Fast Path Planning Using Experience Learning from Obstacle Patterns

Olimpiya Saha and Prithviraj Dasgupta
Computer Science Department
University of Nebraska at Omaha
Omaha, NE 68182, USA
{*osaha,pdasgupta*}*@unomaha.edu*

Abstract

We consider the problem of robot path planning in an environment where the location and geometry of obstacles are initially unknown while reusing relevant knowledge about collision avoidance learned from robots' previous navigational experience. Our main hypothesis in this paper is that the path planning times for a robot can be reduced if it can refer to previous maneuvers it used to avoid collisions with obstacles during earlier missions, and adapt that information to avoid obstacles during its current navigation. To verify this hypothesis, we propose an algorithm called LearnerRRT that first uses a feature matching algorithm called Sample Consensus Initial Alignment (SAC-IA) to efficiently match currently encountered obstacle features with past obstacle features, and, then uses an experience based learning technique to adapt previously recorded robot obstacle avoidance trajectories corresponding to the matched feature, to the current scenario. The feature matching and machine learning techniques are integrated into the robot's path planner so that the robot can rapidly and seamlessly update its path to circumvent an obstacle it encounters, in real-time, and continue to move towards its goal. We have conducted several experiments using a simulated Coroware Corobot robot within the Webots simulator to verify the performance of our proposed algorithm, with different start and goal locations, and different obstacle geometries and placements, as well as compared our approach to a state-of-the-art sampling-based path planner. Our results show that the proposed algorithm LearnerRRT performs much better than Informed RRT*. When given the same time, our algorithm finished its task successfully whereas Informed RRT* could only achieve $10 - 20$ percent of the optimal distance.

Introduction

Autonomous navigation is one of the fundamental problems in robotics used in several real life applications such as unmanned search and rescue, autonomous exploration and surveillance, and domestic applications such as automated waste cleanup or vaccum cleaning. We consider the navigation problem for a robot in an unstructured environment where the location and geometry of obstacles are initially unknown or known only coarsely. To navigate in such an environment, the robot has to find a collision-free path in real-time, by determining and dynamically updating a set of waypoints that connect the robot's initial position to its goal position. While there are several state-of-the-art path planners available for robot path planning (Choset et al. 2005), these planners usually replan the path to the goal from scratch every time the robot encounters an obstacle that obstructs its path to the goal - an operation that can consume considerable time (order of minutes or even hours), if the environment is complex, with many obstacles. Excessively expended path planning time also reduces the robot's energy (battery) to perform its operations, and aggravates the overall performance of the robot's mission.

To address this problem, in this paper, we make the insight that, although obstacles could be geometrically dissimilar in nature, yet there exists some generic features that are common across most obstacles. If a robot can be trained to navigate around obstacles with some basic geometric patterns, it can adapt and synthesize these movements to navigate around more complex obstacles without having to learn the motion around those obstacles from scratch. In this manner, navigation can be learned incrementally by a robot, without having to be trained independently for each new environment it is placed in. To realize our proposed approach we describe an algorithm called LearnerRRT, where, we first train a robot by navigating it in an environment where obstacle shapes have some common geometric patterns. The features of the obstacles perceived by the robot's sensors and its movements or actions to avoid the obstacles are stored in summarized format within a repository maintained inside the robot. Later on, when the robot is presented with a navigation task in a new environment, which requires it to maneuver around obstacles, it retrieves the obstacle feature-action pair, where the retrieved obstacle's features have the highest resemblance to its currently perceived obstacle features. The robot then mimics the actions retrieved from its previous maneuver after adapting them for the current obstacle. We have tested our algorithm on a simulated Corobot robot using the Webots simulator 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*. When given the same time, our algorithm was able to reach the goal in the different environments discussed whereas Informed RRT* had covered approximately about $10 - 20$ percent of the optimal distance between the start and goal locations.

Related Work

Applications of machine learning to robot navigation 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 (Fernández and Veloso 2006) 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 (Da Silva and Mackworth 2010) 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 (Taylor and Stone 2009) 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 owing to irrelevant or 'negative' transfer of knowledge from the source task. In (Taylor, Kuhlmann, and Stone 2007), the authors have addressed this problem using a transfer hierarchy. Approaches such as a human-provided mapping (Torrey et al. 2006) and a statistical relational model (Torrey and Shavlik 2010) 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 (Taylor, Kuhlmann, and Stone 2008) and an experts algorithm which is used to select a candidate policy for solving an unknown Markov Decision Process task (Talvitie and Singh 2007). 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) (Rusu, Blodow, and Beetz 2009) from computer vision to mitigate the problem of negative transfer.

In contrast to using machine learning techniques to learn suitable actions, researchers have also proposed techniques to reuse robots' paths, represented as a set of waypoints, learned from previous navigation experiences. Lien et al. (Lien and Lu 2009) 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 (Phillips et al. 2012), (Phillips and Likhachev 2015), (Hwang et al. 2015) 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 the Lightning and Thunder frameworks (Berenson, Abbeel, and Goldberg 2012; Coleman et al. 2014), a robot learns paths from past experiences in a high-dimensional space and reuses them effectively in the future to solve a given navigation 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* (Gammell, Srinivasa, and Barfoot 2014) to navigate in the target environment. However, in contrast to them, our approach considers a higher level of granularity by learning and reusing actions at 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 size and locations.

Problem Formulation

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 $a_i(q) = q' \in Q_{obs}$. Our proposed LearnerRRT algorithm proceeds in two steps that are described below.

Library Creation

The robot is first trained 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

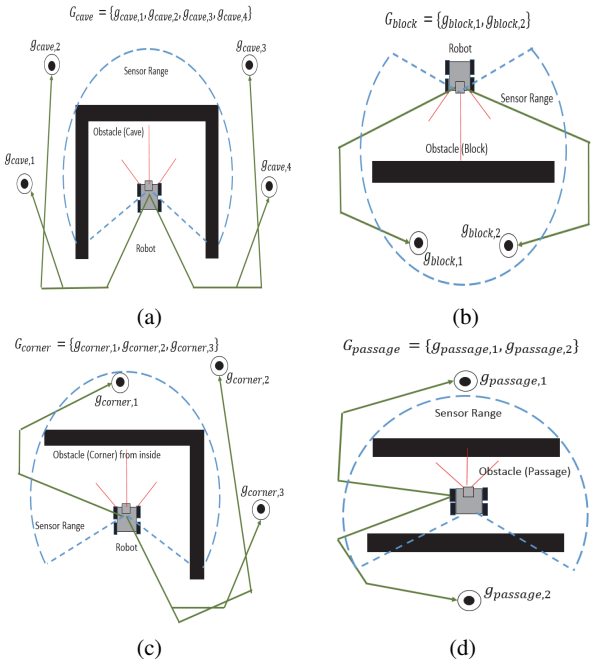


Figure 1: (a) -(d) Different obstacle patterns used as source task to construct the action library. The robot's initial position corresponds to a location from which its sensor can perceive the entire obstacle's inner boundary. Different goal locations for each obstacle are denoted by the set G_{cave},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

with respect to the source tasks, although the orientation and dimensions of individual obstacles in the later environments might vary. We consider four well-defined obstacle geometry patterns as source tasks - cave, column or passage, corner and block, as shown in Figures 1(a) - (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.

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 entire 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 Figures 1 (a) - (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 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 (Gammell, Srinivasa, and Barfoot 2014) 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:

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

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

Obstacle Avoidance Using Learned Navigation Actions

After learning actions for avoiding common obstacle patterns, the robot 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 pseudocode for our proposed LearnerRRT algorithm summarizing the steps discussed above is given in Figure 2. The robot first optimistically assumes that there are no obstacles in the path between its start and goal locations and starts to follow the straight line path connecting these two locations(lines 2-6). While moving towards the goal, when the robot encounters an obstacle obs it first records the proximity data from the obstacle, LS_{obs}. It then uses a state-of-the-art algorithm for aligning object features called Sample Consensus Initial Alignment(SAC-IA) algorithm (Rusu, Blodow, and Beetz 2009), 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 pre-processed 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 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 is given by the ratio of the square roots of the two eigenvalues. 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 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[1]:

[1]If LS_{obs} does not match any LS_{lab} from the action library,

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

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 $\nexists a = (\hat{d}, \hat{\theta}) \in path_{lab,j}$ 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. If a segment of the selected path gets within very close proximity of the obstacle or intersects with the obstacle, then the path segment is extrapolated using the extremity points of the segment, until it does not collide with the obstacle any more. On the other hand, 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.

The robot follows the mapped path retrieved from the action library until it either perceives no obstacle in the direction of q_{goal} or 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.

Experimental Setup and Results

We have verified the performance of the LearnerRRT algorithm using simulated 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 $\times$ 30 cm. For detecting obstacles, the robot is equipped with four infra-red(IR)-based distance sensors, one on each side, oriented at 60° from the front of the vehicle, and two cross beam IR sensors mounted on the front bumper. The robot also has a Hokuyo laser sensor with a 360° field of view and a range of 2 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 photograph of the Corobot robot is shown in Figure 3(a) and the simulated robot within Webots is shown in Figure 3(b).

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 dis-

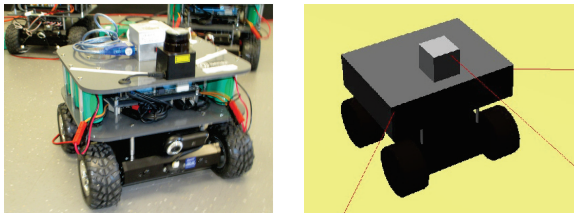
the robot uses the Informed RRT* planner to plan its path around obstacle *obs* instead of attempting to learn from past navigation data.

Algorithm 0.1: LEARNERRRT($currX, currY, goalX, goalY$)

comment: This algorithm uses SAC-IA along with post processing steps to predict a feasible path for the robot in order to reach the goal. Inputs are current location, goal coordinates and action library L

$path \leftarrow \emptyset$
repeat
 $adjustToGoal(goalX, goalY)$
 repeat
  Switch to $MoveToGoal$ and continue
 until obstacle is detected
 Switch to $AvoidObstacle$
 $ls \leftarrow getLaserScan(currX, currY)$
 $ls^{scaled} \leftarrow scaleLaserScan(ls)$
 $pattern, \theta_{match} \leftarrow findBestMatch(ls^{scaled}, L)$
 $path^{best} \leftarrow findBestPath(Lib, pattern)$
 $path^{transf} \leftarrow transformPath(path^{best}, \theta_{match})$
 for each $node \in path^{transf}$
  do if node collides with obstacle
  then $path^{mapped} \leftarrow correctPath(path^{best})$
 for each $edge \in path^{transf}$
 if edge collides with obstacle
  then $path^{mapped} \leftarrow correctPath(path^{transf})$
 if no node collided with obstacle and no edge collided with obstacle
  then $path^{mapped} \leftarrow optimizePath(path^{transf})$
 repeat
  if sensor detects obstacle
   then switch to$AvoidObstacleMode$
  and repeat the rest of the steps
   else $followPath(path^{mapped})$
 until Line of Sight LOS is achievedor $path$ gets exhausted
until $(goalX, goalY)$ is reached

Figure 2: Pseudo-code for the LearnerRRT algorithm.



(a) (b)

Figure 3: (a) Photograph of a Corobot robot, (b) simulated Corobot robot with visible distance sensor rays used for simulations within Webots.

Environment 1		
Test Case	Start	Goal
1	(8.61,11.26)	(19.62,9.21)
2	(2.84,17.82)	(16.81,17.01)
3	(2.62,18.34)	(19.62,9.21)
4	(0.61,3.91)	(16.81,17.01)
5	(16.40,4.18)	(16.81,17.01)
6	(0.47,10.53)	(16.81,17.01)

Environment 2		
Test Case	Start	Goal
1	(0.72,18.37)	(15.93,16.75)
2	(1,11.47)	(16.81,17.01)
3	(0.63,4.99)	(20.34,4.77)
4	(0.60,18.68)	(20.34,4.77)
5	(17.42,4.87)	(11.26,17.87)
6	(0.54,18.27)	(11.37,4.89)

Environment 3		
Test Case	Start	Goal
1	(0.63,0.72)	(20.80,19.88)
2	(1.9,13.9)	(18.27,8.16)
3	(17.97,4.87)	(10.22,12.08)
4	(1.9,13.9)	(10.22,12.08)
5	(17.20,1.68)	(20.80,19.88)
6	(0.86,3.99)	(20.86,4.51)

Environment 4		
Test Case	Start	Goal
1	(20.80,19.88)	(2.5,,4.5)
2	(0.61,3.91)	(19.62,9.21)
3	(17.20,10.40)	(2.5,4.5)
4	(17.05,1.98)	(15.93,16.75)
5	(1.9,15.9)	(19.62,9.21)
6	(0.61,3.91)	(15.93,16.75)

Table 1: Coordinates of the different start and goal locations used as test cases for the four environments shown in Figure 4.

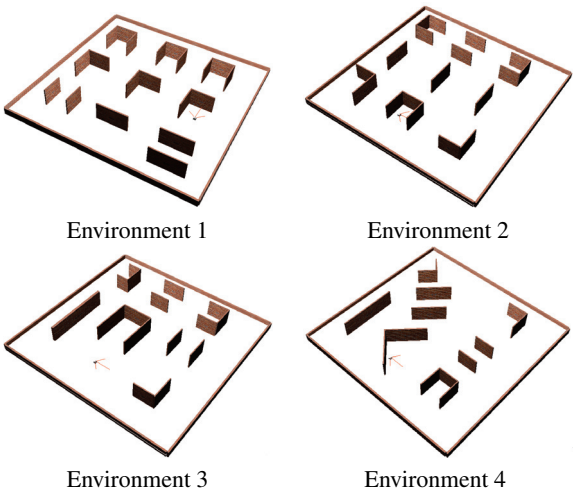


Figure 4: Environments used for testing proposed LearnerRRT algorithm. $(0, 0)$ and $(22, 22)$ correspond respectively to the leftmost and rightmost corners of each environment.

tributions 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 4(a)-(d) above illustrate different test environments used in our experiments; $(0, 0)$ and $(22, 22)$ correspond to the leftmost and rightmost corners of each environment. Motion planning algorithms that plan from scratch(PFS) 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 1. The separation distance between the start and goal locations ranges between $10 - 15$ 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 mainly used three main measures- the planning time to predict the path to be undertaken by the robot, the navigation time which essentially gives 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, while the threshold for matching using the Jaccard Index, JI_{Thr} was set to 0.3.

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 Figure 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 pattern. Figure 5 shows 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 24 test cases executed across 240 runs and uniformly distributed over the four test environments shown in Figure 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 significant 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.

Figure 6 shows the average planning, navigation and total times taken by the LearnerRRT algorithm. It can be observed

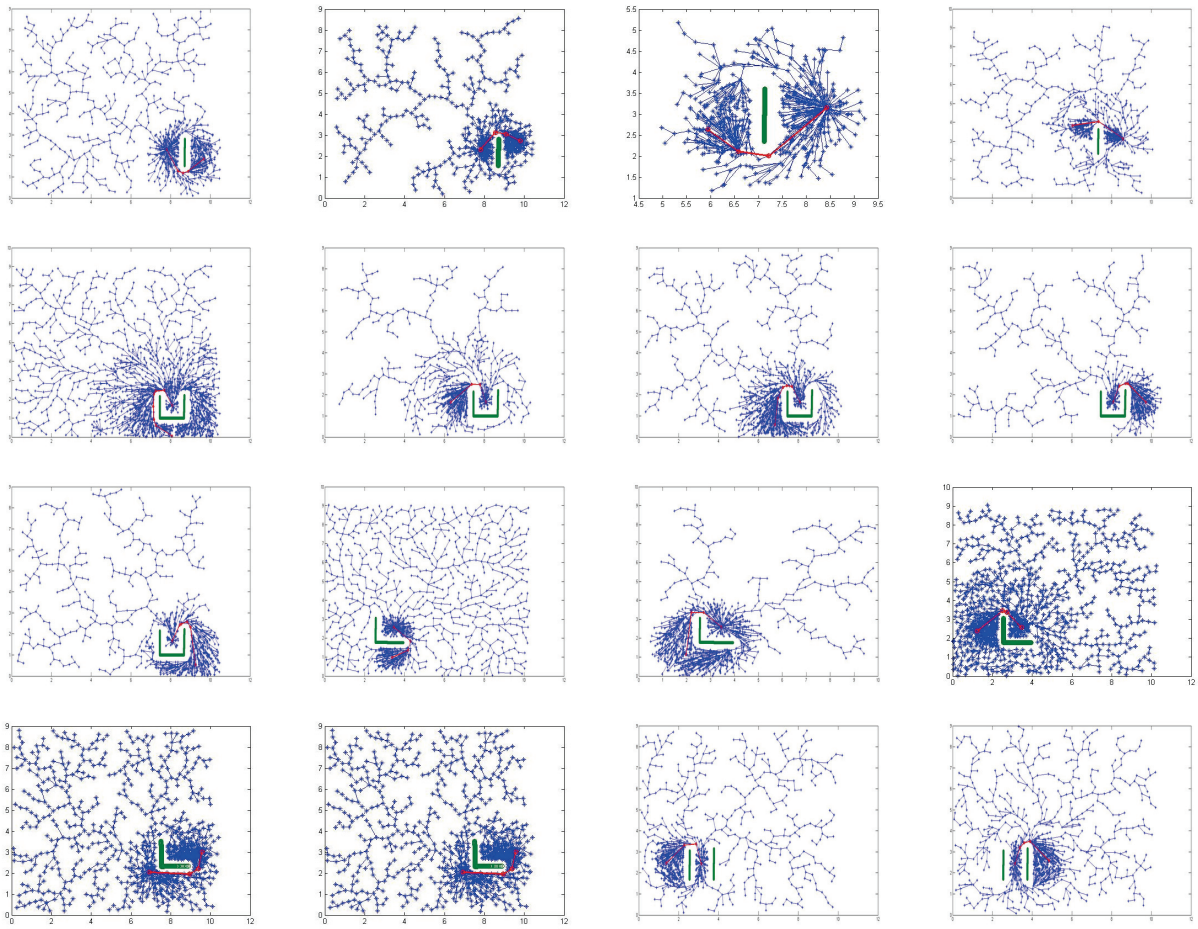
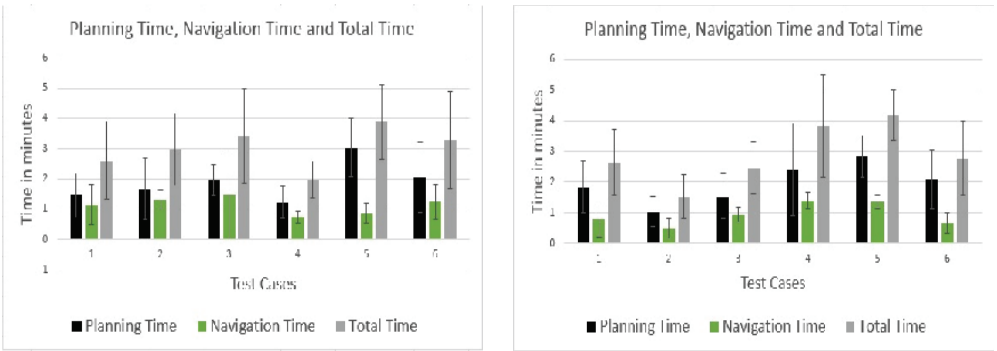


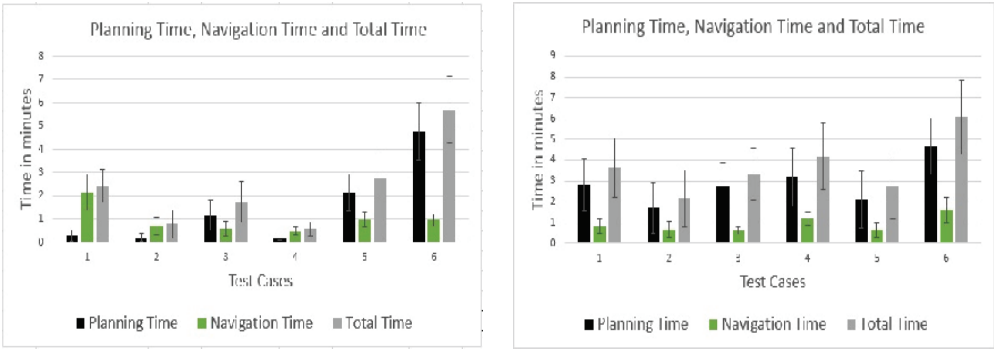
Figure 5: Paths obtained for different source task obstacle patterns using Informed RRT*. The green contour represents the obstacle. The blue starred edges represents the paths generated by Informed RRT* and the red edges represents the final paths obtained for the specified goal around the obstacles.

that for majority of the test cases (22 out of 24) the mean total time taken by the robot to reach its goal by following our LearnerRRT algorithm is approximately about 3 minutes or less. 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. Considering the approximate separation distance between the start and goal locations to be $10 - 15$ 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 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. The mean planning time taken for 22 out of 24 test cases is approximately 2 minutes and the mean navigation time taken is approximately 1 minute. It is worth mentioning here that during planning by LearnerRRT, the robot halts at its last position which means that during planning time, the robot's battery life is not affected. For test case 6 for environment 3, the robot takes a mean total time of approximately 5.72 minutes to reach the goal which consists of a mean planning time of approximately 4.75 minutes and mean navigation time of approximately 1 minute. Similarly test case 6 of environment 4 has planning time, navigation time and total time of respectively 4.65 minutes and 1.34 minutes. We believe that the higher planning time taken by the robot in these cases is due to the presence of multiple cave like patterns in its path which is the most complicated among all four of the obstacle patterns existing in our library.

Figure 7 shows the corresponding distances traveled by the robot for each of the test cases. It can be observed that the mean total distances traveled by the robot is approximately 36 meters. It can be said that although the direct separation distances between the start and goal locations is set between 10 to 15 meters, the presence of significant percentage of obstacles in the paths of each of the test cases, makes the robot navigate a larger amount of distances to circumvent the obstacles in its path.

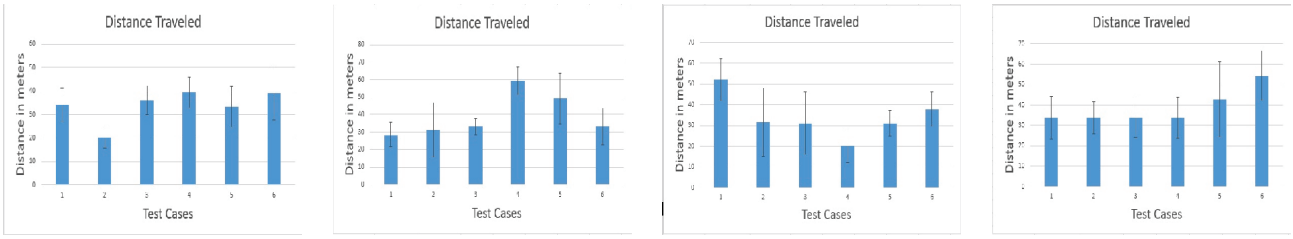


Environment 1 Environment 2



Environment 3 Environment 4

Figure 6: Planning time, navigation time and total time taken for the test cases for environments 1,2,3 and 4.



Environment 1 Environment 2 Environment 3 Environment 4

Figure 7: Total distance traveled by the robot for solving the test cases for environments 1,2,3 and .

In order to compare the performance of our algorithm LearnerRRT with Informed RRT* we have selected a representative test case from each of our test environments. We have provided Informed RRT* the start and goal locations for these test cases and allowed it to run till the mean total time taken by our LearnerRRT algorithm to solve the test cases was reached. When this time was reached Informed RRT* algorithm was terminated and the location reached by the robot was recorded. Then, we compared the percentage of distance that the robot covered upto the recorded location versus the optimal distance to reach the goal. Figure 8 illustrates our results. From the results it can be observed that the robot using Informed RRT* still has approximately $80-90$ percent of the optimal distance left to cover for each test case. We would like to reiterate that in the same time that Informed RRT* has been allowed to perform, our algorithm, LearnerRRT, was able to reach the goal. The relative poor performance of Informed RRT* is due to the fact that it replans from scratch every time the robot encounters an obstacle. This, in effect, emphasizes the efficiency of the experience based learning in our LearnerRRT algorithm with respect to planning from scratch algorithms like Informed RRT*.

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 is the first work of its kind which uses learning experiences from past navigations as sample paths for coded pat-

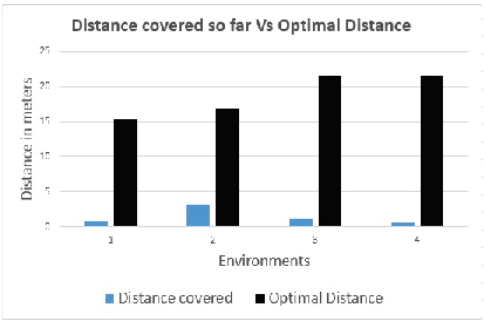


Figure 8: Comparison of distances covered so far to reach the goal by Informed RRT* versus the optimal distance for environments 1,2,3 and 4

terns which is used 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 directions of this work which we plan to explore in the future. We plan to look at other patterns which can be used to expand our library and enable the robot to explore random environments. Another extension to this work could be 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. Yet another extension of this work could be to explore techniques which can merge paths from different patterns to generate an optimal path for the robot to avoid an obstacle in case the encountered obstacle cannot be considerably matched to any of the stored patterns.

References

Berenson, D.; Abbeel, P.; and Goldberg, K. 2012. A robot path planning framework that learns from experience. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, 3671–3678. IEEE.

Choset, H.; Burgard, W.; Hutchinson, S.; Kantor, G.; Kavraki, L. E.; Lynch, K.; and Thrun, S. 2005. *Principles of Robot Motion: Theory, Algorithms, and Implementation*. MIT Press.

Coleman, D.; Sucan, I. A.; Moll, M.; Okada, K.; and Correll, N. 2014. Experience-based planning with sparse roadmap spanners. *arXiv preprint arXiv:1410.1950*.

Da Silva, B. N., and Mackworth, A. 2010. 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-Volume 1*, 317–324. International Foundation for Autonomous Agents and Multiagent Systems.

Fernández, F., and Veloso, M. 2006. Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, 720–727. ACM.

Gammell, J.; Srinivasa, S.; and Barfoot, T. 2014. Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, 2997–3004.

Hwang, V.; Phillips, M.; Srinivasa, S.; and Likhachev, M. 2015. Lazy validation of experience graphs. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, 912–919. IEEE.

Lien, J.-M., and Lu, Y. 2009. Planning motion in environments with similar obstacles. In *Robotics: Science and Systems*. Citeseer.

Phillips, M., and Likhachev, M. 2015. Speeding up heuristic computation in planning with experience graphs. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, 893–899. IEEE.

Phillips, M.; Cohen, B. J.; Chitta, S.; and Likhachev, M. 2012. E-graphs: Bootstrapping planning with experience graphs. In *Robotics: Science and Systems*.

Rusu, R.; Blodow, N.; and Beetz, M. 2009. Fast point feature histograms (fpfh) for 3d registration. In *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, 3212–3217.

Talvitie, E., and Singh, S. P. 2007. An experts algorithm for transfer learning. In *IJCAI*, 1065–1070.

Taylor, M. E., and Stone, P. 2009. Transfer learning for reinforcement learning domains: A survey. *The Journal of Machine Learning Research* 10:1633–1685.

Taylor, M. E.; Kuhlmann, G.; and Stone, P. 2007. Accelerating search with transferred heuristics. In *ICAPS Workshop on AI Planning and Learning*.

Taylor, M. E.; Kuhlmann, G.; and Stone, P. 2008. Autonomous transfer for reinforcement learning. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 1*, 283–290. International Foundation for Autonomous Agents and Multiagent Systems.

Torrey, L., and Shavlik, J. 2010. Policy transfer via markov logic networks. In *Inductive Logic Programming*. Springer. 234–248.

Torrey, L.; Shavlik, J.; Walker, T.; and Maclin, R. 2006. Skill acquisition via transfer learning and advice taking. In *Machine Learning: ECML 2006*. Springer. 425–436.