Composable Learning with Sparse Kernel Representations

Ekaterina Tolstaya[1], Ethan Stump[2], Alec Koppel[2], Alejandro Ribeiro[1]

Abstract— We present a reinforcement learning algorithm for learning sparse non-parametric controllers in a Reproducing Kernel Hilbert Space. We improve the sample complexity of this approach by imposing a structure of the state-action function through a normalized advantage function (NAF). This representation of the policy enables efficiently composing multiple learned models without additional training samples or interaction with the environment. We demonstrate the performance of this algorithm on learning obstacle-avoidance policies in multiple simulations of a robot equipped with a laser scanner while navigating in a 2D environment. We apply the composition operation to various policy combinations and test them to show that the composed policies retain the performance of their components. We also transfer the composed policy directly to a physical platform operating in an arena with obstacles in order to demonstrate a degree of generalization.

I. INTRODUCTION

A key goal in the design of distributed robot teams is the ability to learn collaboratively, so that knowledge and experience gained by one system can be seamlessly incorporated in another. Parallel experiences have already been used to stabilize and hasten joint learning [1], but we argue that there is also a need for learning techniques designed for loosely-coupled teams, in which members may be disconnected for long periods of time and then briefly reconnect to share information and coordinate. We re-consider joint learning problems where coordination is now an infrequent event and not part of the update cycle.

This problem requires *composable learning*, the capability for models learned by different systems to be directly composed as a single model that combines the strengths of each component. We need model representations that can be joined without additional training, which is a major challenge for modern data-driven (deep) machine learning. Different agents with different experiences will invariably develop different representations that cannot be directly combined. Though the agents may use the same deep neural network architecture, each structurally-equivalent weight will be specialized for different purposes.

To capture this notion of composability, we choose instead to use models that live in a Reproducing Kernel Hilbert Space (RKHS), a classic non-parametric learning approach, in which a learned model is represented directly over its training data [2]. Because the domain of the learned model's
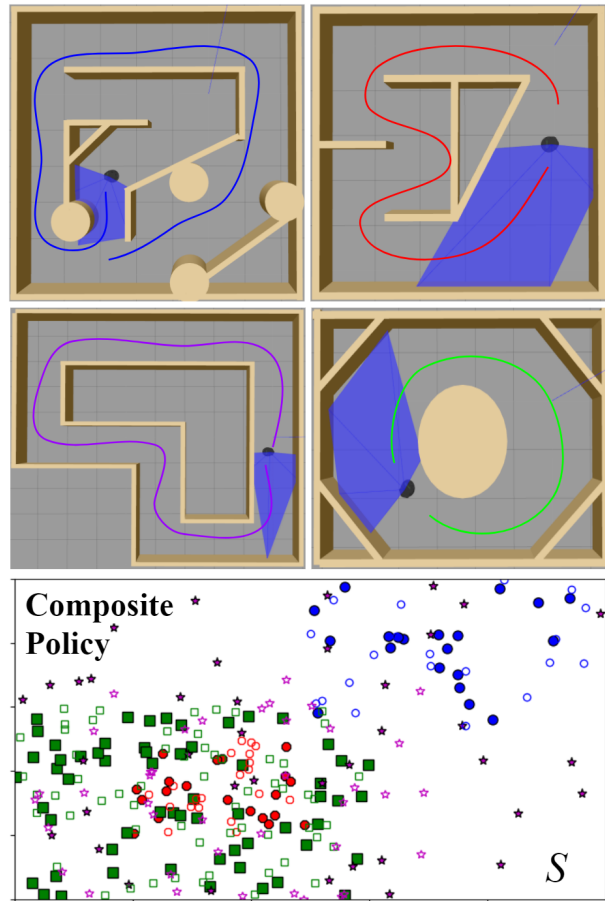
Fig. 1: This drawing illustrates the concept of four policies, trained on individual environments and each represented with their own kernel support, being composed into a single policy with broader support that applies everywhere. These four training environments (referred to as Maze, Circuit 1, Circuit 2, and Round) were used in simulation as part of the demonstration of our approach to composable learning.

applicability is directly represented by its support, we can imagine how models that were developed over separate regions of the data space can combine their supports into one broader model. Recent work on online learning of sparse kernel functions [3] gives us tools for building larger kernel functions while minimizing the number of basis points required in the support.

We apply this model composition approach to the scenario of robots learning to avoid obstacles by experiencing collisions, a well-studied problem that has been successfully solved on physical platforms using methods that blend reinforcement learning with foundational techniques

in optimal control [4] or supervised learning [5]. Because our primary interest is understanding the composability of obstacle-avoiding policies, we opt to learn our policies using Q-learning as a stand-in for more sophisticated approaches, based on recent work in Q-learning with sparse kernel representations [6]. To enable learning control policies in the continuous action space of our problem, we propose a kernelized version of the Normalized Advantage Function Q-learning algorithm [7].

In the context of Q-learning, the density of kernel support of the converged policy is a reflection of the states where we can expect that policy to be near optimal, since these states are drawn from the stationary distribution of the Bellman optimality operator. We suggest the use of the density of the support to decide which component policy should be chosen to dominate in a composition of policies in a region of the state space. We can represent this density directly in the RKHS as an application of the Kernel Mean Embedding [8], but with the addition of our sparsity-inducing procedures.

The **main contributions** of this work are:

1) An algorithm for composing multiple RKHS functions according to their density of kernel support.
2) A Kernel Normalized Advantage Function Q-learning algorithm that extends [7] to learn RKHS policies.
3) A proof-of-concept demonstration in which we: (a) train obstacle-avoidance policies in multiple simulations of a robot equipped with a laser scanner and navigating in a 2D environment; (b) apply composition to various policy combinations and test them to show that the composed policies retain the performance of their components; and (c) transfer the composed policy directly to a physical platform operating in an arena with obstacles in order to elicit a sense of generalization.

II. Normalized Advantage Functions in RKHS

We model an autonomous agent in a continuous space as a Markov Decision Process (MDP) with continuous states $\mathbf{s} \in \mathcal{S} \subseteq \mathbb{R}^p$ and actions $\mathbf{a} \in \mathcal{A} \subseteq \mathbb{R}^q$. When in state $\mathbf{s}$ and taking action $\mathbf{a}$, a random transition to state $\mathbf{s}'$ occurs according to the conditional probability density $\mathbb{P}(\mathbf{s}'|\mathbf{s},\mathbf{a})$. After the agent to a particular $\mathbf{s}'$ from $\mathbf{s}$, the MDP assigns an instantaneous reward $r(\mathbf{s},\mathbf{a},\mathbf{s}')$, where the reward function is a map $r: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$.

In Markov Decision problems[9], the goal is to find the action sequence $\{\mathbf{a}_t\}_{t=0}^{\infty}$ so as to maximize the infinite horizon accumulation of rewards, i.e., the value: $V(\mathbf{s}, \{\mathbf{a}_t\}_{t=0}^{\infty}) := \mathbb{E}_{\mathbf{s}'}[\sum_{t=0}^{\infty} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}'_t) \mid \mathbf{s}_0 = \mathbf{s}, \{\mathbf{a}_t\}_{t=0}^{\infty}]$ [10]. The action-value function $Q(\mathbf{s},\mathbf{a})$ is the conditional mean of the value function given the initial action $\mathbf{a}_0 = \mathbf{a}$:

$$Q(\mathbf{s}, \mathbf{a}, \{\mathbf{a}_t\}_{t=1}^{\infty}) := \tag{1}$$
$$\mathbb{E}_{\mathbf{s}'}\left[\sum_{t=0}^{\infty} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}'_t) \mid \mathbf{s}_0 = \mathbf{s}, \mathbf{a}_0 = \mathbf{a}, \{\mathbf{a}_t\}_{t=1}^{\infty}\right]$$

We consider the case where actions $\mathbf{a}_t$ are chosen according to a stationary stochastic policy, where a policy is a mapping from states to actions: $\pi: \mathcal{S} \to \mathcal{A}$. We define $Q^*(\mathbf{s},\mathbf{a})$ as the maximum of (1) with respect to the action sequence.

It's possible to formulate finding the optimal action-value function as a fixed point problem [11]: shift the index of the summand in (1) by one, make use of the time invariance of the Markov transition kernel, and the homogeneity of the summand, to derive the Bellman optimality equation :

$$Q^*(\mathbf{s},\mathbf{a}) = \mathbb{E}_{\mathbf{s}'}[r(\mathbf{s},\mathbf{a},\mathbf{s}') + \gamma \max_{\mathbf{a}'} Q(\mathbf{s}',\mathbf{a}')] \tag{2}$$

The reason for defining action-value functions is that the optimal Q^* may be used to compute the optimal policy π^* as

$$\pi^*(\mathbf{s}) = \operatorname*{argmax}_{\mathbf{a}} Q^*(\mathbf{s},\mathbf{a}) . \tag{3}$$

Computation of the optimal policy for continuous action problems requires maximizing the Q function (3) which may not have a closed form, and can be challenging. To mitigate this issue, we hypothesize that Q has a special form: it is a sum of the value and an *advantage* [12]:

$$Q(\mathbf{s}, \mathbf{a}) = V(\mathbf{s}) + A(\mathbf{s}, \mathbf{a}) \tag{4}$$

According to the definition of the action-value function, we require that $\max_{\mathbf{a}} Q(\mathbf{s},\mathbf{a}) = V(\mathbf{s})$. Therefore, the optimal action has no advantage: $\max_{\mathbf{a}} A^*(\mathbf{s},\mathbf{a}) = 0$.

For this hypothesis to yield computational savings, we parametrize the advantage function as a quadratic function [7]. We define $L(\mathbf{s}) : \mathcal{S} \to \mathcal{A} \times \mathcal{A}$ as a matrix function, so $L^T(\mathbf{s})L(\mathbf{s})$ is positive definite. The policy is a bounded vector function $\pi : \mathcal{S} \to \mathcal{A}$.

$$A(\mathbf{s},\mathbf{a}) = -\frac{1}{2}(\mathbf{a} - \pi(\mathbf{s}))L^T(\mathbf{s})L(\mathbf{s})(\mathbf{a} - \pi(\mathbf{s})) \tag{5}$$

Thus, we assume the advantage function is quadratic in the actions, which is restrictive, albeit sufficient, for the applications we examine.

To find the optimal policy, we seek to satisfy (2) for all state-action pairs, yielding the cost functional $\tilde{J}(Q)$:

$$\tilde{J}(V, \pi, L) = \mathbb{E}_{\mathbf{s},\mathbf{a}}(y(\mathbf{s},\mathbf{a}) - Q(\mathbf{s},\mathbf{a}))^2, \tag{6}$$

where $y(\mathbf{s},\mathbf{a}) = \mathbb{E}_{\mathbf{s}'}[r(\mathbf{s},\mathbf{a},\mathbf{s}') + \gamma V(\mathbf{s}')]$. Substituting (4), finding the Bellman fixed point reduces to the stochastic program:

$$V^*, L^*, \pi^* = \operatorname*{argmin}_{V, \pi, L \in \mathcal{B}(\mathcal{S})} \tilde{J}(V, \pi, L) . \tag{7}$$

Since searching over value functions that are arbitrary bounded continuous functions $\mathcal{B}(\mathcal{S})$ is untenable, we restrict $\mathcal{B}(\mathcal{S})$ to be a reproducing Kernel Hilbert space (RKHS) $\mathcal{H}$ to which $V(\mathbf{s})$ belongs [10]. Further, the quadratic parameterization of the advantage function means that π and L also belong to $\mathcal{H}$. An RKHS over $\mathcal{S}$ is a Hilbert space is equipped with a reproducing kernel, an inner product-like map $\kappa : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$ [13], [14]:

$$\text{(i)} \langle \pi, \kappa(\mathbf{s}, \cdot) \rangle_{\mathcal{H}} = \pi(\mathbf{s}), \quad \text{(ii)} \mathcal{H} = \operatorname{span}\{\kappa(\mathbf{s}, \cdot)\} \tag{8}$$

In this work, we restrict the kernel used to be in the family of universal kernels, such as a Gaussian kernel with constant diagonal covariance Σ, i.e.,

$$\kappa((\mathbf{s}),(\mathbf{s}',)) = \exp\{-\frac{1}{2}((\mathbf{s})-(\mathbf{s}'))\Sigma((\mathbf{s},)-(\mathbf{s}')^T\} \quad (9)$$

motivated by the fact that a continuous function over a compact set may be approximated uniformly by a function in a RKHS equipped with a universal kernel [15].

To apply the Representer Theorem, we require the cost to be coercive in V, π and L [14], which may be satisfied through use of a Hilbert-norm regularizer, so we define the regularized cost functional $J(V,\pi,L) = \tilde{J}(V,\pi,L) + (\lambda/2)(\|V\|_{\mathcal{H}}^2 + \|\pi\|_{\mathcal{H}}^2 + \|L\|_{\mathcal{H}}^2)$ and solve the regularized problem.

$$V^*, \pi^*, L^* = \underset{V,\pi,L\in\mathcal{H}}{\arg\min} J(V,\pi,L) \quad (10)$$

$$= \underset{V,\pi,L\in\mathcal{H}}{\arg\min} \tilde{J}(V,\pi,L) + \frac{\lambda}{2}(\|V\|_{\mathcal{H}}^2 + \|\pi\|_{\mathcal{H}}^2 + \|L\|_{\mathcal{H}}^2).$$

We point out that this is a step back from the original intent of solving (7) because the assumption we have made about V^*, π^*, and L^* being representable in the RKHS $\mathcal{H}$ need not be true. More importantly, the functional $J(V,\pi,L)$ is not convex in V, π, and L and there is no guarantee that a local minimum of $J(V,\pi,L)$ will be the optimal policy π^*. This is a significant difference relative to policy evaluation problems [16]. In the next section, we describe a method for solving (10) in parallel and then composing sparse solutions.

III. COMPOSABLE LEARNING IN RKHS

To enable efficient learning in diverse environments in parallel, we require a composable representation of the control policy $\pi(\mathbf{s})$, which is obtained through a sparse kernel parametrization. First, we develop an algorithm for learning these policy representations using a sequence of observations from a single environment.

A. Q-Learning with Kernel Normalized Advantage Functions

To solve (10), we follow the semi-gradient TD approach described by [17], which uses the directional derivative of the loss along the direction where $\hat{y}_t(\mathbf{s},\mathbf{a})$ is fixed and independent of Q_t. To obtain the semi-gradient at the sample $(\mathbf{s}_t,\mathbf{a}_t,r_t,\mathbf{s}'_t)$, we define the fixed target value $y_t(\mathbf{s}_t,\mathbf{a}_t,r_t,\mathbf{s}'_t) = r_t + \gamma V_t(\mathbf{s}'_t)$ and the temporal difference as $\delta_t = y_t - Q_t(\mathbf{s}_t,\mathbf{a}_t)$. Then, we apply the chain rule to the resulting functional stochastic directional derivative, together with the reproducing property of the RKHS (8) to obtain the stochastic functional semi-gradients of the loss $J(V,\pi,L)$ as

$$\hat{\nabla}_V J(V,\pi,L) = -\delta_t \kappa(\mathbf{s}_t) \quad (11)$$
$$\hat{\nabla}_\pi J(V,\pi,L) = -L(\mathbf{s}_t)L(\mathbf{s}_t)^T(\mathbf{a}_t - \pi_t(\mathbf{s}_t))\delta_t \kappa(\mathbf{s}_t)$$
$$\hat{\nabla}_L J(V,\pi,L) = L(\mathbf{s}_t)^T(\mathbf{a}_t - \pi_t(\mathbf{s}_t))(\mathbf{a}_t - \pi_t(\mathbf{s}_t))^T\delta_t \kappa(\mathbf{s}_t)$$

As a result, $V,\pi,L \in \mathcal{H}$ are expansions of kernel evaluations only at past observed states and the optimal V, π and L

functions in the Reproducing Kernel Hilbert Space (RKHS) take the following form

$$V(\mathbf{s}) = \sum_{n=1}^{N} w_{Vn}\kappa(\mathbf{s}_n,\mathbf{s}), \quad (12)$$

$$\pi(\mathbf{s}) = \sum_{n=1}^{N} \mathbf{w}_{\pi n}\kappa(\mathbf{s}_n,\mathbf{s}), \quad L(\mathbf{s}) = \sum_{n=1}^{N} \mathbf{w}_{Ln}\kappa(\mathbf{s}_n,\mathbf{s})$$

We propose the Kernel Normalized Advantage Functions (KNAF) variant of Q-Learning to iteratively learn the action-value function while following trajectories with a stochastic policy. This algorithm includes the approximation of a kernel density metric $\rho(\mathbf{s})$, which is later used to compose multiple learned models (14). The V, π, L and ρ function representations are compressed with Kernel Orthogonal Matching Pursuit (KOMP), where we tie the compression to the learning rate to preserve tractability and convergence of the learning process [18], [3].

Algorithm 1 Q-Learning with Kernel Normalized Advantage Functions (KNAF)

Input: l_0, $\{\alpha_t,\beta_t,\zeta_t,\varepsilon_t,\Sigma_t\}_{t=0,1,2...}$
1: $V_0(\cdot) = 0, \pi_0(\cdot) = 0, L_0(\cdot) = l_0 I, \rho_0(\cdot) = 0$
2: **for** $t = 0,1,2,\ldots$ **do**
3:     Obtain trajectory realization $(\mathbf{s}_t,\mathbf{a}_t,r_t,\mathbf{s}'_t)$ using a stochastic policy $\pi_t(\mathbf{s}_t) \sim \mathcal{N}(\pi_t(\mathbf{s}_t),\Sigma_t)$
4:     Compute the target value and Bellman error
    $\hat{y}_t(\mathbf{s}_t,\mathbf{a}_t,r_t,\mathbf{s}'_t) = r_t + \gamma V_t(\mathbf{s}'_t), \quad \delta_t = \hat{y}_t - Q_t(\mathbf{s}_t,\mathbf{a}_t)$
5:     Compute the stochastic estimates of the gradients of J with respect to V, π and L
    $\hat{\nabla}_V J(Q_t) = -\delta_t \kappa(\mathbf{s}_t)$
    $\hat{\nabla}_\pi J(Q_t) = -L(\mathbf{s}_t)L(\mathbf{s}_t)^T(\mathbf{a}_t - \pi_t(\mathbf{s}_t))\delta_t \kappa(\mathbf{s}_t)$
    $\hat{\nabla}_L J(Q_t) = L(\mathbf{s}_t)^T(\mathbf{a}_t - \pi_t(\mathbf{s}_t))(\mathbf{a}_t - \pi_t(\mathbf{s}_t))^T\delta_t \kappa(\mathbf{s}_t)$
6:     Update V, π, L, ρ:
    $V_{t+1} = V_t(\cdot) - \alpha_t\hat{\nabla}_V J(Q_t),$
    $\pi_{t+1} = \pi_t(\cdot) - \beta_t\hat{\nabla}_\pi J(Q_t),$
    $L_{t+1} = L_t(\cdot) - \zeta_t\hat{\nabla}_L J(Q_t)$
    $\rho_{t+1} = \rho_t(\cdot) + \kappa(\mathbf{s}_t)$
7:     Obtain greedy compression of V_{t+1}, π_{t+1}, L_{t+1}, ρ_{t+1} via KOMP with budget ε_t
8: **end for**
9: **return** V,π,L

Alg. 1 produces locally optimal policy representations of low complexity, sparsified using KOMP. These parsimonious representations reduce the complexity of real-time policy evaluation and enable the efficient policy composition procedure described in the next section.

B. Model Composition

The key novelty of our approach is the ability to compose multiple control policies without additional training samples. We summarize a procedure for projecting N policies trained in parallel on different environments using Algorithm 1

$\pi_i(\mathbf{s}) = \sum_j w_{ij} \kappa(\mathbf{s}, \mathbf{s}_{ij})$, for $i = 1, \ldots, N$, to a single function $\Pi(\mathbf{s}) = \sum_j w_j \kappa(\mathbf{s}, \mathbf{s}_j)$.

We seek an interpolation between multiple candidate policies, which preserves the values of the original functions. A simple linear combination or sum is insufficient because we want to set $\Pi(\mathbf{s}) = \pi_i(\mathbf{s})$ for all $\mathbf{s}$ in the kernel dictionaries of the candidate policies. To project multiple functions π_i into one RKHS, we iterate through all dictionary points and perform the following update on $\Pi(\cdot)$ at $\mathbf{s}_{ij}$ given $\pi_i(\mathbf{s}_{ij})$:

$$\Pi(\cdot) = \Pi(\cdot) + (\pi_i(\mathbf{s}_{ij}) - \Pi(\mathbf{s}_{ij})) \kappa(\mathbf{s}_{ij}, \cdot) \quad (13)$$

The next challenge we address is resolving local conflicts among π_i. We score the reliability of the π_i in the neighborhood of $\mathbf{s}$ by the number of gradient steps performed in that neighborhood during training, which is the same as the number of observations at that state in Alg. 1. We cannot directly use the density of kernel dictionary elements because the pruning step of Alg. 1 limits the density of the representation via compression with budget ε_t. To accurately represent the number of observations in a neighborhood of the state space, we propose to augment the approximation of V, π and L with a fourth function, ρ, which is the kernel mean embedding of the observed states, representing the probability distribution of states observed during training [19]. To incorporate this approach into Algorithm 1, we augment Step 6 of Algorithm 1 with a density estimation step:

$$\rho_{t+1} = \rho_t(\cdot) + \kappa(\mathbf{s}_t, \cdot) \quad (14)$$

When ρ is repeatedly pruned along with V, π, and L, the removed weights are projected onto nearby dictionary elements, producing non-unity weights for each dictionary element. This results in a sparse representation of the kernel mean embedding, accurate to ε over the entire state-space. This approach is inspired by kernel density estimation [8].

In Algorithm 2, we first accumulate dictionary points from all candidate policies into one matrix D and initialize the composite policy to zero, $\Pi(\cdot) = 0$. Then, we choose points $\mathbf{s}_{ij}$ from D uniformly without replacement and compare the kernel density at $\mathbf{s}_{ij}$ of π_i, the function of origin for that observation, against the density at $\mathbf{s}_{ij}$ of π_k for $k \neq i$, the other candidate functions. If the density of π_i is greater, we add that point to the composite result because it is deemed most reliable in the neighborhood of $\mathbf{s}_{ij}$ in the state space.

Algorithm 2 Composition with Conflict Resolution

Input: $\{\pi_i(\mathbf{s}) = \sum_j^{M_i} w_{ij} \kappa(\mathbf{s}, \mathbf{s}_{ij}),$
$\qquad \rho_i(\mathbf{s}) = \sum_j^{M_i} v_{ij} \kappa(\mathbf{s}, \mathbf{s}_{ij})\}_{i=1,2\ldots,N}$, ε
1: Initialize $\Pi(\cdot) = 0$, append centers $D = [\mathbf{s}_{11}, \ldots, \mathbf{s}_{ij}, \ldots]$
2: **for** each $\mathbf{s}_{ij} \in D$ chosen uniformly at random **do**
3:    **if** $\rho_i(\mathbf{s}_{ij}) > \max_{k \neq i} \rho_k(\mathbf{s}_{ij})$ **then**
4:       $\Pi = \Pi(\cdot) + (\pi_i(\mathbf{s}_{ij}) - \Pi(\mathbf{s}_{ij})) \kappa(\mathbf{s}_{ij}, \cdot)$
5:    **end if**
6: **end for**
7: Obtain compression of π using KOMP with ε
8: **return** f

For our application, we are able to estimate ρ indirectly by the density of dictionary points of a policy π around state $\mathbf{s}$:

$$\tilde{\rho}(\pi, \mathbf{s}) = \sum_{\mathbf{s}_k \in \pi} \kappa(\mathbf{s}, \mathbf{s}_k) \quad (15)$$

This simplification can be used because the structure of the reward function induces an a approximation of a kernel mean embedding on the kernel dictionary of π itself. This does not hold for all problems. For example, if the optimal value, policy, and L functions are all zero in a region of the state space, then all kernel points in the region will be pruned away and the density of the dictionary will not accurately represent the reliability of the function.

IV. SIMULATED & EXPERIMENTAL EVALUATIONS

We apply the KNAF algorithm to the robotic obstacle avoidance task by training on a variety of environments in simulation, and then validating the learned policies on a physical robot.

A. Simulation Results

Building on the infrastructure designed by [20], we train our algorithm on a wheeled robot traveling through indoor environments. The robot receives laser scans at a rate of 10 Hz, with 5 range readings at at an angular interval of $34°$ with a field of view of $170°$. The robot controls its angular velocity between $[-0.3, 0.3]$ rad/s at a rate of 10Hz while traveling with a constant forward velocity 0.15 m/s. While traveling through the environment, the robot receives a reward of -200 for colliding with obstacles and a reward of $+1$ otherwise. The four training environments pictured in Fig. 1 are Maze, Circuit 1, Circuit 2 and Round, which mimic the appearance of indoor spaces. The Round environment is simplest of the four environments because of its radial symmetry. The Circuit 2 environment incorporates narrow hallways that turn both left and right. The Circuit 1 environment adds multiple tight turns in quick succession. The Maze environment incorporates all of these features in a more complex environment.

Individual environment training. We demonstrate the typical learning progress of Algorithm 1 by reproducing ten trials of training with the Round environment, and plot training rewards, model order, and Bellman error in Fig. 2. The learning progress is extremely reproducible, with average episode rewards of at least 2000 after 100,000 training steps. By the end of training, the robot travels for at least 2000 steps before crashing during every episode. Training Bellman error reliably converges for every trial. The resulting policies are parsimonious, with a limiting model order of fewer than 250 kernel dictionary elements. Next, we demonstrate the performance of Alg 1 on the four environments pictured in Fig. 1 in Table I. All policies achieve zero crashes during testing. We also observe that the Maze environment required many more training steps and achieved a larger model order. We interpret this to mean that the Maze environment is more varied, and therefore requires a higher coverage of the state space.

Environment	Steps	Model Order	Loss	Rewards
Round	230K	224	2.24	**1000**
Maze	630K	779	16.04	**1000**
Circuit 2	280K	467	36.40	**1000**
Circuit 1	500K	578	47.97	**1000**

TABLE I: Limiting model complexity, training loss (Bellman error), and accumulation of rewards over 1,000 testing steps for each of the four environments. All environments are solved by the K-NAF algorithm within 700,000 simulation steps.

Policies / Reward	Round	Maze	Circuit 2	Circuit 1
1 - Round	**1000**	-11663	-608	-608
2 - Maze	**1000**	**1000**	-5	-407
3 - Circuit 2	**1000**	-11663	**1000**	196
4 - Circuit 1	**1000**	-11462	-407	**1000**
1 / 2	**1000**	**1000**	-5	-206
1 / 3	**1000**	-11663	799	-206
1 / 4	**1000**	-11261	-206	799
2 / 3	**1000**	**1000**	**1000**	-5
2 / 4	**1000**	**1000**	-5	799
3 / 4	**1000**	-11462	397	397
1 / 2 / 3	**1000**	**1000**	799	196
1 / 2 / 4	**1000**	**1000**	-5	**1000**
1 / 3 / 4	**1000**	-11663	397	799
2 / 3 / 4	**1000**	**1000**	799	-206
1 / 2 / 3 / 4	**1000**	**1000**	**1000**	598

TABLE II: Cross-validation was performed on all possible compositions of the original 4 policies, each of which was trained on one environment pictured in Fig. 1. Each row is a policy composed using Alg 2 of one or more policies trained using Algorithm 1. All 15 policies were tested on the 4 environments, and compared based on the reward accumulated over 1,000 time steps in each environment.

Generalization across environments. We analyze the inability of a single policy to generalize to other environments in Table II. We observe that policies trained on the Round environment (Policy 1) could not generalize to other environments at all, with an average test reward of -4226. We observe that all policies were able to successfully navigate the Round environment with no crashes, most likely due to its simplicity. The policy trained on the Maze environment (Policy 2) was able to somewhat generalize to the other three environments, while none of the other environments could generalize to the Maze, due to its complexity. Policies trained on Circuit 1 (Policy 4) and Circuit 2 (Policy 3) completely failed on the Maze environment, but were able to transfer somewhat among each other, and to the Round environment.

Dual compositions. We apply the composition algorithm (Alg. 2) to combine two of the four trained policies to observe performance on the original two environments and analyze generalization performance on the remaining environments in Fig. II. All controllers composed from Policy 2 (trained on Maze) achieved perfect reward of 1000 on the difficult Maze environment. Policies composed from Circuit 1 and Circuit 2 were able to achieve positive average rewards on the Circuit 1 and 2 environments. We observe some generalization capabilities in the results for the composite Policy 1/2 (Round and Maze), which performs better in all four test environments than both of the original policies. The other dual composite policies were not able to generalize better than the individual policies.

Composition of multiple policies. We further validate the composition algorithm (Alg. 2) using every combination of four trained policies: an additional eleven policies composed from two, three, and four of the original four policies. The performance of the composite policies was validated on the four training environments in Fig. II. Nearly optimal performance was observed when all four policies (1/2/3/4) were composed into one, and then tested on all of the four environments. Two collisions were observed in the Circuit 1 environment, and none in the other three when testing for 1,000 time steps in each environment.

B. Robot Results

The real-world validation experiments were carried out on a Scarab robot pictured in Fig. 3, equipped with an on-board computer, wireless communication, and a Hokuyo URG laser range finder. It is actuated by stepper motors and its physical dimensions are 30 x 28 x 20 cm with a mass of 8kg [21]. Laser scans were received at a rate of 10Hz and 5 range readings were obtained as in simulation. Angular velocity commands were issued at 10Hz. The test environment was built using laboratory furniture and miscellaneous equipment pictured in Fig. 4. When testing the policy trained only on the Round environment, we observed 3 collisions, with a total reward of 397 over 1,000 time steps. Using the composite policy, which incorporates all 4 policies, no collisions were observed during 1,000 time steps. In Fig. 5, we visualize the trajectory obtained by testing the composite policy on the Scarab robot, which shows that the robot successfully completed multiple loops around the obstacles in the environment.

V. CONCLUSIONS & FUTURE WORK

We envision multi-robot systems which can exchange concise representations of their knowledge to enable distributed learning and control in complex environments with limited communication [22]. In this work, we take the first step towards this goal by developing and demonstrating the Q-Learning with KNAF algorithm, which allows the composition of multiple learned policies without additional training samples. Rather than a single robot collecting information about multiple environments in sequence, multiple robots could work in parallel and combine their models. We also see a clear path to extend this approach to higher-dimensional problems using an auto-encoder to learn policies in an RKHS based on a feature space representation, rather than directly in the state space [4].

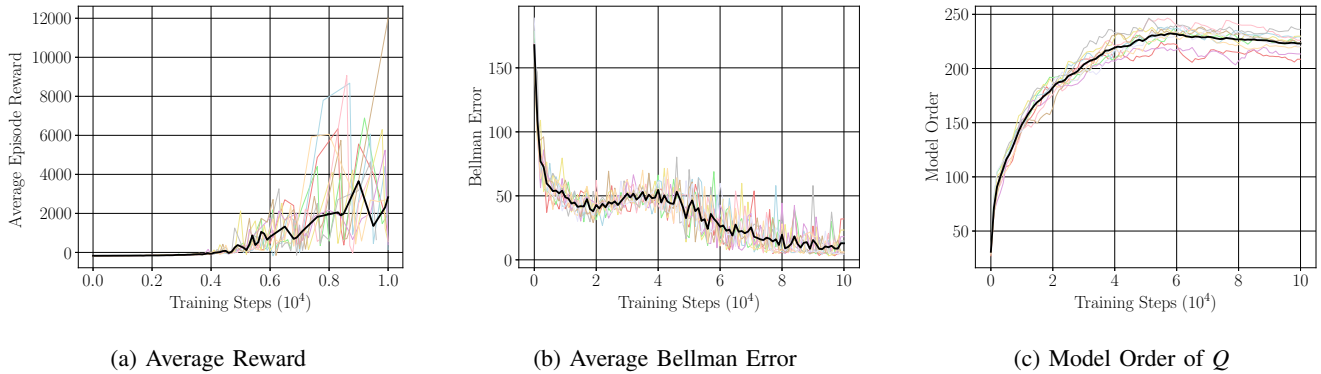(a) Average Reward   (b) Average Bellman Error   (c) Model Order of Q

Fig. 2: Results of 10 experiments over $100,000$ training steps were averaged (black curve) to demonstrate the learning progress for the robotic obstacle avoidance task with the Round environment. Fig. 2a shows the average reward obtained by the stochastic policy during training shows that the robot was able to complete 5000 simulation steps without crashing by the end of training. Fig. 2b shows the Bellman error for training samples converges to a small non-zero value. The model order of the Q approximation remains under 200 for all ten experiments. The exploration variance Σ was 0.2. Constant learning rates were used, $\alpha_t = 0.25$, $\beta_t = 0.25$, $\zeta_t = 0.001$. L was initialized to $L_0 = 0.01I$. For the KOMP Parameters, we used a Gaussian kernel with a bandwidth of $[0.75, 0.75, 0.75, 0.75, 0.75]$ and a pruning tolerance of $\varepsilon_t = 3.0$.
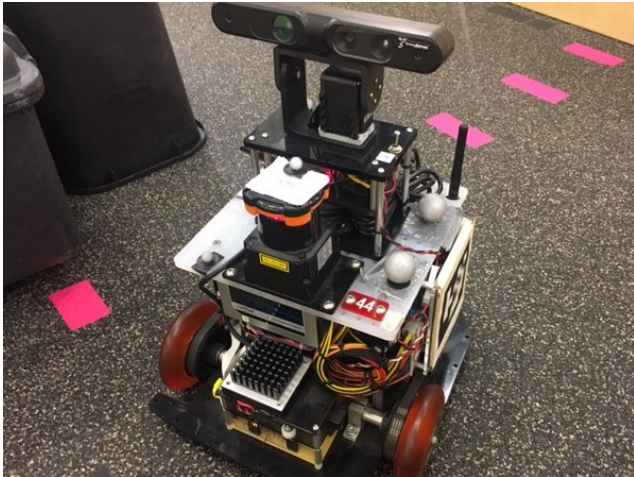

Fig. 3: Scarab robot with equipped with an on-board computer, wireless communication, and a Hokuyo URG laser range finder.
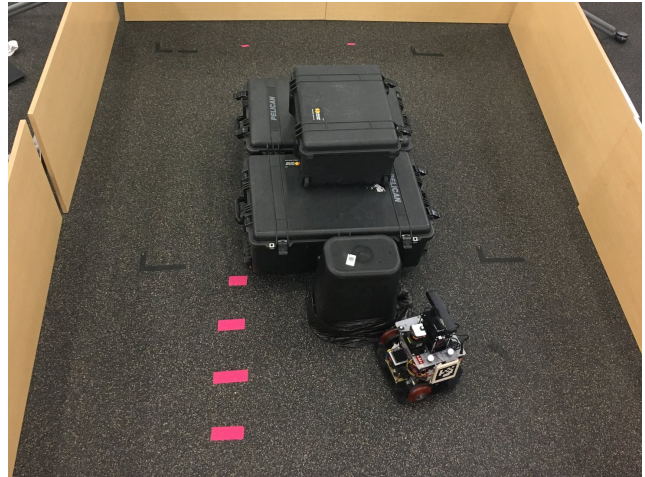

Fig. 4: We validate our approach by testing policies trained in simulation on a real robot in a laboratory environment. The policy trained only on the Round environment experienced 3 crashes over 1,000 testing steps. The composite 1/2/3/4 policy in Table II, combined from all four policies, received a reward of 1,000 during 1,000 testing steps, with no crashes.

REFERENCES

[1] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*, 2016, pp. 1928–1937.

[2] B. Schölkopf and A. J. Smola, *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

[3] A. Koppel, G. Warnell, E. Stump, and A. Ribeiro, "Parsimonious Online Learning with Kernels via Sparse Projections in Function Space," *ArXiv e-prints*, Dec. 2016.

[4] C. Richter and N. Roy, "Safe visual navigation via deep learning and novelty detection," in *Proc. of the Robotics: Science and Systems Conference*, 2017.

[5] G. Kahn, A. Villaflor, B. Ding, P. Abbeel, and S. Levine, "Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[6] E. Tolstaya, A. Koppel, E. Stump, and A. Ribeiro, "Nonparametric stochastic compositional gradient descent for q-learning in continuous markov decision problems," in *American Control Conference*, 2018.

[7] S. Gu, T. P. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep q-learning with model-based acceleration," *CoRR*, vol. abs/1603.00748, 2016.

[8] K. Muandet, K. Fukumizu, B. Sriperumbudur, and B. Schölkopf, "Kernel mean embedding of distributions: A review and beyonds," *arXiv preprint arXiv:1605.09522*, 2016.

[9] R. Bellman, "The theory of dynamic programming," DTIC Document, Tech. Rep., 1954.

[10] A. Koppel, E. Tolstaya, E. Stump, and A. Ribeiro, "Nonparametric stochastic compositional gradient descent for q-learning in continuous markov decision problems," *IEEE Transactions on Automatic Control (under preparation)*, 2017. [Online]. Available: https://koppel.bitballoon.com/pubs/2017_kqlearning_report.pdf

[11] D. P. Bertsekas and S. Shreve, *Stochastic optimal control: the discrete-time case*, 2004.

[12] L. C. Baird, "Reinforcement learning in continuous time: Advantage updating," in *Neural Networks, 1994. IEEE World Congress on*
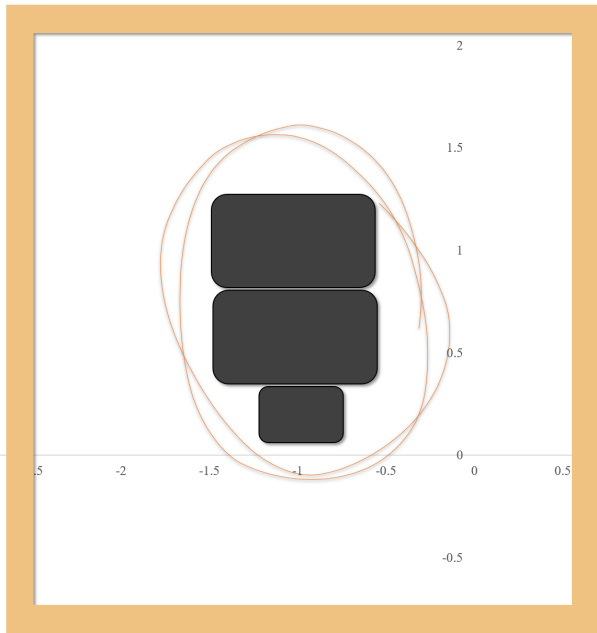
Fig. 5: We visualize the trajectory observed when testing the composite policy (Policy 1/2/3/4) in Table II on the Scarab robot. Position coordinates were computed through onboard motor odometry, which results in a small drift in the position.

Computational Intelligence., 1994 IEEE International Conference on, vol. 4. IEEE, 1994, pp. 2448–2453.

[13] V. Norkin and M. Keyzer, "On stochastic optimization and statistical learning in reproducing kernel hilbert spaces by support vector machines (svm)," *Informatica*, vol. 20, no. 2, pp. 273–292, 2009.

[14] A. Argyriou, C. A. Micchelli, and M. Pontil, "When is there a representer theorem? vector versus matrix regularizers," *Journal of Machine Learning Research*, vol. 10, no. Nov, pp. 2507–2529, 2009.

[15] C. A. Micchelli, Y. Xu, and H. Zhang, "Universal kernels," *Journal of Machine Learning Research*, vol. 7, no. Dec, pp. 2651–2667, 2006.

[16] A. Koppel, G. Warnell, E. Stump, P. Stone, and A. Ribeiro, "Breaking bellman's curse of dimensionality: Efficient kernel gradient temporal difference," *arXiv preprint arXiv:1709.04221 (Submitted to TAC Dec. 2017)*, 2017.

[17] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 2018.

[18] S. G. Mallat and Z. Zhang, "Matching pursuits with time-frequency dictionaries," *IEEE Transactions on signal processing*, vol. 41, no. 12, pp. 3397–3415, 1993.

[19] S. J. Sheather and M. C. Jones, "A reliable data-based bandwidth selection method for kernel density estimation," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 683–690, 1991.

[20] I. Zamora, N. G. Lopez, V. M. Vilches, and A. H. Cordero, "Extending the openai gym for robotics: a toolkit for reinforcement learning using ros and gazebo," *arXiv preprint arXiv:1608.05742*, 2016.

[21] N. Atanasov, J. Le Ny, N. Michael, and G. J. Pappas, "Stochastic source seeking in complex environments," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 3013–3018.

[22] B. Schlotfeldt, D. Thakur, N. Atanasov, V. Kumar, and G. J. Pappas, "Anytime planning for decentralized multi-robot active information gathering," *IEEE Robotics and Automation Letters*, 2018.