Autonomous Waypoint Generation Strategy for On-Line Navigation in Unknown Environments

Sanjeev Sharma and Matthew E. Taylor

Abstract— **This paper introduces a reinforcement learning (RL) based autonomous waypoint generation strategy (AWGS), for on-line path planning in unknown environments. An RL agent intelligently analyzes its surroundings and generates waypoints within the robot's field of view. The RL agent uses an MDP for waypoint generation that is formulated to be independent of the domain, robot model, and state space dimensionality. The RL agent requires no environment-specific information beyond the robot's field of view. A path to the selected waypoint is then generated by a path planner. AWGS is applicable to many path or motion planners. However, for brevity, this paper focuses on path planning without the robot's dynamics constraint. Experiments (i) compare the performance of RL agent's policies with RRTs and $A^\star$, and (ii) show that AWGS can: (a) be trained and then used with different robot models, domains, and state-spaces, and (b) successfully navigate in environments with non-convex obstacles.**

I. Introduction

Autonomous robots are becoming increasingly common in domestic, commercial and military settings. Such robots must be able to quickly plan a safe route from their current location to the goal, while still being flexible and adaptable to unforeseeable obstacles and environmental uncertainties. Despite recent successes, e.g., the Mars rovers [1] and DARPA Urban Challenge [2], on-line navigation in unknown environments remains a challenging problem. This paper presents an autonomous waypoint generation strategy (AWGS) that uses reinforcement learning (RL) [3] for on-line navigation of mobile robots in unknown 2D- and 3D-environments. In AWGS, an RL agent analyzes the local surroundings of the robot and generates a waypoint in its field of view (FOV). An underlying path planner (ECAN [4]) then plans a path to reach the waypoint. One of the key insights is that the RL agent's MDP is formulated to be independent of the domain and space dimensionality, allowing it to generalize its waypoint generation policy across different environments. This space-independence also enables planning the waypoints in 3D, using the identical computations as in 2D, which automatically generalizes learning from 2D- to 3D-space (and vice-versa) and makes AWGS effective for the 2D- and 3D-navigation problems.

Using waypoints or sub-goals for reliable navigation in 2D-environments has been widely discussed. Shiller [5] proposed *exit-points*, fixed locations on the boundary of obstacles for on-line navigation in known environments, which is restricted to 2D-space. Krogh *et al.* [6] presented a geometrical method for selecting sub-goals corresponding to the edge and vertices of *convex polygonal* obstacles in unknown 2D-environments. Thus, the approach may not be applicable for arbitrarily shaped obstacles. Maida *et al.* [7] placed local sub-goals inside a rectangular arena of fixed dimensions, constructed around the robot, at a fixed distance from the robot and at the intersection of line segments (forming the path) avoiding the obstacles. The approach is thus limited to the extraction of waypoints along the generated path (in 2D). In contrast, AWGS intelligently selects a waypoint for the planner, which then plans a path to the waypoint, and is thus not restricted by the abilities of a planner. Wang *et al.* [8] selected midpoints between obstacles, in front of the robot, as sub-goals in 2D-environments. A robot moving in a 3D-space may pass over an obstacle instead of searching for an opening between obstacles. AWGS can select waypoints to pass in between obstacles or to go over an obstacle in 3D-space. While the previous approaches are limited to 2D-space, AWGS is applicable in both the 2D- and 3D-space.

During the last decade, sampling based planners like RRTs [9] have been successfully demonstrated in many robotics applications. RRT methods explore the environment by randomly sampling it to construct trees. On-line planning with RRTs in structured (2D-) environments (e.g., following a road) has been addressed in [10]. The constraint that the vehicle should follow a road effectively provides a direction (forward) for the expansion of RRTs. In contrast, AWGS works in unstructured (no predefined path or road), and in both 2D- and 3D-, environments by intelligently generating the waypoints in the FOV. Karaman *et al.* [11], [12] proposed RRT* for optimal on-line path planning. However, RRT* requires a complete map of the environment for building an initial feasible plan to the goal (an initial tree), before the on-line planning starts. Then during the execution of this initial plan, a *rewiring* step modifies the tree (on-line) to generate an optimal solution [11]. The requirement of an initial solution to the goal may create difficulties in planning with zero prior knowledge of an environment. On the other hand, AWGS requires no prior map and assumes no information of the environment beyond the robot's field of view.

AWGS is applicable to many planners, however, this paper focuses on using the ECAN planner [4] because: (i) ECAN's implementation is similar for both 2D- and 3D-space; (ii) ECAN guarantees collision avoidance for non-convex shaped robots, which makes the presentation of non-convex shaped robots easier; and (iii) ECAN may fall into oscillations when obstacles are non-convex, making it easier to demonstrate

Sanjeev Sharma is a graduate student in the Computing Science Dept. at University of Alberta, Canada. `sanjeev1@ualberta.ca`

Matthew E. Taylor is an assistant professor in the Computer Science Dept., Lafayette College, USA. `taylorm@lafayette.edu`

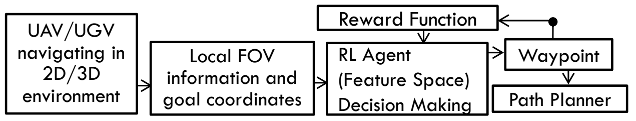

Reward Function
UAV/UGV navigating in 2D/3D environment
Local FOV information and goal coordinates
RL Agent (Feature Space) Decision Making
Waypoint
Path Planner

Fig. 1: AWGS uses domain and space independent RL agent for waypoint selection. The underlying path planner used in experiments is ECAN.

that the AWGS can overcome the shortcomings of a planner[1], and is not restricted by the abilities of an underlying planner. Also, an extension of this paper discussing the multi RL-agents approach and the planning time is available [18].

The contribution of this paper is to present a novel formulation, using RL, for waypoint generation that: (i) works identically for both the 2D- and 3D-navigation problems; and (ii) helps improve the performance of an underlying planner. The rest of this paper proceeds as follows: section-II presents background; section-II-B briefly describes ECAN; section-III describes the feature space construction and the reward function for the RL agent; and section-IV experimentally demonstrates the properties of AWGS.

II. Background

This section provides necessary background, describes the notations used in this paper and presents a selection of related work. This research assumes that the robot: (i) cannot see beyond its field of view (FOV); and (ii) knows its coordinates z_a^t, at each time-step t, and the coordinates of the goal z_g. A point-cloud of m obstacles at time-step t is represented as O^t with $z_{o_i}, i = 1, ..., m$ representing the location of i^{th} point-obstacle in the cloud. A set of n-dimensional positive definite symmetric matrices is denotes as S_{++}^n.

A. Reinforcement Learning

The underlying reinforcement learning (RL) problem of waypoint generation in AWGS is solved as an MDP [3]. In RL, a state-action value function $Q^\pi(s, a)$ for a policy π predicts the long-term reward if an agent takes action $a \in A$ in state $s \in S$ and thereafter follows policy $\pi : S \to A$. The agent's probability of taking an action a in s is given by $\pi(s, a)$. The agent's task is to learn a policy π that maximizes the total expected reward from any $s \in S$, where the reward may be discounted by a discount factor $\gamma \in [0, 1]$. By taking action a in s, agent makes a transition to state s', and receives a reward $r(s, a, s')$. The value function is approximated using a linear function approximation architecture: $Q(s, a) = \phi(s, a)^T w$, where $\phi(s, a) \in \mathbb{R}^k$ is the state-action feature vector for (s, a) and $w \in \mathbb{R}^k$ is learned using samples.

B. ECAN Navigation: Convex Programming Formulation

For navigation, ECAN forms a locally maximal ellipsoid Ψ^t, around the robot while taking the layout of local obstacles into account, oriented in such a way that favors progress towards the goal. The ellipsoid is constrained: (i) to contain the robot, (ii) to keep goal location on the boundary or outside, and (iii) to keep all the obstacles outside its boundary — ensuring collision avoidance. On time-step t, the ellipsoid Ψ^t is parameterized by variables (P^t, q^t, r^t) and is defined as $\Psi^t = \{x | x^T P^t x + x^T q^t + r^t \leq 0\}$, where $P^t \in S_{++}^n, q^t \in \mathbb{R}^n, x \in \mathbb{R}^n, r^t \in \mathbb{R}$ and $n = 2$ for 2D navigation and $n = 3$ for 3D navigation. Let $z \in \mathbb{R}^n$ be an arbitrary location in the space and let $\Psi^t(z) = z^T P^t z + z^T q^t + r^t$. The ellipsoid formation problem is then solved as semi-definite programming (SDP): (α, β are trade-off parameters and I is the identity matrix with the same dimensionality as P^t)

$$\begin{aligned} \textbf{minimize}\ & \Psi^t(z_g) + \alpha||\Psi^t(z_a^t)||_2 + \beta \textstyle\sum_i \Psi^t(z_{o_i}) \\ \textbf{subject to}\ & \Psi^t(z_a^t) \leq -1;\ \Psi^t(z_g) \geq 0;\ \Psi^t(z_{o_i}) \geq 1 \\ & P^t \succeq I;\ z_{o_i} \in O^t;\ i = \{1, ..., m\}. \end{aligned}$$

The first constraint ensures that the (point-) robot lies inside Ψ^t, the second constraint ensures that the goal lies outside or on the boundary of Ψ^t, the third constraint ensures that all the obstacles in the point-cloud (O^t) lie outside Ψ^t, and the positive definite constraint on P^t ($P^t \succeq I$) ensures that Ψ^t is an ellipsoid. Objective $\Psi^t(z_g)$ orients the ellipsoid to point towards the goal; $||\Psi^t(z_a^t)||_2$ checks the ellipsoid's unbounded growth along its principal axis [4]; and $\sum_{i=1}^m \Psi^t(z_{o_i})$ forms a locally maximal ellipsoid, around the robot, bounded by surrounding obstacles. If the robot is finite (i.e., not a point mass), the first constraint is replaced by a constraint that the convex-hull of the robot should lie inside Ψ^t. Next, a navigation direction and a step-length (Δs) are computed using the obtained ellipsoid. However, the navigation direction is biased with the ellipsoid's orientation (often pointing towards the goal), potentially allowing the agent to become trapped between non-convex obstacles [4]. Within the AWGS, z_g in the SDP is replaced by the next waypoint. Thus, instead of planning to reach the goal, ECAN in AWGS always plans to reach the next waypoint.

C. Related Work: Reinforcement Learning

The RL agent in AWGS learns a domain and space independent policy. The reutilization of a policy has been widely addressed in transfer learning [13]. In this context, state of art methods typically suffer when generalizing the learned policy from one 2D-environment to another 2D-environment — at least some amount of retraining in new environment is required (see for example [14]–[17]).

III. Feature Space Construction

This section describes the computation of parameters required to construct the feature space for the RL agent. These parameters are readily available during the navigation and are computed identically in 2D- and 3D-space. Obstacles are represented using a potential map (one of the parameters of feature space). The potential map helps the RL agent to classify safe and unsafe regions for waypoints. The geometric parameters help the RL agent to make progress towards the goal, while ensuring safety. The RL agent's task is to generate a waypoint in the FOV, while taking collision avoidance into account, so that the robot can eventually reach the goal by following waypoints.

[1]AWGS has been implemented successfully for RRTs, $A^\star$, unconstrained 2D- and 3D-splines, and motion planning: (i) with Mixed Integer Programming and (ii) of a Car Like Robot: http://www.searching-eye.com/awgs.mp4

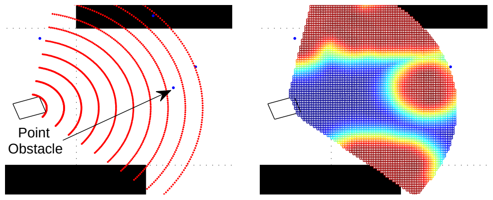


Fig. 2: (a) shows discretized grid-points in the FOV and (b) shows the corresponding potential map; red regions represent higher grid-point values.

The next section introduces the potential map. Then the geometric parameters, which allow goal-directed generation of waypoints, are discussed in section-III-B. Finally the MDP for the RL agent is formulated in section-III-C.

A. Potential Map

A local potential map represents obstacles, for the RL agent, in the robot's FOV at each time-step t. To form the potential map, the FOV is discretized. Using polar coordinate system, robot's view is restricted by $\langle R_{FOV}, \theta_{FOV} \rangle$ in 2D-space and by $\langle R_{FOV}, \theta_{FOV}, \phi_{FOV} \rangle$ in 3D-space. The FOV is thus defined by $\langle r, \theta \rangle$ and $\langle r, \theta, \phi \rangle$ in 2D- and 3D-space respectively, where $r \in (0, R_{FOV}]$, $\theta \in [-\theta_{FOV}, +\theta_{FOV}]$ and $\phi \in [-\phi_{FOV}, +\phi_{FOV}]$. The parameters dr, $d\theta$ and $d\phi$ represent discretization along the respective polar coordinates. The total number of grid-points N, in the robot's FOV is $((2\theta_{FOV}/d\theta) + 1)R_{FOV}/dr$ and $((2\theta_{FOV}/d\theta) + 1)((2\phi_{FOV}/d\phi) + 1)R_{FOV}/dr$ in 2D- and 3D-space respectively. One of these grid-points is then selected as a waypoint by the RL agent. The obstacles in the FOV are converted into an equivalent point-cloud. When obstacles are discovered in the FOV, then the grid-points lying on the faces/edges of these obstacles are added to the point-cloud O^t. Let there be m point-obstacles in the point-cloud (O^t). The potential map, for each of the N grid points, is then computed as:

$$V_q \leftarrow \max_{j:1,\ldots,m} \exp\left(\frac{-(\max\{0, ||l_q - z_{o_j}||_2 - \delta\})^2}{\sigma^2}\right)$$

where $q = \{1, \ldots, N\}$; $V_q \in [0, 1]$ is the value of q^{th} grid-point in the potential map; δ is the radius of the smallest circle encircling the robot ($\delta = 0$ for a point robot); and l_q and z_{o_j} are locations of the q^{th} grid-point and the j^{th} obstacle in the point-cloud, respectively. If $m = 0$ then $V_q = 0, \forall q = \{1, \ldots, N\}$. Fig. 2 shows the grid-points and corresponding potential map in 2D-space.

B. Geometric Parameters Computation: θ^{gr},ρ^{gr},ζ

After computing the potential map, three geometric parameters ($[\theta^{gr}_j, \rho^{gr}_j, \zeta_j]$) are computed for each of the grid-points in the FOV. These parameters, together with the potential map, enable environment independent generation of the waypoints, while ensuring safety. The first two parameters measure the progress towards the goal if the j^{th} grid-point is selected as the waypoint. The first parameter $\theta^{gr}_j \in [-\pi, \pi]$ is an angle between the vectors connecting z^t_a to z_g and z^t_a to the j^{th} grid-point (l_j). The second parameter is a normalized distance of the j^{th} grid-point from the goal location, $\rho^{gr}_j = d^{gr}_j/(2||z^0_a - z_g||_2)$, where $d^{gr}_j = ||z_g - l_j||_2$ and z^0_a is the robot's location at $t = 0$. Fig. 3 (left) depicts these parameters. The third parameter (ζ_j) is a Boolean variable. $\zeta_j = 1$ if a line-segment between the j^{th} grid-point and the robot's current location (z^t_a) intersects any finite obstacle; $\zeta_j = 0$ otherwise. It effectively determines the regions hidden from the robot's view — thus classifying the potentially unsafe regions (Fig. 3, right). The RL agent (as discussed in the following section) gets a large negative reward for selecting the grid-points with $\zeta_j = 1$ as the waypoints.

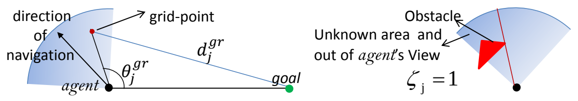


Fig. 3: The left figure shows the first two geometric parameters. The right figure shows a situation where the parameter $\zeta_j = 1$ for the grid-points which lie above the obstacle (red-triangle) and to left of the red-line.

C. Feature Space, Value Function and Reward Function

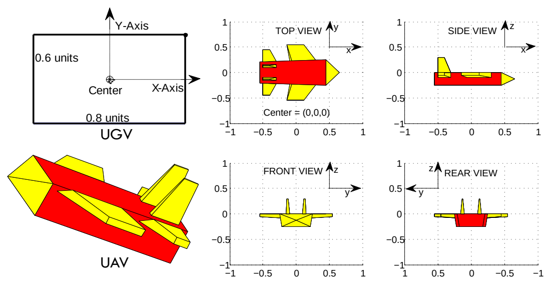
Once the parameters are computed, state-action features are computed for the RL agent. The robot's current location represents state of the RL agent at time-step t, while an action corresponds to selecting one of the grid-points as the waypoint. After the waypoint is selected, it becomes the current goal for ECAN, which then moves the robot a certain distance Δs (if the environment is uncertain) or to the waypoint (if the obstacles in the FOV are known to be static). The entire algorithm is then iterated. The $\Delta s = \min(\delta_1, \delta_2)$, where δ_1 is arbitrarily fixed and δ_2 is computed, with constraint that the robot remains inside Ψ^t, using quadratic programming (see [4]). The robot may not actually navigate to the location suggested by the current action (waypoint) of the RL agent — an action may be only partially executed. The resulting formulation may violate the Markov property, however, it is treated here as an MDP. The state-action feature vector corresponding to the j^{th} grid-point is computed as: $\Phi_j = [1, 2S(\rho^{gr}_j), \cos(\theta^{gr}_j)\exp(-\rho^{gr}_j), V_j, \zeta_j]^T$, where $S(x)$ represents the sigmoid function of $x \in \mathbb{R}$. The RL agent's policy is then learned using reinforcement learning (sarsa), with an appropriate reward function.

Reward Function: The reward function is designed such that it penalizes the RL agent for generating the waypoints in obstructed regions of the FOV ($\zeta_j = 1$) or close to the boundary of obstacles (measured by the grid-point's value in the potential map). If the RL agent selects the goal location z_g as the waypoint, it gets a large positive reward. For selecting the j^{th} grid-point as the waypoint, RL agent receives a reward:

$$r = -10^3\zeta_j - \alpha_1 V_j + \max\{\alpha_2 500, -5\}.$$

$\alpha_2 = 1$ if the waypoint is defined at the goal z_g, and -1 otherwise. The max function respectively returns $+500$ when the waypoint is at the goal and -5 otherwise, encouraging the RL agent to reach the goal in minimum possible waypoint iterations[2]. α_1 is a constant that controls penalty for defining waypoints close to obstacles.

[2]We use number of waypoint iterations (or simply, iterations) in our experiments to measure the performance of a policy.



(a) (b)

Fig. 4: Finite robot models: (a) upper- UGV with x-axis as the direction of motion; lower- UAV; (b) dimensions of UAV with the help of grid markings.

IV. Experiments

AWGS is evaluated with three categories of experiments showing: (i) domain, robot, and space independent waypoint generation; (ii) a comparison of the performance of optimal policies with RRTs and $A^\star$; and (iii) the planning capabilities in unknown complex 2D- and 3D-space. The finite robot models used in the experiments are depicted in Fig. 4. The default values of parameters are: $\langle \delta_1, \alpha_1 \rangle = \langle 1, 200 \rangle$; $\langle R_{FOV}, dr, \phi_{FOV}, d\phi, \sigma^2 \rangle = \langle 5, 0.2, 40^\circ, 2^\circ, 0.5 \rangle$; $\langle \theta_{FOV}, d\theta \rangle = \langle 60^\circ, 1^\circ \rangle$ for 2D and $\langle 40^\circ, 2^\circ \rangle$ for 3D; $\langle \alpha, \beta \rangle = \langle 0.1, 5 \times 10^{-4} \rangle$ in 2D and $\langle 0.1, 10^{-4} \rangle$ in 3D (trade-off parameters in the SDP in ECAN); and learning rate in SARSA [3] is 0.01 with discount factor 0.9. $A^\star$ was implemented in 2D domains by discretizing the domains in 500×500 grid-cells.

A. *Robot-Model Independence and Policy Evaluation*

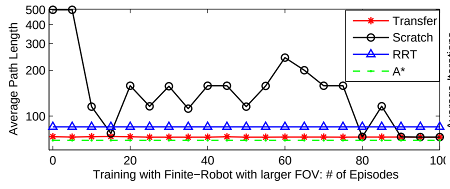
These experiments show that the RL agent's policy is independent of the specific robot model, e.g., dimensionally different robots and FOV. The RL agent is first trained for a point-robot with default FOV parameters. 120 training episodes in the domain of Fig. 7a are used with 50 to 400 random point obstacles. An episode ends when the robot reaches the goal, or the number of waypoint iterations exceeds 200. This policy is then used as an initial policy while training the RL agent on a (2D-) finite robot (Fig. 4, UGV) with $\theta_{FOV} = 80^\circ$. Also, the RL agent learns a new policy for the finite robot from scratch. After every 5 training episodes, both policies are tested in the domain shown in Fig. 7a, with 100 random point obstacles, in 10 different start-goal configurations. In this experiment, obstacles in the FOV are assumed to be static — once the RL agent generates a waypoint, robot reaches it using ECAN, and then the next waypoint is generated. Fig. 5a – 5c show the planning results when learning with point-robot's policy as the initial policy (*transfer*) and when learning from scratch. Navigation to the goal is considered successful if the robot reaches it without colliding with any obstacle and using at most 200 waypoint iterations. If the robot collides with an obstacle, the path-length for that navigation problem is set to 500 and number of iterations is set to 200. Fig. 5a shows the average path-length, averaged over 10 start-goal configurations. Fig. 5b shows the average number of waypoint iterations required to reach the goal and Fig. 5c shows the probability of success, after every 5 training episodes, in 10 different start-goal configurations (i.e., this graph displays the number of experiments in which the robot reached the goal without collision and without violating the 200 iterations limit). These experiments show that the RL agent's policy is independent of the robot-model and FOV — re-learning for the finite-robot with the point-robot's policy (transfer) does not show any improvement. Additionally, the performance of learning from scratch converges to the performance of the point-robot's policy, requiring at least 100 training episodes.

To evaluate the performance of RL agent's optimal policy learned in this domain, the average path-length, over 10 start-goal configurations, produced by RRTs and $A^\star$ are shown in Fig. 5a. In each of the 10 start-goal configurations, RRTs were run 50 times with 10^5 RRT-iterations in each run, producing trees with an average of 75000 edges, in each of the 10 configurations. As shown, the average path-length in 10 start-goal configurations is smaller for AWGS which plans in unknown environments, as compared to RRTs exploring the entire environment. However, the paths are longer than the global $A^\star$ search. Average path-length in each of the 10 configurations is also shown in Fig. 6 (left).

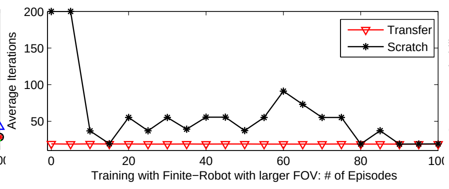
B. *Domain Independence and Policy Evaluation*

Navigation in unknown environments requires the MDP to be domain independent. Experiments in this section show that the RL agent learns a domain-independent policy to generate waypoints. Since the RL agent's policy is robot-independent, the point-robot's policy learned for the domain in Fig. 7a is used as an initial policy for re-learning (*transfer*) in the domain of Fig. 7b, for the finite-robot. Also, a new policy is learned from scratch in this domain to compare how well the transferred policy performs in this new domain. The environment is again assumed static — the next waypoint is generated only when robot reaches the current waypoint, using ECAN. Fig. 5d – 5f compare the performance of learning with transfer and learning from scratch. Both policies are tested after every 5 training episodes, in 15 different start-goal configurations. The policy learned for the domain in Fig. 7a performs optimally in the new domain with non-convex obstacles even without any training. The new policy learned from scratch takes 100 training episodes for a similar performance, and converges to that performance — showing that the initial transfer policy is optimal. Thus, learning is domain-independent — enabling AWGS to plan in unknown environments.

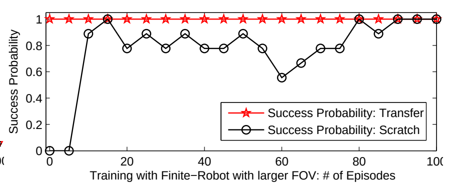
Again the performance of transferred policy (which is optimal for the RL agent) is evaluated with RRTs and $A^\star$. Fig. 5d shows the average path-length over 15 start-goal configurations for RRTs and $A^\star$. Again the AWGS produces shorter paths as compared to RRTs but longer than $A^\star$. Also, the length of the paths produced by AWGS, RRTs and $A^\star$ in each of the 15 configurations is shown in Fig. 6 (right). The average for RRTs was again taken over 50 trials in each configuration, with 10^5 RRT-iterations in each trial, producing trees with an average of 68550 edges in each configuration. The paths produced by AWGS are again shorter than RRTs (for all but one configuration) even when



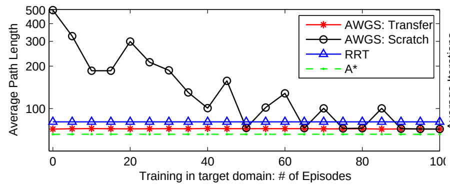
(a)



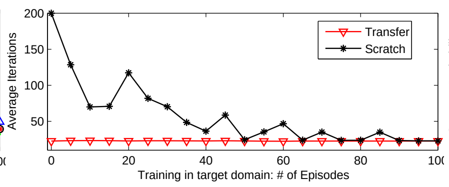
(b)



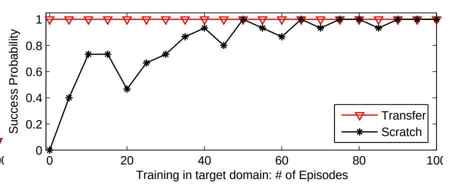
(c)



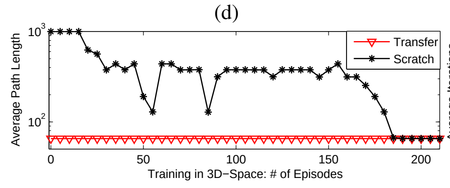
(d)



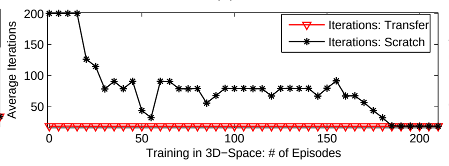
(e)



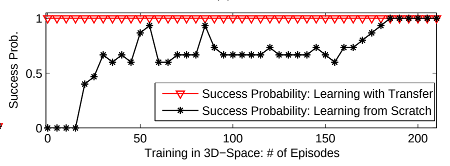
(f)



(g)



(h)



(i)

Fig. 5: a–c empirically show that 1) the RL agent's policy is robot-model independent and can be reused for dimensionally different robots and FOV, and 2) the learned policy produces paths shorter than RRTs and longer than $A^\star$. d–f show that the RL agent performs optimally in novel domain without any retraining and that the performance remains constant during additional training. The learned policy again produces shorter paths than RRTs, but produces longer path than $A^\star$. g–i show that 1) the 2D-space policy performs optimally in 3D-space even with no training in 3D and 2) learning from scratch converges to the same performance.

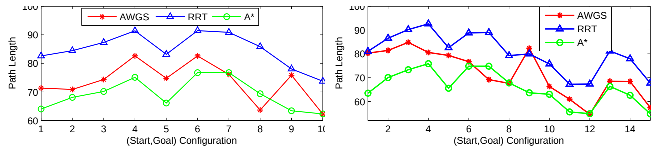


Fig. 6: Comparison of average path-length produced by RRTs, AWGS and $A^\star$ in each of the 10 and 15 configurations in convex (left) and non-convex (right) domains respectively.

planning in unknown environment, while RRTs explore the entire environment.

C. Space Independence

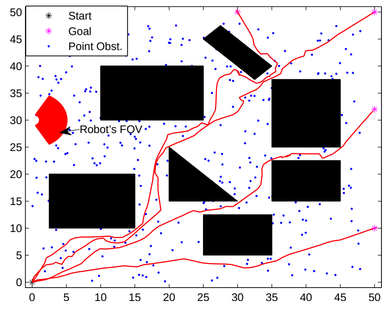
This section experimentally shows that the RL agent's MDP is independent of the space-dimensionality, e.g., training in 2D-space and planning in 3D-space without training from 3D-samples. As in previous experiments, two policies are learned: (i) *transfer-policy*: learning with 3D-samples using the 2D-space policy as an initial policy and (ii) *scratch-policy*: learning from scratch with 3D-samples, for the UAV (Fig. 4). Since the RL agent's policy is environment independent, the test and the training environments are the same. Episodes are defined in the same way as in previous two experiments. After every 5 training episodes, both the policies are tested in the domain shown in Fig. 8, with 15 different start-goal configurations and with 100 random point obstacles. The 2D-space policy was learned in Fig. 7b domain for 200-episodes, with 50 to 1000 random point obstacles. As shown in Fig. 5g – 5i, the transfer-policy performs optimally without any 3D training, while the scratch-policy requires 20-episodes to start reaching the goal (Fig. 5i) and at least 210-episodes to approximate the performance of transfer-policy. Additional training after transfer does not improve the policy's performance. The performance of learning from scratch again converges to the performance of transfer — the initial 2D-space policy performs optimally.

D. Planning in 2D: Convex and Non-Convex Obstacles

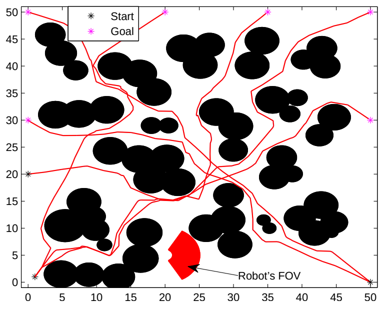
This section shows sample planning results and compares the performance of AWGS (using ECAN as planner) with ECAN to show that AWGS can overcome the shortcomings of a planner. Fig. 7a shows the sample trajectories planned on-line by AWGS, with finite-robot and with $\Delta s = \min(\delta_1, \delta_2)^3$, in a cluttered environment with convex obstacles and 401 random point obstacles. The robot's FOV is also shown for comparison. AWGS successfully navigates the robot to the goal(s), with RL agent defining the waypoints and convex constraints in ECAN ensuring collision avoidance for the finite-robot. Fig. 7b shows navigation with AWGS in a domain cluttered with non-convex obstacles. AWGS successfully avoids the traps in between the obstacles.

Both AWGS and ECAN alone were tested in 100 different start-goal configurations in Fig. 7b. AWGS successfully reached the goal in all runs, while ECAN failed to reach the goal because the ellipsoid always points towards the goal and the robot gets trapped in the concavity of obstacles. These experiments show that AWGS can successfully plan in unknown environments cluttered with convex or non-convex obstacles even when underlying planner is prone to local oscillations among non-convex obstacles. Also, AWGS avoids getting trapped in the concavity of obstacles even when the potential map, one of the features in the RL agent's

[3]Once the waypoint is selected by the RL agent, ECAN plans the path to navigate the robot by a distance Δs and then a new waypoint is generated.



(a)



(b)

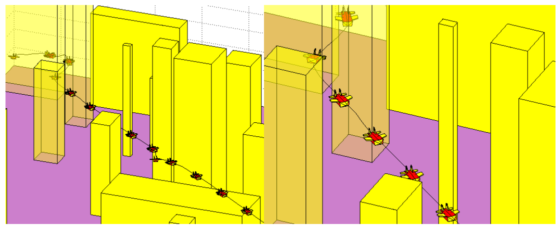

(c)

Fig. 7: (a) Sample trajectories to 4 different goals with finite robot model, (b) planning in unknown environment cluttered with non-convex obstacles; robot's FOV also shown for comparison, and (c) UAV navigating in between convex obstacles (left) and a zoomed figure (right), depicting UAV clearly.

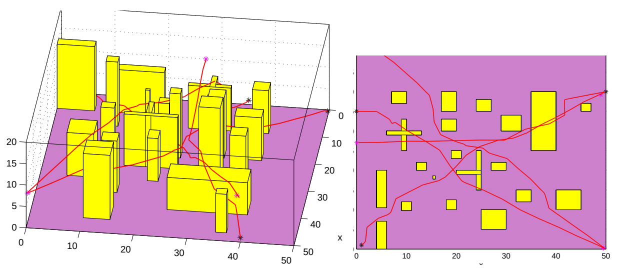


Fig. 8: Showing sample trajectories (left) planned in 3D environment with convex and non-convex obstacles along with projection on x-y (right) plane.

state-action feature vector, may succumb to local minima. The geometric parameters help in goal-directed selection of the waypoints and avoid these local minima.

E. 3D-Space Planning

This section shows successful planning with AWGS ($\Delta s = \min(\delta_1, \delta_2)$) in unknown 3D-spaces, in the presence of both convex and non-convex obstacles. Fig. 7c shows on-line planning for the UAV — the FOV parameters were again set to defaults. AWGS successfully avoids collision and plans directly with finite size UAV in 3D-space cluttered with convex obstacles. Fig. 8 shows sample planning results with AWGS in an environment cluttered with both convex and non-convex 3D-obstacles.

Again, both AWGS and ECAN alone were tested in 100 different start-goal configurations in Fig. 8 domain. AWGS successfully reached the goal in all 100 test cases, while ECAN only succeeded in 36 test cases. ECAN easily gets trapped in between the two non-convex obstacles, while AWGS, due to waypoints, easily avoids getting trapped in between the non-convex obstacles.

V. Conclusion, Discussion and Future Work

This paper presented a novel waypoint generation strategy that facilitates navigation in unknown 2D- and 3D-space. While the existing randomized planners plan once the configuration of obstacles in the environment is known, AWGS on the other hand requires no environment specific information beyond the robot's FOV and produces relatively shorter paths. The waypoint generation is independent of the robot models and space-dimensionality. This makes it possible to learn the RL agent's policy for any kind of robot model and in either 2D- or 3D-space. Future work (i) will present on-line non-holonomic motion planning in unknown environments with AWGS, and (ii) involves implementation of the AWGS on a quad-rotor flying robot and analysis of the performance with noisy robot's coordinate detection.

An implementation of AWGS in Python will be made available at: http://www.searching-eye.com/awsf/

References

[1] K.L. Wagstaff, "Smart Robots on Mars: Deciding Where to Go and What to See", in *Juniata Voices*, vol. 9, 2009.
[2] S. Thurn, "Why we compete in DARPA's Urban Challenge autonomous robot race", in *Communications of the ACM*, 2007.
[3] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, MIT Press, 1998.
[4] S. Sharma, "QCQP-Tunneling: Ellipsoidal Constrained Agent Navigation", in *IASTED International Conference on Robotics*, 2011.
[5] Z. Shiller, "Online Suboptimal Obstacle Avoidance", in *IJRR*, 19(5), 480-497, 2000.
[6] B. H. Krogh and D. Feng, "Dynamic Generation of Subgoals for Autonomous Mobile Robots Using Local Feedback Information", in *IEEE Transactions on Automatic Control*, 34(5), 483-493, 1989.
[7] A.S. Maida, S. Golconda, P. Mejia, A. Lakhotia and C. Cavanaugh, "Subgoal-based local navigation and obstacle avoidance using a grid-distance field", in *Int' Journal of Vehicle Autonomous Systems*, 2006.
[8] D. Wang, D. K. Lit, N. M. Kwok and K. J. Waldron, "A Subgoal-Guided Force Field Method for Robot Navigation", in *Int' Conf. on Mechatronics and Embedded Systems and Applications*, 2008.
[9] S.M. LaValle and J.J. Kuffner, "Randomized Kinodynamic Planning", in *IJRR*, 20(5), 378-400, 2001.
[10] Y. Kuwata, G. Fiore, J. Teo, E. Frazzoli and J.P. How, "Motion Planning for Urban Driving using RRT", in *ICRA*, 2008.
[11] S. Karaman and E. Frazzoli, "Incremental Sampling-based Algorithms for Optimal Motion Planning", in *RSS*, 2010.
[12] S. Karaman, M.R. Walter, A. Perez, E. Frazzoli and S. Teller, "Anytime Motion Planning using the RRT*", in *ICRA*, 2011.
[13] M.E. Taylor and P. Stone, "Transfer Learning for Reinforcement Learning Domains: A Survey", in *JMLR*, 10(1), 1633-1685, 2009.
[14] K. Ferguson and S. Mahadevan, "Proto-Transfer Learning in Markov Decision Process using Spectral Methods", in *ICML Workshop on Transfer Learning*, 2006.
[15] L. Frommberger, "Generalization and Transfer Learning in Noise-Affected Robot Navigation Tasks", in *Artificial Intelligence: EPIA Lecture Notes in Computer Science*, 308-519, 2007.
[16] L. Frommberger, "A Generalizing Spatial Representation for Robot Navigation with Reinforcement Learning", in *FLAIRS*, 2007.
[17] L. Frommberger and D. Wolter, "Structural Knowledge Transfer by Spatial Abstraction for Reinforcement Learning Agents", in *Adaptive Behavior*, 18(6), 507-525, 2010.
[18] S. Sharma and M.E. Taylor, "Autonomous Waypoint Selection for Navigation and Path Planning: A Navigation Framework for Multiple Planning Algorithms", Technical Report, 2012. http://www.searching-eye.com/awsf/awsf.pdf