RRTPI: Policy Iteration on Continuous Domains using Rapidly-exploring Random Trees

Manimaran Sivasamy Sivamurugan and Balaraman Ravindran

***Abstract*— Path planning in continuous spaces has been a central problem in robotics. In the case of systems with complex dynamics, the performance of sampling based techniques relies on identifying a good approximation to the cost-to-go distance metric. We propose a technique that uses reinforcement learning to learn this distance metric on the fly from samples and combine it with existing sampling based planners to produce near optimal solutions. The resulting algorithm - RRTPI can solve problems with complex dynamics in a sample efficient manner while preserving asymptotic guarantees. We provide experimental evaluation of this technique on domains with underactuated and underpowered dynamics.**

I. Introduction

The problem of finding feasible trajectories from a given starting configuration to a goal in dynamical systems is a central problem in robotics. This problem is known to be at least PSPACE-hard [1]. One approach to solve this problem is to directly use numeric solutions of the Hamilton-Jacobi-Bellman equation. Other approaches formulate the problem as a discrete Markov Decision Process (MDP). However these methods suffer from the curse of dimensionality.

A popular class of algorithms that are resilient to this issue are sampling based algorithms. These algorithms are reasonably fast and efficient in terms of space [2]. Rapidly exploring Random Trees (RRTs) are one such method that are widely used [3]. RRTs have good space filling properties and possess *asymptotic completeness*, i.e., they eventually find a solution if one exists. However, they do not provide any guarantees regarding optimality. In fact, it was shown that they almost always converge to a sub-optimal solution [4].

Recently, RRT* an extended version of the RRT method has been developed that also guarantees *asymptotic optimality*, i.e., they eventually converge to an optimal solution as more samples are drawn [4]. However, these guarantees are only asymptotic and the performance of RRT based algorithms is highly dependent on the distance measure used [5]. Even solving a simple two state variable control problem with underpowered dynamics can require an inordinate amount of samples due to poor exploration. The reason being, in complex systems that are underactuated or underpowered, the Euclidean distance between two points is not a good estimate of the geodesic distance or the Carnot-Caratheodory metric on the sub-Riemannian manifold induced by the system dynamics. It has been shown that RRTs explore space efficiently only when the distance metric reflects the true cost-to-go [6].

Thus, recent research has concentrated on identifying the correct domain-dependent metric for efficient exploration. Glassman and Tedrake linearize the system dynamics and use affine quadratic regulators to derive this metric [7]. They show that this results in improvemed exploration. Perez et. al. integrated this technique into the RRT* algorithm to improve exploration while aiming to obtain optimal solutions [8]. They used linear quadratic regulation (LQR) to determine the cost-to-go function as well the tree extension procedure. These methods assume that the dynamics are linearizable and available in closed form. If the robot experiences failure of some joints or picks up new tools, then these dynamics may even change over time. The dynamics can also be discontinuous or too complex to represent in closed form as in the case of an octopus arm [9]. Thus in situations where we cannot define exact closed form dynamics, we need to *learn* the domain dependent metric from experience. This will extend the scope of existing RRT based planners to a wide range of domains that have complex dynamics.

One field that has traditionally looked at learning domain dependent metrics from sample data is Reinforcement Learning (RL). Most RL techniques work by estimating the value function, which is an sample based estimate of the cost-to-go, of states from sample trajectories through the state space [10]. While these techniques have been well studied for discrete domains, continuous domains are more challenging as generalization and approximation is required to learn the value function. Obtaining a correct estimate depends directly on the quality of the samples. A crucial problem that limits the applicability reinforcement learning methods to continuous domains, is ensuring generation of sufficiently representative samples.

In this paper we propose RRTPI a hybrid approach that combines RL with RRT style sampling. RL techniques estimate the cost-to-go of states with minimum domain knowledge, but they require a principled way of generating sufficiently representative sample trajectories in continuous space. On the other hand, RRT based algorithms possess good exploration properties in continuous space, but require an estimate of the cost-to-go in order to work in underactuated and underpowered domains. Combining these approaches enable RRTPI to handle arbitrarily complex domains without making assumptions on the form of the system dynamics and costs. To the best of our knowledge this is the first such hybrid approach combining these two paradigms.

S. S. Manimaran and B. Ravindran are with The Department of Computer Science and Engineering, Indian Institute of Technology Madras, India {ssm,ravi}@cse.iitm.ac.in

We first construct a discrete approximation to the given problem by drawing samples. We then estimate the cost to go pseudo-metric (or the value function) using policy evaluation methods. Following which, we use the estimated value function to again generate samples and form a *better* discrete approximation. The process is thus repeated, iteratively forming better approximations and solving them till the original problem is solved sufficiently well. This method, is in spirit, similar to the iMDP technique of Huynh et. al. [11]. However, they do not handle complex dynamics and rely on Euclidean distance to explore when sampling. Our method works on complex domains with non-linear dynamics and retains the asymptotic completeness and optimality properties of RRT based methods.

II. Preliminaries

A. Reinforcement Learning

We model the problem of finding an optimal path as solving a Markov Decision Process(MDP). A continuous MDP $\mathscr{M}$ is described as $\langle S, A, T, R\rangle$. $S \in \mathbb{R}^n$ is the domain of states. Each state has a corresponding set of allowed actions A_s. This set may be discrete or continuous. The action set A is defined as $\bigcup_s A_s$. $T(s,a,s')$ is the probability of transitioning from state $s \in S$ on taking action a to a state s'. If the transitions are deterministic we may write it as $T(s,a) = s'$. $R(s,a,s')$ is the expectation of real valued rewards(or cost) associated with taking action, a from state $s \in S$, and reaching state s'. These rewards are generally bounded in $[R_{min}, R_{max}]$. Given some starting state s_0, we pick an action a_0 resulting in a state transition to some s_1 according to T and a reward r_1 sampled from R. After a fixed time step, we pick the next action and so on so forth, resulting in a sequence of states actions and rewards $s_0, a_0, r_1, s_1, a_1, r_2, \ldots$. Our objective is to choose actions $a_0, a_1, \ldots$ such that we obtain maximum cumulative reward or *return*. The return is defined as $\sum_{t=0}^{\infty} \gamma^t r_t$, where r_t is the reward at time step t and $\gamma \in (0,1)$ is the discount factor.

A policy π is a mapping from the state space S to the action space A. $\pi : S \times A_s \to [0,1]$ that is, it represents the probability of choosing some action at a given state s. A deterministic policy is usually denoted as $\pi(s) = a$ where $a \in A_s$. The state value function of a state $J^\pi(s)$ is the expected return obtained by following policy π starting from the state s. When the set S is discrete, we may express the value function for a given policy using the Bellman equations as,

$$\forall s \in S,\ J^\pi(s) = \sum_a \pi(s,a) \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma J^\pi(s')] \tag{1}$$

A policy π^* is optimal if $\forall s, \pi\ J^{\pi^*}(s) \geq J^\pi(s)$. The optimal value function also denoted as, J^* can be calculated by replacing the expectation over set of actions in equation 1 with the max operator. When the state space is continuous, we use parametric or non-parametric function approximation techniques to represent J.

B. Policy Evaluation using TD(λ)

Equations 1 can be solved using dynamic programming techniques [10]. In most real world scenarios however, the transition probabilities are not available directly or are too complicated to be represented explicitly. A popular class of RL algorithms solve this problem by *sampling*, and estimate the value function. This is known as policy evaluation.

The TD(λ) family of algorithms is a basic method that uses temporal differences to estimate J^π [12]. Consider a set of N trajectories of the form $\{s_0, a_0, r_1, s_1, a_1, r_2, \ldots s_{M_i}\}_{i=1}^N$. the TD(0) algorithm estimates the state value function using the following update

$$\hat{J}(s_t) \leftarrow (1-\alpha)\hat{J}(s_t) + \alpha(r_{t+1} + \gamma \hat{J}(s_{t+1})) \tag{2}$$

$0 \leq i \leq N, 0 \leq t \leq M_i$. The parameter λ is a measure of how much credit is assigned to earlier states in the trajectory. It serves to trade off bias and variance in the estimates with $\lambda = 0$ having least variance and most bias [12]. TD(λ) converges provably with appropriately decaying values of α.

Methods that handle continuous states usually assume a functional form of J^π to estimate the value [13], [14]. We use TD(λ) on the discrete MDP approximation of the continuous problem, as given by a RRT based sampling technique, to get point estimates of J. We then generalize the point estimates to unseen states, by using nearest neighbor methods. Details on the sampling technique and reasoning behind our choice of the policy estimation technique are given in section IV. We will first describe our sampling technique in the following section.

III. Rapidly exploring Random Sample Trees

In this section we briefly introduce RRTs and describe how we use them to generate a 'sample tree' of trajectories to approximate a given problem. Given a space S, the basic RRT construction is as follows. The algorithm randomly samples a point in the space and calculates the nearest node in the existing tree from the sampled point. A new point is added to the tree by moving a fixed distance in the direction of the sampled state. If the resulting edge from connecting the nearest point to the new point is collision free then the vertex and edge are added to the tree. This is known as extending the tree. Thus the tree is stretched outwards towards lesser explored areas. This requires a distance measure for calculating the nearest vertex and extending the tree. RRTs possess several attractive properties in terms of exploration. Given a RRT G of size n with set of vertices $V(G)$ and edges $E(G)$ constructed in some space S, then

$$\lim_{n\to\infty} Pr(s \in V(G)) \to 1,\ \forall s \in S$$

Also the probability that a node in the tree will be expanded is proportional to the volume of its Voronoi region. This accounts for the *rapidly-exploring* property of RRTs. We will preserve this property while generating samples.

Given a control problem formulated as an MDP $\mathscr{M}$, we would like to generate sample trajectories using a RRT-like procedure. We assume access to a generative model $\widehat{\mathscr{M}}$ as

described by Ng and Jordan [15]. Given a state s and action a, the model returns a sample from distribution of the next state and a sample reward corresponding to a fixed time step. In our algorithm, we generate samples based on a metric defined by the value function. Thus the `Nearest` and `Extend` functions are defined as shown below based on the metric $\|\cdot\|_J$.

Function `Nearest` $(s, X, \|\cdot\|_J)$

Data: Set of states X, distance measure given by $\|\cdot\|_J$ and a state s

Result: Return $x_{near} \in X$ such that, $x_{near} = \arg\max_{x \in X} \|x - s\|_J$

Function `Extend` $(s, s_{target}, \|\cdot\|_J)$

Data: Given state s and a target state s_{target} and a distance metric $\|\cdot\|_J$

Result: A sample $(s_{ext}, a_{ext}, r_{ext})$ such that s_{ext} can be reached from s on performing a_{ext} and is closest to s_{target} as defined by the distance metric

$d_{min} \leftarrow -\infty$
for *every* $a \in A_s$ **do**
  Sample $(r', s') \leftarrow \widehat{\mathscr{M}}(s, a)$
  if $\|s' - s_{target}\|_J > d_{min}$ **then**
    $d_{min} \leftarrow \|s' - s_{target}\|_J$
    $(s_{ext}, a_{ext}, r_{ext}) \leftarrow (s', a, r')$
  end
end

Given a real valued function J (the value function in our case) defined on the state space S, we define

$$\|x - y\|_J = (J(x) - J(y)) \text{ where } x, y \in S$$

Note that in the `Extend` function, when the action space is continuous, we simply sample some k actions uniformly and then choose the best amongst them.

We may now define a sampling procedure that returns a tree of samples G from the given problem based on a distance measure $\|\cdot\|_J$. The algorithm closely resembles the RRT algorithm in constructing the samples. The ConstructRRST procedure as defined in Algorithm 3, constructs a tree with samples of the form $(s_t, a_t, r_{t+1}, s_{t+1})$.

The tree is grown in a *greedy* manner with respect to the value function J as the action that maximizes reward(or minimizes cost) is chosen in the `Extend` function. Using the value function as a distance measure preserves the efficient exploration property of RRTs discussed above. The probability of an edge (s, s') being included in the tree is given by the product of the probability of the following two events - (i) Probability that vertex s is selected for expansion. (ii) Probability that s' is selected. The probability of the first event is proportional to the volume of the Voronoi region of s. This is a property of RRTs. The probability of the second event is proportional to the value of the state $J(s')$. Given a set of samples, we re-evaluate the value function using TD(0) as described in the next section.

Algorithm 3: ConstructRRST$(N, \|\cdot\|_J)$

$E(G) \leftarrow \emptyset, V(G) \leftarrow s_{start}$
$n \leftarrow 0$
while $n < N$ **do**
  Sample a state s_{new} from S
  $s_{near} \leftarrow$ `Nearest`$(s_{new}, V(G), \|\cdot\|_J)$
  $(a_{ext}, s_{ext}, r_{ext}) \leftarrow$ `Extend`$(s_{near}, s_{new}, \|\cdot\|_J)$
  $V(G) \leftarrow V(G) \cup \{s_{ext}\}$
  $E(G) \leftarrow E(G) \cup \{(s_{near}, a_{ext}, r_{ext}, s_{ext})\}$
  $n \leftarrow n + 1$
end
return G

IV. Estimating the value function

Given a sample tree G as described in the previous section, we formulate a discrete approximation to the original problem as follows. From every leaf node in the tree, a path is traced back to the root. This gives us several sample trajectories. We consider a discrete problem whose states are the nodes of the tree $V(G)$. We evaluate a discrete value function $\widehat{J} : V(G) \mapsto \mathbb{R}$ using the $TD(0)$ update given in Equation 2 on the sample trajectories. This can be thought of as backing up values along the trajectories of the tree. $TD(0)$ on discrete domains with a finite set of sample trajectories converges to a fixed value [12].

This value function is discrete and is defined only on specific points. It has to be generalized across the entire state space of the original problem. This a standard regression task. We use nearest neighbor methods that build local models around a given query point. Depending on the nature of the model and definition of the locality there are several variants. We use the following techniques and compare them in experiments described later.

1) Locally constant: Here we simply take the value as the average of the values of k-nearest-neighbors(k-nn) of x. The distance metric used to evaluate the nearest neighbors is Euclidean.

$$J(x) = \sum_{s_i \in \text{Nbr}(x)} \frac{1}{k} \widehat{J}(s_i)$$

For $k = 1$, the value of a state is generalized to its Voronoi region. By varying k we can vary the size of the neighborhood over which we generalize. We can reduce the variance in the estimate by learning local models, such as those discussed by Atkeson et. al. [16].

2) Locally linear: We assume the value of the function is linear within a neighborhood. The parameters of the function β are learnt by minimizing the least-squared error using simple linear regression as described below.

$$J(x) = \beta x = \sum_{j=1}^{n} \beta_j x_j + \beta_0$$

$$\beta = \arg\min_{\beta} \sum_{s_i \in \text{Nbr}(x)} (\beta s_i - \widehat{J}(s_i))^2$$

Here the loss function is defined only within the neighborhood, i.e., we use only the k-nearest-neighbors as training for the linear regression model. In higher dimensional state spaces, locally constant estimates tend to perform poorly because of high variance. In such cases linear methods tend to perform better [16]. This results in a method which has more bias in terms of representation than simple k-nn regression.

3) Locally linear with Gaussian weights: We employ a Gaussian weighting scheme based on the distance of the points in the neighborhood. Closer points are given more weight-age [16]. We use define a Gaussian kernel

$$K(d) = \exp(-d^2)$$

where d is a distance measure between the input states in the neighborhood and the target point. Here we use the Euclidean distance. We assign weights to the inputs using this kernel and regress. The modified error function for estimating β becomes

$$\beta = \arg\min_{\beta} \sum_{s_i \in \text{Nbr}(x)} K(\|x - s_i\|)(\beta s_i - \widehat{J}(s_i))^2$$

We will call these techniques nearest neighbor temporal difference or NN-TD methods. We can define a procedure NN-TD(G) that takes a tree of sample transitions G as input and returns the generalized estimate of the value function. Note that these techniques use a 'lazy' approach to estimate the value at a given point, i.e., they do not perform any calculations until a point is queried and just maintain the set of input points and the corresponding values as such. Thus in an implementation of this method, the generalization is done only when value function is estimated as $J(s)$ in the `Extend` and `Nearest` functions. Such nearest neighbor techniques are preferred as they have low bias in learning and can approximate any arbitrary function given enough data points. Alternatively we can evaluate the value function directly from the set of samples by suitably modifying Fitted Q-Iteration [14]. Here we may use parametric methods such as support vector regression and Gaussian processes regression. However, experimentally they did not perform well. One reason could be the following—these methods operate directly on the vector representation of a state from $\mathbb{R}^n$, whereas $TD(0)$ runs on a tabular representation of states. As the vertices of the set $V(G)$ are actually embedded on a manifold induced by complex dynamics of the system, methods that run on the $\mathbb{R}^n$ representation perform poorly.

Our method based on $TD(0)$ approximates the values better as it operates on the latent space of the system. This due to the fact that the geodesic distance along the trajectories approximate the inherent metric of the manifold. Moreover, the accuracy of the approximation improves as the number of points in the trajectories increases. Given that our sampling technique RRST is asymptotically complete, $TD(0)$ combined with nearest neighbor regression is a favorable method to learn the value function, since it allows us to give asymptotic guarantees on the correctness of approximation. Comparisons between various techniques for policy evaluation are presented in the experiments section.

Now that we have described a class of techniques for estimating the value function from a set of samples, and a technique for generating samples by maximizing the value function, we can define an iterative procedure that alternates between RRST and NN-TD to make progressively better discrete approximations and solve the original problem. We describe our algorithm RRTPI in the following section.

V. RRTPI

The RRTPI algorithm is described in Algorithm 5. Given a control problem, we begin with a uniform estimate for the cost-to-go function J_0. We use this to generate a set of sample transitions using the ConstructRRST method described in Section III. We then estimate the value function J_n from these samples using NN-TD. This estimate is used in the subsequent iteration to construct another sampling tree such that it is grown greedily w.r.t the previously evaluated value function.

Algorithm 4: RRTPI(N)

```
Initialize uniformly J_0 ← 0
n ← 1
while n < N do
    G_n ←ConstructRRST(M_n, ‖·‖_{J_{n−1}})
    J_n ← NN-TD(G_n)
    n ← n + 1
end
```

As we obtain better samples, the estimate of the optimal value function continuous to improve. This proceeds in an iterative manner till we obtain a sufficiently optimal solution. The size of sample set M_n can be changed for different iterations. Typically initial iterations need more samples as the value function might not accurately estimate the optimal cost-to-go.

This method resembles policy iteration, which is a DP based technique for solving discrete MDPs [10]. In policy iteration, an optimal policy π^* is found as follows. First begin with an random policy π. Evaluate this policy, i.e., find the value function J^π using some policy evaluation technique. Then define a new policy that is greedy w.r.t the estimated value function. This step is called policy improvement. The corresponding new value function is again estimated and the steps are repeated till the policy converges to an optimal one. The similarities with our algorithm are now apparent and hence the name RRTPI.

The NN-TD step in our algorithm corresponds to policy evaluation, and the constructRRST step corresponds to policy improvement. We may think of our algorithm as extending policy iteration to continuous domain using samples to estimate both the policy and the value function. Also, we do not require full knowledge of the system dynamics as in the case of policy iteration. A generative model that allows

us to draw samples is sufficient. This is a weaker assumption and allows us to solve a more general class of problems.

The LQR-RRT* technique [8] method describes a similar approach of learning the optimal value function from experience, but assumes knowledge of the system dynamics in linearizable form. It is found that the accuracy of LQR methods falls rapidly as the dimensionality of the domain increases [7]. Thus the assumptions made by LQR-RRT* hinder its ability to model more complex problems effectively. Systems such as the octopus arm, are easy to generate samples from but hard to fully specify in closed form. Our method can handle such domains as it only requires samples. It can also handle arbitrary cost functions. Supporting results are shown in the results section.

VI. Results

We evaluate our algorithm on a variety of domains having underpowered and underactuated dynamics - the mountain car, the acrobot and the octopus arm. The total discounted reward along the current best path to the goal is the evaluation criterion that we use. We compare the performance of our approach against the following baselines:

a) Fixed Discretization: We discretize the space into uniform grids and run Dynamic Programming. The number of discretizations is taken as the no. of samples for comparison.

b) LQR based Policy Evaluation: The policy evaluation technique NN-TD is replaced with an LQR based evaluation technique after Perez et. al. [8].

c) RRTPI variants: We compare the three variants of RRTPI based on the nearest neighbor techniques discussed in section IV. kNN-RRTPI with different values of k corresponds to the locally constant method. LL-RRTPI and LW-RRTPI use the locally linear and the Gaussian weighting scheme correspondingly. The neighborhoods for LL-RRTPI and LW-RRTPI were defined using 5-7 nearest neighbors.

All results are averaged across 100 runs. Simulations were run on a 3.4GHz 4 core system with 16GB of RAM.

A. Mountain Car Domain

In this domain, the goal is to drive an underpowered car in a valley up a steep hill. The state is a 2 dimensional continuous space consisting of the position and velocity of the car along the hill. The actions correspond to acceleration in the positive or negative direction. A small negative reward is given every step, till the goal state is reached. A large positive reward is given upon reaching the target. Detailed descriptions of the dynamics can be found in Singh et. al.[17]. This is an example of a underpowered domain. Comparison of performance is shown in Figure 1.

All methods were successfully able to find feasible solutions. Although the discrete algorithm reaches the optimal performance roughly around the same time as the RRTPI algorithms, the space requirements are much higher. This is because at any given time, the RRTPI algorithms need to store a maximum of M_n nodes and edges plus an additional M_{n-1} values representing the value function. In this

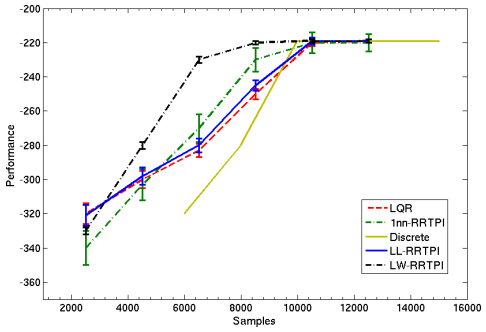

Fig. 1. Comparison of various algorithms on the mountain car domain

experiment M_n was 2000 for all $n > 1$. Whereas the discrete case needed to store 100×100 states. This problem would compound as the dimensionality of the problem increases. Also all algorithms display more efficient use of samples than plain discretization.

Using LQR gives good estimates initially but eventually both LW-RRTPI and 1nn-RRTPI perform better with the same number of samples. The performance of LL-RRTPI is almost similar to LQR. Although as the complexity of the domain increases, this is expected to change. LW-RRTPI performs the best among the algorithms.

B. Acrobot Domain

In this task, an acrobot must be brought to a vertically upright position. The acrobot is a two link robot with one fixed unpowered joint and a free joint powered joint. The system has four states consisting of the angular position and speed of the two joints. This is an underactuated system as only the free joint can be controlled and the robot has to learn to swing up by building momentum. The exact dynamics can be found in Murray and Hauser [18]. The results on this domain are shown in Figure 2.

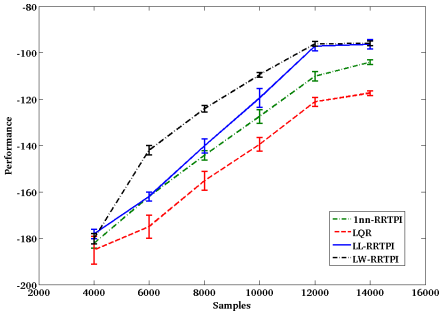

Fig. 2. Comparison of various algorithms Acrobot domain.

In this task, LQR performs poorly compared to the nearest neighbor methods. This is because as the complexity of the dynamics increases, the accuracy of the LQR estimate falls rapidly [7]. Results from using discretization are not reported since we ran out of memory before a solution was found. LL-RRTPI and LW-RRTPI perform better as compare to 1nn-RRTPI due to lower variance in higher dimensions.

C. Octopus arm

The octopus arm possesses a large number of degrees of freedom with high redundancy. The arm is made up of several connected compartments and it is controlled through activations of the muscles on walls of these compartments. We follow the model of the arm in 2-D space with 10 segments introduced by Engel et. al. [9]. The aim of the arm is to reach a specific goal region. The state space corresponds to the position and velocity of the point masses on each segment. Thus the dimensionality of the state space is $22 \times 4 = 88$. We use a subset of 6 possible actions. The cost associated with every action is uniform and reaching the goal results in a positive reward. Each segment has its own dynamics leading to complicated dynamics for the entire arm. This is a case where it is much easier to generate samples for the next state using a one step model of the arm. LQR based methods cannot work because the of the complexity involved in linearizing the dynamics. RRTPI is able to solve such complex high dimensional problems with relatively small number of samples. We show the performance of 1nn-RRTPI and LW-RRTPI on the octopus domain in Figure 3.

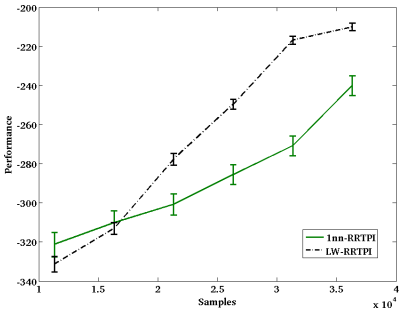

Fig. 3. Comparison of RRTPI algorithms on the octopus arm domain.

VII. Conclusion and Future Work

We present RRTPI the first algorithm that combines RRT style sampling with Reinforcement Learning to solve continuous space control problems with complex dynamics. By estimating the domain dependant distance measure from samples, our algorithm is able to work with complex, underactuated and underpowered systems. The algorithm is able to solve a wider class of problems as compared to previous techniques as it works using only samples and does not make any assumtions on the form of the system dynamics.

The iterative nature of the RRTPI algorithm can be used to interleave planning and actual execution in a real robot. For instance, we may plan for a few iterations and once a satisfactory policy and value function are obtained, the robot can execute the resulting trajectory in real-time. This can be simply done by selecting actions greedily according to the value function. We may then use the resulting trajectory as samples for further iterations. If the same task is repeated several times this allows us to constantly improve performance. It can also be used to improve the accuracy of the one step model.

Transfer learning allows us to use knowledge from solving one particular task in solving a new but related task [19]. The value function estimate can serve as a good representation for transfer learning [20]. From the experiments we can see that RRTPI is sample efficient as compared to discretization and showing sample comlexity bounds on these algorithms would be an interesting direction for future research.

References

[1] J. T. Schwartz and M. Sharir, "On the piano movers problem:II. General techniques for computing topological properties of real algebraic manifolds", *Advances in Applied Mathematics*, vol. 4, pp. 298-351, 1983

[2] S. Lavalle, Planning Algorithms. Cambridge University Press, 2006.

[3] S. M. LaValle and J. J. Kuffner," Randomized kinodynamic planning", *International Journal of Robotics Research*, vol. 20, no. 5, pp. 378-400, May 2001.

[4] Sertac Karaman and Emilio Frazzoli. "Sampling-based algorithms for optimal motion planning". *International Journal of Robotics Research*, 30(7):846-894, June 2011.

[5] S. M. Lavalle, "From dynamic programming to RRTs: Algorithmic design of feasible trajectories," in *Control Problems in Robotics*. Springer-Verlag, 2002.

[6] P. Cheng and S. M. Lavalle, "Reducing metric sensitivity in randomized trajectory design", in *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, 2001, pp. 43-48.

[7] E. Glassman and R. Tedrake, "A quadratic regulator-based heuristic for rapidly exploring state space", in *Proceedings of the IEEE International Conference on Robotics and Automation*, May 2010, pp. 5021-5028.

[8] A. Perez, R. Platt, G. Konidaris, L. Kaelbling, and T. Lozano-Perez,"LQR-RRT : Optimal sampling-based motion planning with automatically derived extension heuristics", in *Proceedings of the IEEE International Conference on Robotics and Automation*, May 2012, pp. 2537-2542.

[9] Engel, Y., Szabo, P., Volkinshtein, D.: "Learning to control an octopus arm with gaussian process temporal difference methods". *Advances in Neural Information Processing Systems* 18, 347-354 (2006)

[10] R.S. Sutton and A.G. Barto. Reinforcement learning: An introduction, volume 28. MIT press, 1998.

[11] V. A. Huynh, S. Karaman, and E. Frazzoli, "An incremental sampling-based algorithm for stochastic optimal control", in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2012, pp. 2865-2872

[12] R. S. Sutton. "Learning to predict by the methods of temporal differences". *Machine Learning*, 3, 1988.

[13] Justin Boyan. "Least-squares temporal difference learning", in *The Proceedings of the Sixteenth International Conference on Machine Learning*, pp. 49-56. Morgan Kaufmann, 1999.

[14] D. Ernst, P. Geurts, and L. Wehenkel. "Tree-based batch mode reinforcement learning". *Journal of Machine Learning Research*, 6:503-556, 2005

[15] Andrew Ng and Michael Jordan. "Pegasus: A policy search method for large mdps and pomdps." *In Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence,* pages 406-415, 2000

[16] Christopher G. Atkeson, Andrew W. Moore, and Stefan Schaal. "Locally Weighted Learning". *Artificial Intelligence*, Rev. 11, 1-5 (February 1997), 11-73

[17] Satinder Singh, Richard S. Sutton, and P. Kaelbling. "Reinforcement learning with replacing eligibility traces." In *Machine Learning*, pages 123-158, 1996.

[18] R. Murray and J. Hauser, "A case study in approximate linearization:The acrobot example", EECS Department, University of California,Berkeley, Tech. Rep. UCB/ERL M91/46, 1991

[19] Matthew E. Taylor and Peter Stone. "Transfer Learning for Reinforcement Learning Domains: A Survey." *Journal of Machine Learning Research*, 10(1):1633-1685, 2009.

[20] Matthew E. Taylor and Peter Stone. "Behavior Transfer for Value-Function-Based Reinforcement Learning."*In The Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 53-59, ACM Press, New York, NY, July 2005.