, hardware results also follow the trend set by the simulation results. The comparative lower time benefits of LearnerRRT over Informed RRT* is a direct consequence of a much smaller environment size (3 × 5

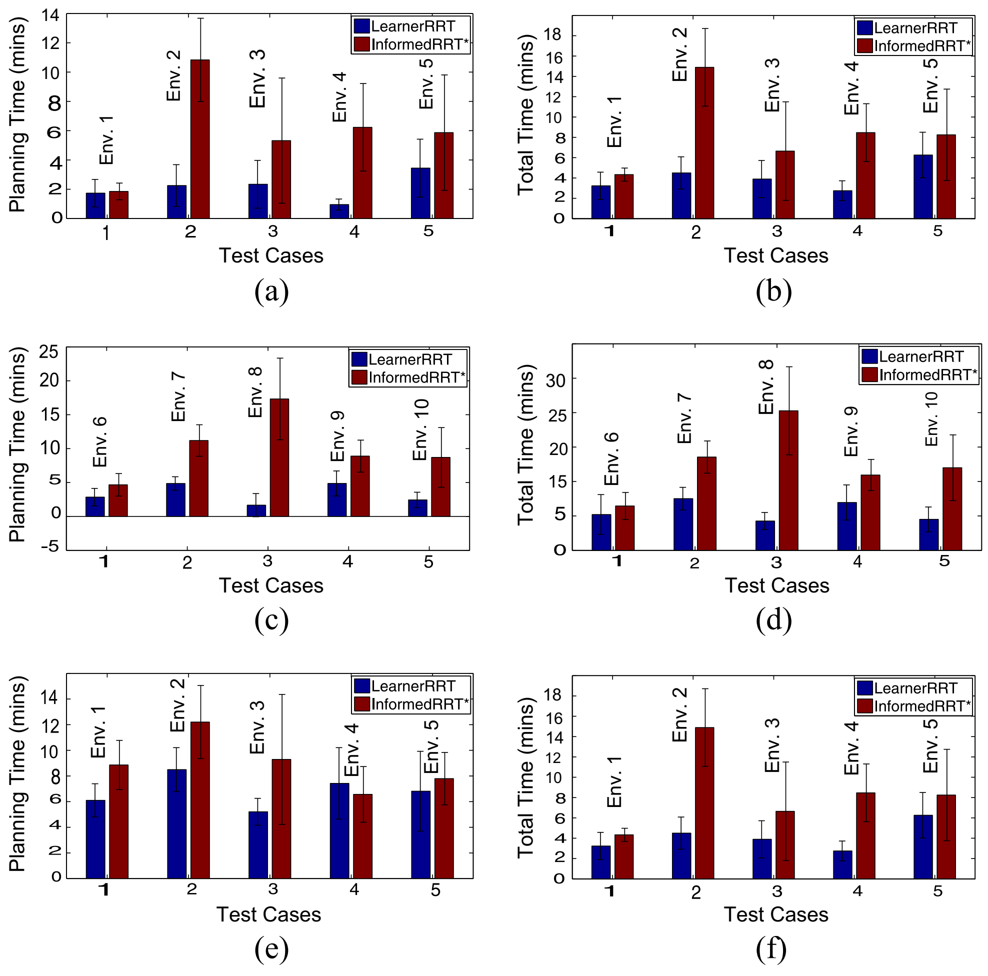

Fig. 11 Planning time and total time taken **a**-**b** for test cases in environments 1 − 5, **c**-**d** for test cases in environments 6 − 10. Total distances traveled **e** for test cases in environments 1 − 5 **f** for test cases in environments 5 − 10

sq.meters) in comparison to the dimensions of the simulation environments (22 × 22 sq.meters), lower number of obstacles(maximum of 2 obstacles) and lower separation distances between the start and the goal(approximately 4 meters). These factors helped sampling based planner Informed RRT* to find paths to the goal in a relatively shorter amount of time. The real advantages of LearnerRRT will be evident when the robot is allowed to autonomously perform navigation tasks in large environments densely populated with complicated obstacles. However, we want to emphasize that even with these favourable factors on part of Informed RRT*, our algorithm offered significant competitive advantage over Informed RRT*.

6 Discussion

From the extensive experimental results above both from the simulations as well as hardware, it can be observed that LearnerRRT performs better than Informed RRT* in terms of planning time and total time. We have selected Informed RRT* as our local planner during the training phase and also as the comparative algorithm instead of any other RRT variants because InformedRRT* is proved to produce more optimized paths compared to RRT* in lesser amount of time. Our proposed algorithm LearnerRRT tries to achieve the virtues of Informed RRT* by using it as the base planner during training. However, it improves significantly on the time requirements by accessing previously recorded information corresponding to the learned basic obstacle patterns. The long planning times required by Informed RRT* especially in the simulation results are the direct consequences of having a large environment of 22 × 22 sq.meters and having several complex obstacles directly aligned along the robot's path to the goal. InformedRRT* takes much lesser time for the hardware experiments where we have used an environment having dimensions of 3 × 5 sq.meters. For both the simulation and the hardware results an important point to note is that unlike most of the conventionally used path planners which are provided with at least an initial rough map of the environment, here the comparison algorithm InformedRRT* has not been provided with any map of the environment. Thus, InformedRRT* can only create an initial plan based on just a small part of the environment that it perceives and has to replan every time the robot's sensors captures other obstacles while navigating to other parts of the environment. It is also important to mention in this regard that LearnerRRT at present works with only four basic patterns. However, it can be extended to include more number of patterns if the other associated scalability factors can be efficiently handled. Also presently our algorithm relies on a manual labeling of the obstacle patterns during the training phase. However, this algorithm can be extended to autonomously determine new patterns and extend the library when and where necessary.

7 Conclusions and Future Work

This is our first step to solve the problem of real-time navigation with the help of experiences coded in the form of paths across most commonly observed sample patterns like caves, corners, passages and blocks. To the best of our knowledge this work is the first which uses learning experiences from past navigation as sample paths to solve path planning locally. One of the insights that is dominant behind this work is that it is not required to generate an entire path from the start to the goal for the robot to successfully reach its goal. In our work we illustrate that the task of navigation can be effectively solved even if the robot is provided with a local plan which helps it to avoid an obstacle in its path and when possible follow the direct path to the goal.

There are many possible future directions of this work which we plan to explore in the future. We plan to generalize our algorithm to varied obstacle geometry patterns which can be used to expand our library and enable the robot to explore random environments with previously unseen obstacle patterns including those formed by combining rudimentary obstacle patterns. Of course with greater number of obstacle patterns in the robot's library there can be scalability issues which need to be addressed as well. In order to overcome the expected scalability challenges posed by increased number of obstacle patterns, we plan to use real-time advanced variants of clustering techniques that will result in faster obstacle recognition during the pattern matching phase of our algorithm.

Another future direction that we plan to explore is the autonomous expansion of the robot's library as the robot encounters previously unseen obstacle patterns anytime during its navigation after the training phase. In order to address this objective, there should be advanced decision making on the part of the robot to determine if a particular obstacle pattern should be included in the library. We aim to investigate techniques that will allow the robot to autonomously determine a feasible path for the novel obstacle pattern by combining paths from different known patterns stored in the robot's library.

In future we also plan to work in partially observable obstacle settings where due to the nature of the obstacle, the robot cannot perceive the entire obstacle boundary. We envision solving this problem by the integration of this experiential learning technique with a formal learning technique like reinforcement learning to elicit intra domain learning in addition to inter domain learning. In this work our robots perceived obstacle through the laser and infra-red sensors which generate clean obstacle boundaries. In future we also plan to investigate the applicability of our proposed approach on relatively irregular data that emerges from other types of sensors like the kinect sensors.

References

1. Alt, H., Buchin, M.: Can we compute the similarity between surfaces? Discret Comput Geom **43**(1), 78–99 (2010)
2. Amato, N.M., Bayazit, O.B., Dale, L.K.: Obprm: An obstacle-based prm for 3d workspaces. In: 3rd International Workshop on Algorithmic Foundations of Robotics (WAFR), pp. 155–168 (1998)
3. Arslan, O., Tsiotras, P.: Use of relaxation methods in sampling-based algorithms for optimal motion planning. In: IEEE International Conference on Robotics and Automation (ICRA), p. 2013. IEEE (2013)
4. Berenson, D., Abbeel, P.: A robot path planning framework that learns from experience. In: IEEE International Conference on Robotics and Automation (ICRA) A robot path planning, p. 2012. IEEE (2012)
5. Bohlin, R., Kavraki, L.E.: Path planning using lazy prm. In: 2000. Proceedings. ICRA'00. IEEE International Conference on Robotics and Automation, vol. 1, pp. 521–528. IEEE (2000)
6. Bruce, J., Veloso, M.: Real-time randomized path planning for robot navigation, vol. 3, pp. 2383–2388. IEEE (2002)
7. Choset, H., Burgard, W., Hutchinson, S., Kantor, G., Kavraki, L.E., Lynch, K., Thrun, S.: Principles of robot motion: theory, algorithms, and implementation. MIT Press, Cambridge (2005)
8. Coleman, D., Sucan, I.A., Moll, M., Okada, K., Correll, N.: Experience-based planning with sparse roadmap spanners. arXiv:1410.1950 (2014)
9. Cui, R., Gao, B., Guo, J.: Pareto-optimal coordination of multiple robots with safety guarantees. Auton. Robots. **32**(3), 189–205 (2012)
10. Cui, R., Li, Y., Yan, W.: Mutual information-based multi-auv path planning for scalar field sampling using multidimensional RRT. IEEE Trans. Syst. Man Cybern. Syst. **46**(7), 993–1004 (2016)
11. Da Silva, B.N., Mackworth, A.: Using spatial hints to improve policy reuse in a reinforcement learning agent. In: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1, pp. 317–324. International Foundation for Autonomous Agents and Multiagent Systems (2010)
12. D'Andrea, R.: Guest editorial: A revolution in the warehouse: A retrospective on kiva systems and the grand challenges ahead. IEEE Trans. Autom. Sci. Eng. **9**(4), 638–639 (Oct 2012)
13. Elbanhawi, M., Simic, M.: Sampling-based robot motion planning: A review. IEEE Access **2**, 56–77 (2014)
14. Elgamal, T., Hefeeda, M.: Analysis of pca algorithms in distributed environments. arXiv:1503.05214 (2015)
15. Ferguson, D., Kalra, N., Stentz, A.: Replanning with rrts. In: Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006, pp. 1243–1248. IEEE (2006)
16. Fernández, F., Veloso, M.: Probabilistic policy reuse in a reinforcement learning agent. In: Proceedings of the fifth international joint conference on Autonomous Agents and Multiagent Systems, pp. 720–727. ACM (2006)
17. Gammell, J., Srinivasa, S., Barfoot, T.: Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In: 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2014), pp. 2997–3004 (2014)
18. Gayle, R., Klingler, K.R., Xavier, P.G.: Lazy reconfiguration forest (lrf)-an approach for motion planning with multiple tasks in dynamic environments. In: Proceedings 2007 IEEE International Conference on Robotics and Automation, pp. 1316–1323. IEEE (2007)
19. Hwang, V., Phillips, M., Srinivasa, S., Likhachev, M.: Lazy validation of experience graphs. In: IEEE International Conference on Robotics and Automation (ICRA), pp. 912–919. IEEE (2015)
20. Karaman, S., Frazzoli, E.: Sampling-based algorithms for optimal motion planning. Int. J. Robot. Res. **30**(7), 846–894 (2011)
21. Kavraki, L.E., Latombe, J.-C.: Probabilistic roadmaps for robot path planning (1998)
22. Kuffner, J.J., LaValle, S.M.: Rrt-connect: An efficient approach to single-query path planning. In: 2000. Proceedings. ICRA'00. IEEE International Conference on Robotics and Automation, vol. 2, pp. 995–1001. IEEE (2000)
23. LaValle, S.M.: Rapidly-exploring random trees: A new tool for path planning (1998)
24. Lien, J.-M., Lu, Y.: Planning motion in environments with similar obstacles. In: Robotics: Science and Systems. Citeseer (2009)
25. Menon, A., Cohen, B., Likhachev, M.: Motion planning for smooth pickup of moving objects. In: Robotics and Automation (ICRA) IEEE International Conference on, pp. 453–460. IEEE (2014)
26. Otte, M., Frazzoli, E.: Rrt^{x}: Real-time motion planning/replanning for environments with unpredictable obstacles. In: Algorithmic Foundations of Robotics XI, pp. 461–478. Springer (2015)
27. Paden, B., Čáp, M., Yong, S.Z., Yershov, D., Frazzoli, E.: A survey of motion planning and control techniques for self-driving urban vehicles. IEEE Trans. Intell. Veh. **1**(1), 33–55 (2016)
28. Phillips, M., Cohen, B.J., Chitta, S., Likhachev, M.: E-graphs: Bootstrapping planning with experience graphs. In: Robotics: Science and Systems (2012)
29. Phillips, M., Likhachev, M.: Speeding up heuristic computation in planning with experience graphs. In: IEEE International Conference on Robotics and Automation (ICRA), p. 2015. IEEE (2015)
30. Rusu, R., Blodow, N., Beetz, M.: Fast point feature histograms (fpfh) for 3d registration. In: 2009. ICRA '09. IEEE International Conference on Robotics and Automation, pp. 3212–3217 (2009)
31. Rusu, R.B., Bradski, G.R., Thibaux, R., Hsu, J.M.: Fast 3d recognition and pose using the viewpoint feature histogram. In: 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2155–2162, Taipei (2010)
32. Saha, O., Dasgupta, P.: Fast path planning using experience learning from obstacle patterns. AAAI Spring Symposium Series. https://www.aaai.org/ocs/index.php/SSS/SSS16/paper/view/12725 (2016)
33. Saha, O., Dasgupta, P.: Real-time robot path planning using experience learning from common obstacle patterns. In: Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems, pp. 1339–1340. International Foundation for Autonomous Agents and Multiagent Systems (2016)
34. Sherwood, R., Mishkin, A., Chien, S., Estlin, T., Backes, P., Cooper, B., Rabideau, G., Engelhardt, B.: An integrated planning and scheduling prototype for automated mars rover command generation. In: Sixth European Conference on Planning (2014)
35. Shi, K., Denny, J., Amato, N.M.: Spark prm: Using rrts within prms to efficiently explore narrow passages. In: 2014 IEEE International Conference on Robotics and Automation (ICRA), pp. 4659–4666. IEEE (2014)
36. Song, B.D., Kim, J., Morrison, J.R.: Rolling horizon path planning of an autonomous system of uavs for persistent cooperative service: MILP formulation and efficient heuristics. J. Intell. Robot. Syst. **84**(1-4), 241–258 (2016)

37. Stolle, M., Atkeson, C.G.: Policies based on trajectory libraries. In: Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006, pp. 3344–3349. IEEE (2006)
38. Stolle, M., Tappeiner, H., Chestnutt, J., Atkeson, C.G.: Transfer of policies based on trajectory libraries. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2981–2986. IEEE (2007)
39. Strandberg, M.: Augmenting rrt-planners with local trees. In: 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on Robotics and Automation, vol. 4, pp. 3258–3262. IEEE (2004)
40. Talvitie, E., Singh, S.P.: An experts algorithm for transfer learning. In: IJCAI, pp. 1065–1070 (2007)
41. Taylor, M.E., Kuhlmann, G., Stone, P.: Accelerating search with transferred heuristics. In: ICAPS Workshop on AI Planning and Learning (2007)
42. Taylor, M.E., Kuhlmann, G., Stone, P.: Autonomous transfer for reinforcement learning. In: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 1, pp. 283–290. International Foundation for Autonomous Agents and Multiagent Systems (2008)
43. Taylor, M.E., Stone, P.: Transfer learning for reinforcement learning domains A survey. J. Mach. Learn. Res. **10**, 1633–1685 (2009)
44. Torrey, L., Shavlik, J.: Policy transfer via markov logic networks. In: Inductive Logic Programming, pp. 234–248. Springer (2010)
45. Torrey, L., Shavlik, J., Walker, T., Maclin, R.: Skill acquisition via transfer learning and advice taking. In: Machine Learning: ECML 2006, pp. 425–436. Springer (2006)
46. Tsardoulias, E., Iliakopoulou, A., Kargakos, A., Petrou, L.: A review of global path planning methods for occupancy grid maps regardless of obstacle density. J. Intell. Robot. Syst. **84**(1-4), 829–858 (2016)
47. Yeh, H.-Y., Thomas, S., Eppstein, D., Amato, N.M.: Uobprm: A uniformly distributed obstacle-based prm. In: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2655–2662. IEEE (2012)
48. Zucker, M., Kuffner, J., Branicky, M.: Multipartite rrts for rapid replanning in dynamic environments. In: Proceedings IEEE International Conference on Robotics and Automation, p. 2007. IEEE (2007)

Olimpiya Saha is a doctoral candidate in the Computer Science Department at the University of Nebraska at Omaha. Her research interest encompasses the applications of machine learning techniques for improved robot navigation and path planning in complicated, unknown environments. She has published her research in multiple conferences and workshops in robotics and machine learning. She received her undergraduate degree in Computer Science and Engineering from West Bengal University of Technology, India.

Prithviraj Dasgupta is the Union Pacific Endowed Professor in the Computer Science Department at the University of Nebraska, Omaha and the director of the CMANTIC Robotics Lab at his university. His research interests are in the area of multi-robot systems, artificial intelligence and game theory. He has authored over 140 journal and conference publications and led several large, federally-funded projects in his research area. He received his Ph.D. and M.S. from University of California, Santa Barbara.