MITSUBISHI ELECTRIC RESEARCH LABORATORIES
http://www.merl.com

Learning to Control Partial Differential Equations: Regularized Fitted Q-Iteration Approach

Farahmand, A.-M.; Nabi, S.; Grover, P.; Nikovski, D.N.

TR2016-145 December 2016

Abstract

This paper formulates a class of partial differential equation (PDE) control problems as a reinforcement learning (RL) problem. We design an RL-based algorithm that directly works with the state of PDE, an infinite dimensional vector, thus allowing us to avoid the model order reduction, commonly used in the conventional PDE controller design approaches. We apply the method to the problem of flow control for time-varying 2D convection-diffusion PDE, as a simplified model for heating, ventilating, air conditioning (HVAC) control design in a room.

IEEE Conference on Decision and Control (CDC)

Learning to Control Partial Differential Equations: Regularized Fitted Q-Iteration Approach

Amir-massoud Farahmand, Saleh Nabi, Piyush Grover, Daniel N. Nikovski

Abstract— **This paper formulates a class of partial differential equation (PDE) control problems as a reinforcement learning (RL) problem. We design an RL-based algorithm that directly works with the state of PDE, an infinite dimensional vector, thus allowing us to avoid the model order reduction, commonly used in the conventional PDE controller design approaches. We apply the method to the problem of flow control for time-varying 2D convection-diffusion PDE, as a simplified model for heating, ventilating, air conditioning (HVAC) control design in a room.**

I. INTRODUCTION

This paper formulates a class of partial differential equation (PDE) control problems as a reinforcement learning (RL) problem. Reinforcement learning is the problem of adaptively finding an optimal policy (i.e., controller) for an unknown nonlinear stochastic dynamical system without the knowledge of the dynamics—using only interaction data [22], [21]. We design an algorithm that can directly work with the state of PDE, an infinite dimensional vector, thus allowing us to potentially overcome the limitations of classical approaches to PDE control. As an example, we consider the problem of optimal control for time-varying 2D convection-diffusion PDE, as a simplified model for heating, ventilating, air conditioning (HVAC) control design in a room.

Conventional approaches to PDE control can be classified into two categories. In the *reduce-then-design* approach, a finite-dimensional ordinary differential equation (ODE) is derived using numerical approximation techniques. Model order reduction is often carried out at this stage, and a linear controller is then designed, typically optimizing a quadratic cost functional [1]. Alternatively, in the *design-then-reduce* approach one directly designs a controller for the PDE, for example using distributed parameter LQR theory, and then later use numerical approximations to find the reduced order control gains [2], [3], [4].

The conventional approaches, while elegant, have some drawbacks, especially when they must be deployed to control a real-world system such as an HVAC system. The first drawback is that the controller is typically only valid in a small neighborhood of the baseline solution. If the boundary conditions are significantly changed, for instance if some furniture is added to the room or even a person enters the room, the controller should be re-designed to maintain the performance, which is often impractical. This re-designing requires the knowledge and time of a control engineer who has expertise in designing PDE controllers. The other drawback is that the linear control design ignores potentially useful nonlinear phenomenon inherent in fluid dynamics problems [11]. Designing a controller based on this simplified model might lead to a suboptimal solution.

To address these issues, we take a data-driven approach to design PDE controllers. We show that the PDE control problem can be seen as an RL problem, therefore allowing us to use powerful RL algorithms that can handle very high-dimensional state spaces [7], [15], [8]. These algorithms have shown great success in controlling complex dynamical systems that violate the usual assumptions in control theory such as the linearity of the dynamics or nonlinearity with a particular known structure. For example, finding a high-performing controller to solve an Atari game using the screen image as the input signal, which can be represented by a several thousand dimensional vector, is indeed an optimal control problem for a nonlinear dynamical system that does not have a clear algebraically manipulatable dynamics. This is a complex dynamical system for which conventional control theory tools cannot be used, but has been successfully formulated and solved in the RL framework [15]. This and other successes motivated us to investigate the use of RL to solve the complex problem of PDE control, as an alternative to the more conventional approaches. The proposed approach does not require any knowledge of the PDE, even its structural form, and only uses data to find the controller.

We propose to use an Approximate Value Iteration (AVI) algorithm to find a close to optimal controller. An AVI algorithm iteratively approximates the optimal value function, which can then be used to find the optimal controller (or policy, as it is called in the RL literature). As the state of PDE is an infinite dimensional vector (or a very high-dimensional vector in simulations), the value function is also defined on an infinite dimensional state space. This makes the value function approximation quite different from the usual problems for which RL algorithms have been applied so far, as almost all of them are designed to deal with problems with finite, and often low, dimensional state spaces. The challenge is to design a method that can easily deal with very high-dimensional state spaces, e.g., 2500 dimensions in our experiments.

To address this challenge, we notice the similarity of an infinite dimensional object such as the temperature scalar field in a 2D (or 3D) convection-diffusion problem with a

All co-authors are with Mitsubishi Electric Research Laboratories (MERL), Cambridge, MA, USA. {farahmand, nabi, grover, nikovski}@merl.com

2D (or 3D) image in computer vision problems.[1] Both of them are scalar (or vector) fields, one defining the solution of a PDE and the other defining the colour of an image. Moreover, both of them often have much spatial regularities, such as local smoothness, too. Indeed visualizing the solution of a PDE is nothing but seeing it as an image. This similarity motivates us to design RL algorithms that directly work with the PDE's infinite dimensional state vector and treat it as if the state is an image.

We propose to solve the RL problem by the Regularized Fitted Q-Iteration (RFQI) algorithm [7] that uses a reproducing kernel Hilbert space (RKHS) [20], defined over an infinite dimensional domain, as its function space. The use of an RKHS enables us to have a flexible function approximator that can easily work with infinite dimensional input spaces (or very high-dimensional approximations of them).

II. PDE CONTROL AS A MARKOV DECISION PROCESS

In this section we explain how a PDE control problem can be formulated as a Markov Decision Process (MDP), which is a mathematical framework to define an RL problem [22].

A *finite-action discounted* MDP is a 4-tuple $(\mathcal{X}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where $\mathcal{X}$ is a measurable state space, $\mathcal{A}$ is a finite set of actions, $P : \mathcal{X} \times \mathcal{A} \to \mathcal{M}(\mathcal{X})$ is the transition probability kernel, and $\mathcal{R} : \mathcal{X} \times \mathcal{A} \to \mathcal{M}(\mathbb{R})$ is the immediate reward distribution. The constant $0 \leq \gamma < 1$ is the discount factor. We use $r(x, a)$ to denote the expected value of the random variable $R \sim \mathcal{R}(\cdot|x, a)$.

We now identify these quantities within the context of PDE control. For concreteness, we consider a time-varying convection-diffusion PDE as a model to describe spatio-temporal evolution of temperature field in presence of a velocity field, but the basic idea behind seeing PDE control as an RL problem is more general and can be applied to other PDEs too.

Consider a domain $\mathcal{Z} \subset \mathbb{R}^2$ (or $\mathbb{R}^3$), which might represent a confined region, e.g., a room. We denote $\partial\mathcal{Z}$ as its boundary, and we assume it has proper regularities. The time set is denoted by I, e.g., $I = \mathbb{R}_+$. Let us denote the temperature field by a time-dependent scalar field $T : \mathcal{Z} \times I \to \mathbb{R}$, and $v : \mathcal{Z} \times I \to \mathbb{R}^2$ (or $\mathbb{R}^3$) as the vector field describing velocity field. Denote $S : \mathcal{Z} \times I \to \mathbb{R}$ as the time-varying source.

The convection-diffusion equation is

$$\frac{\partial T(z,t)}{\partial t} = \frac{1}{\text{Pe}} \nabla^2 T - \nabla \cdot (vT) + S(z,t), \qquad (1)$$

in which $\text{Pe} = \frac{L v_c}{D}$ is the Péclet number with L being the characteristic length of the domain, v_c being the characteristic velocity, and D being the thermal diffusivity constant. ∇^2 and $\nabla\cdot$ represent Laplacian and divergence operators, respectively. The velocity field is divergence free to respect the continuity (conservation of mass), i.e., $\nabla \cdot v = 0$. For a given velocity and source field, the temperature field is $T(\cdot, t) \in \mathcal{T}$, in which $\mathcal{T}$ is the space of all temperature fields for a fixed time. We also use $\mathcal{S}$ to denote the space of all $S(\cdot, t)$. If there is no chance of confusion, we may simply use T, v, and S to refer to $T(\cdot, t)$, $v(\cdot, t)$, and $S(\cdot, t)$ for a particular t.

Let us partition the boundary ∂Z to ∂Z_1 and ∂Z_2, and impose the Dirichlet and Neumann boundary conditions:

$$T(z,t) = T_b(z,t), \qquad \forall z \in \partial Z_1$$
$$\vec{n} \cdot \nabla T(z,t) = 0. \qquad \forall z \in \partial Z_2$$

The Neumann boundary condition signifies an insulated temperature surface and the Dirichlet boundary condition defines a prescribed temperature surface, e.g., provided by the HVAC unit.

We consider the control problem in which we can control the PDE by changing the boundary temperature $T_b(z, t)$ (for $z \in \partial Z_1$) and flow velocity v. For example, the boundary temperature can be changed by turning on/off heaters or coolers. The flow can be controlled by using fans on the walls. For simplicity of our simulations, we consider the case that we can choose v from a given set of divergence-free flows, but in reality v is determined by the Navier-Stokes equation.

We only consider the case that the control commands (T_b and v) belong to a finite action (i.e., control) set $\mathcal{A}$ with $|\mathcal{A}| < \infty$:

$$\mathcal{A} = \{ (T_b^a, v^a) : a = 1, \ldots, |\mathcal{A}| \}.$$

This should be interpreted as choosing action a at time t leads to setting the boundary condition as $T_b(\cdot, t) = T_b^a$ and the flow velocity as $v(\cdot, t) = v^a(\cdot)$.

The dynamics of the PDE at time t is fully described when v, T, and S are all known. The function v is a part of the action that we choose. So the rest define the state of the system, which we denote by a vector $x = (T, S)$. The state space is $\mathcal{X} \triangleq \{ x = (T, S) : T \in \mathcal{T}, S \in \mathcal{S} \}$.

We can compactly write the PDE as

$$\frac{\partial x}{\partial t} = g(x(t), a(t)),$$

in which both the domain and its boundary condition are implicitly incorporated in the definition of function g.[2] To simplify and make it compatible with the MDP framework, we deal with a discrete-time version of the previous set of equations, which can be obtained by integration from time t to time $t + 1$. So we have

$$x_{t+1} = f(x_t, a_t).$$

The choice of 1 as the time step is arbitrary and could be replaced by any Δt, but for simplicity we assume it is indeed equal to 1.

More generally, one can describe the temporal evolution of the PDE by a transition probability kernel:

$$X_{t+1} \sim \mathcal{P}(\cdot|X(t), a(t)).$$

[1] Even though an image has a finite dimensional representation (e.g., a 500×500-dimensional vector in an Euclidean space), it is effectively as high-dimensional as e.g., the temperature and/or flow field of a PDE.

[2] Here we assume that the evolution of $S(z, t)$ can be described by a PDE itself. We can also deal with stochastic PDEs.

We use X instead of x in order to emphasize that it is a random variable. For deterministic dynamics, $\mathcal{P}(x|X(t),a(t)) = \delta(x - f(X(t),a(t))$, in which δ is Dirac's delta function that puts a probability mass of 1 at $f(X(t),a(t))$.

We do not often have access to f or $\mathcal{P}$, but for the moment suppose that we have access to this function, maybe through a numerical simulator. We soon discuss how this assumption is not necessary in RL.

After defining the state space $\mathcal{X}$ and the transition probability kernel $\mathcal{P}$ (or the function $f : \mathcal{X} \times \mathcal{A} \to \mathcal{X}$ for deterministic systems), we specify the reward function $r : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$. We would like this function to reflect the desirability of the current state of the system as well as the cost of the selected action.

An example of how the reward function can be defined is as follows: Consider that the comfort zone of people in the room is denoted by $\mathcal{Z}_p \subset \mathcal{Z}$, and let T^* be the desirable temperature field. This might be a constant or a spatially-varying field temperature profile. One possible definition of the reward function is then

$$r(x,a) = -\left[\int_{\mathcal{Z}_p} |T(z) - T^*(z)|^2 \mathrm{d}z + c_{\text{action}}(a)\right],$$

in which $c_{\text{action}}(a)$ is the cost of choosing the action. This might include the cost of heater or cooler operation and the cost of turning on the fan.

We might also include other terms too. For example, if people dislike the blowing of the fan to their body, we can simply include a cost term in the form of $-\int_{\mathcal{Z}_p} \|v^a(z)\|^2$ to penalize such choice of action. In general, we can include any function of x and a (or even the next state x') in the definition of the reward function, without being restricted to be in a specific analytically amenable form, for example, quadratic form common in LQR.

A measurable mapping $\pi : \mathcal{X} \to \mathcal{A}$ is called a deterministic Markov stationary policy, or simply *policy* in short. Following a policy π in an MDP means that at each time step t, we have $A_t = \pi(X_t)$.

For a policy π, the action-value function Q^π is defined as follows: Let $(R_t)_{t\geq 1}$ be the sequence of rewards when the Markov chain starts from a state-action (X_1, A_1) drawn from a positive probability distribution over $\mathcal{X} \times \mathcal{A}$ and the agent follows policy π. Then the action-value function $Q^\pi : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$ at state-action (x,a) is defined as

$$Q^\pi(x,a) \triangleq \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} R_t \,\middle|\, X_1 = x, A_1 = a\right].$$

For a discounted MDP, we define the *optimal action-value* functions by

$$Q^*(x,a) = \sup_\pi Q^\pi(x,a)$$

for all state-actions $(x,a) \in \mathcal{X} \times \mathcal{A}$. A policy π^* is *optimal* if it achieves the best values in every state, i.e., if $Q^{\pi^*} = Q^*$.

We say that a policy π is *greedy* with respect to (w.r.t.) an action-value function Q if $\pi(x) = \text{argmax}_{a\in\mathcal{A}} Q(x,a)$ for all $x \in \mathcal{X}$. We define function $\hat{\pi}(x;Q) \triangleq \text{argmax}_{a\in\mathcal{A}} Q(x,a)$ (for all $x \in \mathcal{X}$) that returns a greedy policy of an action-value function Q (If there exist multiple maximizers, a maximizer is selected in an arbitrary deterministic manner). Greedy policies are important because a greedy policy w.r.t. the optimal action-value function Q^* is an optimal policy. Hence, knowing Q^* is sufficient for behaving optimally.

The Bellman optimality operator $T^* : B(\mathcal{X} \times \mathcal{A}) \to B(\mathcal{X} \times \mathcal{A})$ is defined as

$$(T^*Q)(x,a) \triangleq r(x,a) + \gamma \int_{\mathcal{X}} \max_{a'} Q(y,a') \mathcal{P}(dy|x,a). \tag{2}$$

The Bellman optimality operator has a nice property that its fixed point is the optimal action-value function, i.e., $Q^* = T^*Q^*$. In the next section, we describe a method to find an approximate solution to the fixed-point of the Bellman optimality operator.

III. Regularized Fitted Q-Iteration

The Regularized Fitted Q-Iteration (RFQI) algorithm [7] is an instance of the family of Approximate Value Iteration (AVI) algorithms [5], [18], [17], [10], [15]. RFQI is a flexible algorithm that can work with a variety of function spaces. Because it uses regularization, it can control the complexity of the estimated value function, so avoiding pitfalls such as overfitting when a large function space is used.

We present RFQI when it uses a reproducing kernel Hilbert space (RKHS) [20] to represent the value function. There are three reasons for this choice: 1) RKHSs are flexible family of function spaces with nice approximation theoretic properties. For example, an RKHS corresponding to a universal kernel is dense (w.r.t. the supremum norm) in the space of continuous functions on $\mathbb{R}^d$. 2) The use of RKHS leads to computationally feasible optimization problems, as we shortly see. 3) By treating the solution of a PDE as if it is an image, we can define and compute the RKHS's kernel, similar to how it is done for computer vision problems. We briefly describe RFQI. For more details, refer to [7] or Chapter 5 of [6].

The RFQI algorithm is an iterative algorithm that approximately performs the Value Iteration (VI). A generic VI algorithm iteratively performs

$$Q_{k+1} = T^*Q_k.$$

Since T^* is a contraction mapping with Q^* as its fixed point, in the limit $Q_k \to Q^*$.

For MDPs with large state space, performing the exact VI is often impractical. In such cases, we can try AVI instead:

$$Q_{k+1} \approx T^*Q_k,$$

in which Q_{k+1} is represented by a function from a function space $\mathcal{F}^{|\mathcal{A}|} : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$. The function space $\mathcal{F}^{|\mathcal{A}|}$ is potentially much smaller than the space of all measurable functions on $\mathcal{X} \times \mathcal{A}$. The choice of $\mathcal{F}^{|\mathcal{A}|}$ is an important aspect of an AVI algorithm. Roughly speaking, if T^*Q_k can be well-approximated within $\mathcal{F}^{|\mathcal{A}|}$, AVI procedure would

perform fine. This suggests that we should pick $\mathcal{F}^{|\mathcal{A}|}$ that can represent a large class of functions. One such example is the Sobolev space $\mathbb{W}^k(\mathcal{X} \times \mathcal{A})$, and another is an RKHS with a universal kernel [20]. We work with RKHS-based variants in this work. The AVI procedure and some particular variants of it have been theoretically analyzed. For error propagation analysis of AVI procedures, refer to [16], [9]. For statistical analysis of some variants of AVI, refer to [17], [7], [6].

In addition to the challenge of dealing with a large state space, we may also face the problem that the integral in the definition of T^*Q_k (2) cannot be computed easily. For instance this might be because we do not have direct access to $\mathcal{P}$, but instead we only have data from interacting with the dynamical system (RL setting). Another possibility is that the model is available, but it is too complex for the exact computation of the integral. In these situations, we assume that we only have a sample $R_i \sim \mathcal{R}(\cdot|X_i, A_i)$ drawn from the reward distribution and a sample $X_i' \sim \mathcal{P}(\cdot|X_i, A_i)$ drawn from the next-state distribution for a finite set of state-action pairs $\{(X_i, A_i)\}_{i=1}^n$.

Notice that for any fixed measurable function Q,

$$\mathbb{E}\left[R(x,a) + \gamma \max_{a' \in \mathcal{A}} Q(X', a') \mid X = x, A = a\right] = (T^*Q)(x,a),$$

that is, the conditional expectation of samples in the form of $R(x,a) + \gamma \max_{a' \in \mathcal{A}} Q(X', a')$ is indeed the same as T^*Q_k. Finding this expectation is the problem of regression, which is well-studied in the machine learning and statistics literature [14], [13], [24]. RFQI is an AVI algorithm that uses regularized least-squares regression estimation.

RFQI works as follows (Algorithm 1). At iteration k, we are given a dataset $\mathcal{D}_n^{(k)} = \{(X_i, A_i, R_i, X_i')\}_{i=1}^n$ with $X_i \sim \nu_{\mathcal{X}}$, a sampling distribution over the state space $\mathcal{X}$, the action $A_i \sim \pi_b(\cdot|X_i)$, a behaviour policy, the reward $R_i \sim \mathcal{R}(\cdot|X_i, A_i)$, and the next state $X_i' \sim \mathcal{P}(\cdot|X_i, A_i)$. The RKHS-based RFQI uses a function space $\mathcal{F}^{|\mathcal{A}|} = \mathcal{H} : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$ corresponding to a kernel function $\mathrm{K} : (\mathcal{X} \times \mathcal{A}) \times (\mathcal{X} \times \mathcal{A}) \to \mathbb{R}$. At each iteration, given the solution $\hat{Q}_k$ of the previous iteration, it computes a new $\hat{Q}_{k+1}$ by solving the following regularized least squares regression problem:

$$\hat{Q}_{k+1} \leftarrow \operatorname*{argmin}_{Q \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \left| Q(X_i, A_i) - \left[R_i + \gamma \max_{a' \in \mathcal{A}} \hat{Q}_k(X_i', a')\right]\right|^2 + \lambda_{Q,n} \|Q\|_{\mathcal{H}}^2 . \tag{3}$$

Here $\lambda_{Q,n} > 0$ is the regularization coefficient.

The function space $\mathcal{H}$, being a Hilbert space, can be infinite dimensional. But for Hilbert spaces that have the reproducing kernel property, one can prove a representer theorem [19] stating that the solution of this optimization problem, even when $\mathcal{H}$ is infinite dimensional, has a finite representation in the form of:

$$\hat{Q}_{k+1}(x,a) = \sum_{i=1}^{n} \alpha_i^{(k+1)} \mathrm{K}\left((X_i, A_i), (x,a)\right),$$

for some vector $\boldsymbol{\alpha}^{(k+1)} = (\alpha_1^{(k+1)}, \ldots, \alpha_n^{(k+1)})^\top \in \mathbb{R}^n$.

Since RFQI works iteratively, it is reasonable to assume that $\hat{Q}_k$ has a similar representation (with $\boldsymbol{\alpha}^{(k)}$ instead of $\boldsymbol{\alpha}^{(k+1)}$). For notational simplicity, assume that the same dataset is used in all iterations. Moreover, suppose that the initial value function is zero, i.e., $\hat{Q}_0 = 0$. We can now replace $\hat{Q}_k$ and $\hat{Q}_{k+1}$ in the optimization problem (3) by their expansions. We use the fact that for $Q(x,a) = \sum_{i=1}^n \alpha_i \mathrm{K}\left((X_i, A_i), (x,a)\right)$, the RKHS squared norm is $\|Q\|_{\mathcal{H}}^2 = \boldsymbol{\alpha}^\top \boldsymbol{K} \boldsymbol{\alpha}$, with $\boldsymbol{K}$ being the Grammian matrix to be defined shortly. After some algebraic manipulations, we get that the solution of (3) is

$$\boldsymbol{\alpha}^{(k+1)} = \begin{cases} (\boldsymbol{K} + n\lambda \mathbf{I})^{-1} \boldsymbol{r} & k = 0, \\ (\boldsymbol{K} + n\lambda \mathbf{I})^{-1} \left(\boldsymbol{r} + \gamma \boldsymbol{K}_k^+ \boldsymbol{\alpha}^{(k)}\right) & k \geq 1. \end{cases}$$

Here $\boldsymbol{r} = (R_1, \ldots, R_n)^\top$. To define $\boldsymbol{K}, \boldsymbol{K}_k^+ \in \mathbb{R}^{n \times n}$, first denote $A_i^{*(k)} = \operatorname{argmax}_{a' \in \mathcal{A}} \hat{Q}_k(X_i', a')$, i.e., the greedy action w.r.t. $\hat{Q}_k$ at the next-state X_i'. We then have

$$[\boldsymbol{K}]_{ij} = \mathrm{K}\left((X_i, A_i), (X_j, A_j)\right),$$
$$[\boldsymbol{K}_k^+]_{ij} = \mathrm{K}\left((X_i', A_i^{*(k)}), (X_j, A_j)\right).$$

If datasets are different at each iteration, we need to slightly change the definitions of $\boldsymbol{K}$ and $\boldsymbol{K}_k^+$ (See Section 5.2 of [6]).

This computation is performed for K iterations to obtain the estimate $\hat{Q}_K(x,a) = \sum_{i=1}^n \alpha_i^K \mathrm{K}\left((X_i, A_i), (x,a)\right)$ of the optimal action-value function Q^*. The obtained policy is the greedy policy of $\hat{Q}_K$, i.e., $\hat{\pi}(x; \hat{Q}_K) = \operatorname{argmax}_{a \in \mathcal{A}} \hat{Q}_K(x,a)$.

The choice of RKHS kernel K is flexible. One possible choice is the squared exponential (i.e., Gaussian) kernel, i.e., $\mathrm{K}(x_1, x_2) = \exp(-\frac{\|x_1 - x_2\|_{\mathcal{X}}^2}{\sigma^2})$, in which $\sigma > 0$ is the bandwidth parameter and $\|\cdot\|_{\mathcal{X}}$ is the norm defined over the state space. This norm measures how close two states x_1 and x_2 are. Since states of a PDE are a scalar or vector field such as the temperature and airflow fields over $\mathcal{Z}$, the norm is defined on an infinite dimensional vector space. To compute this norm, we treat states as if they are images, i.e., a very high but finite dimensional vector with a specific spatial structure. This opens up the possibility of using several distance measures between images that have been studied in the computer vision literature. We may also use norms that have been developed in the fluid mechanics community [23]. The kernel between state-action pairs (x,a) may be defined as $\mathrm{K}((x_1,a_1),(x_2,a_2)) = \mathrm{K}(x_1,x_2)\mathrm{K}'(a_1,a_2)$, in which $\mathrm{K}'(a_1,a_2)$ is a kernel defined over the actions. In our experiments, we simply use Euclidean distance between two

Algorithm 1 Regularized Fitted Q-Iteration (K,$\mathcal{F}^{|\mathcal{A}|}$,$\lambda_{Q,n}$)

// K: Number of iterations
// $\mathcal{D}_n^{(0)}, \ldots, \mathcal{D}_n^{(K-1)}$: Datasets
// $\mathcal{F}^{|\mathcal{A}|}$: The action-value function space
// $\lambda_{Q,n}$: The regularization coefficient
$\hat{Q}_0 \leftarrow 0$
for $k = 0$ to $K - 1$ **do**
  Generate training samples $\mathcal{D}_n^{(k)}$
  $\hat{Q}_{k+1} \leftarrow$ Regularized Regression$(\hat{Q}_k, \lambda_{Q,n}; \mathcal{D}_n^{(k)})$
end for
return $\hat{Q}_K$ and $\pi_K(\cdot) = \hat{\pi}(\cdot; \hat{Q}_K)$

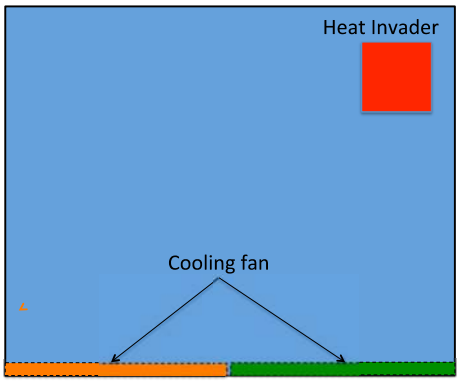

Fig. 1. Heat Invader domain. A heat source starts from a random initial position and moves towards the floor. The agent can turn two cooling fans ON or OFF.

images, thus ignoring the spatial structure within the scalar field. We also use $\mathrm{K}'(a_1, a_2) = \mathbb{I}\{a_1 = a_2\}$.

Finally we would like to remark that one may change the dataset $\mathcal{D}_n^{(k)}$ at each iteration. For example, we may use the obtained policy π_k, instead of a fixed behaviour policy π_b, to generate new samples for the next iteration. In that case, the probability distribution $\nu_{\mathcal{X}}$ is not fixed through all iterations. In the reported experiments, we use a single $\mathcal{D}_n$ for all iterations. We may also change the function space $\mathcal{F}^{|\mathcal{A}|}$ and the regularization coefficient $\lambda_{Q,n}$ at different iterations, so that we adapt to the regularities of the action-value functions at different iterations. For more detail about RFQI and its theoretical properties, refer to Chapter 5 of [6].

IV. EXPERIMENTS

We perform our experiments on a time-varying convection-diffusion domain, which we call *Heat Invader* problem, see Figure 1. In the heat invader problem, a heat source enters a room at a randomly chosen location at the top half of the room. This heat source moves with a constant speed (downward and leftward) across the room until it leaves from the floor. As a result of this moving heat disturbance, the temperature field of the room, which is governed by a convection-diffusion PDE, changes. Heat invader can be thought of as a person entering the room and moving across the room, thus disturbing the temperature field. There are two cooling fans, which can be turned ON or OFF, on the floor. These can be used to fight off the heat invader. The goal is to choose when each of them should be ON or OFF in order to make the room "comfortable" while minimizing the energy.[3]

The state of the system x at time t depends on the temperature field $T(\cdot, t)$. T is a scalar field, so in theory it is an infinite dimensional object. In our simulations, we use a finite volume solver (FiPy by [12]) with a 50×50 grid to represent the room, so T is represented by a 2500 dimensional real-valued vector.[4] The heat invader defines the source S in (1). At the location of the heat invader, a rectangle whose width is $1/5$th of the room's, the value of $S(z, t) = 1$, and it is zero elsewhere. Since the heat invader moves with a downward/leftward velocity, this defines a time-varying source. We choose the Péclet number $\mathrm{Pe} = 500$ in our experiments. Later we describe what input we feed to the RFQI algorithm.

The action set $\mathcal{A} = \{ (T_b^a, v^a) : a = 1, \ldots, |\mathcal{A}| \}$ in our experiments has four elements corresponding to a being one of OFF/OFF, ON/OFF, OFF/ON, and ON/ON settings of the left/right cooling fans. When either of them is ON, the Dirichlet boundary condition of the temperature, T_b^a, at the corresponding side of the floor is set to -0.5, and 0 on the other side (unless both are ON). Moreover, the fan induces a constant upward airflow at the same side of the room as the fan is. That is, $v^a(x, y) = 0e_x + 5e_y$ for all x in the left or right side of the room, and zero on the opposite side ($e_{x/y}$ is the unit vector in the x or y direction). This is a divergence-free field.

Of course, a real fan does not necessarily induce a constant flow v^a, but for simplicity of our simulations, and to avoid solving the Navier-Stokes equations, we assumed these specific divergence-free vector fields. Note that since we effectively control both the temperature on the boundary and the velocity field, the current formulation is indeed a nonlinear (bilinear) control problem due to $\nabla \cdot (vT)$ term in the convection-diffusion

The reward function (the negative of cost) encodes our belief about what a comfortable room setting should be, in addition to the operation cost of the cooling fans. If the temperature field at any given point in the room is within a specific threshold of the desired temperature, there would not be any cost associated. Otherwise, that point is considered undesirable, and it contributes to the cost. The part of the reward due to the uncomfortable temperature is the average value of this criteria over the whole room. The part related to the operation cost is linear in the number of cooling fans that are ON. More precisely,

$$r(T, a) = -\int_{\mathcal{Z}} \mathbb{I}\{|T(z) - T^*(z)| > \Delta T_{\text{threshold}}\} \mathrm{d}\mu(z) - \begin{cases} 0 & a = (\text{OFF,OFF}) \\ c_{\text{actuator}} & a \in \{(\text{ON,OFF}), (\text{OFF, ON})\} \\ 2c_{\text{actuator}} & a = (\text{ON,ON}) \end{cases} \quad (4)$$

In our experiments, we set the desired temperature profile $T^*(z)$ uniformly equal to 0, the threshold $\Delta T_{\text{threshold}} = 0.5$, and $c_{\text{actuator}} = 0.025$. Here μ is a uniform probability measure, i.e., the volume of $\mathcal{Z}$ according to μ is 1. Note that this reward function is not linear or quadratic in T. We set the discount factor $\gamma = 0.9$.

We consider two policies that are obtained using RFQI and several policies that are manually designed. The first RFQI-

[3]The name of *Heat Invader* is inspired from the Space Invader game on Atari 2600, which has been one of the games that is recently attempted to be solved using a related RL algorithm [15]. Heat Invader is the PDE version of the Atari game.

[4]We performed some simple grid studies. A finer 100×100 discretization has essentially the same behaviour, so to save the computation time we performed our experiments only on 50×50 grid.

based policy is the one that uses the whole scalar field T as its state x. This is a very close approximation of the true state of the system. We refer to this policy as "state".

In an application to an HVAC system, however, the current technology does not allow us to have access to the whole temperature field of the room. Instead, we only observe a smaller subset of that field. An example is when we can read the temperature on the walls, for instance using an infrared camera. In our experiments, we read the temperature from four walls, two of which are virtual. The first two are the left and right walls of the room.[5] The other two are virtual walls located at 15 grid cells away from the left and the right walls. These virtual walls are non-intrusive because they do not affect the temperature field or the velocity field. These 4×50-dimensional temperature readings define the input to the RFQI algorithm. We refer to this case as the RFQI with "walls".

We generate the batch of data $\mathcal{D}_n = \{(X_i, A_i, R_i, X_i')\}_{i=1}^n$, to be used by the RFQI algorithm, as follows: We randomly generate the initial position of the heat invader at time $t = 0$ close to the top of the room. The initial temperature $T(\cdot; 0)$ is set to zero at time $t = 0$. We let the temperature diffuse according to the convection-diffusion equation for one time step (since there is no convection field, it is only the diffusion term that acts at this time step). This results in the temperature field $T(\cdot; 1)$, which is a close approximation of the true state of the system. We choose action A_1 uniformly random from the set of all possible actions $\mathcal{A} = \{(\text{OFF/OFF}), (\text{ON/OFF}), (\text{OFF/ON}), (\text{ON/ON})\}$ to obtain the new temperature field at time $t = 2$. The reward at time $t = 1$ is $R_1 = r(T(\cdot; 2), A_1)$ with r being defined in (4). Meanwhile the heat invader moves with a velocity that is constant independent of its initial position. Depending on whether we use the "state" representation or the "walls" representation, we either use $T(\cdot; t)$ or its projections on the walls as X_t. After 40 steps, which defines one episode, we reset the state of the environment to its initial value and then pick a new random location for the heat invader.[6] This procedure is repeated until we obtain n data points.

Table I compares several RL-based and non-learning-based policies. The table shows both the average reward per episode (i.e., $\sum_{t=1}^{T_f} R_t$) and the return (i.e., $\sum_{t=1}^{T_f} \gamma^{t-1} R_t$) with T_f being the episode's length, which is 40 in our experiments.[7] In this table, we report the negative value of the average reward or return, so that all values are shown as positive. As a result, smaller values are better. For non-RL-based algorithms, the empirical average is computed based on 40 independent runs. For the RL-based solutions (RFQI (state) and RFQI (walls) in the table), we first generate a dataset with $n = 20,000$, then compute the policy. Afterwards, we evaluate the policy by running it from 20 random initial states. We repeat this procedure 20 times.

TABLE I

PERFORMANCE COMPARISON FOR SEVERAL DIFFERENT POLICIES. THE EMPIRICAL AVERAGES AND STANDARD DEVIATIONS ARE SHOWN.

Policy Name	**Average Reward**	**Return**
(OFF/OFF)	0.380 ± 0.007	2.172 ± 0.041
(OFF/ON)	0.207 ± 0.03	1.537 ± 0.137
(ON/OFF)	0.204 ± 0.03	1.531 ± 0.164
(ON/ON)	0.0755 ± 0.006	0.977 ± 0.031
Random (all)	0.090 ± 0.017	1.334 ± 0.150
Random (always ON)	0.0752 ± 0.007	1.163 ± 0.091
Simple Controller	0.0724 ± 0.0024	1.152 ± 0.051
Smart Controller	0.0498 ± 0.0032	0.979 ± 0.047
RFQI (state) with $n = 20,000$	0.0417 ± 0.0019	0.901 ± 0.029
RFQI (walls) with $n = 20,000$	0.0557 ± 0.0095	0.964 ± 0.048

We first evaluate several trivial policies. The first one is denoted by (OFF/OFF), and is the policy that never turns any of the cooling fans ON. This is the behaviour of the room when the HVAC system does not react to the heat invader. Both the average reward and the return are quite large. Not surprisingly the performances of policies (OFF/ON), (ON/OFF), and (ON/ON) are much better than (OFF/OFF), with (ON/ON) being the superior among them. The reason is that (OFF/ON) and (ON/OFF) rely on the slow diffusion process to cool down the side of the room whose cooling fan is off, while (ON/ON) can cool both sides at the same time. On the other hand, (ON/ON) is not efficient in terms of operation cost of turning on the cooling fans, cf. the second term of the reward function (4).

The next two policies randomly choose the actions at each time step. Their difference is that Random (all) uniformly chooses between all four actions, but Random (always ON) only chooses between the actions that have at least one ON cooling fans. Both policies are more effective than (OFF/ON) or (ON/OFF). The reason is that by randomly switching between turning on the cooling fans on the left and right side of the room, the discomfort zone created by the heat invader would be cooled down soon. The Random (all) controller, however, chooses OFF/OFF with probability $1/4$, so it is slower in eliminating the discomfort.

The average rewards of (ON/ON) and Random (always ON) are comparable, but their returns are different. The reason is that because of discounting by $\gamma = 0.9$, making the room comfortable as soon as possible is preferable and has a smaller (negative) return. It also does not matter much if the cooling fans are left ON at steps far in the future because their contributions to the operation cost are discounted. So even though (ON/ON) leads to higher average operation cost, its faster cleaning up of the discomfort regions make it better (in return sense) than a slower Random (either all or always ON). Notice that since none of these policies takes into account the location of the heat invader, it is reasonable to expect that they would not be optimal.

We now study the results of two non-trivial manually-

[5]The left and right walls have the Neumann boundary condition.

[6]The heat invader takes at most 25 steps to start from the top of the room until it leaves it. The value of $t = 40$ is chosen accordingly, so that the state of the system gets approximately settled if no action is taken.

[7]Even though the discounted MDP formulation is for infinite horizon problems, for evaluation we truncate the episode at the horizon T_f, mainly to save the computation cost.

Algorithm 2 Smart Controller (T)

// Input: The temperature field T
// Output: An action
{Computing the mean comfortable temperature violation in the left and right side of the room}
$\tilde{T}_L = \int_{\mathcal{Z}_L} \mathbb{I}\{|T(z) - T^*(z)| > \Delta T_{\text{threshold}}\} \mathrm{d}\mu(z)$.
$\tilde{T}_R = \int_{\mathcal{Z}_R} \mathbb{I}\{|T(z) - T^*(z)| > \Delta T_{\text{threshold}}\} \mathrm{d}\mu(z)$.
if $\max\{\tilde{T}_L, \tilde{T}_R\} < c_{\text{actuator}}$ **then**
  return $a =$ (OFF, OFF)
end if
if $\min\{\tilde{T}_L, \tilde{T}_R\} > 2c_{\text{actuator}}$ **then**
  return $a =$ (ON, ON)
end if
if $\tilde{T}_L > \tilde{T}_R$ **then**
  return $a =$ (ON, OFF)
else
  return $a =$ (OFF, ON)
end if

designed policies that take the location of the heat invader into account. These policies directly react to the discomfort level caused by the heat invader. The first one, called "Simple controller", measures the mean temperature in the left half of the room and compares it with the mean temperature in the right half of the room. Depending on which one is higher, the corresponding cooling fan would be ON.

Simple controller does not take into account the operation cost of the cooling fans. It also does not take into account the fact that whenever the temperature of a point is within the threshold $\Delta T_{\text{threshold}}$, it does not contribute to the reward function. Moreover, this policy is myopic, i.e., it does not consider the future states of the system in deciding whether to turn either the left or right cooling fans ON. We would like to remark that this policy has access to the temperature field of the whole room, similar to the "state" case above.

The second policy, called "Smart controller", described in Algorithm 2, addresses some of these issues. It only considers the $\Delta T_{\text{threshold}}$-threshold effect of the temperature. It also takes into account the operation cost by not turning the cooling fans ON whenever the maximum possible reduction in the thermal discomfort is less than the operation cost of turning them ON. In other words, this policy is more mindful of the structure of the reward function than Simple controller. Nonetheless, this policy is still myopic and does not plan for the future horizons.

We see the difference of Smart controller from Simple controller (as well as Random (always ON) or (ON/ON)) in both the average reward and return. Smart controller is aware that within threshold $\Delta T_{\text{threshold}}$, the temperature of a point does not contribute to the reward function. It also considers the operation cost. The result is that both of its measures of performance is much better than any other controller that we have studied so far.

We now turn to the RFQI-based policies (state and walls) with the number of data points $n = 20,000$. We see that the RFQI-based policy that has access to the state information performs significantly better than all other policies in this experiment, both in terms of average reward (which it does *not* optimize) and return (which it does indeed optimize).

The reason for this superiority, comparing to e.g., Smart controller, is that it is not a myopic policy and considers the effect of its actions not only on the immediate reward, but on the future rewards too. When the RFQI algorithm has only access to a subspace of the temperature, i.e., the "walls" case, the performance degrades noticeably. The expected return becomes comparable to the Smart controller's, but its average reward is worse. Even in this case, however, the return, which is the measure that is optimized by RFQI, is as good as any other non-RL-based controller. Since both Smart and Simple controllers use the "state" information, their informational input is richer than the input given to RFQI (walls).

We study the effect of the number of samples on the performance of RFQI-based policies. Figure 2 compares two RFQI-based policies ("state" and "walls") with two manually-designed policies. The horizontal axes of both graphs indicate the number of training samples n used for the RFQI-based policies. The Simple and Smart controllers do not depend on n. On the left graph, we show the average reward obtained per each episode, and on the right graph, we show the expected return. As before, the data is generated by following a random behaviour policy to obtain $20,000$ data points. Each 40 data points correspond to a single episode with a random initial starting point for the heat invader. So each episode is independent from the previous ones, while the samples within each episode are dependent. After collecting data, we only use a subset of the data points (i.e., $n \in \{1240, 2000, 3160, 5000, 7960, 12600, 20000\}$) to obtain the policy using the RFQI algorithm. Depending on whether we consider the "state" or "walls" scenario, we pass all or just a subset of all the temperature field as the input to the RFQI algorithm. After obtaining each policy, we evaluate it with 20 independent initial state. We repeat this procedure for 20 independent runs. The graphs depict the empirical average of the average reward per episode (left) and return (right).

The graphs indicate the progress in the quality of learned policies. When there are not many data points available, the performance is not very good, as expected for a learning-based approach. For instance, when we only use 1240 data points to obtain the policy, the average return is around -1.15 to -1.10, which is comparable to the return of Random (always ON) and Simple controller, but worse than Smart controller's. The average reward, which is not explicitly optimized by RFQI, is comparable to Random (all), but worse than Simple controller's. When the number of samples increases, the performance increases too. Eventually the "state"-based policy surpasses Smart controller at around 7960 samples. The "walls"-based policy, however, takes longer to be comparable to Smart controller in terms of return, and it never outperforms it in term of the average reward. Note that Smart controller uses the state information, while the "walls"-based one does not.

V. CONCLUSIONS AND FUTURE WORK

In this work we proposed to formulate a class of PDE control problems as RL problems. We suggested to directly

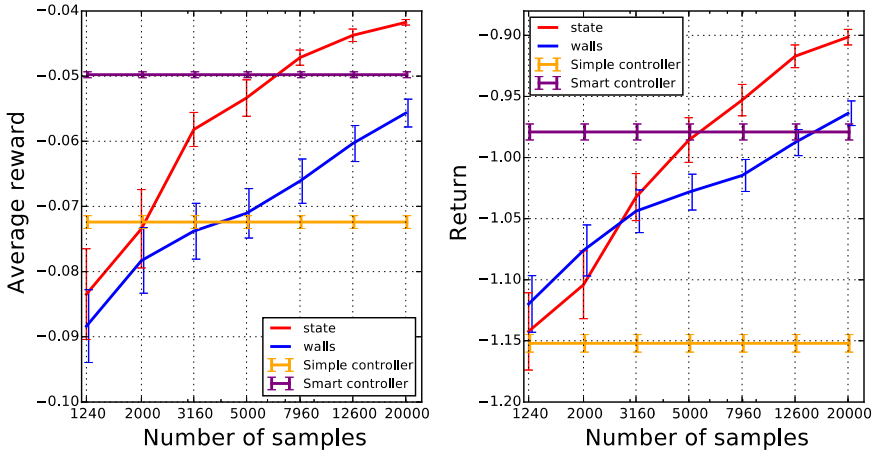

Fig. 2. The average reward per episode (left) and the return (right) of several policies as a function of the number of training points. The red curve shows the RFQI-based policy that uses the whole temperature field, while the blue one shows the policy that has access only to the temperature on "walls". For comparison, the performances of two manually-designed controllers are shown too. The error bars depict one standard error.

work with the state of PDE, instead of reducing the PDE to an ODE and then designing a controller. The advantage of this approach is that it is data-driven, so the knowledge of the PDE is not required. Moreover since we directly work with the state of PDE, we can potentially handle more complex problems such as the control of flows with high Reynolds number, which is challenging for the reduce-then-design-based methods. Studying these flows is a direction for future research.

Our approach is not limited to an RKHS-based solution, or even an RFQI-based method. By seeing the PDE control as a very high-dimensional image-like RL problem, we can use a variety of algorithms that can handle such problems, e.g., deep neural network-based AVI algorithms [15].

Finally notice that in our current simulations we consider that the airflow field v is given. Performing a more realistic simulation that finds v itself by solving the Navier-Stokes equation would be more realistic and interesting.

References

[1] Sunil Ahuja, Amit Surana, and Eugene Cliff. Reduced-order models for control of stratified flows in buildings. In *American Control Conference (ACC)*, pages 2083–2088. IEEE, 2011.

[2] Jeff Borggaard, John A. Burns, Amit Surana, and Lizette Zietsman. Control, estimation and optimization of energy efficient buildings. In *American Control Conference (ACC)*, pages 837–841, 2009.

[3] John A Burns, Xiaoming He, and Weiwei Hu. Feedback stabilization of a thermal fluid system with mixed boundary control. *Computers & Mathematics with Applications*, 2016.

[4] John A Burns and Weiwei Hu. Approximation methods for boundary control of the Boussinesq equations. In *IEEE Conference on Decision and Control (CDC)*, pages 454–459, 2013.

[5] Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research (JMLR)*, 6:503–556, 2005.

[6] Amir-massoud Farahmand. *Regularization in Reinforcement Learning*. PhD thesis, University of Alberta, 2011.

[7] Amir-massoud Farahmand, Mohammad Ghavamzadeh, Csaba Szepesvári, and Shie Mannor. Regularized fitted Q-iteration for planning in continuous-space Markovian Decision Problems. In *Proceedings of American Control Conference (ACC)*, pages 725–730, June 2009.

[8] Amir-massoud Farahmand, Mohammad Ghavamzadeh, Csaba Szepesvári, and Shie Mannor. Regularized policy iteration with nonparametric function spaces. *Journal of Machine Learning Research (JMLR)*, 17(139):1–66, 2016.

[9] Amir-massoud Farahmand, Rémi Munos, and Csaba Szepesvári. Error propagation for approximate policy and value iteration. In *Advances in Neural Information Processing Systems (NIPS - 23)*, pages 568–576. 2010.

[10] Amir-massoud Farahmand and Doina Precup. Value pursuit iteration. In *Advances in Neural Information Processing Systems (NIPS - 25)*, pages 1349–1357, 2012.

[11] DPG Foures, Colm-cille Caulfield, and Peter J. Schmid. Optimal mixing in two-dimensional plane poiseuille flow at finite Péclet number. *Journal of Fluid Mechanics*, 748:241–277, 2014.

[12] Jonathan E. Guyer, Daniel Wheeler, and James A. Warren. FiPy: Partial differential equations with Python. *Computing in Science and Engineering*, 11(3):6–15, 2009.

[13] László Györfi, Michael Kohler, Adam Krzyżak, and Harro Walk. *A Distribution-Free Theory of Nonparametric Regression*. Springer Verlag, New York, 2002.

[14] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2001.

[15] Volodymyr Mnih et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 02 2015.

[16] Rémi Munos. Performance bounds in L_p norm for approximate value iteration. *SIAM Journal on Control and Optimization*, pages 541–561, 2007.

[17] Rémi Munos and Csaba Szepesvári. Finite-time bounds for fitted value iteration. *Journal of Machine Learning Research (JMLR)*, 9:815–857, 2008.

[18] Martin Riedmiller. Neural fitted Q iteration – first experiences with a data efficient neural reinforcement learning method. In *16th European Conference on Machine Learning*, pages 317–328, 2005.

[19] Bernhard Schölkopf, Ralf Herbrich, and Alex J. Smola. A generalized representer theorem. In *Proceedings of the 14th Annual Conference on Computational Learning Theory (COLT)*, pages 416–426, 2001.

[20] Ingo Steinwart and Andreas Christmann. *Support Vector Machines*. Springer, 2008.

[21] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 1998.

[22] Csaba Szepesvári. *Algorithms for Reinforcement Learning*. Morgan Claypool Publishers, 2010.

[23] Jean-Luc Thiffeault. Using multiscale norms to quantify mixing and transport. *Nonlinearity*, 25(2):R1, 2012.

[24] Larry Wasserman. *All of Nonparametric Statistics*. Springer, 2007.