Autonomous Robots
https://doi.org/10.1007/s10514-019-09852-5

Real-time robot path planning from simple to complex obstacle patterns via transfer learning of options

Olimpiya Saha[1] · Prithviraj Dasgupta[2] · Bradley Woosley[2]

Received: 1 May 2018 / Accepted: 22 April 2019

Abstract
We consider the problem of path planning in an initially unknown environment where a robot does not have an *a priori* map of its environment but has access to prior information accumulated by itself from navigation in similar but not identical environments. To address the navigation problem, we propose a novel, machine learning-based algorithm called *Semi-Markov Decision Process with Unawareness and Transfer (SMDPU-T)* where a robot records a sequence of its actions around obstacles as action sequences called *options* which are then reused by it within a framework called *Markov Decision Process with unawareness (MDPU)* to learn suitable, collision-free maneuvers around more complex obstacles in future. We have analytically derived the cost bounds of the selected option by SMDPU-T and the worst case time complexity of our algorithm. Our experimental results on simulated robots within Webots simulator illustrate that SMDPU-T takes 24% planning time and 39% total time to solve same navigation tasks while, our hardware results on a Turtlebot robot indicate that SMDPU-T on average takes 53% planning time and 60% total time as compared to a recent, sampling-based path planner.

Keywords Robot path planning · Transfer learning · Reinforcement learning · Options · Markov decision processes with unawareness

1 Introduction

Autonomous navigation is a crucial aspect of many applications of mobile robots including unmanned search and rescue robots, self-driving vehicles, agricultural robots, extra-terrestrial rovers and assistive robots. In most of these applications, robot navigation must be done in real-time by calculating safe, collision-free trajectories for the robot using a motion planner (Choset 2005). Real-time robot motion planning in complex environments with many obstacles is challenging because robot sensors have limited range. Consequently, the robot's motion plan has to be dynamically updated as its sensors perceive new information from the environment - an expensive operation, both in terms of computation time and energy expended by the robot. To address this problem, one of the approaches that researchers have proposed is to use machine learning-based techniques including imitation learning and transfer learning (Hussein et al. 2017; Taylor and Stone 2009), where a robot learns a suitable maneuver or action model, either from demonstrations by a human expert or through self-rehearsals, to perform a task in a source environment. The robot is then required to perform a similar task in a new, target environment and has to adapt actions from its learned model so that it can perform the task in the new environment efficiently. A central aspect of many of these techniques is that the model learned by the robot is based on the exact actions that were demonstrated to it and does not generalize very well to newer actions that are slightly different from the demonstrated actions. Consequently, these techniques require considerable exploration to discover actions in a new environment. To address this, in this paper, we propose a novel transfer learning framework for learning robot navigation that uses a concept called options which are uninterrupted percept-action sequences to speed-up performing actions in a new, target environment. Specifically, we formulate a navigation problem facing a robot as a decision making problem under uncertainty by proposing a framework called Semi-Markov Decision-making Process with Unawareness and Transfer (SMDPU-T). SMDPU-T integrates a canonical framework for decision making under

✉ Olimpiya Saha
osaha@unomaha.edu

[1] Advanced AI, LG Electronics, Santa Clara, CA, USA

[2] Computer Science Department, University of Nebraska at Omaha, Omaha, NE, USA

uncertainty, Markov Decision Processes (MDPs), with three useful concepts from literature: (i) time-extended actions called *options* (Sutton et al. 1999) that enable the robot to reduce decision making time by performing sequences of actions (maneuvers) through successive states without requiring to recalculate a decision at each state, (ii) including *unawareness* (Rong et al. 2016) in the MDP to enable the robot to explore new actions (maneuvers) in certain situations to discover possibly shorter paths around obstacles with previously unencountered boundary contours, and (iii) using transfer learning to reuse maneuvers learned from similar, past environments in the current environment (Taylor and Stone 2009; Saha and Dasgupta 2017a), to reduce time by not running a full motion planning algorithm. To the best of our knowledge, our work is one of the first attempts to unify these three concepts within a single framework for robot motion planning. Our primary contribution in this paper is the *Semi-Markov Decision Process with Unawareness and Transfer (SMDPU-T)* framework which essentially combines transfer learning with hierarchical reinforcement learning (Sutton et al. 1999) and predictive decision making (Rong et al. 2016) to enhance the robot's capability to perform fast, reliable and efficient navigation without access to any *a priori* map of its current environment by reusing knowledge from its previous navigational experiences in the form of obstacle patterns and associated *options*. The inclusion of the *options* framework from hierarchical reinforcement learning enables the robot to perform path planning over several time steps based on the encountered obstacle pattern. This saves the robot from the overhead of planning at each time step thus contributing towards overall reduction in path planning and navigation times. The incorporation of MDPs with unawareness (MDPUs) facilitates autonomous expansion of the robot's state and action spaces as the robot discovers new states and useful actions (options) in future. This improves the overall navigation process in initially unknown and complicated environments. We have tested the performance of our algorithm to navigate a wheeled Coroware Corobot robot within the Webots simulator and a physical Turtlebot robot within different planar environments with different geometries of obstacles. Our simulated results show that SMDPU-T takes 24% planning time and 39% total time while taking only 8% more distance as compared to a recent sampling-based path planner, InformedRRT*(Gammell et al. 2014). Our hardware experimental results illustrate that SMDPU-T on average takes 53% planning time, 60% total time while covering only 1% more distance as compared to the sampling based path planner to complete the same navigational tasks in the physical environments.

The rest of this paper is structured as follows: in the next section we review existing literature in the area of machine learning-based robot path planning. Section 3 describes our proposed SMDPU-T framework and algorithms for real-time robot navigation in initially unknown environments. Section 4 analyses the performance bounds and complexity of our algorithm. In Sect. 5 we validate the performance of our algorithm with simulated and physical robots in different environments and finally, we conclude. An earlier version of this work has appeared as a conference paper in Saha and Dasgupta (2017c). In this paper, we have extended the conference paper by completely rewriting the introduction and related works section more extensively, added a new section analysing the theoretical performance of our proposed algorithm, and validated our algorithm's performance through new hardware experiments on a physical Turtlebot robot.

2 Related work

Robot motion and path planning is a central problem in robotics and excellent overviews of this topic are available in Choset (2005), Elbanhawi and Simic (2014), Galceran and Carreras (2013), Paden et al. (2016). Applications of machine learning to robotic path planning has been a topic of interest in the robotics community over recent years focusing on imitation learning or learning from demonstrations (LfD) (Hussein et al. 2017; Argall et al. 2009) and reinforcement learning (RL) (Kober et al. 2013). In many of these techniques, a Markov Decision Process (MDP) (Russell and Norvig 2009) that outputs a robot state to action mapping, called a policy, is used as the mathematical framework for the learning mechanism. In LfD, suitable robot actions or movements are demonstrated to the robot either by a human or by another robot in a source domain. Based on this acquired information, the robot has to perform similar operations in a previously unseen and possibly different environment called the target domain. One of the principal challenges in this problem involves determining the correct transformations and adaptations between environment features and corresponding operations performed by the robot across source and target domains (Nehaniv and Dautenhahn 2002). Researchers have addressed this problem using several techniques including automatically integrating demonstrations of a task by an expert into MDP policy updates (Kim et al. 2013), searching in the MDP policy space for learning motor primitives from demonstrations (Kober and Peters 2011; Manschitz et al. 2015), using statistical models such as Gaussian Mixture Regressions to reproduce demonstrations between source and target domain in human robot interaction tasks (Calinon 2009) and learning semantic labels from verbale human commands (Boularias et al. 2016). Chernova and Veloso (2009), proposed a technique where a robot evaluates its confidence in performing actions in the target domain and interactively requests more human demonstrations to update its MDP policy so that it can perform tasks more effectively in the target domain. In Levine et al. (2016), authors proposed a deep, convolution neural network-based technique for robustly extracting object features in the target domain, followed by a guided policy search technique to map sensor inputs directly to motor primitives for a robot learning to perform manipulation tasks from video demonstrations. LfD has also been shown to work successfully for robot navigation on physical ground and aerial robot platforms (Ng et al. 2006; Sofman et al. 2006; Silver et al. 2010).

Reinforcement learning (RL) techniques including Q-learning have also been used extensively to learn suitable robot navigation and motion planning strategies. In one of the earliest works in this direction, Moore and Atkeson (1993) developed a Q-learning-based prioritized sweeping algorithm. Subsequently, several researchers have used RL for motion planning problems including environment map building via object feature detection followed by trajectory optimization (Kollar and Roy 2008), determining robot path costs from obstacle features (Ratliff et al. 2009), and for terrain classification from visual data, using image processing and computer vision techniques (Happold et al. 2006; Kim et al. 2006; Erkan et al. 2007; Bajracharya et al. 2008). In Fernández et al. (2010), authors proposed a policy reuse technique for learning robot navigation (Fernández et al. 2010). Additional RL-based techniques for motion planning included combining one-shot learning and RL for supervised and unsupervised portions of the problem respectively (Takeuchi and Tsujino 2010), hierarchical RL for robot motion planning by dividing a task into subtasks (Bischoff et al. 2013; Jeni et al. 2007), and transferring data from simulation to a physical robot as Bayesian non-parametric prior (Cutler and How 2015; Deisenroth and Rasmussen 2011) for improved task learning. In most of these techniques, the source and target environment are identical or very similar. To enable the robot to perform actions in a new target environment, researchers have proposed transfer learning (Taylor and Stone 2009). Within transfer learning, to mitigate the problem of learning incorrect or unnecessary information from demonstration or policies in the source domain into the target domain (called negative transfer), several researchers have investigated techniques including support vector regression (Torrey et al. 2005), using a compliance metric with probabilistic mapping selection (Fachantidis et al. 2015), using the average trajectory from motion demonstrations to calculate the execution trajectory (Bowen et al. 2015), updating the regions in the MDP policy where there is a larger deviation between demonstration and execution (Mendoza et al. 2015), and inverse reinforcement learning to determine the reward function in RL more accurately using reward shaping (Brys et al. 2015; Wulfmeier et al. 2017; Burchfiel et al. 2016). In Hester and Stone (2013), authors proposed an online learning technique to determine better exploration strategies based on rewards yielded by those strategies. Our work is complementary to these directions, while additionally, we integrate options to speed-up robot actions as well as unawareness to handle previously unseen states or percepts. Majority of transfer learning approaches in reinforcement learning utilize the control policies learned in one or more source environments to solve the task in the target environment. In contrast to this, in our proposed approach, the robot does not apply a full control policy from one of its source environments but applies *options* or sub-policies based on the closest matched obstacle pattern from the source environments. This minimizes the problem of negative transfer commonly associated with transfer learning approaches as well as reduces the computational requirements as the robot only needs to learn and preserve the options instead of the full policies. Related to transfer learning, albeit not using RL, authors have also proposed experience-based learning (Berenson et al. 2012; Bruin et al. 2016; Saha and Dasgupta 2017a) as an effective technique for robots to navigate in initially unknown environments by identifying previously learned landmarks and transfering previously learned paths after environment specific adaptations. However, these approaches are usually limited to a restricted set of simple obstacle boundary patterns. Our work generalizes this direction where new states (obstacle patterns) and options (robot actions) are incrementally discovered with time.

Temporally extended actions or options (Sutton et al. 1999) have been shown to improve planning time when combined with primitive actions in contrast to using only primitive one-step actions. Extending this direction, (Konidaris and Barto 2007; Simsek et al. 2005) have proposed the idea of portable options which can be transferred across domains to solve an RL problem in a related but distinct domain. However, most of these RL-based techniques assume that the set of states and actions are already known and the decision problem is to determine suitable actions for each state that maximizes the expected reward from doing the actions. This assumption might not be valid for many real-world robotics problems including robot navigation where all states (locations) of the robots and/or all actions (e.g., a new path) might not be known *a priori*. In Rong et al. (2016), authors introduced an extension of Markov Decision Processes (MDPs) called MDPs with Unawareness (MDPU) where an agent intermittently tries a new, *explore* action to discover new actions and states. Our work extends MDPUs for the robot navigation problem and combines it with options using a layered approach to improve the path planning and navigation time for robots. Complementary to our approach, very recently, researchers have proposed techniques including abstract MDPs (Gopalan et al. 2017) and policy sketches (Andreas et al. 2017) that attempt to generalize MDP policies to work across different, new environments. Our work is complementary to the concept of portable options proposed by Konidaris and Barto (2007). However, we consider more generalized agent space descriptors which can have transformations in different environments. Although we generally discuss path planning in 2D planar environments, our approach can be easily extended in higher dimensional path planning problems like 3D navigation by UAVs and object manipulation by robotics arms in humanoids. Recently, Bacon et al. proposed the option critic architecture (Bacon et al. 2017) which is capable of learning the internal policies as well as the termination conditions of options corresponding to the policy over option. However, their experimental evaluation has only been conducted in RL domains and has not been evaluated on real-time robotic tasks.

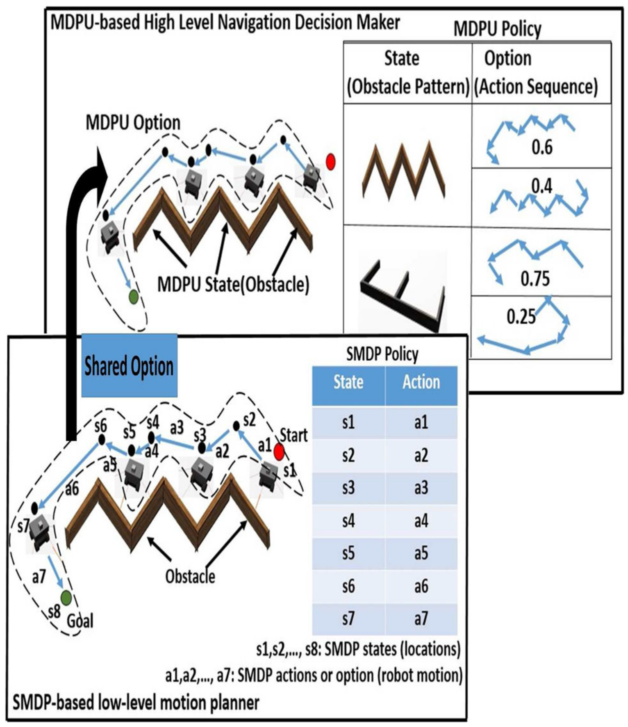
Recent success in deep learning has led to its formal applications in robotics. In Mahler et al. (2017), Mahler and Goldberg (2017), the authors have used the Dexterity Network(Dex-Net) 2.0 and a special type of convolutional neural network (CNN) called Grasp Quality Convolutional Neural Network (GQ-CNN) to plan robust grasps for general object manipulation and for robotic bin picking. Dex-Net 2.0 is a dataset consisting of 1500 3D object models and associated grasps and grasp-quality metrics. Our work has some similarity with this work but instead of using object models for the problem of grasp planning, we utilize obstacle patterns and associated options learned by the robot for real-time path planning in combination with predictive decision making embedded in a reinforcement learning framework. In Mnih et al. (2013), authors formally introduced the concept of deep reinforcement learning by proposing *deep Q-networks (DQN)* which utilizes deep networks to approximate the action value function associated with reinforcement learning and was used by an agent which learned to play the game of Atari. Following this research direction, deep reinforcement learning has been applied to robotic navigation and grasp planning. In Lillicrap et al. (2016), the authors proposed deep deterministic policy gradient (DDPG) as an extension of deterministic policy gradient (DPG) (Silver et al. 2014) which has been applied to various robot control tasks in the continuous domain. Some of the other recent works which used deep reinforcement learning to solve continuous robot control tasks like robot manipulation skills or robot locomotion include (Levine et al. 2016; Gu et al. 2017; Zhu et al. 2017; Montgomery et al. 2017). Although these recent works have illustrated promising results in improving different robotic applications, however, deep reinforcement learning still requires considerable time and diverse data to train the underlying reinforcement learning model for it to generalize to a new, previously unseen task. This limitation makes their utilization difficult for real-time robotic applications. In contrast, our proposed approach requires much less training data for the robot at the beginning as the robot accumulates its own training data as it navigates different environments and encounters various obstacle patterns, thus, ensuring path planning and navigation in real-time.

3 Robot navigation using semi-Markov decision processes with unawareness

We consider a wheeled robot situated within a bounded environment $\mathcal{Q} \subseteq \Re^2$; $\mathcal{Q}_{free} \subset \mathcal{Q}$ denotes the free space, and $\mathcal{Q}_{obs} = \mathcal{Q} - \mathcal{Q}_{free}$ denotes the space occupied by obstacles, respectively. The robot is initially unaware of the location and geometry of obstacles in $\mathcal{Q}_{obs}$. A robot's primitive action is denoted as $a = (\theta, d)$, where θ is its desired bearing and d is the distance it needs to move; the set of primitive actions is denoted by A_P. The robot first rotates in place to achieve its desired bearing θ and then navigates forward in the achieved orientation until it covers distance d. Each robot is equipped with a depth sensor that can record a point cloud representing an obstacle. We also assume that each robot is localized w.r.t. its environment using a localization device.[1] The robot determines its distance from obstacles in its proximity using the point cloud information from its depth sensor and its localization information retrieved from the localization device. The robot effectively leverages this information to avoid collision with obstacles in its vicinity. The objective of the robot is to solve a navigation task by finding a collision free path as a sequence of actions between two points, $(q_{start}, q_{goal}) \in \mathcal{Q}_{free}$. To avoid collisions along its navigation path, the robot has to decide upon a sequence of actions when it encounters an obstacle. Due to limited perception range of the sensors and features of the obstacle boundary, the robot might not be able to perceive the entire obstacle boundary and is uncertain about the actions that correspond to the shortest path around the obstacle that takes it closest to q_{goal}. Following the work of Lien and Lu (2009), we assume that obstacles are characterized by the robot according to the contour or pattern of their boundary perceived by the robot. We assume that there is a finite set of obstacle boundary patterns, denoted by $P = \{P_i\}$ where each obstacle pattern denoted by set P_i contains a sequence of 2D coordinates $\{(x, y)\}$. The robot initially does not know the set P. It is taught through training to maneuver around a subset $P_L \subset P$ of basic boundary patterns. Its objective is to efficiently (time and distance-wise) maneuver around obstacles with previously un-encountered boundary patterns in real time, while utilizing the maneuver information in P_L, and also incrementally updating P_L as new obstacle boundary patterns are encountered.

[1] To achieve localization SLAM-based techniques could also be used; we do not discuss localization issues further to focus on the learning problem.

We formulate the robot's problem of navigating around previously unencountered obstacles as a decision making problem under uncertainty by proposing a framework called Semi-Markov Decision-making Process with Unawareness and Transfer (SMDPU-T). Our framework employs a hierarchical planner architecture that is shown in Fig. 1. A lower level planner handles decision making for individual motion steps around an obstacle using a Semi-Markov Decision Process (SMDP) (Sutton et al. 1999), while a higher level planner decides on the general trajectory that the robot should follow when it has limited knowledge of the obstacle boundary (e.g., possibly only perceived a smaller portion of the obstacle's entire boundary) using an MDP with Unawareness (MDPU) (Rong et al. 2016). These two MDP-based planners are coupled through time-extended actions called *options* - the individual actions control the movement of the robot around obstacles within the SMDP framework, while the same actions, considered as a sequence or options, control the trajectory of the robot within the MDPU framework. Finally, our proposed framework also uses a transfer learning technique (Taylor and Stone 2009) to speed up the robot's trajectory computation by opportunistically reusing trajectories remembered from previously encountered navigation maneuvers. We now describe each of these concepts more formally.



State	Action
s1	a1
s2	a2
s3	a3
s4	a4
s5	a5
s6	a6
s7	a7

Fig. 1 Schematic of navigation approach used by SMDPU-T using an MDP with unawareness (MDPU) as the high level decision maker and a semi-MDP (SMDP) as the low level motion planner for a navigation task. The option is shared between the SMDP and the MDPU

Background Markov Decision Processes (MDPs) are a widely used technique for decision making under uncertainty. An MDP is specified by a tuple (S, A, T, R) where S is the set of states in the decision problem, A is the set of actions that the agent or robot can make at each state, $T : S \times S \times A \rightarrow [0, 1]$ specifies a transition probabililty for going between states $(s, s') \in S$ while taking action $a \in A$, and R is a real-valued reward associated with this transition. A detailed description of MDPs is available in Russell and Norvig (2009).

3.1 Representing robot action sequences as options in SMDP.

For motion-related decision making for avoiding obstacles by the lower-level planner, a robot in our proposed framework uses a Semi-Markov Decision Process (SMDP)- a variant of an MDP where sequences of actions are treated as a single unit called an option (Konidaris and Barto 2007). Options afford faster robot maneuvers around obstacles by not requiring to determine individual collision avoiding actions at each waypoint on the path around the obstacle every time; rather they can be determined once initially and used as a (mostly) uninterrupted action sequence to avoid the same obstacle later on. The state space S_{SMDP} for our SMDP corresponds to a set of waypoints $q \in \mathcal{Q}_{free}$ along a path around an obstacle. An SMDP option has three components: (i) a policy $\pi_{SMDP} : S_{SMDP} \rightarrow A_P$ that recommends a collision avoidance action from the primitive action set for each state perceived by the robot, (ii) a termination condition $\beta : S_{SMDP} \rightarrow [0, 1]$ that gives a probability, at each state, to stop following the policy and (iii) an initiation set $I \subset S_{SMDP}$ that gives the set of initial states that triggers the option.

To build the initial set of states and options, a robot goes through a *training phase* - it is allowed to solve a set of navigation tasks (move between different start and goal locations) using a motion planner, while encountering different obstacles whose boundaries are one of the basic boundary patterns in P_L. For each pattern $P_i \in P_L$, the waypoints of the path around the obstacle returned by the motion planner correspond to the states of the SMDP, S_{0,P_i}, The sequence of maneuvers made by the robot to follow the path along those waypoints define the initial option O_{0,P_i}. In this manner, one SMDP is created for each obstacle boundary pattern in P_L. A set of primitive actions $A_{prim_{0,P_i}} = \{(\theta, d)\}$ is also created for each P_i. Each action $a \in A_{prim_{0,P_i}}$ ensures that the robot will not collide with the corresponding obstacle pattern P_i by following that action. In this work, we have considered polygonal actions to keep the action space simple as the focus of our work emphasizes more on the high level decision making by the robot and not on the low level actions controlling the robot's motion.

3.2 Selecting suitable options at obstacles based on MDPU.

Within the higher level planner, the robot treats its navigation problem as deciding on a trajectory around a perceived obstacle. For making this decision, the states of the higher level planner correspond either to obstacle states represented by the obstacle boundary pattern perceived on its sensor when the robot's path towards q_{goal} is blocked by an obstacle, or, a *free* state, where no obstacles are perceived in direction of q_{goal}. The trajectory to be taken around an obstacle (state) is considered as the recommended option for that state. To find the correct decision—which trajectory would be the best for a certain perceived obstacle boundary pattern, we propose to use an extension of MDPs called MDP with Unawareness (MDPU) Rong et al. (2016) that affords an *explore* action, in addition to existing options to discover new, possibly shorter path trajectories around an obstacle. Formally, an MDPU is represented as a tuple $M^u = (S_{M^u}, O, T, R, a_0, D, R^+, R^-)$
where

S_{M^u} is a set of states
O is a set of options
$T : S_{M^u} \times S_{M^u} \times O \rightarrow [0, 1]$ is a transition function.
$R : S_{M^u} \times S_{M^u} \times O \rightarrow \Re^+$ is a reward function
$a_0 \notin O$ is an explore action
$D : \mathbb{Z}^+ \times \mathbb{Z}^+ \times S_{M^u} \rightarrow [0, 1]$ is a discovery function.
R^+, R^- : is the exploration reward for explore action at s

The first four parameters of an MDPU are similar to those in an MDP, as described earlier. We consider that the obstacle patterns encountered during the training phase form the initial set of MDPU states S_{M^u}, while the MDPU's options are the same as the SMDP's options (collision avoiding paths around obstacle). In an MDPU, a special action called *explore*, denoted by a_0, is used to discover new options and states. For our case, the explore action is implemented as finding a new, collision-free path at an MDPU state (obstacle) using a motion planner. The discovery probability function $D(.)$ in the MDPU specifies the probability that a new option can be discovered by performing the explore action at a state. The reward R^+ for performing the explore action and discovering a new option (similarly, R^- for not discovering new option) is proportional to $D(.)$. The output of the MDPU is a policy, $\pi^u : S_{M^u} \times O \rightarrow [0, 1]$ that probabilistically prescribes a utility-maximizing sequence of options for the obstacle pattern (MDPU state) encountered which allows the robot to navigate through obstacle states until it reaches a free state. Note, that when a new option and new state is discovered via the explore action, all the MDPU parameters are updated to accommodate the new state and option. In order to avoid redundancy, the robot rejects any option generated by the explore action which is within a similarity threshold of any of the existing options in the library. In such a scenario, the robot utilizes the explore action again to generate a new option.

3.3 Transferring states and options across environments.

Transfer learning techniques (Taylor and Stone 2007, 2009) provide a mechanism by which state-action pairs used to solve a problem in one domain or environment can be transferred and re-used with suitable adaptations, in another similar environment. In our framework, the set of state and option pairs, S_{0,p_i} and O_{0,p_i}, $\forall P_i \in P_L$ learned during the training phase is stored in a library $\mathcal{L}$. Later on, when a robot is presented with a navigation task in a new environment and encounters an obstacle, it retrieves a suitable option from library $\mathcal{L}$ and reuses it. Transfer is implemented by function $\tau : P \rightarrow S \times O$ that takes a currently encountered obstacle boundary pattern p_{cur} (in new environment) and returns an obstacle state s (encountered in an earlier environment), along with the set of options o that was used to navigate around s. τ uses a well-known feature matching algorithm called SAC-IA (Sample Consensus Initial Alignment) to determine obstacle state s with the highest degree of match with p followed by a transformation to best align the geometry of p with s; the same transformation is then used on the options corresponding to s to calculate o so that the transformed options can be used in for the currently perceived obstacle. The extent of match between the currently encountered obstacle and the matched obstacle from $\mathcal{L}$ is determined by evaluating their $fitness\text{-}score$ or Jaccard Index which measures the extent of overlap between two geometric shapes. Further details of this technique are described in Saha and Dasgupta (2017a).

3.4 Navigating robot in new environment using SMDPU-T

The navigation algorithm used by the robot is shown in Algorithms 1 and 2. The main symbols used in the algorithm and their descriptions are given in Table 1. Initially, when the robot is placed in an environment, it aims to navigate towards the given goal in a straight line path until it encounters an obstacle. The robot maintains this behavior of orienting itself towards the goal and traveling towards it whenever it ends up in an obstacle free region in the environment during its navigation. This specific behavior also helps the robot in situations in certain situations. For example, if the centroid of all training maps is covered with obstacles while, during testing, the center of the environment is free space, the robot would navigate towards the goal by orienting itself towards the goal

Table 1 Symbols used in problem formulation and their description

Symbols	Description
Q, Q_{free}, Q_{obs}	Bounded environment where the robot operates, free space in the environment, space occupied by obstacles in the environment
q, q_{start}, q_{goal}	A configuration of the robot, configuration of the robot at the start, configuration of the robot at the goal
a, A_p	An action of the robot, set of primitive actions of the robot
P, P_i, P_L	Set of obstacle boundary patterns, a single obstacle boundary pattern, set of initial obstacle boundary patterns
S_{SMDP}	State space of the SMDP which consists of a set of waypoints of a path around an obstacle
π_{SMDP}, I, β	SMDP policy that recommends a collision avoidance action for each SMDP state encountered by the robot, the initiation set that gives the set of initial SMDP states that trigger a specific option, the termination probability at each state of the SMDP giving probability of terminating the option
$S_{0,P_i}, O_{0,P_i}, A_{Prim_{0,P_i}}$	The state space corresponding to an initial obstacle pattern P_i, the set of options around an obstacle pattern P_i consisting of maneuvers to avoid P_i, set of primitive actions created for a specific obstacle pattern P_i (MDPU state) which ensures that the robot will not collide with P_i
M^u	Markov Decision Process with Unawareness (MDPU)
S^{M^u}, O, R, T	Set of MDPU states (obstacle boundary patterns), set of options for MDPU states (series of actions around obstacles), MDPU reward function giving real-valued rewards for MDPU states and option, MDPU transition function giving probability of MDPU states and options
$a_0, D, R^+/R^-$	Explore action in MDPU such that $a_0 \notin O$, discovery function specifying the probability of a new option being discovered by performing the explore action in a state, exploration reward at an MDPU state gives the probability that a new option can be discovered at an MDPU state by following the explore action
π^u	MDPU policy giving probability of an option at an MDPU state
L	Library containing the set of state and option pairs
τ, *transf*	Transfer function that takes the currently encountered obstacle pattern as input and returns the best matched state, the fitness score and the set of options for that state, transformation matrix returned by τ aligning the current obstacle pattern with the matched state
s, s_{new}	MDPU state, newly discovered MDPU state
o, o_{new}, o_{cum}	Option followed by the robot, newly discovered option corresponding to the new MDPU state s_{new}, cumulative option formed by combining all the previous option with the latest interrupted option
o'	Part of option o followed by the robot before being interrupted by a previously unseen obstacle or by termination probability β
p, p_{new}, p_{cum}	Obstacle pattern, newly detected obstacle pattern, cumulative obstacle pattern formed by combining previously encountered obstacle patterns with new obstacle pattern p_{new}
$L_{coll}, S_{coll}, A_{Prim_{s_{coll}}}$	Collision library storing the locations (SMDP states) where options got interrupted, collision state space storing the corresponding MDPU state where the options got interrupted, the set of primitive actions corresponding to the set of collision states S_{coll}
$Q(s, o)$	SMDP Q-value corresponding to state option pair (s, o)

and moving through the free space, instead of attempting to navigate around an obstacle pattern from its library. Whenever the robot encounters an obstacle with obstacle pattern p, it uses the transfer function τ to determine the MDPU state $s \in S_{M^u}$, the $fitness\text{-}score$ of the MDPU state with the currently encountered obstacle pattern along with the transformation matrix $transf$ (Line 3). It also retrieves the corresponding best possible option using Algorithm 2 (Line 4). The robot uses the selected option o to navigate until it encounters an obstacle while following o, or o gets terminated according to the SMDP termination condition β before the robot reaches a *free* state. If either of these conditions occur,

Algorithm 1: SMDPU-T algorithm

```
Input: Perceived obstacle boundary pattern $p$, transfer function $\tau$
Output: An option $o$ from start to a $free$ state
1  Initial MDPU policy $\pi_0 : S \times A \rightarrow [0, 1]$ is evaluated from a uniform distribution of states and actions
2  $p_{cum} \leftarrow p$; $o_{cum} \leftarrow \{\varnothing\}$; $\mathcal{L}_{coll} \leftarrow \{\varnothing\}$; $S_{coll} \leftarrow \{\varnothing\}$
3  $(s, fitness\text{-}score, transf) \leftarrow \tau(p_{cum})$
4  $o \leftarrow selectOption(s, transf, M^u, p_{cum})$
5  repeat
6      Navigate robot while following actions in option $o$
7      if obstacle is detected OR option is terminated then
8          Update termination condition $\beta$ of option $o$
9          Update rewards and Q-values associated with option $o$
10         $o_{cum} \leftarrow o_{cum} \cup o' | o' \in o$
11         $p_{new} \leftarrow$ Perceived obstacle boundary pattern
12         $p_{cum} \leftarrow p \cup p_{new}$
13         $(s, fitness\text{-}score, transf) \leftarrow \tau(p_{cum})$
14         $last\text{-}score \leftarrow fitness\text{-}score$
15         if fitness score of $p_{cum}$ < fitness threshold then
16             $(s, transf) \leftarrow \tau(p_{new})$
17             $S_{coll} \leftarrow S_{coll} \cup s$
18             $\mathcal{L}_{coll} \leftarrow \mathcal{L}_{coll} \cup$ collision location
               $o \leftarrow SelectOption(s, transf, M^u, p_{new})$
19         else
20             $o \leftarrow SelectOption(s, transf, M^u, p_{cum})$
21 until Free state is reached;
22 if $last\text{-}score$ < fitness threshold then
23     $s_{new} \leftarrow p_{cum}$
24     $o_{new} \leftarrow optimizeOption(o_{cum}, S_{coll}, \mathcal{L}_{coll})$
25     Update $M^u$ with the updated option $o_{new}$ and new state $s_{new}$
26 Update $M^u$ with reward
27 Evaluate policy $\pi^u$ after updating $Q$
```

the robot updates the termination probability and the associated rewards and Q-values. The termination probability is incremented by a constant factor which increases the probability of termination of the option at the current state. After each update, the termination probabilities at each option state are updated to maintain the probability constraint. The robot then updates the option followed so far o_{cum} by combining it with the interrupted option o' (Line 10). o' represents the part of the option o that the robot followed before being terminated by an unseen obstacle pattern or by the termination condition of o and thus $o' \in o$. At this point, the robot determines if the obstacle pattern $p_{cum} = p \cup p_{new}$ formed by combining the previously detected obstacle pattern p with the newly detected obstacle pattern p_{new}, is substantially similar to one of the existing MDPU states (obstacle patterns) in S_{M^u} by evaluating a similarity measure (*fitness score*) for p_{cum}.[2] If this *fitness score* is below a certain fitness threshold, the robot determines the best matched state based on the current pattern p_{new} only.[3] We noted that the extent of the match between the perceived obstacle pattern and the obstacle pattern in the robot's library depends on the value of the fitness threshold. If this value is set too high, the robot requires a very close match with one of the obstacle patterns in the library in order for a match to be achieved. In contrast, if the fitness threshold is set too low, the robot achieves a match even if the perceived obstacle is considerably different from the obstacles in the robot's library. Hence, there is a clear trade-off between the accuracy of match requires to be achieved and the generalization of the proposed method to different environments with varied obstacles. In our experiments, we have empirically varied the fitness threshold value between 0.6 and 0.9 to note the performance of the robot and found that the robot performed best for the threshold value of 0.75. Hence, all our experiments were performed using the fitness threshold value of 0.75. Instead of using a simple thresholding function, an advanced function can be used to determine the appropriate value of this parameter. We leave this aspect to the scope of future work as this is not the central focus of this work.

The robot also records the location(s) where the option got interrupted inside a collision library $\mathcal{L}_{coll}$ as well as the state s inside a cumulative collision state S_{coll}, so that they can be used for updating the MDPU at the end of the navigation to avoid the current obstacle. It is important to mention here that although we use the term collision, the robot does not actually collide with the obstacle(s) but determines a new option to be followed in accordance to the newly perceived obstacle. However, if the robot continued to follow the same option, it would eventually lead to collision. Once the matching MDPU state is determined, Algorithm 2 is called to determine the best option o and the robot navigates around the obstacle using the actions in the option. This process continues until the robot reaches a *free* state (Line 21). Upon reaching the *free* state, the current obstacle is avoided. The *fitness score* at this point gives the match between the entire obstacle boundary pattern p_{cum} and the best matched MDPU state in S_{M^u}. If this *fitness score* is below a certain fitness threshold, it means that the obstacle pattern does not correspond to any existing MDPU state. In that case, the MDPU is updated with a new state s_{new} corresponding to the entire obstacle boundary pattern and the corresponding options used to navigate the robot around the obstacle for this new state after optimizing the option (Line $22-25$). We discuss the option optimization technique used by SMDPU-T in Subsect. 3.4.1. Finally the rewards for s_{new} as well as the intermediary states are evaluated, the corresponding Q-values are updated and the policy π^u is evaluated (Lines $26-27$).

[2] *Fitness score* is measured by calculating the Jaccard Index(JI) between the detected obstacle pattern and each obstacle pattern in P as described in Saha and Dasgupta (2017a).

[3] If the *fitness score* is still below the fitness threshold, a local motion planner is called to construct a trajectory around the detected obstacle pattern.

3.4.1 Post-optimization of new options by sampling new states.

After the robot finally arrives at the *free* state, MDPU M^u is updated with a new state state s_{new} in case s_{new}, obtained by combining all the encountered obstacle patterns from the obstacle, was not matched with any existing state in S_{M^u}. The followed option so far o_{cum}, corresponding to the newly discovered state s_{new}, is optimized to form a new option o_{new} (Line 24). In Sutton et al. (1999), it was shown that better options can be created from previously seen options, if options are allowed to terminate before they terminate according to their termination probability β. In SMDPU-T we use a post-optimization technique where an option, after its initial construction, can be modified based on the evaluated Q-values. For this, the robot records the location(s) (or SMDP state(s)) $\{(x_{coll}, y_{coll})\}$ where the option got obstructed either due to previously unseen obstacle(s) or due to the SMDP termination with probability β, in a collision locations library $\mathcal{L}_{coll}$. The corresponding MDPU state(s) are recorded in a collision state library S_{coll}. The robot samples new locations or SMDP states $\{(x_{new}, y_{new})\}$ in the vicinity of collision state(s) $\{(x_{coll}, y_{coll})\}$ following the actions available in $A_{prim_{S_{coll}}}$ and records them in $\mathcal{L}_{new} = \{(x_{new}, y_{new})\}$. Finally, the robot creates a set of modified options O_{new} by modifying the SMDP policy of o_{cum} by replacing the SMDP state(s) $\{(x_{prev}, y_{prev})\}$, the predecessor of SMDP state (x_{coll}, y_{coll}) with each new state in $\mathcal{L}_{new}$, as $\pi_{SM}^{mod} \leftarrow \pi_{SM} \setminus \{(x_{prev}, y_{prev}), a\} \cup \{(x_{prev}, y_{prev}), a_{new}\}$ where $a, a_{new} \in A_P$. It then replaces the initial option o_{cum} with the new option o_{new} only if $Q(s_{new}, o_{new}) > Q(s_{new}, o_{cum})$. This condition replaces undesired SMDP state(s) (robot navigation waypoints) inside o_{cum} by more desirable states (waypoints giving shorter navigation path), and generates higher immediate reward(s). So this option optimization technique used ensures the generation of an improved option in comparison to the initial option whenever possible. After the robot has discovered a new MDPU state s_{new}, it evaluates a set of primitive actions $A_{Prim_{s_{new}}} = \{(\theta, d)\}$ for the new state s_{new} following similar approach as discussed in the SMDP background section earlier.

3.4.2 Selecting options maintaining exploration-exploitation.

In this subsection, we discuss about the details of Algorithm 2 and how it selects a suitable option for the encountered MDPU state by the robot. In order to maintain the exploration-exploitation balance, Algorithm 2 invokes *explore* action a_0 on the newly discovered states according to the discovery probability $D(.)$ (Lines 4-5). If *explore* action is selected, a local path planner is invoked to create a new option for the new state s following which the MDPU reward for exploration $R^+(s, a_0)$ and Q-value $Q(s, a_0)$ are updated (Lines 6-9). On the contrary, if exploration is not selected, one of the past options is chosen based on the current policy $\pi^u(s, o)$ (exploitation). In order to give higher preference to exploration, we consider that the first option for a new state is always the *explore* action which means exploration can be selected even in the exploitation phase. If *explore* action is selected under exploitation, depending on whether $\mathcal{L}_{coll}$ is empty or not, the MDPU M^u receives a reward of $R^-(s, a_0)$ or $R^+(s, a_0)$ (Line 9 and 20). In the former case, the probability function is called again to select another option according to π^u (Lines 12-13). In the latter case, a local motion planner is called to create a new option for the current state s (Line 18-19). To enable the robot to select a suitable option at an obstacle state that has a higher probability of resulting in a shorter collision-free path around the obstacle, we use a reinforcement learning technique called SMDP Q-learning (Sutton et al. 1999). The Q-value update rule of an option o starting at state s is given by:

$$Q(s, o) = Q(s, o) + \alpha \left[r_s^o + \gamma^k \max_{o' \in \mathcal{O}_{s'}} Q(s', o') - Q(s, o) \right] \tag{1}$$

where s' is the state where option o terminates, α is the learning rate, γ is the discount factor while r_s^o is the expected reward for option o being initiated in the state s which can be expressed as below:

$$r_s^o = \beta \sum_{k=1}^{\infty} \left(\sum_{l=1}^{k} \gamma^{l-1} r_{t+l} \right) \tag{2}$$

where $t + l$ is the random time at which o terminates probabilistically depending on the termination condition β. Equation 1 is the objective function that the robot optimizes. The Q function maximizes a multi-objective reward function which can be expressed as $r_t = w' \phi(s)$, where w is a weight vector and $\phi(s)$ is the feature vector corresponding to the SMDP state that the robot is in at time t. The feature vector captures three important attributes of efficient and safe navigation namely distance to the closest obstacle, length of the path and distance to the goal. Details about the state features formulating the reward can be found in Saha and Dasgupta (2017b). Although time is not directly involved in the objective function, it is minimized as it is directly related to the distance traveled by the robot. The MDPU policy π^u is updated after the robot reaches the goal and the optimal policy is selected as below:

$$\pi^{u^*} = \arg\max \pi^u(Q(s, o)) \; \forall s, o \tag{3}$$

Algorithm 2: SelectOption Algorithm

```
Input: Matched state $s$, Transformation $transform$, MDPU $M^u$, Obstacle pattern $P$
Output: An option
1  foreach option $o$ in $M^u$ do
2      Scale and transform $o$ using $transform$
3  if MDPU state $s \notin S_{0_{M^u}}$ then
4      Select explore action $a_0$ with prob. $D(.)$ OR option
       recommended by current policy with prob. $1 - D(.)$
5      if explore action selected then
6          Construct option for $a_0$ using local path planner
7          Increment $R^+(s, a_0)$, update $Q(s, a_0)$ (eqn.1)
8          Add $o$ as a new option for s, select $o$
9      else
10         repeat
11             Select option $o$ with probability $\pi^u(s, o)$ OR explore
               action $a_0$ w/ prob. $1 - \pi^u(s, o)$
12             if explore action selected then
13                 if $\mathcal{L} - coll = \{\varnothing\}$ then
14                     Decrement $R^-(s, a_0)$, update $Q(s, a_0)$ (eqn.1)
15                 else
16                     Construct option for $a_0$ using local path
                       planner
17                     Increment $R^+(s, a_0)$, update $Q(s, a_0)$ (eqn.1)
18                     Add $o$ as a new option for $s$, select $o$
19         until option $o$ is selected;
20 else
21     Select option $o$ with probability $\pi^u(s, o)$
22 Scale and transform option $o$
23 return $o$
```

4 Analysis

Lemma 1 *For any detected obstacle pattern, SMDPU-T is guaranteed to generate a collision-free option to the free state so that the robot can successfully avoid the encountered obstacle.*

Proof (by contradiction) Let us assume that SMDPU-T does not guarantee a collision-free option to the *free* state. While navigating around an obstacle, two cases can arise:

Case I. The robot encounters one of the known obstacle patterns corresponding to a state in $S_{0_{M^u}}$. This case is intuitive as the lemma follows from the collision avoidance guarantee offered by the local motion planner.

Case II. The robot encounters a novel obstacle pattern. In this scenario, the robot was following an option o prescribed by the MDPU policy $\pi^u(s)$, where $s \in S_{M^u}$ is the MDPU state corresponding to the boundary pattern of the obstacle p. While doing so, the robot encounters a previously unseen part of the same obstacle with boundary pattern p_{new}. Algorithm 1 then tries to match the currently encountered obstacle p_{new} with the known obstacle patterns $S_{0_{M^u}}$. If p_{new} can be matched to an existing state in S_{M^u} within the fitness threshold, the option prescribed by MDPU policy $\pi^u(s)$ is used to navigate around the obstacle, which is guaranteed to be collision free. If p_{new} does not have sufficient degree of match with any state in S_{M^u}, the motion planner is used to navigate, which again guarantees a collision-free around the obstacle. So, in both cases, SMDPU-T algorithm guarantees a collision free option. Therefore, our assumption was incorrect. Hence proved. □

Theorem 1 *The cost of the option selected by SMDPU-T for a given obstacle pattern or MDPU state is guaranteed to be within a bound of $(Peri_{lg}^{opt}/CH) * D_{opt}$ where $Peri_{lg}^{opt}$ is the optimized perimeter of the largest obstacle pattern in M^u, CH is the convex hull of the obstacle pattern and D_{opt} is the cost of the optimal option.*

Proof Let us consider that the length of the optimal option corresponding to an MDPU state s is D_{opt}. Considering the start and goal locations to be on the opposite side of the obstacle, the best path or option around s is given by $D_{opt} = CH/2$ where CH is the length of the convex hull of the obstacle (while ignoring the constant minimum distance that the robot needs to maintain from the obstacle). [4] During navigation, the robot can encounter various situations which can be classified into two cases.

Case I. *The robot knows that it has perceived the entire obstacle pattern.* In this case the robot selects an option o from the set of options O corresponding to s according to the policy $\pi^u(s, o)$. The expected option length can be calculated as $EL = \sum_{i=1}^{i=|O|}(\pi(s, o_i) \times \sum_{j=1}^{j=|o_i|} d_j)$ where d is the length of each primitive action a consisting each option o in O. If we consider the initial policy assigns uniform preference to all the available options o at state s, $EL = Peri/2 - OF$ where $Peri$ denotes the length of the obstacle boundary and OF denotes the length reduced by option optimization. Thus, the cost of option o selected by SMDPU-T in relation to D_{opt} is given by $(Peri - 2OF_c)/CH = Peri_{opt}/CH$. We can denote $Peri - 2OF_c$ as the optimized perimeter of the obstacle pattern under consideration.

Case II. *The robot does not know if it has perceived the entire obstacle pattern.* If the robot perceives only the partial obstacle pattern (state s), it can either select option o_c corresponding to the actual obstacle pattern or option o_{ic} corresponding to an incorrect obstacle pattern according to the MDPU policy π^u. Let us consider that π^u selects o_c with probability p_r. Selection of wrong option can be further classified into the following two scenarios.

Scenario I. *The actual obstacle pattern is larger than the perceived obstacle pattern p but the robot selects an incorrect option corresponding to a previously learned smaller obstacle pattern.* In this scenario, as the robot selects a smaller

[4] If we consider $Dis_{thresh} \neq 0$, from our analysis, $D_{opt} = CH/2 + 3Dis_{thresh}$.

option than the obstacle pattern, once the option gets terminated, the robot selects another option corresponding to the combined obstacle. It covers the perimeter of the actual obstacle $Peri_c/2$. The expected length of the option in this scenario can thus be expressed as $EL = p_r(Peri_c/2 - OF_c) + (1 - p_r)(Peri_c/2)$. In the worst case, we consider that $p_r \rightarrow 0$ i.e. the wrong option is selected. Hence, EL reduces to $Peri_c/2$. Thus, the cost of the option in this scenario can be expressed in relation to D_{opt} as $Peri_c/CH$.

Scenario II. *The actual obstacle pattern is smaller than the perceived obstacle pattern p but the robot selects an incorrect option corresponding to a larger obstacle pattern* In this scenario the robot covers a greater distance as it selected the wrong option o_{ic} corresponding to a larger obstacle pattern in comparison to the actual obstacle pattern. We can express the expected option length in this scenario as $EL = p_r(Peri_c/2 - OF_c) + (1 - p_r)(Peri_{ic}/2 - OF_{ic})$ where $Peri_c$ and OF_c are respectively the perimeter and optimization lengths of the correct obstacle pattern and $Peri_{ic}$ and OF_{ic} are respectively the perimeter and optimization lengths of the incorrect obstacle pattern. Under the worst case situation, the above equation becomes $EL = Peri_{ic}/2 - OF_{ic} = Peri_{ic}^{opt}$ (follows similarly from proof of *Scenario I*). In the worst case scenario, o_{ic} will correspond to the largest obstacle pattern in the state space S of M^u whereas the actual obstacle will correspond to the smaller partially perceived obstacle by the robot. Thus, the cost of the option in the worst case can be expressed as $Peri_{lg}^{opt}/CH * D_{opt}$ where $Peri_{lg}^{opt}$ is the optimized perimeter of the largest obstacle pattern in M^u. We specifically consider the optimized perimeter of the largest obstacle pattern in the MDPU library, as we are concerned about the worst case bound of the option cost. Optimized perimeter represents the length of the obstacle boundary reduced by a factor by which our algorithm optimizes the length of the option. □

Theorem 2 *SMDPU-T algorithm has a time complexity of $\mathcal{O}(|\mathcal{L}_{coll}|^2.|O|)$ where $|\mathcal{L}_{coll}|$ is the size of the collision library and $|O|$ is the size of the option space of M^u at a given state.*

Proof Following (Rusu et al. 2009), the time complexity in Algorithm 1 derives from executing the transfer function τ and calculating the *fitness score* for an obstacle pattern (Lines 13−14). It is given by $\mathcal{O}(|S_{M^u}|.n_1.n_2)$ where $|S_{M^u}|$ is the size of the MDPU state space, n_1 and n_2 are constants representing the number of 2D points in the perceived obstacle pattern and in the obstacle patterns representing states in S_{M^u}. The time complexity of Algorithm 2 depends on the times taken by *-explore* or *exploit* which is probabilistically selected by M^u's discovery function $D(.)$. For *explore*, a sampling based robot motion planner is used. Such planners have time complexity given by $\mathcal{O}(N^2)$ where N is the sample capacity or the number of vertices in the sample. For *exploit*, the time complexity depends on the selection of the *explore* action and the other options available at the state s. A specific option o inside *exploit* is selected by the policy $\pi^u(s, o)$. If in case the option selected corresponds to *explore*, it can lead to two situations. If *explore* is correctly selected, then the time complexity, as discussed before, is $O(N^2)$ whereas if it is wrongly selected, π^u is used again to select a different option $o \neq a_0$. Hence, the total expected time of execution of Algorithm 2 can be given by $D(.)\mathcal{O}(N^2) + (1 - D(.)\{\pi^u(s, a_0)(\mathcal{O}(N^2) + \mathcal{O}(|O|)) + (1 - \pi^u(s, a_0))(\mathcal{O}(|O|))\})$ where $|O|$ is the total number of options in MDPU state s. As the discovery function $D(.)$ is proportional to the size of the collision library $\mathcal{L}_{coll}$, the above expression can be rewritten as $|\mathcal{L}_{coll}|\mathcal{O}(N^2) + (1 - |\mathcal{L}_{coll}|)\{\pi^u(s, o)(\mathcal{O}(N^2) + \mathcal{O}(|O|)) + (1 - \pi^u(s, o))(\mathcal{O}(|O|))\})$. Considering the policy $\pi^u(s, o)$ to be constant, the resultant time complexity for selecting an option is given by $\mathcal{O}(|\mathcal{L}_{coll}|.|O|)$. The time required by *optimizeOption* which is performed after the robot reaches the *free* state is $\mathcal{O}(|\mathcal{L}_{coll}|.|A_{prim}|)$ where $|A_{prim}|$ is a finite size of the primitive actions in that state. The transfer function τ and *SelectOption* module are called as many times as the selected option gets obstructed which is equal to the size of the collision library $\mathcal{L}_{coll}$. Hence, the total time required by SMDPU-T is $\mathcal{O}(|\mathcal{L}_{coll}|.|S_{M^u}|) + \mathcal{O}(|\mathcal{L}_{coll}|^2.|O|) + \mathcal{O}(|\mathcal{L}_{coll}|.|A_{prim}|) = \mathcal{O}(|\mathcal{L}_{coll}|^2.|O|)$ (ignoring lower order terms). □

Lemma 2 *The number of switches between different options performed by SMDPU-T to reach the goal by avoiding a given obstacle is guaranteed to be within a bound of $\frac{peri}{\sum_{i=1}^{s_{range}/s_{res}} \sqrt{s_d^2 + s_d'^2 - 2s_d * s_d' cos(s_{res})}}$ where peri is the perimeter of the obstacle boundary, s_{range} is the range of view of the robot's sensor, s_{res} is the sensor resolution, s_d and s_d' are respectively the distances returned by the sensor readings corresponding to obstacle detections at consecutive sensor angle resolutions.*

Proof Let us consider that obs_r is the ratio of the obstacle that the robot initially detects. Once the obstacle is detected, the SMDPU-T algorithm determines a suitable option o to be followed by the robot. The robot follows o until the option is terminated as the robot detects a previously unseen part of the obstacle. Let us consider that the new ratio of the obstacle detected by the robot is given by obs_r'. Hence, the part of the obstacle left to be detected by the robot is $1 - (obs_r + obs_r')$. In this situation the robot determines a new option o' and starts following it. This process continues until the robot avoids the entire obstacle i.e. $\sum_i^n obs_{r_i} = 1$ and eventually reaches the goal.

In the worst case scenario, the robot 's option will be terminated after each new part of the obstacle is detected by the robot. The number of times the robot detects a previously unseen part of the obstacle depends directly on the obstacle perimeter $peri$ and inversely on the range of the onboard sensors. Let us consider that the robot's onboard sensor has a range of s_{range} and angular resolution of s_{red} and two consecutive distances returned by the sensor reading when part of an obstacle is detected are s_d and s'_d. Let the two coordinates where the sensor detects the obstacle be (x, y) and (x', y') and the corresponding sensor angles by θ and θ'. Adopting the formulation from polar to rectangular conversion, (x, y) and (x', y') can be expressed as $(s_d cos\theta, s_d sin\theta)$ and $(s'_d cos\theta', s'_d sin\theta')$. The distance between these two consecutive coordinates can be expressed as $\delta d = \sqrt{(s_d cos\theta - s'_d cos\theta')^2 + (s_d sin\theta - s'_d sin\theta')^2}$. After expansion and simplification, the above distance formulation reduces to $\delta d = \sqrt{(}s_d^2 + s_d'^2 - 2s_d s'_d cos(\theta - \theta'))$. As the angular difference between two consecutive sensor readings should be equivalent to the angular resolution of the sensor s_{res}, hence, the above expression can be rewritten as $d = \sqrt{s_d^2 + s_d'^2 - 2s_d s'_d cos(s_{res})}$.

The total length of the obstacle detected by the sensor in each run d can be deduced as the summation of the ratio between the angular range and the angular resolution of the sensor s_{range}/s_{res} i.e. $d = \sum_{i=1}^{s_{range}/s_{res}} \delta d = \sum_{i=1}^{s_{range}/s_{res}} \sqrt{s_d^2 + s_d'^2 - 2s_d s'_d cos(s_{res})}$. In the worst case scenario, the number of switches will be equivalent to the ratio between the length of the obstacle boundary $peri$ and the length of the obstacle detected by the sensor in one run d. Hence, the number of switches between different option is guaranteed to be within a bound of $\frac{peri}{\sum_{i=1}^{s_{range}/s_{res}} \sqrt{s_d^2+s_d'^2-2s_d * s'_d cos(s_{res})}}$. □

We would like to mention here that the number of switches between options in case of SMDPU-T and the number of replanning for the InformedRRT* algorithm will both be equivalent and will depend on the length of the obstacle under consideration and the sensor parameters as shown in Lemma 2. The main advantage of SMDPU-T over InformedRRT* can be derived from the relative time complexity between the two algorithms. A sampling-based algorithm encounters a high overhead for replanning an initially selected path (order of N^2 where N is in the order of 1000s for large environments) when the robot encounters an initially unseen part of the obstacle. On the other hand, the time taken by SMDPU-T to perform replanning is governed only by two parameters- the sizes of the collision library and the option space of MDPU $|O|$. The collision library has the worst case bound equivalent to the maximum number of switches as derived in Lemma 2 and the size of the option space $|O|$ is much lower (in the order of 10s) compared to the number of samples or vertices of InformedRRT* and most importantly is independent of the environment dimensionalities. The performance of SMDPU-T will converge to that of InformedRRT* if value of one of the discussed parameters reaches the sample complexity of InformedRRT*.

5 Experimental setup and results

5.1 Simulated experiments

We have verified the performance of the SMDPU-T algorithm using simulated Corobot robots in the Webots simulator. The Corobot robot is a four-wheeled differential drive robot equipped with a laser sensor for detecting obstacles. A GPS and a compass node was used on the simulated robot to emulate the behavior of the indoor localization device that is available on the hardware robot. A photograph of the simulated and physical Corobot robot is shown in Fig. 2(a)-(b). Simulations were run on a computer with four, 3.20 GHz cores and 12 GB RAM; algorithms were implemented in C++.

Our test environments (shown in Fig. 3 (a)-(h)) are 22×22 m^2 in size and have different types of obstacles varying in geometry, scale and location. The selection of obstacle patterns in the test environments is inspired from the geometric patterns of obstacles commonly observed in indoor environments like buildings, warehouses, and offices. Although there can be a huge variety of obstacles across different environments, we believe that these obstacle geometries are most commonly observed in various forms in indoor environments. Hence, it is imperative for the robot to learn to navigate efficiently around these obstacle geometries. We would also like to mention that our environments mainly consist of obstacle patterns covering the central part of it. This is because in our current work, the main objective for the robot is to navigate across the obstacle pattern centered in the environment. Given the complexity of the obstacle patterns, we have selected the external part of the environment to be free. However, this can be extended to scattered obstacle patterns in future. Moreover, we believe presence of more obstacles scattered in the environment will only contribute towards additive components in the time complexity of our algorithm. This has already been accounted for in the time complexity calculation in Theorem 2.

Different test cases were created by selecting different start-goal pairs in each of the environments. It was also ensured that the straight line path connecting the robot's start to the goal location intersects one or more previously un-encountered obstacles. Consequently, the robot has to use SMDPU-T algorithm to navigate around the obstacle when it encounters the obstacle while navigating between start and goal. To compare our algorithm, we have used a recent version of the widely used motion planning algorithm RRT* called Informed RRT* (Gammell et al. 2014). Informed RRT* is a sampling-based motion planner that

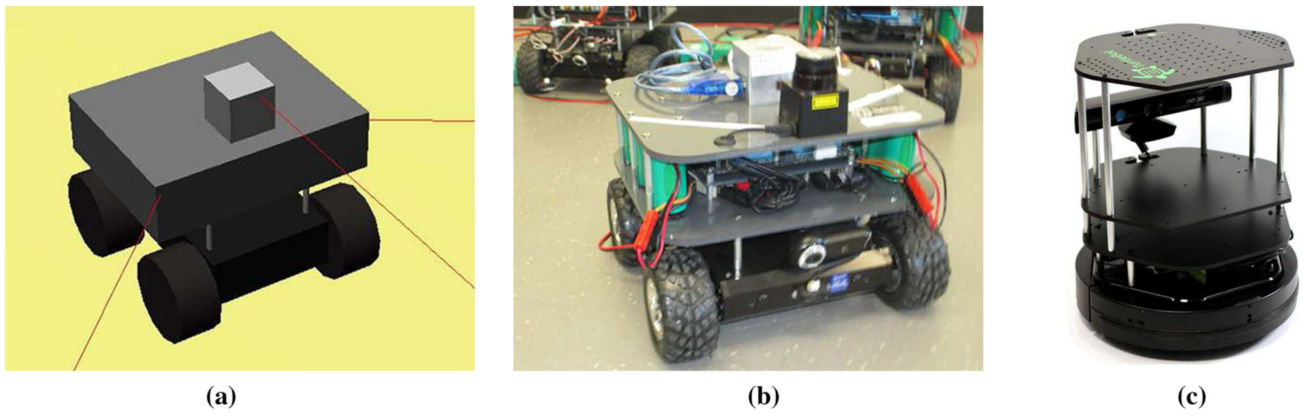


Fig. 2 **a** Simulated corobot, **b** hardware corobot and **c** hardware turtlebot

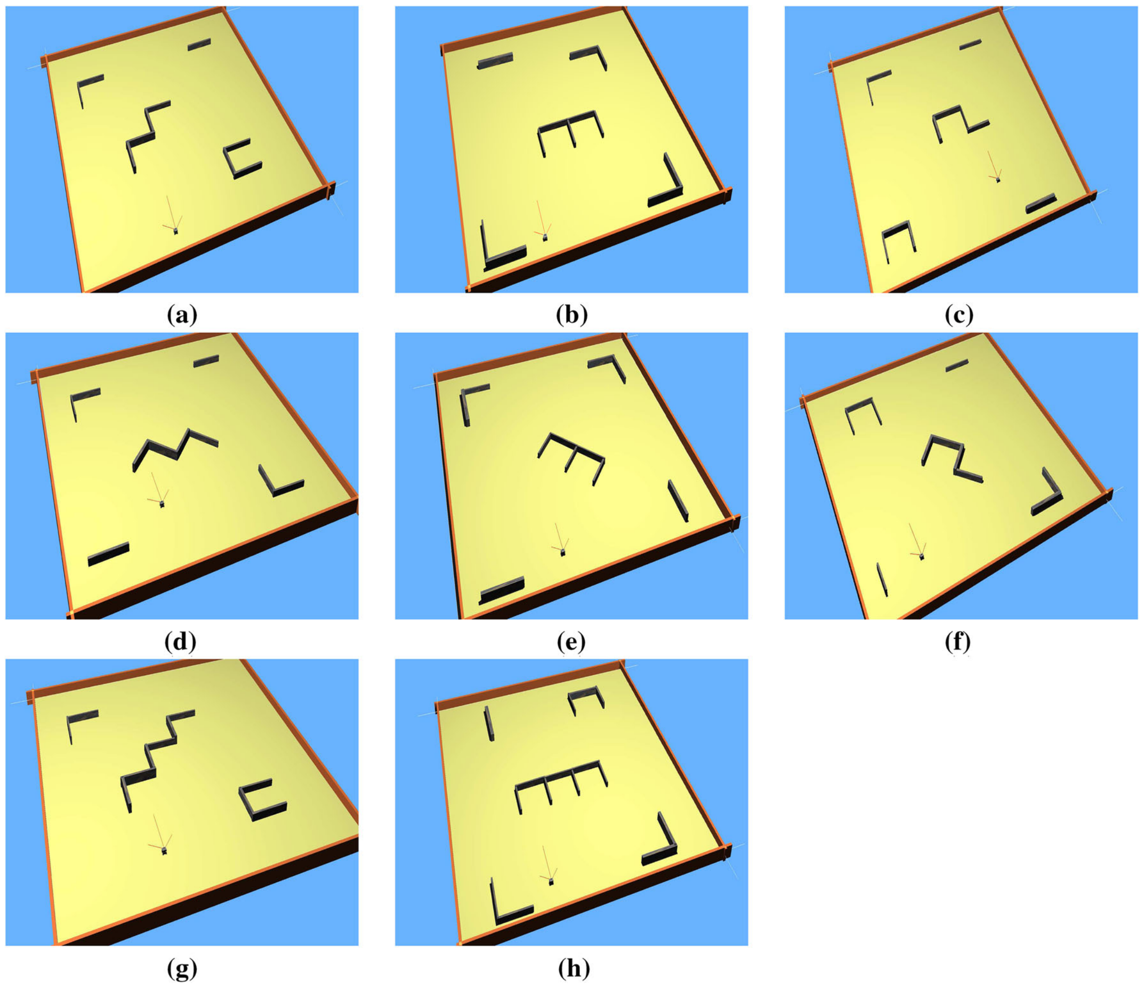


Fig. 3 **a**–**h** Different simulated environments used for testing proposed SMDPU-T algorithm

has been proved to generate shorter paths faster compared to conventional RRT*. In contrast to other path planning algorithms like repeated A* and D*, InformedRRT* and RRT* in general are more memory efficient as they do not require to maintain separate node lists for replanning paths for unknown environments. This also makes RRT* based algorithms suitable to be applied to large scale environments and path planning in 3D and higher dimensional spaces.

SMDPU-T algorithm performs path planning and navigation in a *one-shot* manner without access to any *a priori* map of the environment. Most learning-based motion planning algorithms tend to learn an optimal policy by allowing the robot to solve the navigation tasks repeatedly in the same environment or need considerable amount of robot motion data for training(Fachantidis et al. 2011; Fernández et al. 2010; Lillicrap et al. 2016). In contrast, we consider the problem where the robot is allowed to solve the navigation problem in only a single trial with no resets in the environment, without access to an initial map, and, transfer this knowledge to a new target environment which has some similarities, but might not be identical with the prior environments. To the best of our knowledge, most of the existing learning-based algorithms relax one or more of these assumptions. Hence, we selected Informed RRT* as our baseline comparison algorithm instead of a learning-based robot path planning algorithm. We would also like to mention here that SMDPU-T utilizes Informed RRT* as the underlying planner if for some reason the selected option fails in determining a collision-free path around an obstacle. Hence, we believe using Informed RRT* as a baseline gives offers a clear understanding of the robot's performance with and without our proposed algorithm.

A total of 11 test cases distributed across 8 environments were used for the experiments. To account for uncertainty/noise in the robot's sensing and motion, each test case was performed for 5 runs making a total of 55 runs for each of the algorithms. For our experiments, the used parameters of SMDP and MDPU are discount factor $\gamma = 0.85$ and learning rate $\alpha = 0.125$. We have adopted these values as reported by Sutton et al. in Sutton et al. (1999). During our experiments we made minor modifications to these values but did not notice any significant change in the robot's performance. For each option in the SMDP, we have initialized the termination condition β as 1 for the last state in the option and 0 for all the other states indicating that initially the option is continued till the last state where it is terminated. We have initialized the Q-values for each option as 0 and the initial MDPU policy π^u is initialized by assigning uniform probability to all the options corresponding to each state in S_{M^u}. The initial reward for each option r_s^o in $\mathcal{L}$ is set to 0 and an option receives a high reward if by following it the robot ends up in $free$ state. In the MDPU, the value returned by $D(.)$ should be a function of the number of available actions at the MDPU state where explore action is attempted (Rong et al. 2016). We have estimated this number of available actions as proportional to the number of times the initial option at that MDPU state got interrupted when the state was first discovered.

We have divided our test cases into two sets according to the extent of the environment's complexity in which the test cases occur. Figure 3a–c illustrates the environments in which our first five test cases were performed. These environments were obtained by changing the obstacle patterns learned during the training ($S_{0_{M^u}}$), thus making the test environments more complicated compared to the training environments. Figure 3d–h shows the environments in which our next six test cases were performed. These environments were created either by constructing more complicated obstacle patterns or by positioning obstacles at random orientations compared to the obstacle patterns learned in the environments of Fig. 3a–c. Figure 4 shows the comparative results from our experiments.

We have reported three performance metrics for our algorithm- the planning (decision-making) time to calculate the path between start and goal, the total time (planning + navigation) and the distance to reach from start to goal. Our results in Fig. 4 show that the average planning time and total time for SMDPU-T is lower in comparison to Informed RRT*. From our results, we found that SMDPU-T on average takes only 24% of the planning time and 39% of the total time to complete the same navigation tasks as Informed RRT*. As can be observed from the graphs, our proposed approach takes planning time in the order of minutes. Hence, we would like to mention here that it is not 'real-time' in the true sense of the term. However, given that the robot has no access to any a priori map of its environment, it takes much less time compared to the other conventional learning-based algorithms. This is a direct consequence of the fact that unlike general learning-based algorithms, our algorithm takes minimal time for offline training. In addition to this, it also outperforms InformedRRT* which is a non-learning based path planner due to its capability to leverage navigational knowledge from previous experiences.
When we compared the average distances covered by both the algorithms, we found that Informed RRT* does take lesser distance compared to SMDPU-T. However, this distance reduction is achieved at the cost of a much higher total time. When compared we found that SMDPU-T on average takes only 8% more distance to complete the same tasks as Informed RRT*. As our objective was to develop a fast path planning technique for robots with satisfactory path quality, we used Informed RRT* for our comparisons instead of RRT* which achieves faster path planning with much poorer path quality.

To further analyze the performance of SMDPU-T, we have compared the navigational performance of SMDPU-T with

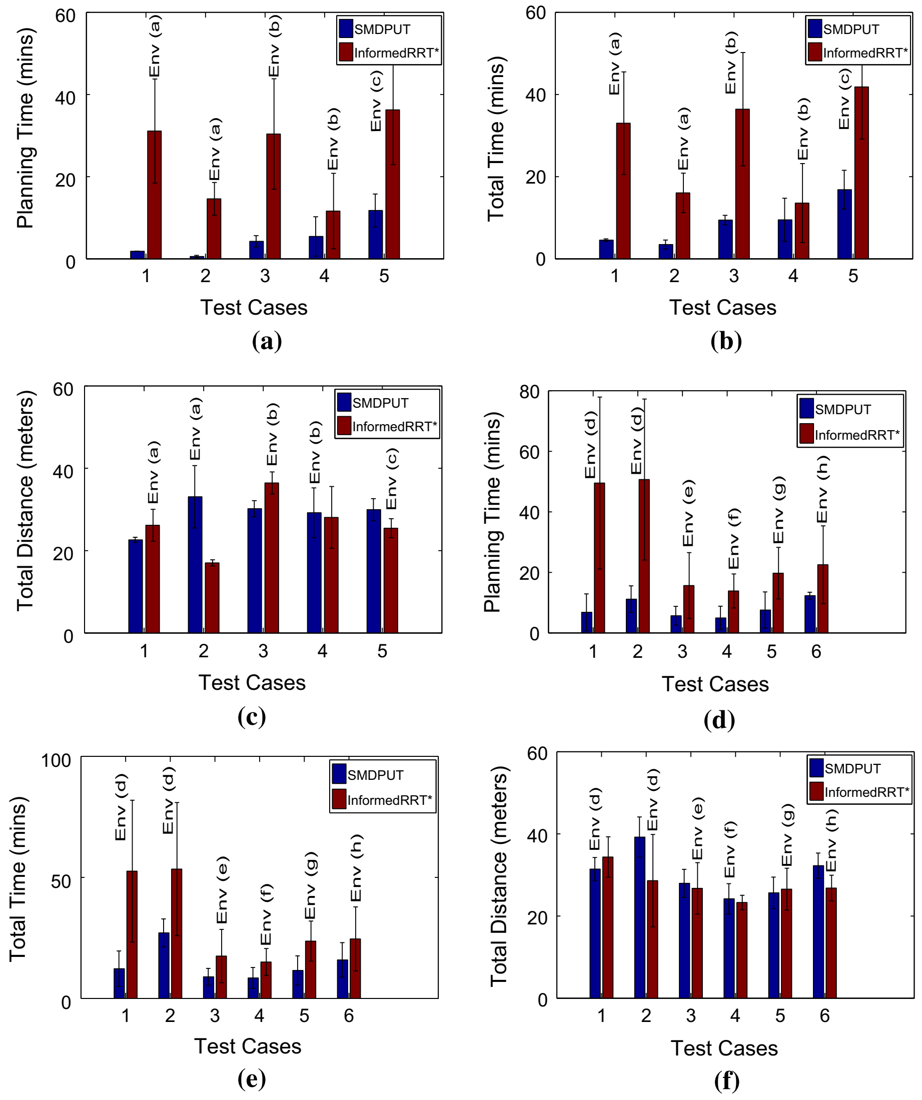


Fig. 4 **a** Planning Time, **b** total time, **c** total distance comparisons between SMDPU-T and Informed RRT* in environments with one step modification over training environments. **d** Planning time, **e** total time, **f** total distance comparisons between SMDPU-T and Informed RRT* in environments with two step modifications over training environments

baseline RRT* algorithm. Figure 5 illustrates the comparative performance of the two algorithms in the simulated environments shown in Fig. 3. From the figures it can be observed that in majority of the test cases, the baseline RRT* algorithm requires much lesser planning time and total time to reach the specific goals in the given environments. However, in terms of total distance covered by the two algorithms, in 10 out of 11 test cases, baseline RRT* covers much higher total distance compared to SMDPU-T. We separately analyzed the total distance covered in environments with one step modification (Fig. 3a–c) and two step modifications (Fig. 3d–h) and found that SMDPU-T on average requires 24% and 25% lesser distance respectively to reach the goal compared to baseline RRT*. The results obtained were intuitive. As baseline RRT* does not guarantee optimality, it can predict paths to the goal in a much lower amount of time compared to SMDPU-T which predicts near-optimal paths with respect to the local geometry. Consequently, the paths generated by baseline RRT* requires higher distance to reach the goal compared to SMDPU-T. In general, a high extent of variance can be observed across all the navigational metrics for RRT* as the algorithm contains very high level of randomness (Fig. 6).

We analyzed the quality of the newly learned options on discovery of a new MDPU state. Figure 3i–j illustrate the perception data as captured by the robot representing the obstacle contour and the initial option learned and the option after optimization in environments (a) and (b) in Fig. 3. The blue bold line in the graph represents the obstacle boundary perceived by the robot's laser, the red dotted line represents the robot's initial option while the green solid line represents the optimized option constructed by SMDPU-T. From the figures it can be observed that option optimization improves the initial option by eliminating states which are too close to the obstacle and including states away from the perceived obstacle in addition to reducing the overall length of the opti-

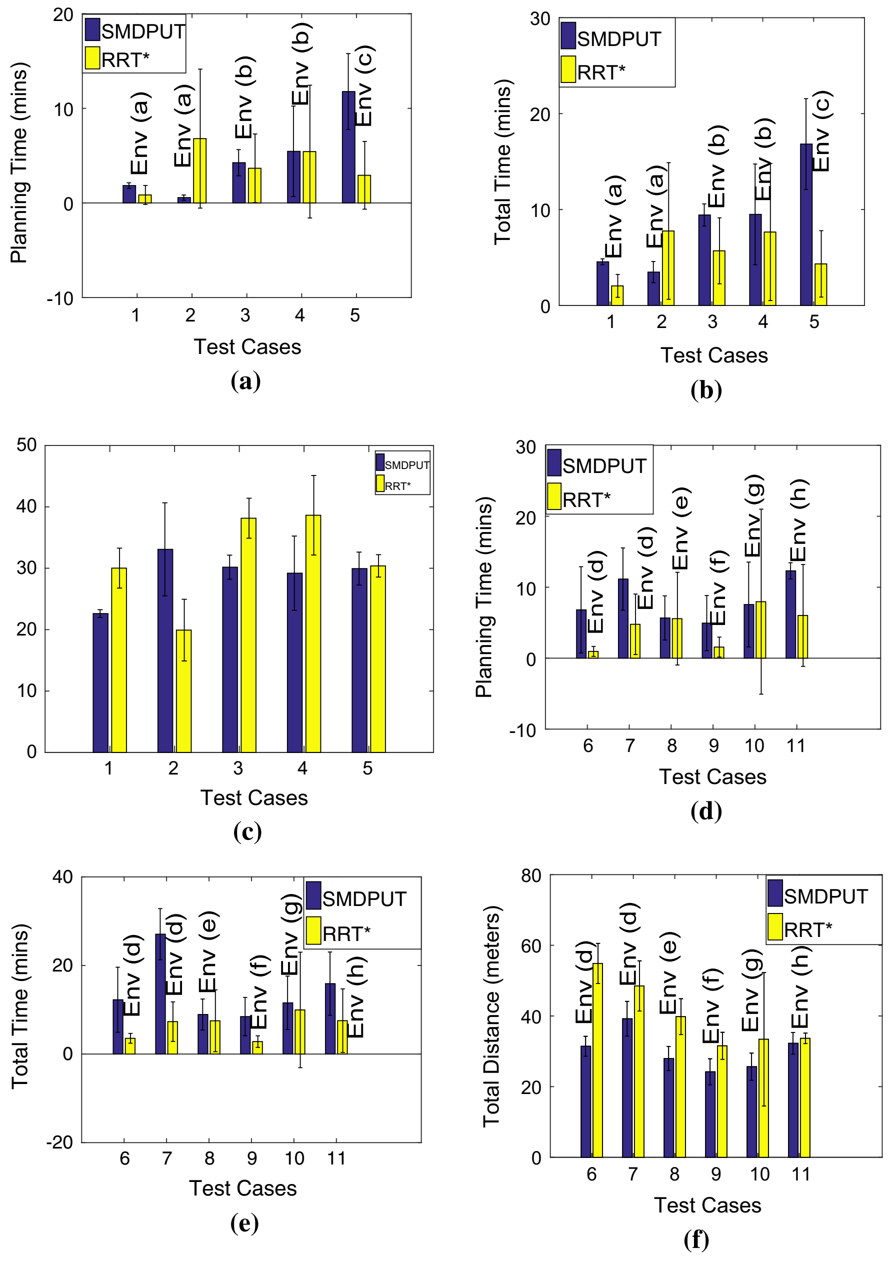


Fig. 5 **a** Planning time, **b** total time, **c** total distance comparisons between SMDPU-T and RRT* in environments with one step modification over training environments. **d** Planning time, **e** total time, **f** total distance comparisons between SMDPU-T and RRT* in environments with two step modifications over training environments

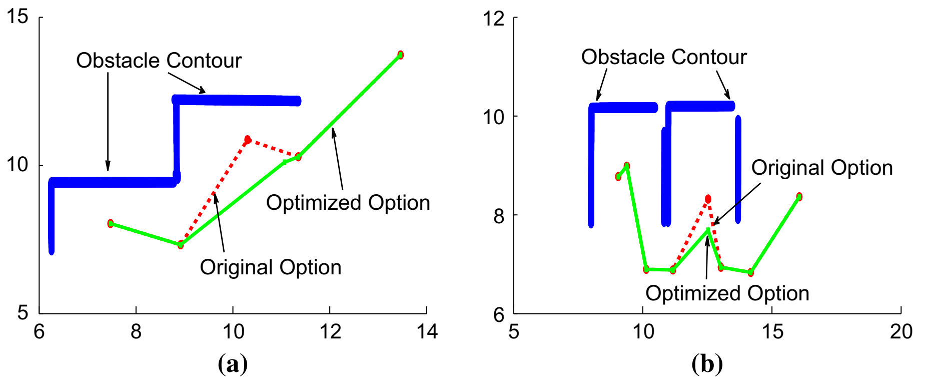


Fig. 6 **a**–**b** Perception data (blue bolded lines) captured by the robot for environments in Fig. 3a and b and corresponding options constructed by SMDPU-T algorithm. Red dotted lines show original option while green solid lines show path-optimized (shorter path length) options (Color figure online)

Table 2 Initial policy with uniform probability distribution

	Opt.1 prob	Opt. 2 prob	Opt. 3 prob	Opt. 4 prob	Opt. 5 prob
Initial policy					
Obs.1	0.25	0.25	0.25	0.25	–
Obs.2	0.2	0.2	0.2	0.2	0.2
Obs.3	0.5	0.5	–	–	–
Obs.4	0.2	0.2	0.2	0.2	0.2

The values in each cell indicate the initial probability with which the option specified in the corresponding column for the corresponding MDPU state as specified in the row, will be selected

Table 3 Final policy with biased probabilities based on navigational experiences

	Opt.1 prob	Opt.2 prob	Opt.3 prob	Opt.4 prob	Opt.5 prob	Opt.6 prob	Opt.7 prob	Opt.8 prob
Final policy								
Obs.1	0.06	0.06	0.06	0.79	–	–	–	–
Obs.2	0.02	0.02	0.02	0.02	0.02	0.88	0.02	0.02
Obs.3	0.13	0.86	–	–	–	–	–	–
Obs.4	0.03	0	0	0	0	0.26	0.70	0
Obs.5	0.03	0.93	0.03	–	–	–	–	–
Obs.6	0.06	0.94	–	–	–	–	–	–
Obs.7	0.09	0.90	–	–	–	–	–	–
Obs.8	0.22	0.77	–	–	–	–	–	–
Obs.9	0.30	0.69	–	–	–	–	–	–

The values in each cell indicate the updated probability with which the option specified in the corresponding column for the corresponding MDPU state as specified in the row, will be selected

mized option. We also assessed the learning performance of SMDPU-T while performing the different navigation tasks. Figure 7 illustrates the evolution of the average expected rewards for the options over time as the robot solves navigation tasks sequentially. As can be observed from the results, the robot achieves higher average expected rewards as it learns newer options corresponding to newly discovered state(s). At the end of the training phase, the underlying MDPU had only 5 states and 16 associated options but by the end of our experiments, the MDPU evolved to contain 10 states and 33 associated options. Along the overall process, the initial policy got updated to reflect the modification of the Q-values as well as the addition of new states and options. Tables 2 and 3 illustrate respectively the initial MDPU policy at the beginning of our experiments and the evolved policy achieved at the end. Each row in Tables 2 and 3 represents a state and each column represents the probability of selection of the specific option for that state. From the tables it can be observed that the initial policy uses a uniform distribution and assigns equal preference to all the options for a particular state whereas the final policy assigns higher preference to specific options which were learned to be 'good' options (higher Q-values) during the robot's navigation in different environments. Table 4 shows the initial and final values for the termination condition β for the obstacle pattern in Fig. 3(b). Each row in Table 4 represents an option and each column represents an SMDP state associated with that option. At the beginning of our experiments, the termination probability reflects that the options should terminate in their last state i.e. when the option execution is complete, which gradually changes as the options get interrupted with the robot's navigation experience. Hence, the table at the bottom shows the increase in values for the intermediary SMDP states indicating the increase of probability for the termination of the option when the robot encounters those states.

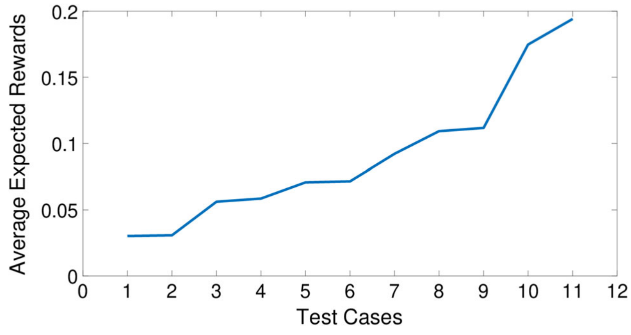


Fig. 7 Average expected rewards obtained by the robot over time for the different test cases

Table 4 Evolution of termination probability β over time for test case 3 performed in environment shown in Fig. 3b

Initial termination probabilities				
Obs.1	0	0	0	1
Obs.2	0	0	0	1
Obs.3	0	0	0	1
Obs.4	0	0	0	1
Obs.5	0	0	0	1

Final termination probabilities													
Obs.1	0	0.15	0	0.85	–	–	–	–	–	–	–	–	–
Obs.2	0	0.15	0	0.85	–	–	–	–	–	–	–	–	–
Obs.3	0	0	0	1	–	–	–	–	–	–	–	–	–
Obs.4	0	0	0	1	–	–	–	–	–	–	–	–	–
Obs.5	0	0	0	1	–	–	–	–	–	–	–	–	–
Obs.6	0.15	0	0	0.18	0.66	–	–	–	–	–	–	–	–
Obs.7	0	0	0	0	0	0	0	0	1	–	–	–	–
Obs.8	0	0	0	0	0	0	0	0	0	0	0	0	1

5.2 Hardware experiments

We have performed extensive hardware experiments to validate the performance of our proposed algorithm SMDPU-T on hardware robot platform. We have used Turtlebot $2e$ robot for our hardware experiments. Turtlebot $2e$ is a popular, low-cost robot platform with a Kobuki base, an Asus Xtion Pro live as the $3D$ kinect sensor and an on-board laptop with 4 GiB RAM and Intel Core $i3$ processor with processing speed of 1.70 GHz that runs the Robot Operating System (ROS). Figure 2(c) illustrates an image of the turtlebot robot. As GPS is not available in indoor environments, we have used the Vicon motion tracking system for localizing the Turtlebot in the environment. The Vicon motion tracking system captures the motion of mobile objects and people using vision data from overhead cameras installed uniformly around the experimental arena. The hardware implementation of the algorithm was done in C++ using ROS hydro. The real-world test environments used in the hardware experiments are illustrated in Fig. 8. The environments had a dimension of $3 \times 2\ m^2$ and consisted of obstacles with different geometries and orientation. Similar to our simulated experiments, we selected distinct start and goal locations for the robot in each environment ensuring that the straight line path from the start to the goal will be obstructed multiple times by the respective obstacle pattern in each environment. It was also ensured that the start and goal locations are located diametrically opposite to each other thus maximizing the distance between them. As before, we compared the performance of SMDPU-T with Informed RRT*. We used a total of 5 test cases across 5 environments. In order to account for the randomness present in both SMDPU-T and Informed RRT*, each test case was evaluated across 10 runs (50 runs in total) for both the algorithms and the mean performance was recorded. We compared the performance of SMDPU-T and Informed RRT* in terms of three standard navigation metrics- planning time taken by the robot to plan (or replan) the path to the goal, total time taken by the robot to reach the goal from its initial location and total distance traversed by the robot to reach its goal. In addition to this, we also noted the learning performance of SMDPU-T in terms of the expected reward and expected Q-value achieved by SMDPU-T with its increasing knowledge by solving the navigational tasks in the test environments shown in Fig. 8.

Figure 9a–c illustrate the comparative navigational performance of SMDPU-T with InformedRRT* where each of the algorithms solved the same navigational tasks in the respective test environments. As can be observed from Fig. 9a, SMDPU-T takes lower planning time compared to InformedRRT* in majority (4 out of 5) of the test cases indicating that SMDPU-T takes lesser time to plan the paths around the obstacle patterns in the respective environment. From our analysis, we found that SMDPU-T on average requires only 53% of planning time compared to InformedRRT* to solve the same tasks. Figure 9(b) illustrates the comparative difference between SMDPU-T and InformedRRT* with respect to the total time taken to solve the navigation task. It can be observed from the figure that SMDPU-T requires lower total time to solve 4 out of 5 test cases, thus, emphasizing the overall time effectiveness of SMDPU-T over InformedRRT*. Our analysis indicates that, on average, SMDPU-T takes only 60% of the total time compared to InformedRRT* to complete identical navigation tasks. The first test case in Fig. 9b requires higher total time compared to InformedRRT*. This was caused mainly due to some inaccuracies in scaling and the robot's sensor not capturing considerable portion of the surrounding obstacles due to the robot's current orientation. For assessing the scale of the perceived obstacle, our algorithm uses a basic version of the Principal Component Analysis (PCA) algorithm which results in some errors in the estimation of the scaling factor of the perceived obstacle with respect to the obstacle(s) in the robot's library. The scaling error can be reduced in future by using advanced methods for determining scaling factors. Figure 9c demonstrates the total distance covered by the two algorithms to reach the goal in the respective test environment. It can be observed from the figure that SMDPU-T travels comparable or lower distance to reach the goal in 3 out of 5 test cases. From our analysis, we found that on average, SMDPU-T covers approximately 1% higher distance compared to SMDPU-T. However, it is important to mention that this slight increase in distance traveled is negligible when compared to the high time benefits that SMDPU-T offers over InformedRRT*. Figure 10a–b illustrate the learning performance of SMDPU-T

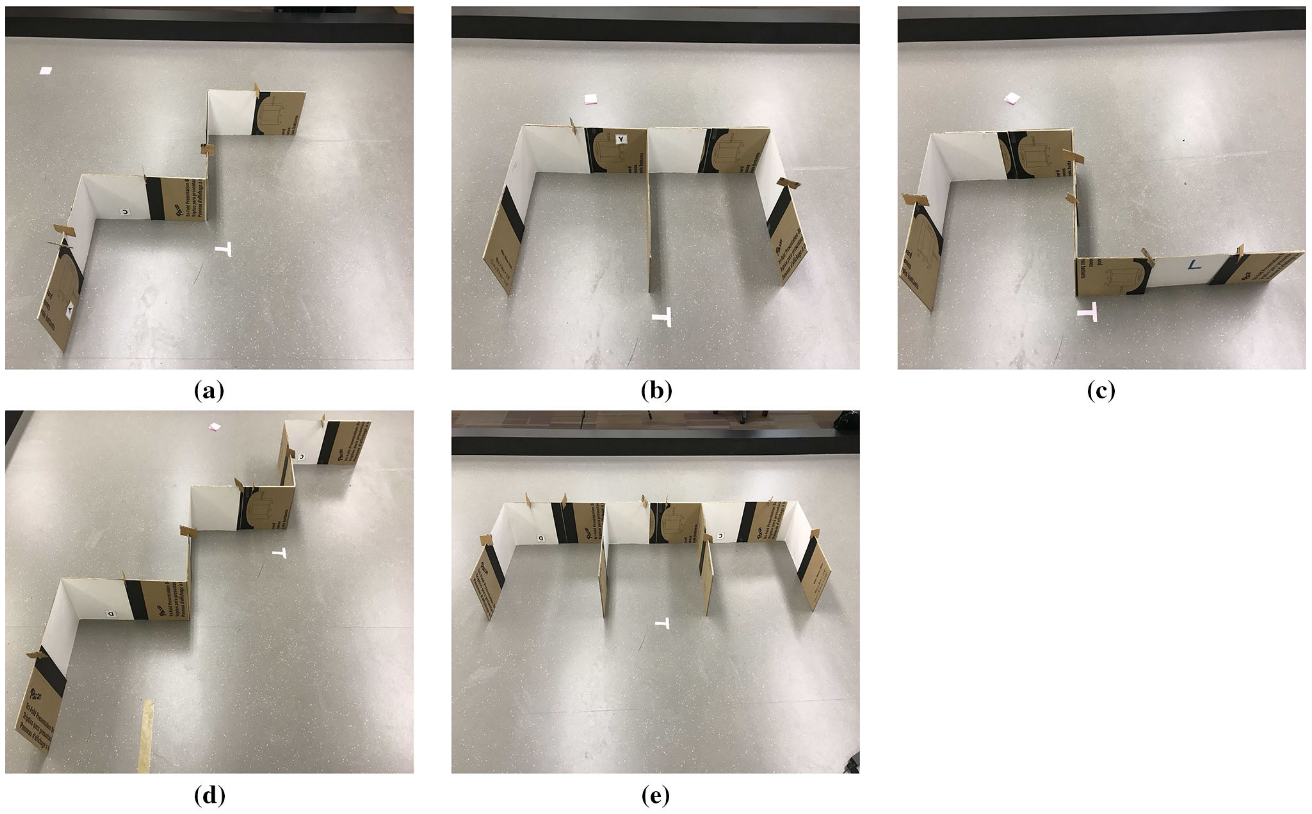


Fig. 8 **a**–**h** Different real-world environments used for testing proposed SMDPU-T algorithm

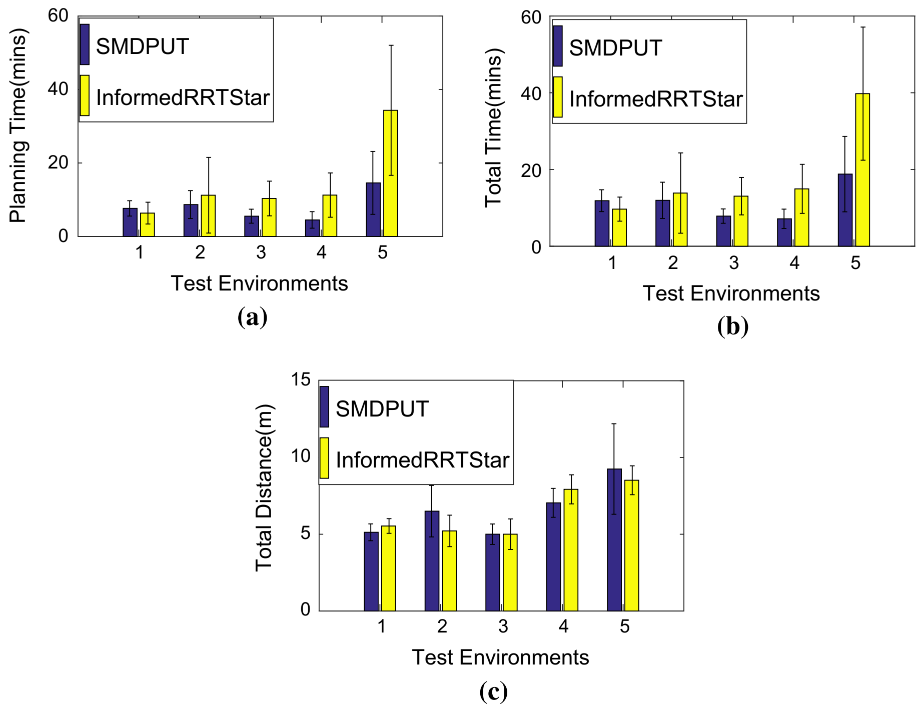


Fig. 9 **a** Planning Time, **b** total time and **c** total distance traveled by SMDPU-T in hardware environments shown in Figure 8

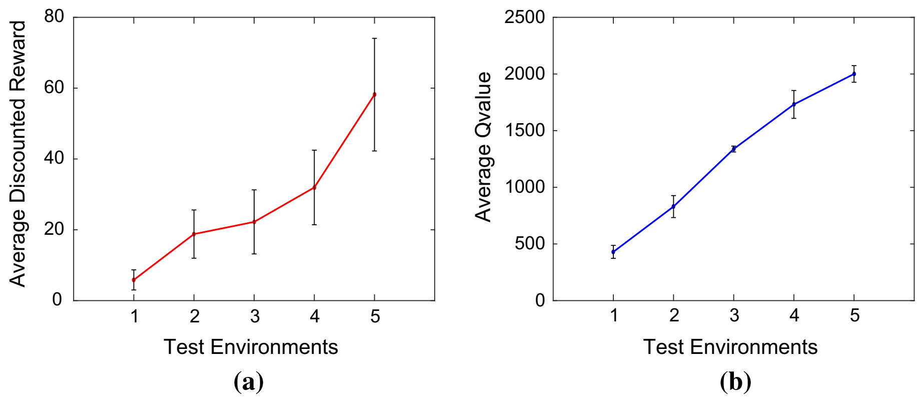


Fig. 10 **a** Average discounted reward and **b** Average Q-value achieved by SMDPU-T in hardware environments shown in Figure 8

in terms of the average rewards and average Q-value achieved by the algorithm with time. It can be observed from the figures that SMDPU-T consistently achieves higher rewards and effectively higher Q-values with each test case indicating the learning benefits of the algorithm. From our analysis, we found that SMDPU-T, on average, achieves approximately 10 times of its initial reward and 4.6 times of its initial Q-value at the end of our experiments implying its improved performance with learning of new obstacle patterns and corresponding options to circumnavigate the learned obstacles.

Although most of our experiments were conducted in environments consisting of single obstacles, our proposed algorithm can be easily applicable to environments with multiple scattered obstacles. In such scenarios, once the robot ends up in free space after following the SMDPU-T algorithm, it can start traveling towards the goal after appropriately orienting itself until it detects another obstacle. As soon as the robot detects an obstacle, it can invoke the SMDPU-T algorithm which predicts a path for the robot to follow and efficiently navigate around the obstacle. If the robot arrives at an obstacle-free region after following the predicted path, it can again resume its motion towards the goal. This robot can iteratively continue this process until it reaches the goal. Similarly, the algorithm can also be extended to environments with large obstacles as has been demonstrated in some of our experiments. The robot can only perceive part of an obstacle, plans the path according to the partial obstacle information and starts following it. The robot replans its path when it perceives the other portion(s) of the obstacle while following the previously planned path. Hence, the robot continues this interleaved planning, replanning and following the planned path until it reaches a *free* region from where it travels directly towards the goal. We would like to mention that in environments with multiple large and scattered obstacles, the robot will require higher time to achieve the navigational task due to the complexity of the environment.

6 Conclusions, limitations and future directions

In this work, we proposed an environment independent learning algorithm utilizing which robots can learn to autonomously navigate around obstacle patterns. Our main contribution was to use the concept of time-extended actions called option to represent the robot's motion plan. The robots can reuse and dynamically update the learned options when they incrementally discover more complicated obstacles while solving future navigation tasks in different environments. We considered that the robot starts learning with partial knowledge about its states and actions and gradually updates its knowledge. Our results validate that robots can significantly reduce their total time for solving a navigation task as they learn more options with time. The proposed technique can be used across spatially dispersed robots operating in different environments across geographically dispersed locations and at different times.

Our work made some assumptions which are described below. Although the environments in which the robot navigates are different, there should be some inherent obstacle patterns which are universally present in majority of the environments. If the robot's navigation environments do not share any common features, the benefits of our proposed algorithm will not be leveraged to improve the navigation capability of the robot and our algorithm would perform similar to learning from scratch. However, in reality, most indoor environments possess variations and repetitions of common obstacle patterns and environmental structures like corners, passages and cubicles.

One of the limitations of our work is that as the robot continues to navigate in various environments and learn different obstacle patterns encountered, the number of obstacle patterns as well as the corresponding options increase. This results in the increase in obstacle recognition and classification times, thus affecting the robot's real-time navigation performance. In future, this issue related to the exponential growth of options needs to be resolved for the proposed approach to be scalable across large number of environments. Another limitation of our work is in the option optimization phase after the robot learns a new option corresponding to a newly learned obstacle pattern. Our option optimization technique uses a discrete action space to improve the initial option which does not always result in near-optimal options. In contrast, a continuous action space would result in the generation of highly improved options as it would consider a much wider action range. At this point, our work only deals with static obstacles and has to be extended for scenarios involving dynamic obstacles. Finally, the proposed algorithm has only been tested in a single robot platform. A more complex transfer function might be required for knowledge transfer across multiple heterogeneous robot platforms.

This work can have multiple future directions of research that arise from the limitations discussed above. For the robot to maintain its real-time performance in presence of large amounts of obstacle information, it has to determine the significant obstacle patterns from its collected sensor data of obstacle boundaries. As a future direction, a clustering procedure could be utilized as a preprocessing step to detect the more important obstacle patterns which could then be combined with the underlying MDPU in our algorithm. This would prevent the uncontrolled expansion of the MDPU state space and ensure the real-time processing capabilities of the proposed robot navigation system. Another future direction could involve the incorporation of a deep reinforcement learning (deep RL) framework to improve the post-optimization process for a newly learned option. The deep RL technique can help in determining a near-optimal option in a continuous action space. Finally, as a future direction, this work can also be extended to involve multiple heterogeneous robots interacting with each other to solve the navigation tasks simultaneously by sharing their knowledge among themselves via a common cloud infrastructure. Using multiple interacting robots will offer the benefits of higher diversity in experiences which can lead to improved navigation performance. In addition to serving as a platform to share experiences, the incorporation of the cloud infrastructure can enable the robots to significantly reduce their computations by offloading high-level, time-intensive computations to the cloud like point cloud registration, pattern matching, obstacle recognition and option optimization using deep RL and would improve the real-time performance of the robots. These research directions can be explored in future to address the limitations of our current approach. We believe this can ultimately lead to a fast, reliable, dynamic, environment independent, learning-based navigation
system enabling robots to successfully navigate in initially unknown and geographically dispersed complicated environments.

References

Andreas, J., Klein, D., & Levine, S. (2017). Modular multitask reinforcement learning with policy sketches. In *ICML* (Vol. 70, pp. 166–175). PMLR.

Argall, B., Chernova, S., Veloso, M. M., & Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, *57*(5), 469–483.

Bacon, P., Harb, J., & Precup, D. (2017). The option-critic architecture. In *AAAI* (pp. 1726–1734). Menlo Park: AAAI Press.

Bajracharya, M., Tang, B., Howard, A., Turmon, M. J., & Matthies, L. H. (2008). Learning long-range terrain classification for autonomous navigation. In *ICRA* (pp. 4018–4024). IEEE.

Berenson, D., Abbeel, P., & Goldberg, K. (2012). A robot path planning framework that learns from experience. In *ICRA* (pp. 3671–3678). IEEE.

Bischoff, B., Nguyen-Tuong, D., Lee, I., Streichert, F., & Knoll, A. (2013). Hierarchical reinforcement learning for robot navigation. In *ESANN*.

Boularias, A., Duvallet, F., Oh, J., & Stentz, A. (2016). Learning qualitative spatial relations for robotic navigation. In *IJCAI* (pp. 4130–4134). IJCAI/AAAI Press.

Bowen, C., Ye, G., & Alterovitz, R. (2015). Asymptotically optimal motion planning for learned tasks using time-dependent cost maps. *IEEE Transactions on Automation Science and Engineering*, *12*(1), 171–182.

Bruin, T. de, Kober, J., Tuyls, K., & Babuska, R. (2016). Improved deep reinforcement learning for robotics through distribution-based experience retention. In *IROS* (pp. 3947–3952). IEEE.

Brys, T., Harutyunyan, A., Taylor, M. E., & Nowé, A. (2015). Policy transfer using reward shaping. In *AAMAS* (pp. 181–188). ACM.

Burchfiel, B., Tomasi, C., & Parr, R. (2016). Distance minimization for reward learning from scored trajectories. In *Proceedings of the thirtieth aaai conference on artificial intelligence* (pp. 3330–3336). AAAI Press.

Calinon, S. (2009). *Robot programming by demonstration—a probabilistic approach*. Lausanne: EPFL Press.

Chernova, S., & Veloso, M. M. (2009). Interactive policy learning through confidence-based autonomy. *Journal of Artificial Intelligence Research*, *34*, 1–25.

Choset, H. M. (2005). *Principles of robot motion: Theory, algorithms, and implementation*. Cambridge: MIT press.

Cutler, M., & How, J. P. (2015). Efficient reinforcement learning for robots using informative simulated priors. In *ICRA* (pp. 2605–2612). IEEE.

Deisenroth, M. P., & Rasmussen, C. E. (2011). PILCO: A model-based and data-efficient approach to policy search. In *ICML* (pp. 465–472). Omnipress.

Elbanhawi, M., & Simic, M. (2014). Sampling-based robot motion planning: A review. *IEEE Access*, *2*, 56–77.

Erkan, A., Hadsell, R., Sermanet, P., Ben, J., Muller, U., & LeCun, Y. (2007). Adaptive long range vision in unstructured terrain. In *IROS* (pp. 2421–2426). IEEE.

Fachantidis, A., Partalas, I., Taylor, M. E., & Vlahavas, I. (2015). Transfer learning with probabilistic mapping selection. *Adaptive Behavior*, *23*(1), 3–19.

Fachantidis, A., Partalas, I., Taylor, M. E., & Vlahavas, I. P. (2011). Transfer learning via multiple inter-task mappings. In *EWRL* (Vol. 7188, pp. 225–236). Springer.

Fernández, F., García, J., & Veloso, M. M. (2010). Probabilistic policy reuse for inter-task transfer learning. *Robotics and Autonomous Systems*, *58*(7), 866–871.

Galceran, E., & Carreras, M. (2013). A survey on coverage path planning for robotics. *Robotics and Autonomous Systems*, *61*(12), 1258–1276.

Gammell, J. D., Srinivasa, S. S., & Barfoot, T. D. (2014). Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *IROS* (pp. 2997–3004). IEEE.

Gopalan, N., desJardins, M., Littman, M. L., MacGlashan, J., Squire, S., Tellex, S., et al. (2017). Planning with abstract markov decision processes. In *ICAPS* (pp. 480–488). AAAI Press.

Gu, S., Holly, E., Lillicrap, T. P., & Levine, S. (2017). Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *ICRA* (pp. 3389–3396). IEEE.

Happold, M., Ollis, M., & Johnson, N. (2006). Enhancing supervised terrain classification with predictive unsupervised learning. In G. S. Sukhatme, S. Schaal, W. Burgard, & D. Fox (Eds.), *Robotics: Science and systems*. Cambridge: The MIT Press.

Hester, T., & Stone, P. (2013). TEXPLORE: Real-time sample-efficient reinforcement learning for robots. *Machine Learning*, *90*(3), 385–429.

Hussein, A., Gaber, M. M., Elyan, E., & Jayne, C. (2017). Imitation learning: A survey of learning methods. *ACM Computing Surveys*, *50*(2), 21:1–21:35.

Jeni, L. A., Istenes, Z., Korondi, P., & Hashimoto, H. (2007). Hierarchical reinforcement learning for robot navigation using the intelligent space concept. In *ICIES* (pp. 149–153).

Kim, B., Farahmand, A., Pineau, J., & Precup, D. (2013). Learning from limited demonstrations. In *NIPS* (pp. 2859–2867).

Kim, D., Sun, J., Oh, S. M., Rehg, J. M., & Bobick, A. F. (2006). Traversability classification using unsupervised on-line visual learning for outdoor robot navigation. In *ICRA* (pp. 518–525). IEEE.

Kober, J., Bagnell, J. A., & Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, *32*(11), 1238–1274.

Kober, J., & Peters, J. (2011). Policy search for motor primitives in robotics. *Machine Learning*, *84*(1–2), 171–203.

Kollar, T., & Roy, N. (2008). Trajectory optimization using reinforcement learning for map exploration. *The International Journal of Robotics Research*, *27*(2), 175–196.

Konidaris, G., & Barto, A. G. (2007). Building portable options: Skill transfer in reinforcement learning. In *IJCAI* (pp. 895–900).

Levine, S., Finn, C., Darrell, T., & Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, *17*(1), 1334–1373.

Lien, J., & Lu, Y. (2009). Planning motion in environments with similar obstacles. In J. Trinkle, Y. Matsuoka, & J. A. Castellanos (Eds.), *Robotics: Science and systems*. Cambridge: The MIT Press.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., et al. (2016). Continuous control with deep reinforcement learning. In *ICLR*.

Mahler, J., & Goldberg, K. (2017). Learning deep policies for robot bin picking by simulating robust grasping sequences. In *Corl* (Vol. 78, pp. 515–524). PMLR.

Mahler, J., Liang, J., Niyaz, S., Laskey, M., Doan, R., Liu, X., et al. (2017). Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. In N. M. Amato, S. S. Srinivasa, N. Ayanian & S. Kuindersma (Eds.), *Robotics: Science and systems XIII*. Cambridge, MA: Massachusetts Institute of Technology.

Manschitz, S., Kober, J., Gienger, M., & Peters, J. (2015). Learning movement primitive attractor goals and sequential skills from kinesthetic demonstrations. *Robotics and Autonomous Systems*, *74*, 97–107.

Mendoza, J. P., Veloso, M. M., & Simmons, R. G. (2015). Plan execution monitoring through detection of unmet expectations about action outcomes. In *ICRA* (pp. 3247–3252). IEEE.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., et al. (2013). Playing atari with deep reinforcement learning. arXiv:1312.5602

Montgomery, W., Ajay, A., Finn, C., Abbeel, P., & Levine, S. (2017). Reset-free guided policy search: Efficient deep reinforcement learning with stochastic initial states. In *ICRA* (pp. 3373–3380). IEEE.

Moore, A. W., & Atkeson, C. G. (1993). Prioritized sweeping: Reinforcement learning with less data and less time. *Machine learning*, *13*(1), 103–130.

Nehaniv, C. L., & Dautenhahn, K. (2002). Imitation in animals and artifacts. In K. Dautenhahn & C. L. Nehaniv (Eds.), (pp. 41–61). Cambridge, MA: MIT Press.

Ng, A. Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., et al. (2006). Autonomous inverted helicopter flight via reinforcement learning. In *Experimental robotics ix* (pp. 363–372). Springer.

Paden, B., Cáp, M., Yong, S. Z., Yershov, D. S., & Frazzoli, E. (2016). A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on intelligent vehicles*, *1*(1), 33–55.

Ratliff, N. D., Silver, D., & Bagnell, J. A. (2009). Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots*, *27*(1), 25–53.

Rong, N., Halpern, J. Y., & Saxena, A. (2016). Mdps with unawareness in robotics. In *UAI* (pp. 627–636). AUAI Press.

Russell, S., & Norvig, P. (2009). *Artificial intelligence: A modern approach*. Upper Saddle River: Prentice Hall.

Rusu, R. B., Blodow, N., & Beetz, M. (2009). Fast point feature histograms (FPFH) for 3d registration. In *ICRA* (pp. 3212–3217). IEEE.

Saha, O., & Dasgupta, P. (2017). Experience learning from basic patterns for efficient robot navigation in indoor environments. *Journal of Intelligent and Robotic Systems*, *92*, 545–564.

Saha, O., & Dasgupta, P. (2017). Improved reward estimation for efficient robot navigation using inverse reinforcement learning. In *AHS* (pp. 245–252). IEEE.

Saha, O., & Dasgupta, P. (2017). Real-time robot path planning around complex obstacle patterns through learning and transferring options. In *2017 IEEE international conference on autonomous robot systems and competitions, ICARSC 2017, coimbra, portugal, april 26–28, 2017* (pp. 278–283).

Silver, D., Bagnell, J. A., & Stentz, A. (2010). Learning from demonstration for autonomous navigation in complex unstructured terrain. *The International Journal of Robotics Research*, *29*(12), 1565–1592.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, M. A. (2014). Deterministic policy gradient algorithms. In *ICML* (Vol. 32, pp. 387–395). JMLR Org.

Simsek, Ö., Wolfe, A. P., & Barto, A. G. (2005). Identifying useful subgoals in reinforcement learning by local graph partitioning. In *ICML* (Vol. 119, pp. 816–823). ACM.

Sofman, B., Lin, E., Bagnell, J. A., Cole, J., Vandapel, N., & Stentz, A. (2006). Improving robot navigation through self-supervised online learning. *Journal of Field Robotics*, *23*(11–12), 1059–1075.

Sutton, R. S., Precup, D., & Singh, S. P. (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, *112*(1–2), 181–211.

Takeuchi, J., & Tsujino, H. (2010). One-shot supervised reinforcement learning for multi-targeted tasks: RL-SAS. In *ICANN* (2) (Vol. 6353, pp. 204–209). Springer.

Taylor, M. E., & Stone, P. (2007). Cross-domain transfer for reinforcement learning. In *ICML* (Vol. 227, pp. 879–886). ACM.

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

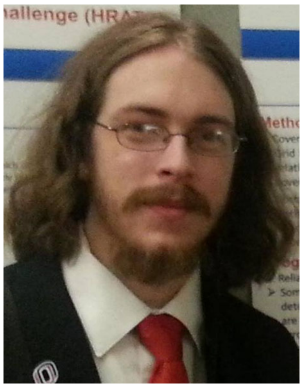
Torrey, L., Walker, T., Shavlik, J., & Maclin, R. (2005). Using advice to transfer knowledge acquired in one reinforcement learning task to another. In *Ecml* (pp. 412–424).

Wulfmeier, M., Rao, D., Wang, D. Z., Ondruska, P., & Posner, I. (2017). Large-scale cost function learning for path planning using deep inverse reinforcement learning. *The International Journal of Robotics Research*, *36*(10), 1073–1087.

Zhu, Y., Mottaghi, R., Kolve, E., Lim, J. J., Gupta, A., Fei-Fei, L., et al. (2017). Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *ICRA* (pp. 3357–3364). IEEE.

Olimpiya Saha is a Senior AI Research Scientist in the Advanced AI group at LG Electronics, USA. Prior to joining LG, she received her Ph.D. and M.S. degrees in Computer Science from the University of Nebraska at Omaha, and, her undergraduate degree in Computer Science from West Bengal University of Technology, India. Her research interests involve applications of machine learning techniques and knowledge transfer to computer vision problems as well as robotics particularly for the improvement of autonomous robot navigation in unfamiliar environments.

Prithviraj Dasgupta is the Union Pacific Endowed Professor with the Computer Science Department at the University of Nebraska, Omaha. He is the director of the CMANTIC Robotics Lab at his university that does research in the area of multi-robot systems, swarm robotics, and modular robots. He has led several large federally funded projects in his area of research, published more than 130 papers in leading conferences and journals, and received the highest research award at his university called ADROCA in 2017. He received his M.S. and Ph.D. in Computer Engineering from University of California, Santa Barbara in 2001, and, B. Engg. in Computer Science from Jadavpur University, India in 1995.



Bradley Woosley is a Ph.D. candidate in the Computer Science Department at the University of Nebraska at Omaha (UNO) and Research Assistant in the CMANTIC lab. He received his M.S. degree from UNO in computer science in 2015, and his B.Sc. in Computer Engineering from the University of Nebraska, Lincoln in 2013.