RL-RRT: Kinodynamic Motion Planning via Learning Reachability Estimators from RL Policies

Hao-Tien Lewis Chiang[1,2], Jasmine Hsu[1], Marek Fiser[1], Lydia Tapia[2], Aleksandra Faust[1]

Abstract—This paper addresses two challenges facing sampling-based kinodynamic motion planning: a way to identify good candidate states for local transitions and the subsequent computationally intractable steering between these candidate states. Through the combination of sampling-based planning, a Rapidly Exploring Randomized Tree (RRT) and an efficient kinodynamic motion planner through machine learning, we propose an efficient solution to long-range planning for kinodynamic motion planning. First, we use deep reinforcement learning to learn an obstacle-avoiding policy that maps a robot's sensor observations to actions, which is used as a local planner during planning and as a controller during execution. Second, we train a reachability estimator in a supervised manner, which predicts the RL policy's time to reach a state in the presence of obstacles. Lastly, we introduce RL-RRT that uses the RL policy as a local planner, and the reachability estimator as the distance function to bias tree-growth towards promising regions. We evaluate our method on three kinodynamic systems, including physical robot experiments. Results across all three robots tested indicate that RL-RRT outperforms state of the art kinodynamic planners in efficiency, and also provides a shorter path finish time than a steering function free method. The learned local planner policy and accompanying reachability estimator demonstrate transferability to the previously unseen experimental environments, making RL-RRT fast because the expensive computations are replaced with simple neural network inference.
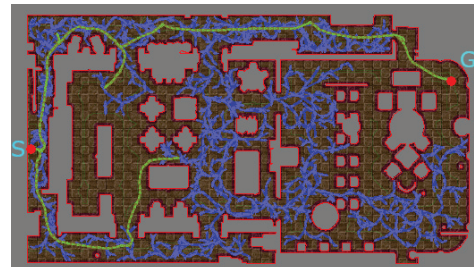
Index Terms—Motion and Path Planning, Learning and Adaptive Systems, Deep Learning in Robotics and Automation

(a) RL-RRT and SST in Map 1 (46.1 x 49.5 m)

(b) The Fetch robot

(c) Trjectory execution of Fetch in Map 2 (46.1 x 49.5 m)

Fig. 1. (a) Example trees constructed with RL-RRT (yellow) and SST [15] (blue) for a kinodynamic car navigating from start (S) to goal (G). (b) The Fetch robot. (c) RL-RRT (green) and the real-world trajectory executed (cyan) from the start (green dot) towards the goal (blue dot) in Map 2. Map 2 is a SLAM map of an actual office building.

I. INTRODUCTION

CONSIDER motion planning for robots such as UAVs [17], autonomous ships [3], and spacecrafts [23]. The planning solution needs to satisfy two criteria. First, the solution path must be feasible, meaning that the path must be collision-free and satisfy kinodynamic constraints, e.g. velocity and acceleration bounds even in the presence of sensor noise. Second, the path needs to be efficient, i.e. near optimal with respect to objectives such as time to reach the goal. For example, a motion plan for a car-like robot should avoid obstacles, reach the goal promptly, not make impossibly sharp turns, and maintain enough clearance to compensate for sensor noise.
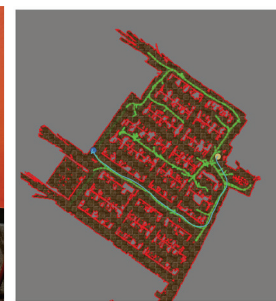
Current state of the art kinodynamic motion planners search the robot's feasible state space by building a tree data structure of possible robot motions rooted at the robot's current state. The methods iteratively use a local planner to attempt to grow the tree until the goal is reached. While some tree-based methods grow the tree by randomly propagating actions, others guide the tree growth to focus state space expansion thus requiring the local planner to be a steering function, a control policy that guides a robot to a specific goal in obstacle-free space, while satisfying the kinodynamic constraints. For example, consider a car-like robot needing to translate a small distance to the left, a motion resembling parallel parking. This motion plan is difficult, even in the absence of obstacles, and requires a steering function to steer the car to the goal. Computing the steering function requires solving an optimal control problem, and is generally NP-Hard [28]. To date, only very limited robot dynamics such as linear

[27] and differential drive [20] systems have optimal steering functions.

Besides the existence of steering functions, there are two additional difficulties facing efficient tree-based kinodynamic motion planning. First, tree-based methods that use steering functions require identifying the state in the tree from which to grow. This requires a function that compare the distance between states and return those that are expected to be easily solved by the steering function. An effective distance function for kinodynamic planning is the Time To Reach (TTR) between states using an optimal steering function [20]. TTR, however, is often expensive to compute as it involves numerically integrating the steering function [20]. Second, neither the steering functions nor the related TTR are informed by sensors, and, as a result, do not account for potential obstacles. For example, if a goal is occluded by a wall, the steering function is not able to see the wall due to the lack of sensory input, and TTR would return a value as if an agent could go through the wall.

Recently, deep Reinforcement Learning (RL) emerged as a promising near optimal steering function for kinodynamic systems [13]. In addition, deep RL algorithms can learn policies that map noisy lidar or camera observations directly to robot actions, thus enabling obstacle avoidance while navigating between states for differential drive robots [4], [5]. With the recent development of AutoRL [4], which uses evolutionary algorithms to largely eliminate the need to hand-tune hyper-parameters, network structure and reward functions. This combination offers the promise of deep RL being employed for local planning, i.e., providing both steering function and obstacle avoidance. However, RL policies often lack long-term planning capabilities [18] and get trapped in environments with complex obstacles [6].

To address the lack of available steering functions, good distance functions for aiding tree growth, and obstacle-awareness facing kinodynamic motion planning, we propose RL-RRT, which combines RL and sampling-based planning. It works in three steps. First, we learn an obstacle-avoiding point-to-point (P2P) policy with AutoRL. This is a mapless, goal-conditioned policy that maps sensor readings to control. These P2P policies generalize to new environments without re-training [4]. Second, we train a supervised obstacle-aware reachability estimator that predicts the time it takes the P2P policy to guide the robot from a start to goal state in presence of obstacles, using local observations such as lidar. The key insight is that the AutoRL policy and the estimator implicitly learn the topology of the obstacles. This allows reasonably accurate estimates of time to reach in new environments. Lastly, presented with a motion planning problem in a new envrionment, in a RRT setting, we use the RL policy as a local planner and the reachability estimator as the distance function. The combination of these two learning solutions offers two primary advantages. First, by using RL policies as an obstacle avoiding local planner, RL-RRT can be applied to a variety of kinodynamic systems without optimal steering functions. Second, by using the obstacle-aware reachability estimator, RL-RRT can prune out randomly sampled states

that are un-reachable from the tree, e.g., the policy is expected to be unsuccessful, and identify nodes with small TTR to the sampled state. In the example of a car in front of a wall, the RL policy will go around the wall, and the estimator will predict that the time to reach will be longer because of the wall.

We evaluate RL-RRT in two environments with three kinodynamic robots. Results indicate that AutoRL policies are effective obstacle-avoiding local planners. The obstacle-aware reachability estimators, one for each robot, are 74-80% accurate in identifying if a goal state is reachable. Compared to a state of the art steering function free method, SST [15], RL-RRT is up to 2.3 times more likely to identify a path within a fixed time budget and the identified path is up to 4.5 times shorter. RL-RRT typically identifies dynamically-feasible paths in very few iterations – 11 in this case – thanks to intelligent node selection and the obstacle-avoiding local planner (Figure 1a). The enclosed video demonstrates RL-RRT tree construction and trajectory execution on a physical differential drive robot.

II. RELATED WORK

Steering function-based kinodynamic planners, such as kinodynamic RRT* [27] and D-FMT [24] rely on a steering function to "pull" the tree to achieve rapid exploration [22] and a proper distance function [27], [20], [28]. RL-RRT uses AutoRL [4] to learn steering functions, thus bypassing the challenging two-point boundary value problem.

Steering function free-based approaches, such as EST [22] and SST [15], propagate random actions from a selected node. These methods can be applied to a variety of robot dynamics, although they tend to "wander" [1], thus they can take a long time to identify a solution.

Recent research has offered several solutions for P2P obstacle-avoidance policies on a differential drive robot from raw sensory input, including learning from demonstration [21], curriculum learning [29], and reinforcement learning [26], [4]. Other research offers hierarchical solutions to navigation, where the RL agent executes a path identified by another planner, e.g., from a grid [5], PRMs [6], [8], or manually selected waypoints [12]. However, none of those methods are designed for kinodynamic robots, leading to failures at milestones due to dynamic constraints [8].

Designing an effective distance function for sampling-based kinodynamnic motion planning is challenging [20]. The commonly used Euclidean and weighted Euclidean distance for configuration space planning is inefficient as kinodynamic robot states have limited reachability [14]. The minimum TTR between states is a highly effective distance function [20], [28] but is often too computationally-expensive to be used as a distance function [20]. While learned TTR of a near-optimal differential drive steering function [20] can overcome the computational complexity, this approach still requires a near-optimal steering function. Indirect optimal control has also been used to generate training samples composed of minimum TTR and optimal control actions along trajectories [28]. However, this approach currently only

works for low dimensional systems such as inverted pendulum and does not handle limited action bounds. Our approach addresses these challenges by bypassing the necessity of a near-optimal steering function via RL. Unlike previous methods, we also take into account obstacle avoidance, which can significantly change the minimum TTR.

III. METHODS

RL-RRT is a kinodynamic motion planner that learns local planner and distance function w.r.t the individual robot dynamics. It has three main steps. First, we learn an obstacle-avoiding point to point policy with AutoRL [4]. Next, since the RL policy avoids obstacles, we can use the policy to generate obstacle-aware reachability training samples by repeatedly rolling out the learned policy. An obstacle-aware reachability estimator is trained to predict the time to reach between two robot states in the presence of obstacles. Policy and estimator training is done once per robot in training environments. Third, during planning, RL-RRT creates dynamically-feasible motion plans using the RL policy as the local planner and the reachablity estimator as the distance function. Note, that the training and planning simulators require simulated depth measurements (e.g. lidar) around the robot, which can be thought of as analogous to motion planning with information about clearance.

A. AutoRL Local Planner

We train a RL agent to perform a P2P task that avoids obstacles. The output of the training is a policy that is used as a local planner, an execution policy, and a data generation source for the obstacle-aware reachability estimator. Using one RL policy for both local planning and path execution is inspired by [9]. This allows the planner to account for potential noise during path execution. To train a policy robust against noise, we model the RL policy is a solution for a continuous state, continuous action, partially observable Markov decision process (POMDP) given as a tuple $(\Omega, S, A, D, R, \gamma, O)$ of observations, state, actions, dynamics, reward, scalar discount, $\gamma \in (0, 1)$, and observation probability. The observations are the last three measurements of the noisy robot lidar and potentially noisy relative goal position and robot velocity. We define states as the true robot configuration and its derivative. A black-box robot dynamics simulator, which maps states-action pairs to states, is an input to the RL training environment. Another black-box simulator maps the robot state to noisy lidar observations w.r.t. obstacles. The goal is to train the agent to reach a goal state, G, within radius, d_G. Note that AutoRL identifies a policy that maps noisy sensor and state observations to action. We explore simulated lidar measurement noise in this work and left state estimation and process noise to future work. AutoRL training is required only once for a given robot.

AutoRL [4] over DDPG [16], used for learning the RL agent policy, takes as input: observations, actions, dynamics, goal definition, (G, r), and a parametrized reward, $R : O \times \theta_r \to \mathbb{R}$,. The agent is trained to maximize the probability of reaching the goal without collision. This is achieved by using evolutionary algorithms over populations of agents to find a dense reward that maximizes successful goal reaching. Each generation of agents is trained with a new reward, selected based on the previous experience. At the end, the fittest agent that performs P2P tasks best, is selected as the P2P policy. In this work, all three agents use the same observations, goal definitions, and neural network architectures, but differ in the robot dynamics and reward features used.

As an example, we explain the training of the Asteroid robot here (details of the robot are in the Appendix). Details for the Differential Drive and Car robot can be found in [4] and [8]. The observation is a vector of $3N_{\text{beams}}$ noisy lidar returns concatenated with the relative planar position of the goal, the robot velocity and orientation ($3N_{\text{beams}} + 5$ dimensional vector). The state is the planar position, velocity and orientation of the robot. The action is the amount of forward thrust and turn rate. The parameterized reward includes

$$R_{\boldsymbol{\theta}_{r_{\text{DD}}}} = \boldsymbol{\theta}^T [r_{\text{goal}} r_{\text{goalDist}} r_{\text{collision}} r_{\text{clearance}} r_{\text{speed}} r_{\text{step}} r_{\text{disp}}],$$

where r_{goal} is 1 when the agent reaches the goal and 0 otherwise, r_{goalDist} is the negative Euclidean distance to the goal, $r_{\text{collision}}$ is -1 when the agent collides with obstacles and 0 otherwise, $r_{\text{clearance}}$ is the distance to the closest obstacle, r_{speed} is the agent speed when the clearane is below 0.25m, r_{step} is a constant penalty step with value 1, and r_{disp} is sum of displacement between the current and positions 3, 6 and 9 steps before. $\boldsymbol{\theta}$ is the weight vector tuned by AutoRL.

B. Obstacle-Aware Reachablity Estimator

We further improve upon work in [20] by learning the TTR of an obstacle-avoiding P2P RL policy learned in Section III-A. Our obstacle-aware reachability estimator provides the following benefits: 1) It does not need an engineered near-optimal steering function for each robot dynamics. This allows TTR learning for robot systems without near-optimal steering functions. 2) Due to the presence of obstacles, the minimum TTR between states is a function of both robot dynamics and obstacles. Since RL policies can also learn to avoid obstacles, the obstacle-aware reachability estimator can provide additional benefit over TTR estimators that consider only obstacle dynamics such as [20].

1) Training data collection: Algorithm 1 summarizes the training data collection. First, for each episode, we initialize the robot with randomly chosen start and goal states (Alg. 1 line 2). Next, we execute the policy until the episode terminates (lines 4-11) due to reaching the goal, collision, or reaching a time horizon T_{horizon}. During execution, we record the robot observation at each time step (line 8) and compute and record the TTR cost (lines 9-10). The TTR cost is set to Δt at every time step. To classify whether the robot can reach the goal, we use a simple heuristic that penalizes trajectories that do not reach the goal. If the robot is in collision or the time horizon is reached (elapsedTime equals to T_{horizon}), the TTR cost of that time step is set to $\Delta t + T_{\text{horizon}}$, and the

Algorithm 1 Training data collection

Input: $\pi(o)$: Obstacle avoiding P2P RL policy, N_{episode}: Number of episodes, Δt: Time step size, T_{horizon}: Reachability horizon
Output: trainingData $= (o_1, y_1), (o_2, y_2), \cdots, (o_N, y_N)$.
1: **for** $i = 1, \cdots N_{\text{episode}}$ **do**
2:    s, g = sampleStartAndGoal()
3:    elapsedTime = 0
4:    **while** isDone is False **do**
5:       elapsedTime += Δt
6:       o = makeObservation()
7:       executePolicy($\pi(o)$, Δt)
8:       obsHistory.append(o)
9:       c, isDone = getTTRCost(elapsedTime, T_{horizon})
10:      costHistory.append(c)
11:    **end while**
12:    cfc = computeCumulativeFutureCost(costHistory)
13:    **for** j=0, len(obsHistory) **do**
14:       trainingData.append((o = obsHistory[j], y = cfc[j]))
15:    **end for**
16:    obsHistory.clear(); costHistory.clear()
17: **end for**
18: **return** trainingData

Algorithm 2 RL-RRT

Input: $\pi(o)$: Obstacle avoiding P2P RL policy, Δt_{tree}: Tree extension time step size, Δt: policy time step size, T_{horizon}: Reachability horizon, P_{goalBias}: Goal bias, x_{root}: Current robot state, k_c: Number of candidate nodes
Output: $\mathcal{P}$: Motion plan.
1: iteration = 0
2: $\mathcal{T}$.add(makeNode(x_{root}, None))
3: **while** termination condition not met **do**
4:    iteration += 1
5:    goodXrndFound = False
6:    **while** not goodXrndFound **do**
7:       x_{rnd} = sampleCollisionFreeStateSpace(P_{goalBias})
8:       candidateNodes = findNearestNodesEu($\mathcal{T}$, x_{rnd}, k_c)
9:       n_{nearest} = findNearestNode(candidateNodes, x_{rnd})
10:      TTR = getAvgTTR(n_{nearest}, x_{rnd})
11:      **if** TTR $<$ TTR$_{\text{threshold}}$ or rnd $> P_{\text{prune}}$ **then**
12:         goodXrndFound = True
13:      **end if**
14:    **end while**
15:    $x_{\text{new}} = n_{\text{nearest}}$.state; $t_{\text{extend}} = 0$
16:    **while** not ($t_{\text{extend}} > t_{\text{maxExtend}}$ or reach(x_{new}, x_{rnd}) or x_{new} is in collision) **do**
17:       t_{extend} += Δt
18:       o = makeObservation(x_{new}, x_{rnd})
19:       x_{new} = propagateDynamics($\pi(o)$, x_{new})
20:       **if** x_{new} is not in collision and t_{extend} % $\Delta t_{\text{tree}} = 0$ **then**
21:         $\mathcal{T}$.add(makeNode(x_{new}, x_{rnd}))
22:       **end if**
23:    **end while**
24: **end while**
25: **return** $\mathcal{P}$ = extractMotionPlan($\mathcal{T}$)

---

episode is terminated immediately by setting isDone to true.

After an episode terminates, we compute the cumulative future TTR cost for all states along the trajectory, i.e., remaining cost-to-go to the end of the trajectory (line 12). The observation and cumulative future cost of each time step form a training sample and is recorded (line 14). The process repeats for $N_{\text{episode}} = 1000$ episodes. We designed the TTR cost heuristic such that if the robot reaches the goal, the cumulative future cost of each state along the trajectory is the TTR between that state and the goal. Conversely, if the robot failed to reached the goal due to collision or the episode reaches time horizon, all cumulative future cost along the trajectory will be larger than T_{horizon}. By employing a common machine learning technique that uses a regressor and a threshold value as a classifier [11], we can quickly classify whether a goal state can be reached during planning.

2) Reachability Estimator Network: We train the obstacle-aware reachability estimator network with the training data collected above. The network input is the robot observation o and the output is the estimated TTR. We use a simple three-layer fully-connected network with [500, 200, 100] hidden neurons with each a dropout probability of 0.5. We use the L2 loss between estimated TTR and the V-value label from the training data.

C. RL-RRT

Alg. 2 describes RL-RRT. While the standard RRT algorithm was utilized, modifications were made to efficiently utilize the obstacle-aware reachability estimator and the obstacle-avoiding RL local planner.

Within RL-RRT, the obstacle-aware reachability estimator can provide insight into the best samples to enhance tree growth. However, as we began to use the estimator, it became clear that the obstacle-aware reachability estimator can take longer than the standard Euclidean distance metric to compute (about 0.5 ms vs. 7 μs for Euclidean). Therefore, to enhance computation time in large trees, the estimator

was integrated into a hierarchical nearest neighbor selector. Similar to [2], the method first identifies k_c candidate nodes closest to x_{rnd} using Euclidean distance (Alg. 2, line 8), and subsequently these choices are filtered by the obstacle-aware TTR between each candidate node and x_{rnd}. To alleviate noise in the TTR estimator, we take the average of the TTR between the selected node and $N_{\text{TTR sample}}$=10 target states around x_{rnd}, i.e., within a hypercube of $d_{TTRsample}$=0.3 units (line 10). The node with the lowest average TTR is selected for RRT extension (line 9). In addition, the obstacle-aware reachability estimator can also be used to check whether the randomly sampled state x_{rnd} is reachable from the nearest node n_{nearest}. Recall that the TTR reward in Section III-B is setup such that any x_{rnd} unreachable from n_{nearest}.state has an associated V-value larger than T_{horizon}. As the result, the estimated TTR can be used to prune out x_{rnd} that are un-reachable from the tree within T_{horizon}. However, since the estimated TTR is not exact, we made the pruning probabilistic, i.e., if x_{rnd} is deemed unreachable, it will be pruned with probability P_{prune} (line 10). If x_{rnd} is pruned, it is rejected and a new x_{rnd} is sampled (line 6).

After the nearest node is selected, RL-RRT uses the RL policy π as the local planner (lines 15-24). Specifically, an observation o which includes simulated lidar, robot state, and goal information is made at every policy time step Δt (line 17). This observation is fed to the RL policy, which produces an action that can be used to forward propagate the dynamics to a new state x_{new} (line 18). This process repeats and a new node storing x_{new} is created, and added to the tree every Δ_{tree} seconds (line 21), until x_{new} is in collision, a maximum

extension time is reached (line 20), or x_{rnd} is reached (line 20).

RL-RRT terminates when either the tree reaches the goal or after a fixed amount of computation time is exhausted (line 3). If the tree reaches the goal, a dynamically-feasible motion plan can be returned (line 25).

IV. EVALUATION

To demonstrate RL-RRT, we evaluate our method on three kinodynamic robots in two environments unseen during training, and we experimentally verify the method on a physical differential drive Fetch robot from Fetch Robotics.

A. Setup

The three robots we evaluate are: Car, Asteroid, and Fetch. Car is a kinematic car with inertia [19] with a maximum steering angle $30°$, and a $1.0 \ m/s^2$ maximum acceleration and speed of $1.0 \ m/s$. Asteroid has similar dynamics to those found in the popular video game Asteroid, and we chose it since it is highly kinodynamic, unintuitive for a human to control, and has no known optimal steering function. The details are available in the supplemental materials. The Fetch robot has a radius of $0.3 \, \mathrm{m}$, 1.0 m/s maximum speed and 2.0 rad/s turn rate. The sensor noise is simulated by a zero mean Gaussian with a standard deviation of 0.1 m. We use the Fetch robot as a differential drive platform for on-robot experiments.

All point-to-point policies are trained in the environment depicted in Figure 4a. We evaluate RL policies and plan in two office building environments, Map 1 (Figure 1a) and Map 2 (Figure 1c), which are roughly 15 and 81 times larger than the training environment, respectively. Map 1 is is generated from a floor plan, while Map 2 is generated using a noisy SLAM of the Fetch physical testbed where we ran the experiments. These environments include parts that are cluttered, as seen in Map 1, and very narrow corridors, such seen in Map 2.

We compare RL-RRT to SST [15], a state of the art steering function free kinodynamic motion planner. For Fetch robot, we also compare to RRT with Dynamic Window Approach (DWA) [7] as local planner (denoted RRT-DW). Additionally, we test disabling the clearance term of DWA, essentially turning it into a MPC-based steering function (denoted RRT-S). All experiment are repeated 50 times. Besides AutoRL training, all computation was done on an Intel Xeon E5-1650 @ 3.6GHz using TensorFlow 1.x (Google release) and Python 2.7. AutoRL policies were implemented with Google Vizier [10] and TFAgents [25].

B. AutoRL Policy Performance

We use pre-trained P2P policies for Fetch [4] and Car [8] robots. Their short description is available in the Appendix. The Asteroid P2P policy is original to this paper. All agents are trained with AutoRL over DDPG [4]. The goals are randomly placed within $10 \, \mathrm{m}$. We train 100 agents in parallel

Robot	Confusion Matrix		Prec. (%)	Recall (%)	Accur. (%)
		True (%)			
Fetch	Predicted (%)	42.7 21.6 3.6 32.1	66.4	92.2	74.8
Car	Predicted (%)	44.5 14.2 4.8 36.5	75.8	90.2	81.0
Asteroid	Predicted (%)	26.5 16.3 9.6 47.6	61.9	73.4	74.1

TABLE I
REACHABILITY ESTIMATOR CONFUSION MATRIX, PRECISION, RECALL, AND ACCURACY IN THE TRAINING ENVIRONMENT.

over 10 generations as in [4]. The training took roughly 7 days.

Figure 2 shows the success rate of the P2P agents compared to goal distance. Notice that when the goal distance is $10 \, \mathrm{m}$ or farther than the trained policy, the performance degrades. We also notice that the Car policy is best performing, while the Asteroid policy is the most challenging. These results show that AutoRL produces, without hand-tuning, effective local planners, i.e., both a steering function and an obstacle avoidance policy for a variety of robot dynamics.

C. Reachability Estimator Performance

The obstacle-aware reachability estimator is trained in the training environment with goals sampled within $20 \, \mathrm{m}$ from the initial states, twice the distance used for P2P training. The estimator network was trained on 1000 episodes with about 100,000 samples. Data generation takes about 10 minutes. The reachability thresholds are 20 seconds for differential drive and Asteroid, and 40 seconds for Car. Each estimator was trained over 500 epochs and took about 30 minutes.

Accuracy of the models is between 70% and 80% (Table I). Notice that a high recall means that the estimator misses fewer nodes, and suggests that the paths RL-RRT produces should be near-optimal. On the other hand, relatively low precision implies that RL-RRT will explore samples that end up not being useful. This means that we can speed-up RL-RRT further by learning a more precise predictor.

The reachability estimator overestimates the TTR of reachable states across all robots (Fig. 3). However, overestimation disappears when trained and evaluated only on reachable states (see Fig. 1 in Appendix for more detail). This suggests that the overestimation of TTR is likely due to the TTR cost heuristic uses a penalty for states unreachable within T_{horizon}. We leave identifying better TTR cost heuristics and estimator network architectures for future work.

In general, the estimator captures the regions of start states that cannot reach the goal (blue dot) (Fig. 4). This is most visible at the bottom right region of the environment, which has a TTR larger than the 40s horizon which indicates that the policy failed to escape that region. We also see that the estimated TTR captures the dynamics of Car robot, i.e., since the goal orientation is facing right, it takes less time to reach the goal from the left, top or bottom than from the right. Note that the network is never trained on trajectories that start inside of obstacles and thus cannot accurately predict
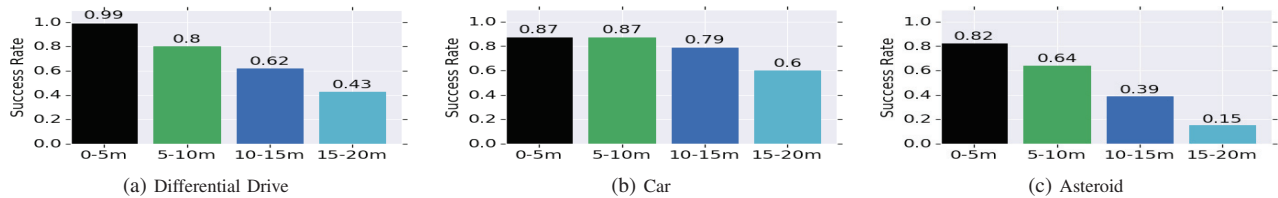
(a) Differential Drive

(b) Car

(c) Asteroid

Fig. 2. AutoRL P2P navigation success rate as a function of start and goal distance for (a) Fetch, (b) Car and (c) Asteroid robot. The success rates are evaluated in Map 1 with randomly sampled start and goal states.
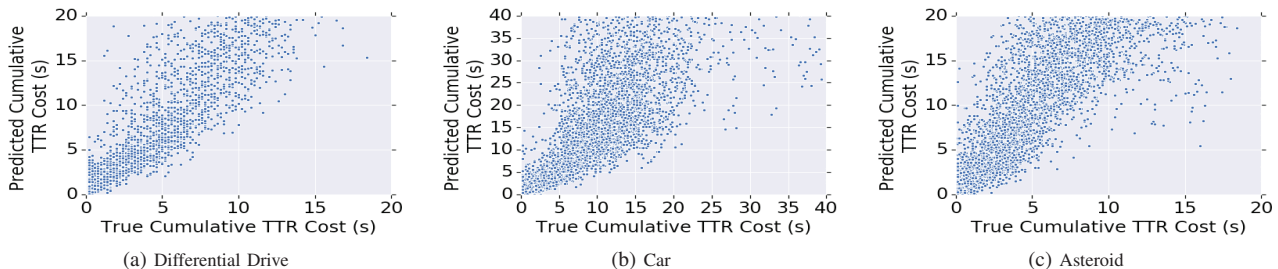

(a) Differential Drive

(b) Car

(c) Asteroid

Fig. 3. Predicted cumulative future time to reach cost v.s. true value for various robots.

(a) Training environment (22.7 x 18.0 m)
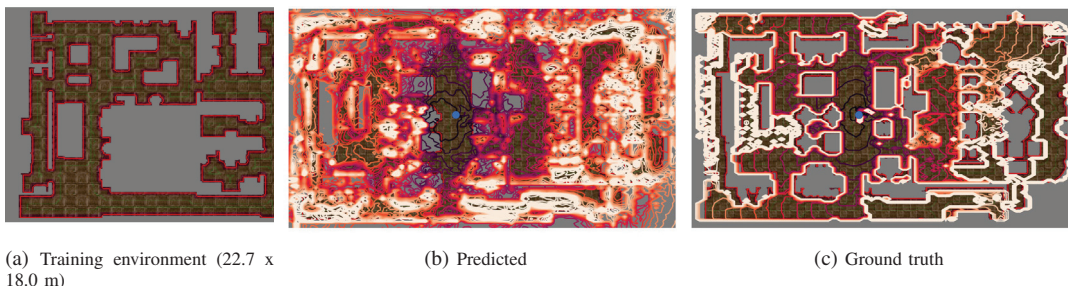
(b) Predicted

(c) Ground truth

Fig. 4. (a) The training environment. Contour plot of (b) Predicted future cumulative time to reach cost v.s. (c) the true value for Car to reach the goal near the center marked by the blue dot. The white regions have time to reach value over the 40s horizon, i.e., un-reachable. All start states and the goal have 0 as linear speed and orientation.

TTR starting from those states, an event which should not occur in sampling-based planning.

D. Planning Results

RL-RRT finds a solution faster than SST for all three robots in both environments (Fig. 5a, 5b, 5c). Note that Car shows the best improvement over the baseline (up to 2.3 times faster), which matches the high success rate of the P2P Car policy. Conversely, the least improvement is for Asteroid, which is the most challenging for the RL agent. Figure 5a also shows that RL-RRT finds a solution faster than steering function-based methods, where DWA was used as the steering function (yellow, RRT-S) and obstacle-avoiding steering function (red, RRT-DW). These results are expected as RL-RRT learns a obstacle-avoiding local planner that can often go through very narrow corridors and move around corners (Figure 1a). In comparison, DWA often gets stuck around corners. To separate the impact of the RL local planner as compared to the reachability estimator, we tested RL-RRT without the estimator and use Euclidean distance to

identify the nearest state in the tree instead. Figures 5a, 5b and 5c show that RL-RRT without the reachability estimator (magenta curves) performs worse than RL-RRT for all robots. This is expected as the reachability estimator prunes potentially infeasible tree-growth, thereby biasing growth towards reachable regions. Also, the reachabilty estimator encodes the TTR and is thus more informative than the Euclidean distance for kinodynamic robots such as Asteroid.

The finish time of trajectories identified by RL-RRT are significantly shorter (up to 6 times shorter) than SST for all robots (Fig. 5d, 5e, 5f) and comparable to RRT-DWA and RRT-S on differential drive. This is expected as SST does not use steering functions. Instead, it randomly propagates actions, resulting in a "jittery" behavior (visible in Figure 1a) and long finish time. The comparable finish time with steering function-based methods show that RL-RRT learns a near-optimal steering function.

E. Physical Robot Experiments

In order to verify that the RL-RRT produces motion plans that can be used on real robots, we executed the motion
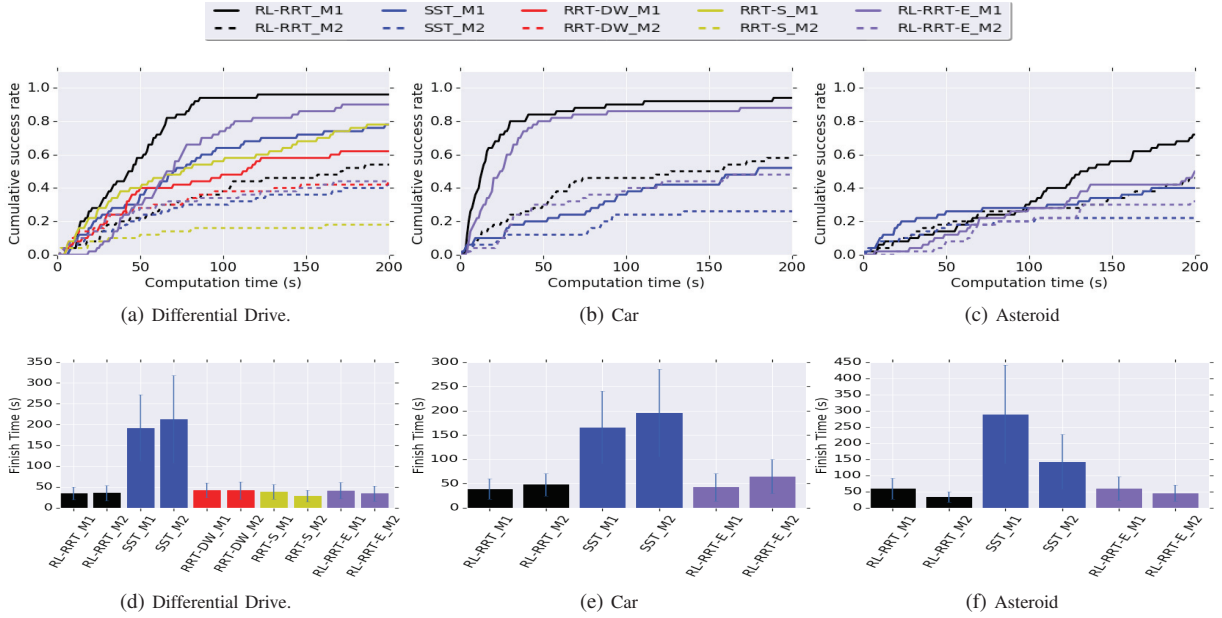
Fig. 5. Success rate (top) and Finish time (bottom) of RL-RRT (black) compared to, SST (blue), RRT-DW (red, RRT with DWA obstacle-avoiding steering function), RRT-S (yellow, RRT with DWA as the steering function) and RL-RRT-E (magenta, RL-RRT using Euclidean distance instead of the reachability estimator) in Map 1 (M1) and Map 2 (M2).

plans on the Fetch robot (Figure. 1b) in Map 2 environment. We ran 10 different motion plans, repeated 3 times. Figure 1c presents one such trajectory. The straight line distance between the start and goal is 20.8 m. In green are tree nodes for a path, and the blue line is the executed robot path with the P2P AutoRL policy. We notice two things. First, the path is similar to the one humans would take. The shortest path leads through cubicle space, which is cluttered. Because the P2P policy does not consistently navigate the cubicle space, the TTR estimates are high in that region and the tree progress slowly in that area. At the same time, in the uncluttered space near the start position (left and right) the tree grows quickly. The executed trajectory (in blue) stays close to the planned path. Enclosed video contains the footage of the robot traversing the path.

V. DISCUSSION

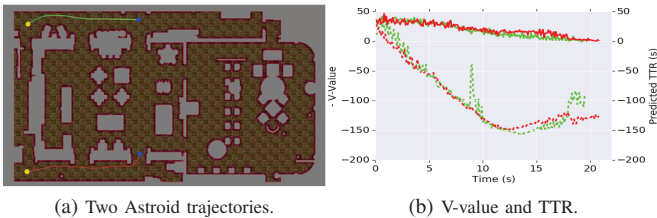(a) Two Astroid trajectories.   (b) V-value and TTR.

Fig. 6. (a) Two trajectories (green and red) of the Asteroid robot from the yellow dots to blue dots. (b) The corresponding predicted TTR (solid lines) and the negative of V-value from DDPG's critic net (dashed lines).

Deep actor-critic RL methods approximate the cumulative future reward, i.e., state-value function with the critic net. Intuitively, the state-value function captures the progress towards the goal and may be used as a distance function

during planning. Here we show that this is *not* the case when proxy rewards are used. AutoRL uses proxy rewards (shown in Section III-A) since they significantly improve learning performance, especially for tasks with sparse learning sigals such as navigation [4]. Fig 6a shows examples of two Asteroid trajectories and Fig. 6b shows the corresponding the estimated TTR (solid lines) and negative of DDPG state-value function extracted form the critic net (dashed lines). The obstacle-aware reachability estimator correctly predicted the TTR while the DDPG's critic net has a significant local maximum, thus unsuitable as a distance function. This finding motivated the supervised reachability estimator.
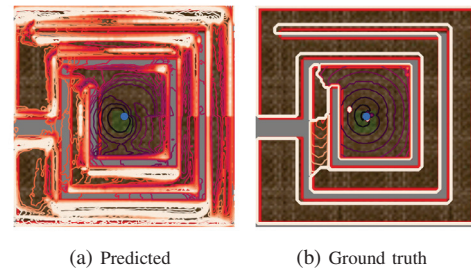

(a) Predicted   (b) Ground truth

Fig. 7. Contour plot of (a) Predicted future cumulative time to reach cost v.s. (b) the true value for Car to reach the goal near the center marked by the blue dot. The white regions have time to reach value over the 40s horizon, i.e., un-reachable. All start states and the goal have 0 as linear speed and orientation. The environment size is 50 m by 40 m.

One limitation of RL-RRT is that the obstacle-aware reachability estimator approximates reachability using only local information such as simulated lidar measurements around the robot. However, the true reachability is often impacted significantly by large-scale obstacle structures. Figure 7 demonstrates this limitation. The ground truth shows that

the Car policy generally fails to reach the goal outside of the center box due to the complex maze-like obstacles (Figure 7b). The reachability estimator failed to predict this as some regions outside of the center box are incorrectly predicted as reachable (Figure 7a). On the other hand, we also demonstrated that the estimator performs well when the training and planning environments are similar (Figure 4). This suggests that the reachability estimator should to be trained in environments similar to the planning environment or perform online adaptation/learning during planning. We leave the latter to future work.

VI. CONCLUSIONS

This paper contributes RL-RRT, a kinodynamic planner which works in three steps: 1) learning obstacle-avoiding local planner; 2) training an obstacle-aware reachability estimator for the learned local planner; and 3) using the estimator as the distance function and to bias sampling in RRT. Unlike traditional kinodynamic motion planners, RL-RRT learns a suitable steering and distance function. The robot is trained once, and the policy and estimator transfer to the new envrionments. We evaluated the method on three kinodynmic robots in two simulated environments. Compared to the baselines, RRT plans faster and produces shorter paths. We also verified RL-RRT on a physical differential drive robot. For future work, following PRM-RL, we plan to improve the noise robustness of RL-RRT by Monte Carlo roll-outs during tree extensions. We also plan to identify better TTR cost heuristics, network architectures and online adaptation of the reachability estimator.

ACKNOWLEDGMENT

REFERENCES

[1] R. Allen and M. Pavone. A real-time framework for kinodynamic planning with application to quadrotor obstacle avoidance. In *AIAA Guidance, Navigation, and Control Conference*, page 1374, 2016.

[2] H.-T. L. Chiang, A. Faust, S. Satomi, and L. Tapia. Fast swept volume estimation with deep learning. In *Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR)*, page To appear, 2018.

[3] H.-T. L. Chiang and L. Tapia. Colreg-rrt: An rrt-based colregs-compliant motion planner for surface vehicle navigation. *Robotics and Automat. Lett.*, 3(3):2024–2031, 2018.

[4] L. Chiang, A. Faust, M. Fiser, and A. Francis. Learning navigation behaviors end-to-end with autorl. *IEEE Robotics and Automation Letters (RA-L)*, 2019.

[5] T. Fan, X. Cheng, J. Pan, P. Long, W. Liu, R. Yang, and D. Manocha. Getting robots unfrozen and unlost in dense pedestrian crowds. *Robotics and Automat. Lett.*, 2019.

[6] A. Faust, O. Ramirez, M. Fiser, K. Oslund, A. Francis, J. Davidson, and L. Tapia. PRM-RL: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 5113–5120, Brisbane, Australia, 2018.

[7] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robot. & Automation Mag.*, 4(1):23–33, 1997.

[8] A. Francis, A. Faust, H.-T. L. Chiang, J. Hsu, J. C. Kew, M. Fiser, and T.-W. E. Lee. Long-range indoor navigation with prm-rl. *arXiv preprint arXiv:1902.09458*, 2019.

[9] A. Francis, A. Faust, H.-T. L. Chiang, J. Hsu, J. C. Kew, M. Fiser, and T.-W. E. Lee. Long-range indoor navigation with prm-rl. volume 1, 2019.

[10] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley. Google vizier: A service for black-box optimization. In *Proc. of ACM Intl. Conference on Knowledge Discovery and Data Mining*, pages 1487–1495. ACM, 2017.

[11] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[12] Y. Kato, K. Kamiyama, and K. Morioka. Autonomous robot navigation system with learning based on deep q-network and topological maps. In *2017 IEEE/SICE International Symposium on System Integration (SII)*, pages 1040–1046, Dec 2017.

[13] A. Layek, N. A. Vien, T. Chung, et al. Deep reinforcement learning algorithms for steering an underactuated ship. In *IEEE Int. Conf. on Multisensor Fusion and Integration for Intell. Sys. (MFI)*, pages 602–607. IEEE, 2017.

[14] Y. Li and K. E. Bekris. Learning approximate cost-to-go metrics to improve sampling-based motion planning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 4196–4201. IEEE, 2011.

[15] Y. Li, Z. Littlefield, and K. E. Bekris. Asymptotically optimal sampling-based kinodynamic planning. *Int. J. Robot. Res.*, 35(5):528–564, 2016.

[16] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[17] S. Liu, M. Watterson, K. Mohta, K. Sun, S. Bhattacharya, C. J. Taylor, and V. Kumar. Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments. *Robotics and Automat. Lett.*, 2(3):1688–1695, 2017.

[18] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[19] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Trans. on intel. vehicles*, 1(1):33–55, 2016.

[20] L. Palmieri and K. O. Arras. Distance metric learning for rrt-based motion planning with constant-time inference. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 637–643. IEEE, 2015.

[21] M. Pfeiffer, M. Schaeuble, J. I. Nieto, R. Siegwart, and C. Cadena. From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots. *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 1527–1533, 2017.

[22] J. M. Phillips, N. Bedrossian, and L. E. Kavraki. Guided expansive spaces trees: A search strategy for motion-and cost-constrained state spaces. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, volume 4, pages 3968–3973. IEEE, 2004.

[23] A. Richards, T. Schouwenaars, J. P. How, and E. Feron. Spacecraft trajectory planning with avoidance constraints using mixed-integer linear programming. *J. of Guidance, Control, and Dynamics*, 25(4):755–764, 2002.

[24] E. Schmerling, L. Janson, and M. Pavone. Optimal sampling-based motion planning under differential constraints: the drift case with linear affine dynamics. In *IEEE Conf. on Decision and Control (CDC)*, pages 2574–2581. IEEE, 2015.

[25] Sergio Guadarrama, Anoop Korattikara, Oscar Ramirez, Pablo Castro, Ethan Holly, Sam Fishman, Ke Wang, Ekaterina Gonina, Chris Harris, Vincent Vanhoucke, Eugene Brevdo. TF-Agents: A library for reinforcement learning in tensorflow. https://github.com/tensorflow/agents, 2018.

[26] L. Tai, G. Paolo, and M. Liu. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 31–36, Sept 2017.

[27] D. J. Webb and J. Van Den Berg. Kinodynamic rrt*: Asymptotically optimal motion planning for robots with linear dynamics. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 5054–5061. IEEE, 2013.

[28] W. J. Wolfslag, M. Bharatheesha, T. M. Moerland, and M. Wisse. RRT-colearn: towards kinodynamic planning without numerical trajectory optimization. *Robotics and Automat. Lett.*, 3(3):1655–1662, 2018.

[29] J. Zhang, J. T. Springenberg, J. Boedecker, and W. Burgard. Deep reinforcement learning with successor features for navigation across similar environments. In *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, pages 2371–2378. IEEE, 2017.