Learning-Guided Exploration for Efficient Sampling-Based Motion Planning in High Dimensions

Liam Schramm and Abdeslam Boularias

***Abstract*— Optimal motion planning is a long-studied problem with a wide range of applications in robotics, from grasping to navigation. While sampling-based motion planning methods have made solving such problems significantly more feasible, these methods still often struggle in high-dimensional spaces wherein exploration is computationally costly. In this paper, we propose a new motion planning algorithm that reduces the computational burden of the exploration process. The proposed algorithm utilizes a guidance policy acquired offline through model-free reinforcement learning. The guidance policy is used to bias the exploration process in motion planning and to guide it toward promising regions of the state space. Moreover, we show that the gradients of the corresponding learned value function can be used to locally fine-tune the sampled states. We empirically demonstrate that the proposed approach can significantly reduce planning time and improve success rate and path quality.**

I. Introduction

Sampling-based optimal motion planning algorithms have become a staple of robotics research due to their ability to quickly find viable paths [1]–[3]. However, while these methods may quickly find good paths in low-dimensional problems, they still sometimes struggle to find good-quality paths in high-dimensional spaces due to the exorbitant time it would take to fully explore the state space. The popular planning algorithm Stable Sparse-RRT (SST) [3], for instance, has a probabilistic worst-case time to find a near-optimal path that is exponential in the dimension of the state space.

Model-free Reinforcement Learning (RL) has made significant strides in recent years due to increased availability of computing power, but suffers from a lack of robustness or theoretical guarantees. Many reinforcement learning methods used today, such as Deep Deterministic Policy Gradient (DDPG), have no guarantee of convergence [4], while many others, such as Hindsight Experience Replay (HER), are known to be asymptotically biased [5]. This is especially problematic in robotics where unsafe actions may damage the robot, and we wish to have a guarantee that the robot will solve the task, or at least not take a potentially dangerous path. Additionally, model-free RL algorithms struggle to deal with long-time horizons, sparse rewards, and environments requiring time-efficient exploration. These problems frequently prevent the use of model-free RL for applications where it would otherwise be useful. Finally, training a reinforcement learning agent to convergence may be extremely-resource intensive even when the model does converge.

The authors are with the Computer Science Department of Rutgers University. {lbs105,ab1544}@cs.rutgers.edu. This work is supported by NSF awards IIS-1734492, IIS-1846043 and 2132972.

We propose Policy-guided Stable Sparse-RRT (PSST), a sampling-based kinodynamic planning algorithm that makes use of RL-learned policies while retaining the desirable properties of sampling-based planning algorithms such as Stable-Sparse-RRT (SST) [3]. We do this by having the planner sample actions from the policy some set fraction of the time while exploring with random actions for the rest. Additionally, we perform gradient descent on the sampled states used for the tree extension in order to bias the tree growth towards the goal. Importantly, our method does not require that the learned agent is trained to convergence, and benefits even when the RL agent's performance is very poor. This allows our method to be used when computational resources are not sufficient to train an RL agent to convergence. We find that the proposed method outperforms the state-of-the-art planner SST, standard reinforcement learning methods, and the hybrid method RL-RRT on a variety of challenging kinodynamic tasks.

II. Problem

Optimal motion planning attempts to solve the following problem. Suppose we have a dynamics function that governs the time evolution of a robotic system in a given environment, such that $\dot{x}(t) = f(x(t), u(t))$, where $x(t)$ is a point in the robot's configuration space, $u(t)$ is a point in a control space, and $\dot{x}(t)$ is the time derivative of $x(t)$. Given an initial state $x(0) = s_0$ and a set of goal states G, and a cost function $C(x(t), u(t))$, our goal is to find a control function $u(t)$ and interval $[0, t_{term}]$ that minimizes $\int_0^{t_{term}} C(x(t), u(t))dt$ under the constraint that $x(t_{term}) \in G$. Intuitively, we want to find a sequence of controls that puts our system in a goal state and minimizes the total cost of the corresponding path, assuming that the system is deterministic.

III. Related Work

Anytime sampling-based motion planners such as PRM [6], RRT [1], RRT* [2], and SST [3] have become popular in recent years due to their efficient exploration and relatively good performance in higher dimensions. These methods construct a search tree by randomly sampling nodes, expanding them with random controls, and removing nodes from the tree that have been obsoleted by new paths. These methods are capable of quickly finding solutions to difficult planning problems and can be extended to high-dimensional planning problems. Additionally, SST in particular comes with a number of theoretical guarantees, such as asymptotic δ-robust completeness and asymptotic δ-robust near-optimality. However, its performance deteriorates in higher dimensions,

as its worst-case performance is exponential in both the dimensionality of the state-space and the action space.

Sampling-based planning methods often attempt to minimize their search time through the use of heuristics. Several methods, such as Informed RRT* [7], Deformable RRT* [8], BIT* [9], and RABIT* [10] use heuristics to select where points in the space should be sampled from, in order to reduce the size of the search space. However, this type of selective sampling is not always possible for problems with more complex dynamics. The geometric bound used by these methods (sampling only within the ellipse where it is possible for the optimal path to lie) is only usable when there is a known lower bound on the cost between two points, which is a constant multiple of Euclidean distance. This bound may form only a loose bound, and may not be known for arbitrary dynamics.

Another common approach to solving optimal motion problems is model-free reinforcement learning. Reinforcement learning can solve many planning problems in robotics, such as grasping [5], [11]–[19] and navigation [20]. These methods often scale well to very high-dimensional problems such as Atari [21] and Go [22]. A number of recent methods have also explored methods that attempt to integrate model-free RL with model-based planning methods [23]. For example, Temporal Difference Models (TDMs) [24] predict the displacement vector from the goal state after t steps, and train a policy to minimize the norm of that displacement vector. Universal Planning Networks (UPNs) [25] learn a dynamics model for planning in a latent space, then optimize actions directly with gradient descent in the latent space.

Several methods have been proposed that use machine learning to augment sampling-based motion planning [26] [27] [27] [28] [29] [30]. Arslan and Tsiotras [28] use an adaptive approach to predict if a given sample is collision-free and whether it is likely to improve on the current best path. They use this method to reduce the number of collision checks needed during planning and focus the search on promising regions of the state space. Ichter, Harrison, and Pavone [27] learn a conditional variational autoencoder to generate samples that are likely to lie near the optimal path. They find that their sampling method is able to significantly reduce the number of samples needed by focusing the search to regions of state space where near-optimal paths are likely to occur.

BELT [31] proposes a planning method for using using learned policies to form plans over multiple sequential tasks. This approach uses an imitation-learned goal-conditioned policy together with a classifier that predicts if the point is efficiently reachable. They propose a sampling-based planning algorithm which is similar to performing RRT in a space of task manipulations. The BELT algorithm differs from our proposed method in a number of ways, but most significantly in that BELT uses human demonstrations to train its policy, as opposed to learning them from scratch.

RL-RRT [26] is likely the work most closely related to our approach. It attempts to improve the performance of the sampling-based motion planning algorithm RRT by biasing the sampled control sequences toward ones that avoid collisions. It does this by first training an obstacle-aware policy and a time-to-reach function that is used to estimate whether a given sampled point is reachable. The algorithm then samples points until it finds one that is reachable by the policy, and uses the policy to sample a control sequence that reaches that point. The authors demonstrate that this approach significantly improves both the planning time and path quality of discovered paths compared to SST on several 2D environments.

IV. Proposed Algorithm

The proposed Policy-guided SST (PSST), explained in Algorithm 1, is a kinodynamic motion planning algorithm that leverages reinforcement learning to improve convergence of the original SST algorithm [3]. The SST algorithm iteratively builds a nearest-neighbor graph G that sparsely covers the state space, while taking into account the dynamics transition function f and the presence of obstacles when connecting neighbors together. Due to space constraints, we omit here details related to the original algorithm [3] and focus only on our main contribution, which is related to how states and actions are sampled in the first place. We show in Section V that learning-driven samples lead to dramatic improvements over the original SST.

Offline, we use goal-conditioned reinforcement learning to learn two sub-optimal policies, π and $\tilde{\pi}$, one to move to arbitrarily chosen states, and one to move to arbitrary goals. The only difference between these two policies is that π is trained to navigate to a point in the original state space (e.g, position and orientation), whereas $\tilde{\pi}$ is trained to navigate to goals defined in a subspace (e.g, position only). This distinction is important for applications such as object manipulation, where a task may be defined as "Move the object to location x", and the goal specifies nothing about the position of the robotic arm. Although these policies may be learned by any model-free reinforcement learning algorithm, we opt to train both policies using Soft Actor Critic [32] with Hindsight Experience Replay [5]. Unlike the standard implementation of SAC, we do not set the policy to be deterministic at runtime in order to encourage exploration in the planner. We then use these policies to modify the SST algorithm in two ways. Instead of using purely random actions, during each expansion we randomly select a policy from $\pi, \tilde{\pi}$, and a uniform random policy to draw actions from (function `RLProp`, called in line 7 of Algorithm 1). Additionally, when sampling points to guide the growth of the tree, we perform gradient descent on the sampled points in order to bias the tree towards the goal region (function `Gradient_Descent_Sampling`, called in line 5 of Algorithm 1). We will show in Section V that the proposed algorithm finds near-optimal solutions in several robotic domains even when the guidance policies π and $\tilde{\pi}$ are significantly sub-optimal due to limited training time. Thanks to the use of the state transition function f in the online planning process, we also show that π and $\tilde{\pi}$ can be

trained offline in a domain that is substantially different from the test domain.

The algorithm for expanding the search tree then works as follows; We randomly sample a point S in the state space. We then perform a random number of gradient descent steps on S with respect to $-Q(S, \tilde{\pi}(S), g)$ to find a new point S', where g is the goal, Q is the value function of a pre-trained agent, and $\tilde{\pi}$ is the policy of a pre-trained agent. We select from the nearest-neighbor graph the lowest cost node x within a radius δ of S'. If there are no nodes within δ, we instead take the nearest neighbor as x. We then randomly select one of three policies to expand x; a random state-conditioned policy $\pi(\cdot, S')$ that attempts to reach S', and goal-conditioned policy $\tilde{\pi}(\cdot, g)$ that attempts to reach the goal, and a purely random policy that just selects a random action from a uniform distribution. We expand x by rolling out the policy for a random number of steps in the range $(0, t_{max})$ to obtain a new state x_{new}. The new node is then added to the planning tree using the same algorithm for adding new nodes and deleting old ones described in SST [3]. The described algorithm is layed out in Figure 1

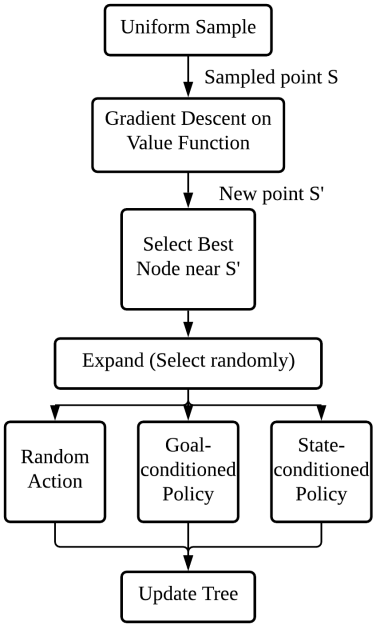

Fig. 1. PSST Algorithm

A. Action Sampling

We replace SST's uniform random control sampling with sampling from a mixture of these policies, using random controls, the goal-conditioned policy $\tilde{\pi}$, and the state-conditioned policy π. Like RL-RRT, the state-conditioned policy attempts to reach the sampled point to which the current node is a nearest neighbor, while the goal-conditioned policy always tries to reach the goal region, defined as a point in a subspace of the original state space. Note that the goal is an input of policy $\tilde{\pi}$, which is trained offline with HER [5] to aim for any given goal region, even though $\tilde{\pi}$ is sub-optimal.

The motivation for the use of two distinct policies is to guarantee a certain density of both efficient exploration and efficient exploitation. RL-RRT demonstrated that learning to navigate towards random points is an effective exploration strategy. However, we were concerned that in high dimensions the probability of sampling a point near a goal state decays exponentially to zero, and in some tasks it is not trivial to sample the goal state directly (e.g. grasping). Thus, the purpose of the goal-conditioned policy is to navigate directly to the goal once the goal becomes reachable from the tree. Figure 2 visualizes the different kinds of expansions used by the proposed method. In the case where the goal space and state space are the same, the same neural network may be used for both π and $\tilde{\pi}$, saving training time.

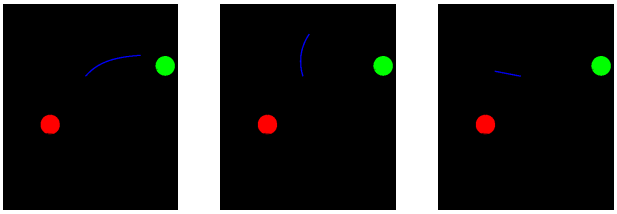

Fig. 2. The three kinds of node expansions in PSST – Goal seeking (left), random-state-seeking (center), and random (right). The task is to navigate from start (red) to goal (green). The blue path denotes the expansion of the current node using the selected policy. Sampling from a mixture of policies allows PSST to efficiently trade off between exploration and exploitation. Random-state-seeking expansions utilize a Voronoi bias to efficiently reach new regions of state space while avoiding obstacles, while goal-seeking expansions encourage the tree to grow towards the goal.

B. State Sampling with Gradient Descent

Secondly, we propose drawing samples from the uniform random distribution over the state space, and then performing gradient descent on the sampled points before connecting them to their nearest neighbors in the search graph. The motivation for this is that many interesting problems have no simple way of sampling from the set of reachable goal states. By sampling random states and then moving them closer to the goal by performing gradient descent on an RL-learned value function, we can bias the growth of the search tree towards goal states without an explicit goal-sampler.

Since the policy and value functions are both neural networks, it is simple to calculate the gradient of the value $\nabla_s Q(s, \pi(s), g)$ with respect to the state s with backpropagation. We sample the number of gradient descent steps to take from a geometric distribution with stopping probability θ and subtract one ($n \sim \text{Geom}(\theta) - 1$). This ensures that we always retain at least a θ chance of taking 0 gradient descent steps. Thus a portion θ of the samples are drawn from the uniform random distribution, which means we retain all the asymptotic-sampling properties of SST. It also ensures that for any sampled point, we have a lower-bounded chance of converging to within ϵ of the local minimum for any ϵ. If the minimum of the learned value function is in a goal state (which is very likely as these are terminal states with maximum reward), then we have a lower-bounded chance of sampling arbitrarily close to the goal state. We expect this approach to be most useful in high-dimensions, where the proportion of goal-states to the total volume may become extremely small. As we can see from Figure 3, performing gradient descent on the value with respect to the state increases the density of sampled points near the goal. The more gradient descent steps are performed, the greater the likelihood that the search tree will expand the node closest to the goal. Thus we can see the average number of gradient descent steps acts as a greediness parameter, in that it determines how aggressively the search will focus on points near the goal.

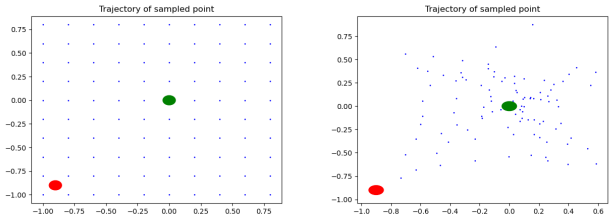

Fig. 3. Distribution of sample points after 0 (left) and 3 (right) gradient descent steps. Sampled points (blue dots) move towards the goal (green circle) as more gradient descent steps are performed, focusing the search on the goal region

Gradient descent on the sampled states is performed by using the Q-value function of policy $\tilde{\pi}$ learned offline through HER. Using the standard notation for goal-conditioned reinforcement learning, we represent $Q(s,a,g)$ as a neural network that takes as inputs a starting state s, an action a, and a goal state g and returns a Q-value. By using the back-propagation algorithm, the gradient $\nabla_s Q(s,a,g)$ is computed analytically and used to push sampled states s toward the goal ($s \leftarrow s + \alpha \nabla_s Q(s,a,g)$). Here again, learned function Q and its gradient $\nabla_s Q(s,a,g)$ are generally inaccurate due to limited training time, and yet significantly improve SST by pointing to the direction of the goal state.

Function `RL_Prop` $(x_{selected}, x_{target}, f, g, \pi, \tilde{\pi}, t_{max}, \epsilon_{rand}, \epsilon_{policy})$**:**

$t_{prop} = SampleUniform(0, t_{max})$
Randomly select policy to use as π_{sample}.
$P(\pi_{sample} = \pi(., x_{target})) = \epsilon_{policy}$.
$P(\pi_{sample} = random) = \epsilon_{rand}$.
$P(\pi_{sample} = \tilde{\pi}(., g)) = 1 - (\epsilon_{policy} + \epsilon_{rand})$.
return $x_{new} = x_{selected} + \sum_{t=0}^{t_{prop}} f(x(t), \pi_{sample}(x(t)))\delta t$

Function `Gradient_Descent_Sampling` (g, Q, α, θ)**:**

$n = SampleGeometric(\theta) - 1$
$s = SampleRandomState()$
for $i = 0;\ i < n;\ i = i + 1$ **do**
  $s \leftarrow s + \alpha \nabla_s Q(s,a,g)$; `// Perform gradient descent on the (negative) value function`
return s

V. Experimental Results

We evaluate our method on a variety of challenging tasks, including two 2D kinodynamic planning problems and four tasks with the *Fetch Robot* arm simulated in *MuJoCo*. We also test on a modified version of each of the 2D environments, aimed at showing that the RL agent continues to be useful to the planner even when distributional shift causes the agent's performance to suffer.

All experiments (including both model training and planning) were performed on an Alienware Aurora-R9, with 8 i7 CPU cores. The GPU was not used for model training, as CPU parallelization proved to be more efficient. All RL agents were trained with SAC, using a 4-layer network with hidden width of 256 for both policy and critic networks, a learning rate of .001, and an entropy regularization term of

Algorithm 1: Policy-Guided SST (PSST)

Inputs : x_0, the initial state; g, a goal state; ϵ, a success threshold; $Q(s,a,g)$, a value function; α, a step-size for the gradient descent; θ: a stopping probability for a geometric distribution over the number of gradient-descent steps; f, a state transition function; π, a goal-conditioned policy; $\tilde{\pi}$, a goal-conditioned policy in the subspace of the goal constraints; t_{max}, maximum expansion length; ϵ_{rand}, probability of random expansions; ϵ_{policy}, probability of π-guided expansions;

Output: $(x_0^*, \mu_0^*, \ldots, x_{t_{term}}^*, \mu_{t_{term}}^*)$ a path of states-actions that satisfies $\|x_{t_{term}}^* - g\| \leq \epsilon$;

$V_{active} \leftarrow x_0, V_{inactive} \leftarrow \emptyset, V \leftarrow (V_{active} \cup V_{inactive})$ $E \leftarrow \emptyset, G = \{V, E\}$
$s_0 \leftarrow x_0, s_0.rep = x_0, S \leftarrow \{s_0\}$;
for *N iterations* **do**
  $s_{sample} \leftarrow$ `Gradient_Descent_Sampling` (g, Q, α, θ)
  $x_{selected} \leftarrow$ `Best_Near`$(V_{active}, s_{sample}, \delta_{BN})$
  $x_{new} \leftarrow$ `RL_Prop`$(x_{selected}, s_{sample}, f, g,$
  $\pi, \tilde{\pi}, t_{max}, \epsilon_{rand}, \epsilon_{policy})$
  if `CollisionFree`$(x_{selected} \rightarrow x_{new})$ **then**
    if `Is_Node_Locally_the_Best`(x_{new}, S, δ_s) **then**
      $V_{active} \leftarrow V_{active} \cup \{x_{new}\}$;
      $E \leftarrow E \cup \{x_{nearest} \rightarrow x_{new}\}$;
      `Prune_Dominated_Nodes` $(x_{new}, V_{active}, V_{inactive}, E)$
Return the path with the minimum cost from x_0 to g in constructed graph G;

0.01. 50 trials were run on all environments. As one of our main objectives is demonstrating that the proposed method PSST works well with suboptimal policies, we do not train all policies to convergence.

We compare our method to two baselines. Since our method requires some degree of training before planning begins, we decided it would be unfair to compare to SST directly. Instead, as a baseline we run both SST and the pre-trained goal-conditioned RL policy and take the better solution. We can therefore take improvements over this baseline to be a result of our method, and not a result of the RL policy alone outperforming SST. We also compare to RL-RRT [26]. In all experiments, the RL-RRT policy is trained for the same amount of time PSST's policy is.

We aim to answer two main questions about the proposed method. Firstly, does it outperform SST and pure reinforcement learning methods on challenging tasks? We test on an obstacle-based navigation problem and several difficult object-manipulation problems to measure the proposed method's effectiveness. Secondly, how good does the learned policy need to be in order for our method to see noticeable improvement over SST and pure reinforcement learning methods? We evaluate by examining how PSST's performance changes as the reinforcement learning agent

is trained, as well as its performance when the policy is trained on one environment, and then transferred to a similar environment.

A. Obstacle Maze task

Similar to the RL-RRT test environments [26], we train agents to navigate a two-dimensional area and reach a goal without colliding with obstacles. Each agent observes its position and 16 laser rangefinder sensors that inform it of the locations of obstacles. This environment is a modification of the Limited-Range-2D environment in gym-extensions [33]. 1000 iterations are allowed for planning.

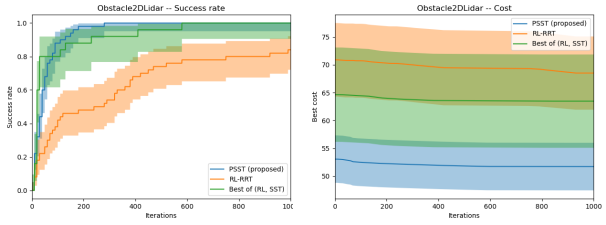

Fig. 4. Success rates and path cost on 2D obstacle environment. SST is competitive with PSST on success rate, but PSST outperforms it on path cost

We find from Figure 4 that while PSST only slightly outperforms the better of SST and pure RL in terms of success rate, it reduces the average path cost by nearly 20%.

B. Fetch Robot tasks

Lastly, we test our algorithm on a series of challenging object manipulation tasks with a simulated Fetch robot arm. The tasks are as follows:

FetchPush: Use a robotic arm to push a block to a desired location. **FetchPickAndPlace**: Use a robotic arm to pick up an object from a table and move it to a desired point in space. **FetchSlide**: Use a robotic arm to slide an object across a table so that it stops at the desired point

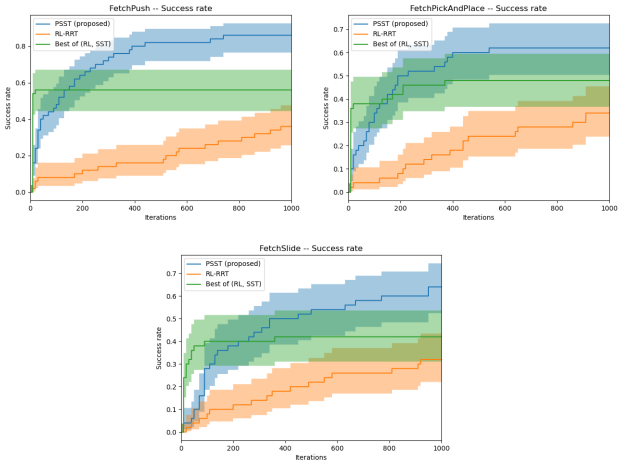

Fig. 5. Success rates on Fetch robot arm environments. The proposed method PSST performs best on all tasks

We find that in all Fetch tasks, PSST (proposed) outperforms both RL-RRT and the better of SST and the baseline RL agent. Additionally we find that PSST delivers paths of comparable or better quality than the other methods. We hypothesize that this improvement is due in part to the improved exploration ability of PSST. PSST benefits from the Voronoi bias of SST, however the policy's tendency to select actions that move towards the goal (even when imperfectly) and the similar tendency of gradient descent sampling helps to focus the search on the region of space that is likely to contain good paths. We attribute the gap in performance between RL-RRT and PSST primarily to the difference in the dimensionality of the goal space. RL-RRT's goal sampling attempts to reach a specific point in a high-dimensional (here, 30+ dimensions) state space, while PSST's goal sampling attempts to reach a 3 dimensional goal, which is much easier to achieve.

C. 2D Environments

We examine two 2D kinodynamic planning problems.

Simple 2D: This environment is a simple two-dimensional double-integrator control problem with no friction. The state is 4-dimensional, including the agents position and velocity. The robot controls its acceleration vector directly, and is attempting to reach a given position. 500 iterations are allowed for planning.

Asteroids: This problem is similar to the Atari game "Asteroids". The state is 5-dimensional, including position, velocity, and rotation. The robot controls its rotation and forward/backward acceleration to reach a given position. It experiences drag proportional to its velocity. 500 iterations are allowed for planning.

Since all three compared methods are able to solve this problem with close to a 100% success rate, we focus on the solution quality instead, measured in the number of steps required to reach the target.

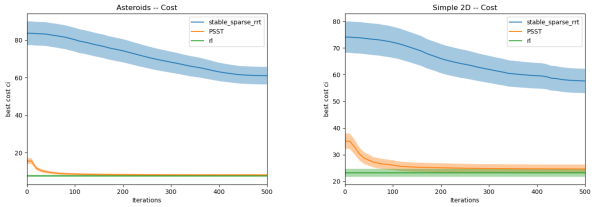

Fig. 6. Performance on 2D environments. RL finds the optimal solution for these problems

As we can see, the reinforcement learning policy is able to produce a near-optimal solution that is difficult to outperform with a planner. Next, we explore how the the proposed method PSST behaves when this the optimal solution is changed due to distribution shift.

D. 2D Environments under distribution shift

Here, we introduce two changes to the 2D environments. Controls become less responsive as the agents move away from the center of the work space, to a minimum of .05 times the original control vector. Additionally, controls are biased slightly off-center by $1/10$th of the maximum control,

causing the agents to drift unless this effect is compensated for. The RL agent is not retrained on an environment with these changes. These modifications are introduced to test the planner's ability to make use of a sub-optimal policy.

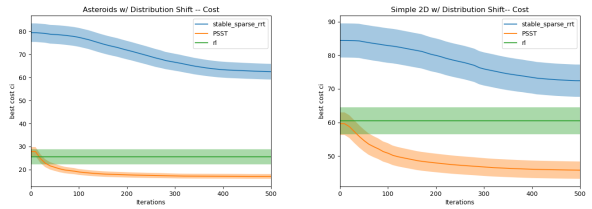

Fig. 7. Performance on 2D environments after distributional shift. RL is suboptimal on the new environments, but PSST is still able to use the suboptimal policies to find a near-optimal solution

We find that while the RL agent's performance suffers slightly due to these changes, PSST is able to adapt while still making use of the agent's bias towards actions that performed well on the previous environments, leading it to outperform both SST and the learned agent. This implies that PSST is able to make use of the agent's bias towards certain actions to narrow the search space, even when those actions may be suboptimal. This behavior illustrates one of the key benefits of this approach – that the planner is able to provide robustness when the RL agent is an incomplete solution to the problem, due to any number of reasons such as incomplete convergence, instability, or distribution shift.

E. Impact of policy quality on performance

One question posed by this method is how good the policy has to be in order to see noticable improvements. We attempt to answer this question by running the FetchPush experiment multiple times with policies that have been trained for varying lengths of time.

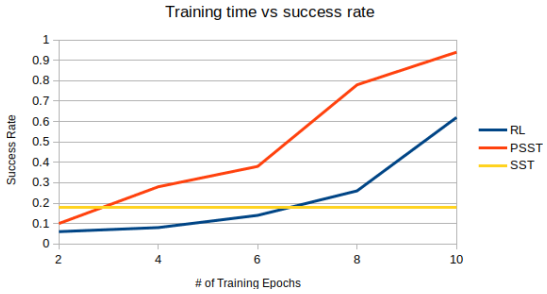

Fig. 8. PSST (proposed) success rates as a function of training time. PSST improves over SST even with minimally trained policies, and benefits from additional training even more than the base policy

Surprisingly, we find that the proposed method PSST shows improvement from training earlier and faster than the policy itself – the proposed method PSST shows noticeable improvement between epochs 2 and 4, while the pure RL agent does not see noticeable improvement until epoch 6. This is possibly because the policy learns to move in the right general direction before it learns to reliably reach the goal. While "the right general direction" will not lead to a non-trivial number of successes for the agent until its control improves further, it can be a useful heuristic for PSST. Thus we see the surprising result that the policy benefits PSST enough to outperform SST *before* the policy even appears to be improving by the "reward-per-episode" metric. Interestingly, PSST also seems to derive benefit from training at a faster rate than the baseline policy. The gap between PSST and the baseline policy grows as the policy receives more training, until PSST begins to approach a 100% success rate.

F. Sample Solutions to Double Integrator Problem

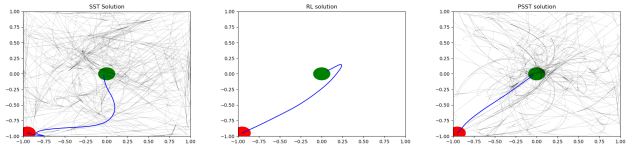

Fig. 9. Sample solution for double integrator problem. **Task**: Navigate from start (red) to goal (green). **Gray**: Paths in the search tree. **Blue**: Final path returned by the method **Left**: SST, **Center**: RL, **Right**: PSST (Proposed)

The focusing effect of the proposed PSST algorithm can be seen in Figure 9, which demonstrates its search for a solution to a double integrator problem. SST has no way of biasing the search towards the goal, so it explores fully randomly, finding a poor path that does not attempt to seek the goal directly. The RL agent successfully aims for the target, but misses it due to non-convergence and is forced to loop back to hit the target. PSST's solution also aims for the target, but is able to locate a near-optimal path more easily due to the bias from the policy and gradient descent sampling. Note that PSST's explored paths (grey) tend to focus around the goal state more than SST's

VI. Conclusion

In this work, we proposed PSST, a sampling-based kinodynamic planning algorithm that leverages trained RL agents to narrow its search and improve efficiency. Our approach trains two RL agents, one for efficient exploration, and one for goal-seeking, and samples from agents' policies and a purely random policy in order to efficiently explore the space. Additionally, our approach optimizes the value function of sampled points by gradient descent in order to bias the growth of the search tree towards the goal. We evaluated this approach on a number of tasks, including several challenging object-manipulation tasks using a simulated robot arm. On all tasks, we demonstrated a significant increase in performance against both pure reinforcement learning and the state-of-the-art planning algorithm SST.

In the future, we plan to expand this work by using the planner as a supervisor to train the policies and the value functions from scratch, hopefully allowing us to make stronger convergence guarantees than other common reinforcement learning algorithms offer. Finally, we plan to demonstrate the returned paths on real robotic systems in the context of object grasping and manipulation in cluttered environments.

REFERENCES

[1] S. LaValle, "Rapidly-exploring random trees : a new tool for path planning," *The annual research report*, 1998.

[2] S. Karaman and E. Frazzoli, "Sampling-based Algorithms for Optimal Motion Planning," *arXiv:1105.1186 [cs]*, May 2011, arXiv: 1105.1186. [Online]. Available: http://arxiv.org/abs/1105.1186

[3] Y. Li, Z. Littlefield, and K. E. Bekris, "Asymptotically Optimal Sampling-based Kinodynamic Planning," *arXiv:1407.2896 [cs]*, Feb. 2016, arXiv: 1407.2896. [Online]. Available: http://arxiv.org/abs/1407.2896

[4] 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." in *ICLR*, Y. Bengio and Y. LeCun, Eds., 2016. [Online]. Available: http://dblp.uni-trier.de/db/conf/iclr/iclr2016.html#LillicrapHPHETS15

[5] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight Experience Replay," *arXiv:1707.01495 [cs]*, Feb. 2018, arXiv: 1707.01495 version: 3. [Online]. Available: http://arxiv.org/abs/1707.01495

[6] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, Aug. 1996. [Online]. Available: http://ieeexplore.ieee.org/document/508439/

[7] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed RRT*: Optimal Sampling-based Path Planning Focused via Direct Sampling of an Admissible Ellipsoidal Heuristic," *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2997–3004, Sept. 2014, arXiv: 1404.2334. [Online]. Available: http://arxiv.org/abs/1404.2334

[8] F. Hauer and P. Tsiotras, "Deformable rapidly-exploring random trees," Cambridge, Massachusetts, USA., 2017. [Online]. Available: https://rss2017.lids.mit.edu/program/papers/17/

[9] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Batch Informed Trees (BIT*): Sampling-based Optimal Planning via the Heuristically Guided Search of Implicit Random Geometric Graphs," *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3067–3074, May 2015, arXiv: 1405.5848. [Online]. Available: http://arxiv.org/abs/1405.5848

[10] S. Choudhury, J. D. Gammell, T. D. Barfoot, S. S. Srinivasa, and S. Scherer, "Regionally accelerated batch informed trees (RABIT*): A framework to integrate local information into optimal path planning," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016, pp. 4207–4214.

[11] A. Boularias, J. A. Bagnell, and A. Stentz, "Learning to manipulate unknown objects in clutter by reinforcement," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, 2015, pp. 1336–1342. [Online]. Available: http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9360

[12] J. Fu, S. Levine, and P. Abbeel, "One-shot learning of manipulation skills with online dynamics adaptation and neural network priors," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct 2016, pp. 4019–4026.

[13] M. P. Deisenroth and C. E. Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," in *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ser. ICML'11. USA: Omnipress, 2011, pp. 465–472. [Online]. Available: http://dl.acm.org/citation.cfm?id=3104482.3104541

[14] S. Levine, N. Wagener, and P. Abbeel, "Learning contact-rich manipulation skills with guided policy search," in *IEEE International Conference on Robotics and Automation*, May 2015, pp. 156–163.

[15] P. Falco, A. Attawia, M. Saveriano, and D. Lee, "On policy learning robust to irreversible events: An application to robotic in-hand manipulation," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1482–1489, July 2018.

[16] J. A. Bagnell and J. G. Schneider, "Autonomous helicopter control using reinforcement learning policy search methods," in *IEEE Int. Conf. on Rob. and Aut.*, vol. 2, May 2001, pp. 1615–1620.

[17] A. S. Polydoros and L. Nalpantidis, "Survey of model-based reinforcement learning: Applications on robotics," *J. of Intel. & Rob. Sys.*, vol. 86, no. 2, pp. 153–173, May 2017.

[18] J. Ko, D. J. Klein, D. Fox, and D. Haehnel, "Gaussian processes and reinforcement learning for identification and control of an autonomous blimp," in *IEEE Int. Conf. on Rob. and Aut.*, April 2007, pp. 742–747.

[19] M. P. Deisenroth, P. Englert, J. Peters, and D. Fox, "Multi-task policy search for robotics," in *IEEE Int. Conf. on Rob. and Aut.*, May 2014, pp. 3876–3881.

[20] H. Surmann, C. Jestel, R. Marchel, F. Musberg, H. Elhadj, and M. Ardani, "Deep Reinforcement learning for real autonomous mobile robot navigation in indoor environments," *arXiv:2005.13857 [cs]*, May 2020, arXiv: 2005.13857. [Online]. Available: http://arxiv.org/abs/2005.13857

[21] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," *arXiv:1312.5602 [cs]*, Dec. 2013, arXiv: 1312.5602. [Online]. Available: http://arxiv.org/abs/1312.5602

[22] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, Oct. 2017. [Online]. Available: http://www.nature.com/articles/nature24270

[23] B. Amos, I. D. J. Rodriguez, J. Sacks, B. Boots, and J. Z. Kolter, "Differentiable MPC for End-to-end Planning and Control," *arXiv:1810.13400 [cs, math, stat]*, Oct. 2019, arXiv: 1810.13400. [Online]. Available: http://arxiv.org/abs/1810.13400

[24] V. Pong, S. Gu, M. Dalal, and S. Levine, "Temporal Difference Models: Model-Free Deep RL for Model-Based Control," *arXiv:1802.09081 [cs]*, Feb. 2020, arXiv: 1802.09081. [Online]. Available: http://arxiv.org/abs/1802.09081

[25] A. Srinivas, A. Jabri, P. Abbeel, S. Levine, and C. Finn, "Universal Planning Networks," *arXiv:1804.00645 [cs, stat]*, Apr. 2018, arXiv: 1804.00645. [Online]. Available: http://arxiv.org/abs/1804.00645

[26] H.-T. L. Chiang, J. Hsu, M. Fiser, L. Tapia, and A. Faust, "RL-RRT: Kinodynamic Motion Planning via Learning Reachability Estimators from RL Policies," *arXiv:1907.04799 [cs]*, July 2019, arXiv: 1907.04799. [Online]. Available: http://arxiv.org/abs/1907.04799

[27] B. Ichter, J. Harrison, and M. Pavone, "Learning sampling distributions for robot motion planning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 7087–7094.

[28] O. Arslan and P. Tsiotras, "Machine learning guided exploration for sampling-based motion planning algorithms," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 2646–2652.

[29] A. Sivaramakrishnan, E. Granados, S. Karten, T. McMahon, and K. E. Bekris, "Improving Kinodynamic Planners for Vehicular Navigation with Learned Goal-Reaching Controllers," *arXiv:2110.04238 [cs]*, Oct. 2021, arXiv: 2110.04238. [Online]. Available: http://arxiv.org/abs/2110.04238

[30] S. Karten, A. Sivaramakrishnan, E. Granados, T. McMahon, and K. E. Bekris, "Data-Efficient Learning of High-Quality Controls for Kinodynamic Planning used in Vehicular Navigation," *arXiv:2201.02254 [cs]*, Jan. 2022, arXiv: 2201.02254. [Online]. Available: http://arxiv.org/abs/2201.02254

[31] B. Ichter, P. Sermanet, and C. Lynch, "Broadly-exploring, local-policy trees for long-horizon task planning," 2020.

[32] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 1861–1870. [Online]. Available: http://proceedings.mlr.press/v80/haarnoja18b.html

[33] P. Henderson, W.-D. Chang, F. Shkurti, J. Hansen, D. Meger, and G. Dudek, "Benchmark environments for multitask learning in continuous domains," *ICML Lifelong Learning: A Reinforcement Learning Approach Workshop*, 2017.