Learning Vector Policy Fields for Continuous Control

Tanmay Shankar
Robotics Institute
Carnegie Mellon
tshankar@andrew.cmu.edu

Abstract

We introduce Deep Vector Policy Fields (DVPF), a framework to extend the differentiable planners introduced in VIN (Tamar et al. [2016]) and RLN (Shankar et al. [2016]) to the domain of continuous control, by representing policies as continuous vector fields. DVPF addresses learning continuous control policies from expert demonstrations; learning MDP transition dynamics as convolutional filters; and deep inverse reinforcement learning in partially observable environments. We apply our framework to the problem of quadrotor navigation, and establish the potential of our framework to reconstruct expert demonstrations in simulation and on real quadrotors.

1 Motivation and Related Work

Recent advances in the learning community have addressed the problem of *learning* robot control, often from high-dimensional sensor inputs. While a majority of these approaches have operated in discrete action spaces, few among them, such as Lillicrap et al. [2015], Heess et al. [2015], Schulman et al. [2015], Gu et al. [2016] have targeted *continuous* control. Such approaches construct monotlithic deep neural networks in order to reconcile with high-dimensional sensor inputs and generalize across scenarios. As a result, these approaches often ignore the underlying structure of planning, and are difficult to interpret.

An orthogonal approach has been to connect deep neural networks with reinforcement learning (RL) in a natural manner, as in the Value Iteration Networks (VIN) from Tamar et al. [2016], and the 3 reinforcement learning networks (RLN) from Shankar et al. [2016]. This new class of algorithms combine the structure inherent to classical planning frameworks Markov Decision Processes (MDPs), with thse potential for invariance and generalization afforded by deep learning (DL) approaches.

In particular, Tamar et al. [2016] and Shankar et al. [2016] demonstrated the equivalence between the Bellman updates in Value Iteration, and the architectural elements of recurrent convolutional neural networks (RCNNs). This equivalence is achieved by representing forward-rollouts with the transition dynamics of an MDP using convolutional filters, and the choice of optimal actions as an adapted 'max-pooling' across actions. The resulting networks, VIN and RLN, use backpropagation to learn transition models and reward functions associated with the underlying MDP in an end-to-end fashion.

Tamar et al. [2016] and Shankar et al. [2016] thus introduced the paradigm of *differentiable planning*, which preserves the structure of classical planning in a *learnable* manner. This upcoming paradigm has afforded new insights on traditional planning, as evident in a number of recent papers. Karkus et al. [2017], like Shankar et al. [2016], address planning under partial observability, and strive to generalize such planning to new environments. Gupta et al. [2017] introduce the concept of *cognitive mapping*, and jointly learn to navigate and map environments in a latent space. Finally, Niu et al. [2017] extend the VIN to generalized graph convolutions, allowing its application to non-lattice structured data.

10703 Class Project (2016).

Despite the wide variety of problems and simulation environments that [Niu et al., 2017, Karkus et al., 2017, Tamar et al., 2016, Shankar et al., 2016, Gupta et al., 2017] are applied to, the discrete convolutions and finite number of convolutional "action" filters in RLN and VIN restrict their application to discrete state and action spaces respectively. However, a number of problems where these frameworks could prove to be powerful do not adhere to such discrete settings. Consider for instance the robot planning problem, where continuous state *and* action spaces are frequently encountered. Naively discretizing the action space may lead to discontinuous and jerky trajectories, or radically different behaviors from those expected; operating over the original continuous action space makes optimization over even a single action non-trivial.

In this paper, we introduce Deep Vector Policy Fields (DVPF), to extend to the scope VIN [Tamar et al., 2016] and RLN [Shankar et al., 2016] continuous domains. DVPF learns control policies in continuous state and action spaces, directly from expert demonstrations. We apply DVPF to the problem of quadrotor navigation in a continuous 4 dimensional space (3 spatial dimensions and yaw), an instance of the robot planning problem. DVPF selects continuous domain actions by representing the learnt policy as a continuous vector field in this $4D$ space.

At the heart of DVPF are the Belief Propagation RCNN (*BP RCNN*) and *QMDP RCNN* introduced by Shankar et al. [2016]. The *BP RCNN* implements Bayesian belief prediction differentiably, by representing it as a convolution. Backpropagation in the *BP RCNN* learns an estimate of the transition dynamics (as convolutional filters). Backpropagation in the *QMDP RCNN* learns a reward function that a set of provided expert demonstrations implicitly optimize, as a form of imitation learning. The *QMDP RCNN* then chooses optimal actions given a belief of state from the *BP RCNN*. We direct readers to [Shankar et al., 2016] for an in-depth explanation of the *BP RCNN* and *QMDP RCNN*.

We adapt the *BP RCNN* and *QMDP RCNN* to operate over continuous state spaces and output continuous domain actions, by probabilistically interpreting our chosen discrete state-action representation. We adapt the *BP RCNN* to learn the transition dynamics of the underlying MDP, despite the prescence of non-discrete actions. This enables us to quantify the quadrotor's ability to achieving a commanded velocity; as a error-measure in the controller. Finally, we extend the *QMDP RCNN* to learn reward functions *conditioned on sensor observations* directly from the provided *continuous* demonstrations. This powerful improvement gives DVPF the potential to generalize to unseen environments, as a form of deep inverse reinforcement learning [Wulfmeier et al., 2015, 2016] under partial observability. Together, these contributions allow us to learn continuous *vector policy fields*.

We evaluate the ability of DVPF to reconstruct expert demonstrations in a learning from demonstration setting. We utilize data collected both in simulation and from a real quadrotor, and show that DVPF is able to outperform traditional methods employing discretization of the space. Finally, we show that DVPF demonstrates potential to generalize its trajectories to unseen environments, by learning appropriate parametrizations of its reward function.

2 Method

Consider the problem of quadrotor navigation. We consider the state s of the quadrotor as the 3-dimensional position $x, y, z \in \mathbb{R}^3$, along with yaw orientation $\psi \in \mathbb{SO}^1$. At any time point t, we have state $s_t \in S$, where state space S is $\mathbb{R}^3 \times \mathbb{SO}^1$. We further consider the actions of the quadrotor to be specified by the velocities in this space, thus the action a_t at time t, may be described by the velocities *commanded* to the quadrotor, (v_x, v_y, v_z, v_ψ). Thus given a set of continuous 4-dimensional expert trajectory demonstrations, $\{s_t, a_t, s_{t+1}...\}$, where $s_t, a_t \in \mathbb{R}^3 \times \mathbb{SO}^1 \; \forall \; t$, we would like to learn the transition dynamics and the reward function of the underlying MDP. The *BP RCNN* and the *QMDP RCNN* introduced by Shankar et al. [2016] afford us the machinery to do so.

State-Action representation in DVPF: Reconciling with the continuous-discrete gap.

To learn in the discrete setting inherent to the *BP RCNN* and the *QMDP RCNN*, we first construct a probabilistic interpretation of interpolation, that allows us to reconcile with the continuous-discrete gap. Consider a discrete representation of our state space S, consisting of $X \times Y \times Z \times \Psi$ discrete lattice points $\hat{s}_i$ in the $4D$ space. We use multilinear interpolation to interpolate these continuous states into this discrete state space. Thus a continuous state, s_t, may be expressed as $s_t = \sum_{i=1}^{2^d} \alpha_i \hat{s}_{ti}$, where $d = 4$ is the dimensionality of our space, α_i's represent the coefficients of the discrete lattice points at that time step ($\hat{s}_{ti}$). We obtain these coefficients α_i via standard multilinear interpolation in $4D$. Note that $\{\hat{s}_{ti}\} \subset \{\hat{s}_i\}$.

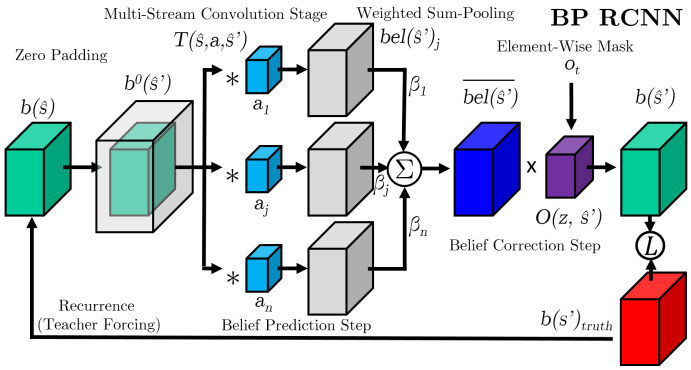

Figure 1: Multi-Action Stream Network Architecture of the Belief Propagation RCNN. The BPRCNN propagates beliefs forward in time using Bayesian Filtering, and minimizes the mean square error between the ground truth belief (in red), and the predicted belief (in green), by backpropagation.

We represent our action space as the span of basis vectors in $\mathbb{R}^3 \times \mathbb{SO}^1$, i.e. $\pm x, \pm y, \pm z, \pm \psi$, denoted by $\hat{e}_j \; \forall \; j = 1, 2, ...8$. Velocities a_t in $\mathbb{R}^3 \times \mathbb{SO}^1$ may be represented as a linear combination of basis vectors $\hat{e}_j$, with coefficients β_j. Thus $a_t = \sum_j \beta_j \hat{e}_j$; the non-zero β_j's define the $4D$ *orthant* in which action a_t lies. Coefficients β_j are obtained as $\beta_j = {}^{a_t . \hat{e}_j}/_{||a_t||_1}$; as a result $\sum_j \beta_j = 1$.

Note that in both VIN [Tamar et al., 2016] and RLN [Shankar et al., 2016], a *single* discrete action is taken at any given timepoint. By allowing a *soft assignment* of coefficients β_j instead (analogous to the soft assignment of α_i's), we allow DVPF to explicitly consider (and execute) actions that are continuous in direction. Key to DVPF is this notion of assigning such a weighting, or *membership* of a continuous variable (s_t or a_t), to a set of discrete points or basis vectors ($\hat{s}_i$ or $\hat{e}_i$); these memberships may be loosely interpreted as the *probability* associated with that discrete state $\hat{s}_i$ or basis vector $\hat{e}_j$, conditioned on the given continuous state s_t or action a_t. Formally, we may express this as $\alpha_i \equiv p(\hat{s}_i | s_t)$, and $\beta_j \equiv p(\hat{e}_j | a_t)$.

In reality, we do not directly have access to the continuous state s_t, but to an observation o_t from which we *infer* a belief distribution over states $b(\hat{s}_t)$. Motivated by the probabilistic interpretation of interpolation, we may construct this belief distribution over discrete states, $b(\hat{s}_i)$ from these *memberships* α_i's, observing that $b(\hat{s}_i) = p(\hat{s}_i | s_t) \equiv \alpha_i$. This probabilistic interpretation is made further apparent by observing that $s_t = \mathbb{E}[\hat{s}_i] = \sum_{\hat{s}_i} b(\hat{s}_i) \hat{s}_i$, which is analogous to the interpolated form, $s_t = \sum_i \alpha_i \hat{s}_{ti}$. An identical interpretation may be made considering the interpolation of actions, i.e. $a_t = \mathbb{E}[\hat{e}_j] = \sum_j p(\hat{e}_j | a_t) \hat{e}_j \equiv \sum_j \beta_j \hat{e}_j$.

Belief Propagation in DVPF: Learning the Transition Dynamics

As emphasized above, the 'expert' demonstrations afford us observations o_t, rather than state s_t. Given a demonstration $\{o_t, a_t, o_{t+1}, a_{t+1} ...\}$, it is necessary to infer and propagate beliefs over the state of the agent (in our case, the quadrotor). The machinery of the *BPRCNN* allows us propagate such beliefs of state, and hence learn transition dynamics associated with the underlying MDP.

Given a belief $b(\hat{s}_t)$ at time t, a particular choice of action, a_t and an observation o_t, we may propagate this belief forward in time via Bayesian Filtering. Thus the belief at time $t+1$, $b(\hat{s}_{t+1})$ may be expressed as $b(\hat{s}_{t+1}) = \eta \; O(s_{t+1}, o_{t+1}) \sum_s T(s_t, a_t, s_{t+1}) b(\hat{s}_t)$; where, $O(s_{t+1}, o_{t+1})$ is the observation model, corresponding to the probability $p(s_{t+1} | o_{t+1})$; $T(s_t, a_t, s_{t+1})$ represents the transition probabilities $p(s_{t+1} | s_t, a_t)$; η is the normalization factor.

The vanilla Bayes Filters uses a *single* action, a_t, to propagate beliefs. By representing continuous actions as linear combinations of a discrete action space, DVPF propagates the beliefs of state with *multiple* discrete actions $\hat{e}_j$ simultaneously. We thus replace transition probabilities $p(\hat{s}_{t+1} | \hat{s}_t, a_t)$ with $\sum_j p(\hat{s}_{t+1} | \hat{s}_t, \hat{e}_j) p(\hat{e}_j | a_t)$. Recalling $p(\hat{e}_j | a_t) \equiv \beta_j$, we have $\sum_j p(\hat{s}_{t+1} | \hat{s}_t, \hat{e}_j p(\hat{e}_j | a_t) = \sum_j \beta_j T(\hat{s}, \hat{e}_j, \hat{s}')$. Thus belief propagation in DVPF can be represented as a weighted sum of the beliefs propagated by each action, weighted by coefficients β_j, as depicted in equation 1. Formally, the prediction and update steps of the *multi-action* Bayes filter are as follows:

$$\overline{b(\hat{s}_{t+1})} = \sum_j \beta_j \; T(\hat{s}, \hat{e}_j, \hat{s}') * b(\hat{s}_t) \tag{1}$$

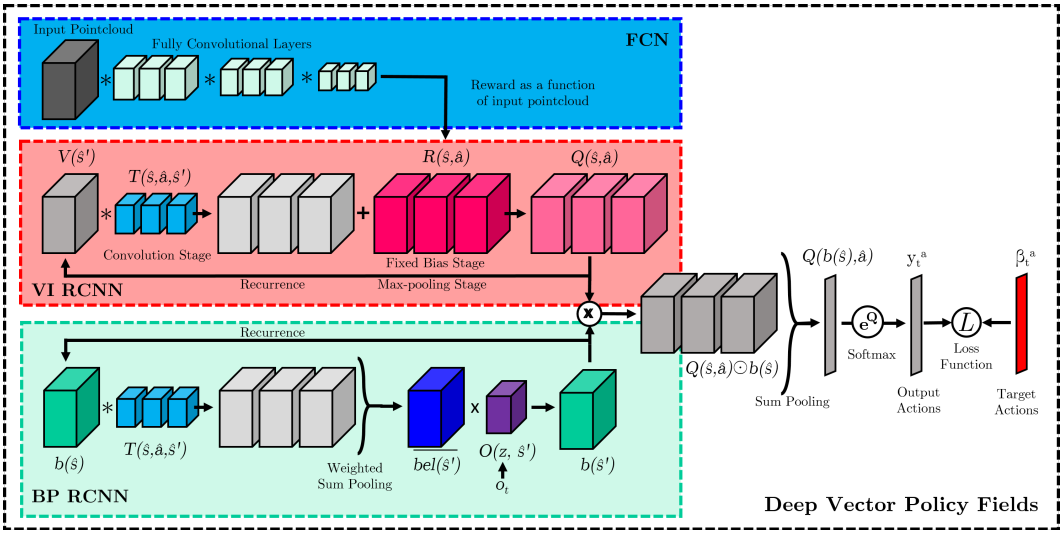

Figure 2: Network architecture for Deep Vector Policy Fields. The network chooses near-optimal continuous domain actions given a belief of state (from the *BP RCNNN*). Backpropagation through the network learns a reward that can reconstruct expert trajectories on which it is trained.

$$b(\hat{s}_{t+1}) = \eta\, O(\hat{s}_{t+1}, o_{t+1}) \odot \overline{b(\hat{s}_{t+1})} \tag{2}$$

The intuition behind equation 1 is to predict future belief based on the relative likelihoods of executing each of the discrete actions involved. The *BPRCNN* may thus be represented as a generic multi-stream architecture, where each stream corresponds to a discrete action, as reflected in Figure 1.

Note that while Figure 1 depicts the state space and filters in $3D$, the transition probabilities $T(s_t, a_t, s_{t+1})$ are actually a set of $4D$ convolutional filters, of size $3 \times 3 \times 3 \times 3$, with one kernel for each of the 8 discrete actions. The circular nature of the yaw dimension may be elegantly handled by implementing the convolution along the yaw dimension as a *circular* convolution. We consider a $4D$ Gaussian kernel as the observation model; the observation o_{t+1} is the mean of the Gaussian, and has a fixed Σ as its positive definite covariance matrix, thus $p(s_{t+1}|o_{t+1}) = \mathcal{N}(o_{t+1}, \Sigma)$.

In order to learn the transition dynamics associated with the MDP (and correspondingly the *error* model of the quadrotor's ability to attain a certain velocity), we follow the training regime followed by Shankar et al. [2016]. We minimize the squared loss between the predicted belief $b(\hat{s}'_i)$, and the ground truth belief $b(s'_i)_{truth}$, i.e. $L_t = \sum_{\hat{s}_i}(b(\hat{s}'_i) - b(s'_i)_{truth})^2$. We use stochastic gradient descent, with teacher-forcing during training, which serves to decouple training timepoints, and prevents uncertainty estimates from collapsing. The ground truth belief is constructed using solely the *mean* of the observation, o_t; the transition kernels are initialized to random values.

Choosing Continuous Domain Actions in DVPF

Making an optimal choice of action in partially observable settings amounts to solving a POMDP, which quickly becomes intractable in continuous spaces. DVPF reconciles with this complexity by invoking the QMDP approximation and approximate value iteration. Given an estimate of the optimal Q values constructed from approximate value iteration, the QMDP approximation considers that the optimal action, a^*, maximizes the *expected* Q value, given the current belief of state $b(\hat{s})$. Mathematically, we have $a^* = \arg\max_a Q(b(\hat{s}), a)$, where belief-space Q values $Q(b(\hat{s}), \hat{a}) = \mathbb{E}_{b(\hat{s})} Q(\hat{s}, \hat{a}) = \sum_{\hat{s} \in S} b(\hat{s}) Q(\hat{s}, \hat{a})$.

In order to extend DVPF to continuous domain actions, we treat the output of the final softmax activation layer of our network, y_t, as the probabilities of each discrete action, $p(\hat{e}_j|a_t)$. Formally, we have $y_t = e^{Q(b(\hat{s}),\hat{e}_j)} / \sum_{\hat{e}_k} e^{Q(b(\hat{s}),\hat{e}_k)}$. Revisiting the equivalence established between $p(\hat{e}_j|a_t)$ and β_j, our network outputs coefficients β_j of discrete actions $\hat{e}_j$, thus specifying a continuous direction in the $4D$ space we consider. We may thus retrieve the (near) optimal *continuous* action at every discrete $\hat{s}_i$ in the state space, as $\pi(\hat{s}_i) = \sum_j \frac{e^{Q(b(\hat{s}),\hat{e}_j)}}{\sum_{\hat{e}_k} e^{Q(b(\hat{s}),\hat{e}_k)}} \hat{e}_j$. Note that we may retrieve this action for a *continuous state* s as well, by constructing belief $b(s)$ from coefficients $\alpha_i \equiv p(\hat{s}_i|s)$. This

Algorithm 1 Inverse Reinforcement Learning via Backpropagation

procedure TRAINING DVPF ON EXPERT DEMONSTRATIONS
  Initialize $R(\hat{s}, \hat{a})$, $D \leftarrow$ Expert Demonstrations
  for $i \in \{1, 2, ..., N_{demo}\}$ **do**
    $\xi = \{(o_1^{(i)}, a_1^{(i)}, o_2^{(i)}, a_2^{(i)} ... o_T^{(i)}, a_T^{(i)})\}$
    for $t \in \{1, 2, ..., T\}$ **do**
      $\beta \leftarrow$ Coefficients(a_t)
      Belief Update: Forward Pass of *BP RCNN*
      $\overline{b(\hat{s}_{t+1})} \leftarrow \sum_{j=1}^{|A|} \beta_j \, T(s, a_j, s') * b(\hat{s}_t)$
      $b(\hat{s}_{t+1}) \leftarrow \eta \, O(s_{t+1}, a_t, o_{t+1}) \odot \overline{b(\hat{s}_{t+1})}$
      QMDP Update: Forward Pass of *QMDP RCNN*
      $Q(b(\hat{s}), \hat{a}) \leftarrow \sum_{\hat{s} \in S} b(\hat{s}) Q(\hat{s}, \hat{a})$.
      $y_t \leftarrow \frac{e^{Q(b(\hat{s}), \hat{a})}}{\sum_{\hat{a}'} e^{Q(b(\hat{s}), \hat{a}')}}$
      Reward Update: Backward Pass of *QMDP RCNN*
      $L_t \leftarrow -\sum_j \beta_j \log y_t^{(j)}$
      $R(\hat{s}, \hat{a}) \leftarrow R(\hat{s}, \hat{a}) - \alpha \frac{\partial L_t}{\partial R(\hat{s}, \hat{a})}$
      $\theta_R \leftarrow \theta_R - \alpha \frac{\partial L_t}{\partial \theta_R}$
    Value Iteration Update: Forward Pass of *VI RCNN*
    $Q(\hat{s}, \hat{a}) \leftarrow R(\hat{s}, \hat{a}) + \gamma T(\hat{s}, \hat{a}, \hat{s}') * V(\hat{s}')$
  Return $R(\hat{s}, \hat{a})$

provides us a continuous policy, in the form of a vector field defined over the $4D$ space. We call this field a *vector policy field*.

Deep Inverse Reinforcement Learning: Backpropagation in the *QMDP-RCNN*

By combining the planning embedded in forward passes of the VIN [Tamar et al., 2016] and the *VI RCNN* [Shankar et al., 2016] with our method of constructing continuous vector policy fields from Q-value estimates, we derive a method to learn reward functions from the expert demonstrations provided. In order to reconstruct the expert demonstrations provided, our objective is to learn a reward function that induces a policy whose actions match those observed in the demonstrations.

Our approach is a form of deep inverse reinforcement learning, applied to the learning from demonstration problem. We achieve this objective by maximizing the conditional log-likelihood of the actions selected in the demonstrations, given the current belief of state $b(\hat{s})$, with respect to some parametrization of the reward function. This is equivalent to training the modified *QMDP RCNN* to minimize the cross-entropy loss (alternately, the KL Divergence) between a target action distribution, constructed from the demonstrations, and the network's predicted actions.

DVPF maintains the target action distribution for continuous domain actions, by virtue of the soft assignment of coefficients β_j. Thus given a demonstration $\{o_t, a_t, o_{t+1}, a_{t+1} ...\}$, we may compute the set of coefficients $\beta_j \; \forall j$, for each action a_t, as $\beta_j = {}^{a_t \cdot \hat{e}_j}/_{||a_t||_1}$. By treating these β_j's as $p(\hat{e}_j | a_t) \; \forall \; j$, we construct such a *target action distribution*. The network's predicted actions are retrieved from the final softmax activation layer, y_t, as described in the previous subsection. The use of cross entropy (or KL Divergence) as a measure of how much $y_t^{(j)}$ differs from β_j is motivated by noting that $\sum_j \beta_j = \sum_j y_t^{(j)} = 1$. Thus we minimize the cross entropy loss, $L_t = -\sum_j \beta_j \log y_t^{(j)}$, using backpropagation through the *QMDP RCNN*.

However, naively maximizing this log-likelihood for the given beliefs does not generalize to unseen states. In order to *propagate* learnt rewards to portions of the state space that have not been encountered, DVPF updates Q value estimates by running forward passes of the *VI RCNN* from Shankar et al. [2016], using the learnt transition dynamics, and the *current* estimate of the reward function. We note that these forward passes are analogous to solving an MDP, a common step in traditional IRL algorithms.

Further generalization across environments may be achieved by conditioning the reward on sensor observations, as observed in [Wulfmeier et al., 2016, 2015]. We thus provide a $3D$ sensor input to DVPF, in the form of a RGB pointcloud. This is treated as input to a fully convolutional *reward*

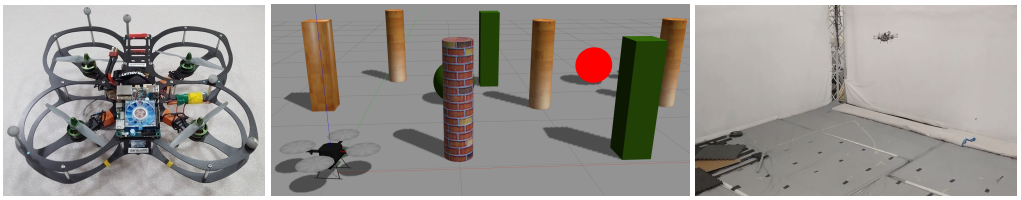

Figure 3: (Left) The Lumemier Danaus quadrotor with which real quadrotor data was collected. (Centre) Sample simulation environment in which we collect quadrotor trajectories, observe the various obstacles, and the red sphere goal location. (Right) VICON arena in which we collect real quadrotor trajectories.

Table 1: Evaluation of DVPF's ability to learn transition dynamics: Cosine similarity values between ground truth and expected velocities learnt under the transition dynamics, for the 3 spatial dimensions.

Experimental Domain	Control Noise	Actions Selected						Across all actions
		$+x$	$-x$	$+y$	$-y$	$+z$	$-z$	
Simulated Trajectories	No noise	0.9322	0.9243	0.9073	0.8918	0.9057	0.9440	0.9174
	Artificial Noise	0.8917	0.8327	0.8465	0.8809	0.8427	0.9251	0.8699
Real Quadrotor Trajectories	No Fans	0.9272	0.9350	0.9693	0.9612	0.7977	0.9734	0.9273
	With Fans	0.9083	0.9225	0.9822	0.9787	0.7103	0.9152	0.9029

network, which outputs reward values used in the *VI RCNN*. The reward function is hence no longer tabular, but is a highly non-linear function of features of this input pointcloud. The fully differentiable nature of DVPF (and the underlying *QMDP RCNN*) implies no additional changes need be made to the architecture to train the reward network [1].

The corresponding architecture is presented in Figure 2. The architecture depicted in Figure 2 resembles the original *QMDP RCNN*, but is implemented in 4 dimensions (3 spatial dimensions and yaw) for generality. The algorithm for training the *QMDP RCNN* is presented in algorithm 1. We note that the transition dynamics parameters in the *BP RCNN* and the *VI RCNN* blocks are frozen during training. Additionally, rather than running full forward passes of value iteration at every time step, we emperically found that using *delayed feedback*, i.e. running *single* passes of step 18 after training on a single trajectory prevents initial trajectories from dominating the training.

3 Experiments and Results

In order to quantify the performance of our framework, we collect a series of quadrotor trajectories in simulation and from a real quadrotor. We first describe the environments used and data collected, followed by a description of our experiments, and finally our results.

Simulated Trajectories: We use the publicly available simulator, Hector Quadrotor [Meyer et al., 2012], to validate DVPF in simulation. Hector quadrotor is built on the Gazebo simulator; this allows us to also simulate an external world and provide simulated sensor inputs to our network. We construct a set of simulated environments in which we collect trajectories; these environments consists of a series of Gazebo shapes (such as cylinders, cuboids, etc.) with different textures (wood, brick, etc.), populated in random locations. A sample environment is depicted in Figure 3 (Centre).

We collect a total of 30 trajectories in these environments, where an expert user pilots the quadrotor through these random obstacles to navigate to the red sphere. The trajectories range from 300 to 1200 timesteps long. We inject small control noise to the executed velocity in a subset of these trajectories, to observe how well DVPF is able to learn transition dynamics in the presence of such noise. We also simulate a Kinect sensor in a third party perspective, faced towards the obstacles and the quadrotor in these environments. The RGB pointcloud provided by this Kinect is voxelized, as in [Maturana and Scherer, 2015], and fed into DVPF as an input to the reward network stream at the top of the DVPF architecture.

[1] We note that experiments with the reward network stream (conditioning on sensor inputs) are carried out in the simulation domain, while experiments with real quadrotor trajectories are carried out in the tabular setting. This choice is motivated purely by hardware limitations.

Quadrotor Trajectories: We utilize data collected from a Lumenier Danaus, a medium-sized quadrotor aerial robot, shown in Figure 3. A total of 10 horizontal and vertical circular trajectories were collected, ranging from 2500 to 15000 time steps long. The quadrotor uses a cascaded proportional-derivative (PD) control scheme to track a reference trajectory (in position and velocity). We utilize state estimates from an indoor Vicon motion capture system at 100 Hz. We executed a subset of these trajectories in the presence of significant wind disturbances coming from 8 fans positioned around the outside of the Vicon arena. For our actions, we record the *commanded* velocities transmitted to the quadrotor, rather than the velocities it executes.

Learning Transition Dynamics: Velocity error models

The *BP RCNN* embedded in DVPF learns transition dynamics, and hence provides us an estimate of the error of the quadrotor in achieving a certain *commanded* velocity. To evaluate the learnt transition dynamics, we compute the similarity betwee the expected velocity under the learnt transition model, q_t, and the ground truth velocities executed, v_t. In the absence of a ground truth transition model for the real quadrotor, the similarity serves as an evaluation metric for learnt transition models. Formally the similarity is $c = {}^{v_t \cdot q_t}/_{||v_t||_2||q_t||_2}$, where q_t is computed as $\mathbb{E}_{\hat{s}',\hat{e}_j}[T(\hat{s},\hat{e}_j,\hat{s}')|\hat{s}=\hat{s}_i] = \sum_{\hat{s}'} \overrightarrow{(\hat{s}')} \sum_{\hat{e}_j} p(\hat{s}'|\hat{s},\hat{e}_j)p(\hat{e}_j|v_t)$.

The average similarities computed across trajectories under various conditions are presented in Table 1. Note that the similarities with ground truth actions are, on average, higher in the trajectories without fans as compared to those with, due to the increased control noise experienced with fans. The overall high degree of similarity indicates our framework is able to learn models useful for planning, even in continuous domains. By quantifying the deviation between expected and ground truth velocities, DVPF learns policies that account for the presence of disturbances.

Table 2: Average deviation and maximum acceleration of reconstructed trajectories learnt from data collected from the real quadrotor. Deviation and accelerations are presented in normalized coordinates.

Experimental Domain	Control Noise	Deep Vector Policy Fields		Vanilla QMDP RCNN (discrete actions)	
		Trajectory Deviation	Max acceleration	Trajectory Deviation	Max acceleration
Real Quadrotor Trajectories	No Fans	0.0440	0.0319	0.2375	0.1828
	With Fans	0.0826	0.0591	0.3590	0.2301

Learning vector policy fields from reward functions.

Quadrotor Trajectories: We evaluate the ability of DVPF to learn continuous domain policies to reconstruct the provided demonstrations, on a real quadrotor platform and in simulation. We train our framework on trajectories of similar shapes (for example, on horizontal circular motions, or vertical circular motions). We learn a reward function by the variant of deep inverse reinforcement learning specified in algorithm 1. After running the *VI RCNN* forward for several steps to propagate this reward throughout the space, we finally extract a vector policy field as earlier specified.

We then follow this policy by performing rollouts from various initialization states. In order to observe how this compares to the expert trajectory, we compute the deviation from the original trajectory (as the average $L2$ distance between the trajectories normalized by the size of the state space), when initialized in the vicinity of the original starting point. As noted in Table 2, DVPF is able to reconstruct trajectories more adeptly than the vanilla *QMDP RCNN*, achieving significantly lower average deviation. Naive discretization often cannot capture small motions and subtleties of the original trajectories. Further, the discretization of the action space causes large accelerations (presented in *normalized* coordinates in Table 2) in the reconstructed trajectories, which is mitigated in the case of DVPF's continuous actions.

Upon executing this learnt policy, we observe it brings the quadrotor close to the height of the original demonstrated circle, executes a single circular motion, and then descends as observed in the demonstration. This shows the ability of the *QMDP RCNN* to reconstruct demonstrated trajectories to an appreciable extent; we note the overall shape, extent, and form of the trajectory is preserved very well. A video of the quadrotor running this policy may be found at https://goo.gl/S4oCBo.

We visualize the learnt policy for this horizontal circular trajectory as a $3D$ vector field (yaw is encoded as color for clarity), in Figure 4. Towards the edges of the field, and in the region of the

Table 3: Average distance to goal and success rate of DVPF in learning goal-directed policies across environments, when rewards are conditioned on input pointclouds. Compared against the vanilla *QMDP RCNN*, operating with discrete states and actions and tabular rewards.

Experimental Domain	Environment Type	Deep Vector Policy Fields		Vanilla QMDP RCNN (discrete actions)	
		Distance from goal	Percentage Success	Distance from goal	Percentage Success
Simulated Trajectories	Training	0.081	73.28 %	0.142	53.13 %
	Novel	0.196	42.59 %	0.772	13.34 %

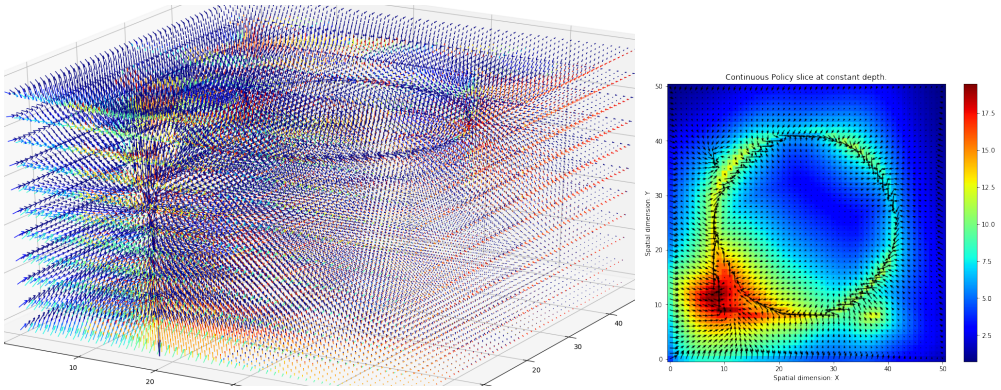

Figure 4: We display the Vector Policy Field in 3D (Left); the arrows at any point depict the velocity policy commanded for each state. The color of the vector depicts the commanded yaw. This vector field was generated by training on a horizontal circular behavior, as evident by the strong velocity vector in a circle. The trajectory may be thought of as a streamline in this vector field. The figure on the right displays shows a slice of the continuous vector policy field at constant depth; depicted over the value function.

demonstration the policy field commands actions that induce circular trajectories as flow lines. We extract a constant-depth slice of this policy, and display it in Figure 4 to the right. This is displayed at the height of the original trajectory. The *continuous field lines* are clearly visible, the policy commands actions continuous in both direction and magnitude.

We observe that in regions close to the original trajetory, the model learns with high confidence the actions that ought to be taken (i.e. along the circle). In unencountered regions of the space, the policy acts as an attractor towards the region where the original demonstration was enacted.

Simulated Trajectories: To test the capacity of DVPF to generalize across different environments, we initialize the policy in a new environment, and compute the average distance achieved from the desired goal location. We also compute the success rate of coming within a certain threshold of this goal location (roughly 0.1 times the state space size). We present the results from DVPF and the vanilla *QMDP RCNN* in Table 3. The vanilla *QMDP RCNN* fails to generalize, due to its tabular rewards. By virtue of conditioning the reward on the input pointcloud, we observe DVPF is able to achieve *some* level of success in moving towards the goal location in new environments. We note, however, that there is a large scope for improvement in the ability of DVPF to generalize. DVPF is also able to achieve a smaller distance to goal in the training environments.

4 Conclusion & Future Work

In this paper, we introduced Deep Vector Policy Fields, a framework for learning continuous control policies from demonstrations. By learning estimates of transition dynamics, DVPF is able to quantify and account for disturbances while planning, in the form of control noise or external disturbances. As demonstrated in simulation and on a quadrotor platform, DVPF is capable to reconstructing trajectories to an appreciable extent.

We emphasize that DVPF preserves the structure of planning in a learnable manner; it is this end-to-end differentiable representation of planning that allows us to extend DVPF to learn parametrized reward functions. This step affords DVPF the potential to generalize demonstrations to new environments without the need for handcrafted features.

Interesting directions of future research include extending such end-to-end planning to non-lattice like structures, and moving into the realm of manipulation. We also intend to explore replacing the 3rd person perspective pointclouds with 1st person sensor input. An additional avenue of interest is providing spatial memory to DVPF, to learn rewards over *entire mapped pointclouds*, rather than the current field of view.

References

Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2829–2838, New York, New York, USA, 20–22 Jun 2016. PMLR. URL `http://proceedings.mlr.press/v48/gu16.html`.

S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik. Cognitive Mapping and Planning for Visual Navigation. *ArXiv e-prints*, February 2017.

Nicolas Heess, Greg Wayne, David Silver, Timothy P. Lillicrap, Yuval Tassa, and Tom Erez. Learning continuous control policies by stochastic value gradients. *CoRR*, abs/1510.09142, 2015. URL `http://arxiv.org/abs/1510.09142`.

Peter Karkus, David Hsu, and Wee Sun Lee. Qmdp-net: Deep learning for planning under partial observability. *arXiv preprint arXiv:1703.06692*, 2017.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015. URL `http://arxiv.org/abs/1509.02971`.

Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Pittsburgh, PA, September 2015.

Johannes Meyer, Alexander Sendobry, Stefan Kohlbrecher, Uwe Klingauf, and Oskar von Stryk. Comprehensive simulation of quadrotor uavs using ros and gazebo. In *3rd Int. Conf. on Simulation, Modeling and Programming for Autonomous Robots (SIMPAR)*, page to appear, 2012.

S. Niu, S. Chen, H. Guo, C. Targonski, M. C. Smith, and J. Kovačević. Generalized Value Iteration Networks: Life Beyond Lattices. *ArXiv e-prints*, June 2017.

John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *CoRR*, abs/1506.02438, 2015. URL `http://arxiv.org/abs/1506.02438`.

Tanmay Shankar, Santosha K Dwivedy, and Prithwijit Guha. Reinforcement learning via recurrent convolutional neural networks. In *Pattern Recognition (ICPR), 2016 23rd International Conference on*, pages 2592–2597. IEEE, 2016.

Aviv Tamar, YI WU, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2154–2162. Curran Associates, Inc., 2016. URL `http://papers.nips.cc/paper/6046-value-iteration-networks.pdf`.

Markus Wulfmeier, Peter Ondruska, and Ingmar Posner. Deep inverse reinforcement learning. *CoRR*, abs/1507.04888, 2015. URL `http://arxiv.org/abs/1507.04888`.

Markus Wulfmeier, Dominic Zeng Wang, and Ingmar Posner. Watch this: Scalable cost-function learning for path planning in urban environments. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2016, Daejeon, South Korea, October 9-14, 2016*, pages 2089–2095, 2016. doi: 10.1109/IROS.2016.7759328. URL `https://doi.org/10.1109/IROS.2016.7759328`.