In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'07), Rome, Italy (2007).

Value Function Approximation on Non-Linear Manifolds for Robot Motor Control

Masashi Sugiyama[*,†] Hirotaka Hachiya[†] Christopher Towell[†] and Sethu Vijayakumar[†]

Abstract— **The least squares approach works efficiently in value function approximation, given appropriate basis functions. Because of its smoothness, the Gaussian kernel is a popular and useful choice as a basis function. However, it does not allow for discontinuity which typically arises in real-world reinforcement learning tasks. In this paper, we propose a new basis function based on *geodesic Gaussian kernels*, which exploits the non-linear manifold structure induced by the Markov decision processes. The usefulness of the proposed method is successfully demonstrated in a simulated robot arm control and Khepera robot navigation.**

I. Introduction

Value function approximation is an essential ingredient of reinforcement learning (RL), especially in the context of solving Markov Decision Processes (MDPs) using policy iteration methods [1]. In problems with large discrete state space or continuous state spaces, it becomes necessary to use function approximation methods to represent the value functions. A *least squares* approach using a linear combination of predetermined *under-complete* basis functions has shown to be promising in this task [2]. Fourier functions (trigonometric polynomials), Gaussian kernels [3], and wavelets [4] are popular basis function choices for general function approximation problems. Both Fourier bases (global functions) and Gaussian kernels (localized functions) have certain smoothness properties that make them particularly useful for modeling inherently smooth, continuous functions. Wavelets provide basis functions at various different scales and may also be employed for approximating smooth functions with local discontinuity.

Typical value functions in RL tasks are predominantly smooth with some discontinuous parts [5]. To illustrate this, let us consider a toy RL task of guiding an agent to a goal in a grid world (see Fig.1(a)). In this task, a state corresponds to a two-dimensional Cartesian position of the agent. The agent can not move over the wall, so the value function of this task is highly discontinuous *across* the wall. On the other hand, the value function is smooth *along* the maze since neighboring reachable states in the maze have similar values (see Fig.1(b)). Due to the discontinuity, simply employing Fourier functions or Gaussian kernels as basis functions tend to produce undesired, non-optimal results around the discontinuity, affecting the overall performance significantly. Wavelets could be a viable alternative, but are *over-complete* bases—one has to appropriately choose a *subset* of basis functions, which is not a straightforward task in practice.

Recently, the article [5] proposed considering value functions defined not on the Euclidean space, but on *graphs* induced by the MDPs (see Fig.1(c)). Value functions which usually contain discontinuity in the Euclidean domain (e.g., across the wall) are typically smooth on graphs (e.g., along the maze). Hence, approximating value functions on graphs can be expected to work better than approximating them in the Euclidean domain.

The spectral graph theory [6] showed that Fourier-like smooth bases on graphs are given as minor eigenvectors of the *graph-Laplacian* matrix. However, their global nature implies that the overall accuracy of this method tends to be degraded by local noise. The article [7] defined *diffusion wavelets*, which posses natural multi-resolution structure on graphs. The paper [8] showed that diffusion wavelets could be employed in value function approximation, although the issue of choosing a suitable subset of basis functions from the over-complete set is not discussed—this is not straightforward in practice due to the lack of a natural ordering of basis functions.

In the machine learning community, Gaussian kernels seem to be more popular than Fourier functions or wavelets because of their locality and smoothness [3], [9], [10]. Furthermore, Gaussian kernels have ‘centers’, which alleviates the difficulty of basis subset choice, e.g., uniform allocation [2] or sample-dependent allocation [11]. In this paper, we therefore define Gaussian kernels on graphs (which we call *geodesic Gaussian kernel*), and propose using them for value function approximation. Our definition of Gaussian kernels on graphs employs the *shortest paths* between states rather than the Euclidean distance, which can be computed efficiently using the *Dijkstra algorithm* [12], [13]. Moreover, an effective use of Gaussian kernels opens up the possibility to exploit the recent advances in using Gaussian processes for temporal difference learning [11].

When basis functions defined on the state space are used for approximating the state-action value function, they should be extended over the action space. This is typically done by simply copying the basis functions over the action space [2], [5]. In this paper, we propose a new strategy for this extension, which takes into account the transition after taking actions. This new strategy is demonstrated to work very well when the transition is predominantly deterministic.

The authors acknowledge financial support from MEXT (Grant-in-Aid for Young Scientists 17700142 and Grant-in-Aid for Scientific Research (B) 18300057), the Okawa Foundation, and EU Erasmus Mundus Scholarship.

[*]Department of Computer Science, Tokyo Institute of Technology, 2-12-1, O-okayama, Meguro-ku, Tokyo, 152-8552, Japan sugi@cs.titech.ac.jp

[†]School of Informatics, University of Edinburgh, The King’s Buildings, Mayfield Road, Edinburgh EH9 3JZ, UK. H.Hachiya@sms.ed.ac.uk, C.C.Towell@sms.ed.ac.uk, sethu.vijayakumar@ed.ac.uk

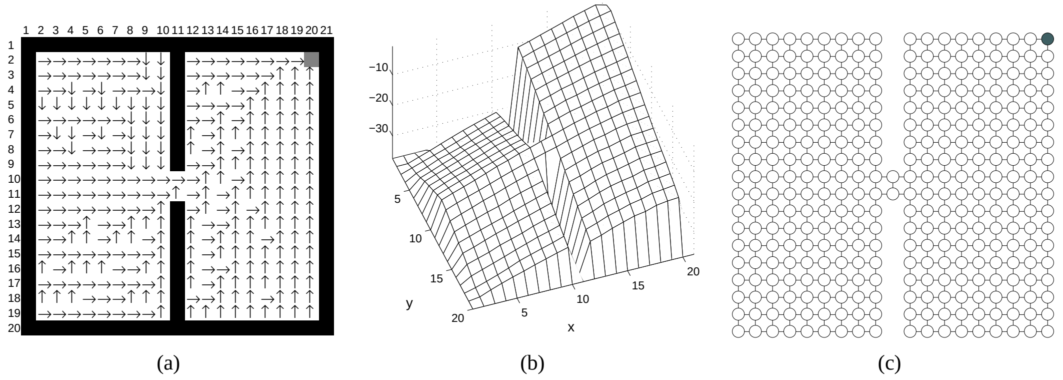


Fig. 1. An illustrative example of an RL task of guiding an agent to a goal in the grid world. (a) Black areas are walls over which the agent cannot move while the goal is represented in gray. Arrows on the grids represent one of the optimal policies. (b) Optimal state value function (in log-scale). (c) Graph induced by the MDP and a random policy.

II. Formulation of the Reinforcement Learning Problem

In this section, we briefly introduce the notation and reinforcement learning (RL) formulation that we will use across the manuscript.

A. *Markov Decision Processes*

Let us consider a Markov decision process (MDP) $(\mathcal{S}, \mathcal{A}, \mathcal{P}, R, \gamma)$, where $\mathcal{S} = \{s^{(1)}, s^{(2)}, \ldots, s^{(n)}\}$ is a finite[1] set of states, $\mathcal{A} = \{a^{(1)}, a^{(2)}, \ldots, a^{(m)}\}$ is a finite set of actions, $\mathcal{P}(s, a, s') : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the joint probability of making a transition to state s' if action a is taken in state s, $R(s, a, s') : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ is an immediate reward for making a transition from s to s' by action a, and $\gamma \in [0, 1)$ is the discount factor for future rewards. The expected reward $\mathcal{R}(s, a)$ for a state-action pair (s, a) is given as

$$\mathcal{R}(s, a) = \sum_{s' \in \mathcal{S}} \mathcal{P}(s, a, s') R(s, a, s'). \tag{1}$$

Let $\pi(s) : \mathcal{S} \rightarrow \mathcal{A}$ be a deterministic policy which the agent follows. In this paper, we focus on deterministic policies since there always exists an optimal deterministic policy [2]. Let $Q^\pi(s, a) : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ be a state-action value function for policy π, which indicates the expected long-term discounted sum of rewards the agent receives when the agent takes action a in state s and follows policy π thereafter. $Q^\pi(s, a)$ satisfies the following *Bellman equation*:

$$Q^\pi(s, a) = \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s, a, s') Q^\pi(s', \pi(s')). \tag{2}$$

The goal of RL is to obtain a policy which results in maximum amount of long-term rewards. The optimal policy $\pi^*(s)$ is defined as $\pi^*(s) = \operatorname{argmax}_{a \in \mathcal{A}} Q^*(s, a)$, where $Q^*(s, a)$ is the optimal state-action value function defined by $Q^*(s, a) = \max_\pi Q^\pi(s, a)$.

[1]For the moment, we focus on discrete state spaces. In Sec.III-D, we extend the proposed method to the continuous state space.

B. *Least Squares Policy Iteration*

In practice, the optimal policy $\pi^*(s)$ can not be directly obtained since $\mathcal{R}(s, a)$ and $\mathcal{P}(s, a, s')$ are usually unknown; even when they are known, direct computation of $\pi^*(s)$ is often computationally intractable.

To cope with this problem, the paper [2] proposed approximating the state-action value function $Q^\pi(s, a)$ using a linear model:

$$\widehat{Q}^\pi(s, a; \boldsymbol{w}) = \sum_{i=1}^{k} w_i \phi_i(s, a), \tag{3}$$

where k is the number of basis functions which is usually much smaller than the number of states, $\boldsymbol{w} = (w_1, w_2, \ldots, w_k)^\top$ are the parameters to be learned, $^\top$ denotes the transpose, and $\{\phi_i(s, a)\}_{i=1}^k$ are pre-determined basis functions. Note that k and $\{\phi_i(s, a)\}_{i=1}^k$ can depend on policy π, but we do not show the explicit dependence for the sake of simplicity. Assume we have roll-out samples from a sequence of actions: $\{(s_i, a_i, r_i, s'_i)\}_{i=1}^t$, where each tuple denotes the agent experiencing a transition from s_i to s'_i on taking action a_i with immediate reward r_i. Under the Least Squares Policy Iteration (LSPI) formulation [2], the parameter $\boldsymbol{w}$ is learned so that the Bellman equation (2) is optimally approximated in the least squares sense[2]. Consequently, based on the approximated state-action value function with learned parameter $\widehat{\boldsymbol{w}}^\pi$, the policy is updated as

$$\pi(s) \longleftarrow \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, \widehat{Q}^\pi(s, a; \widehat{\boldsymbol{w}}^\pi). \tag{4}$$

Approximating the state-action value function and updating the policy is iteratively carried out until some convergence criterion is met.

III. Gaussian Kernels on Graphs

In the LSPI algorithm, the choice of basis functions $\{\phi_i(s, a)\}_{i=1}^k$ is an open design issue. Gaussian kernels have traditionally been a popular choice [2], [11], but they

[2]There are two alternative approaches: *Bellman residual minimization* and *fixed point approximation*. We take the latter approach following the suggestion in the reference [2].

can not approximate discontinuous functions well. Recently, more sophisticated methods of constructing suitable basis functions have been proposed, which effectively make use of the *graph* structure induced by MDPs [5]. In this section, we introduce a novel way of constructing basis functions by incorporating the graph structure; while relation to the existing graph-based methods is discussed in the separate report [14].

A. MDP-Induced Graph

Let G be a graph induced by an MDP, where states $\mathcal{S}$ are nodes of the graph and the transitions with non-zero transition probabilities from one node to another are edges. The edges may have weights determined, e.g., based on the transition probabilities or the distance between nodes. The graph structure corresponding to an example grid world shown in Fig.1(a) is illustrated in Fig.1(c). In practice, such graph structure (including the connection weights) are estimated from samples of a finite length. We assume that the graph G is connected. Typically, the graph is *sparse* in RL tasks, i.e., $\ell \ll n(n-1)/2$, where ℓ is the number of edges and n is the number of nodes.

B. Ordinary Gaussian Kernels

Ordinary Gaussian kernels (OGKs) on the Euclidean space are defined as

$$K(s,s') = \exp\left(-\frac{\mathrm{ED}(s,s')^2}{2\sigma^2}\right), \tag{5}$$

where $\mathrm{ED}(s,s')$ are the Euclidean distance between states s and s'; for example, $\mathrm{ED}(s,s') = \|\boldsymbol{x} - \boldsymbol{x}'\|$ when the Cartesian positions of s and s' in the state space are given by $\boldsymbol{x}$ and $\boldsymbol{x}'$, respectively. σ^2 is the variance parameter of the Gaussian kernel.

The above Gaussian function is defined on the state space $\mathcal{S}$, where s' is treated as a center of the kernel. In order to employ the Gaussian kernel in the LSPI algorithm, it needs to be extended over the state-action space $\mathcal{S} \times \mathcal{A}$. This is usually carried out by simply 'copying' the Gaussian function over the action space [2], [5]. More precisely: let the total number k of basis functions be mp, where m is the number of possible actions and p is the number of Gaussian centers. For the i-th action $a^{(i)}$ $(\in \mathcal{A})$ $(i = 1, 2, \ldots, m)$ and for the j-th Gaussian center $c^{(j)}$ $(\in \mathcal{S})$ $(j = 1, 2, \ldots, p)$, the $(i + (j-1)m)$-th basis function is defined as

$$\phi_{i+(j-1)m}(s,a) = I(a = a^{(i)})K(s, c^{(j)}), \tag{6}$$

where $I(\cdot)$ is the indicator function, i.e., $I(a = a^{(i)}) = 1$ if $a = a^{(i)}$ otherwise $I(a = a^{(i)}) = 0$.

Gaussian kernels are *shift-invariant*, i.e., they do not directly depend on the absolute positions $\boldsymbol{x}$ and $\boldsymbol{x}'$, but depend only on the difference between two positions; more specifically, Gaussian kernels depend only on the *distance* between two positions.

C. Geodesic Gaussian Kernels

On graphs, a natural definition of the distance would be the *shortest path*. So we define Gaussian kernels on graphs based on the shortest path:

$$K(s,s') = \exp\left(-\frac{\mathrm{SP}(s,s')^2}{2\sigma^2}\right), \tag{7}$$

where $\mathrm{SP}(s,s')$ denotes the shortest path from state s to state s'. The shortest path on a graph can be interpreted as a discrete approximation to the *geodesic distance* on a non-linear manifold [6]. For this reason, we call Eq.(7) a *geodesic Gaussian kernel* (GGK).

Shortest paths on graphs can be efficiently computed using the *Dijkstra algorithm* [12]. With its naive implementation, computational complexity for computing the shortest paths from a single node to all other nodes is $O(n^2)$, where n is the number of nodes. If the *Fibonacci heap* is employed, the computational complexity can be reduced to $O(n \log n + \ell)$ [13], where ℓ is the number of edges. Since the graph in value function approximation problems is typically sparse (i.e., $\ell \ll n^2$), using the Fibonacci heap provides significant computational gains. Furthermore, there exist various approximation algorithms which are computationally very efficient (see [15] and and references therein).

Analogous to OGKs, we need to extend GGKs to the state-action space for using them in the LSPI method. A naive way is to just employ Eq.(6), but this can cause a 'shift' in the Gaussian centers since the state usually changes when some action is taken. To incorporate this transition, we propose defining the basis functions as the expectation of Gaussian functions after transition, i.e.,

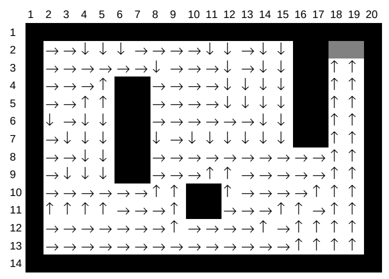
$$\phi_{i+(j-1)m}(s,a) = I(a = a^{(i)}) \sum_{s' \in \mathcal{S}} \mathcal{P}(s,a,s')K(s', c^{(j)}). \tag{8}$$

This shifting scheme is expected to work well when the transition is predominantly deterministic (see Sec.IV and Sec.V-A for experimental evaluation).

D. Extension to Continuous State Spaces

So far, we focused on discrete state spaces. However, the concept of GGKs can be naturally extended to continuous state spaces, which is explained here. First, the continuous state space is discretized, which gives a graph as a discrete approximation to the non-linear *manifold* structure of the continuous state space. Based on the graph, we construct GGKs in the *same* way as the discrete case. Finally, the discrete GGKs are interpolated, e.g., using a linear method to give *continuous* GGKs.

Although this procedure discretizes the continuous state space, it must be noted that the discretization is only for the purpose of obtaining the graph as a discrete approximation of the continuous non-linear manifold; the resulting basis functions themselves are continuously interpolated and hence, the state space is still treated as continuous as opposed to other conventional discretization procedures.


(a) Sutton's maze

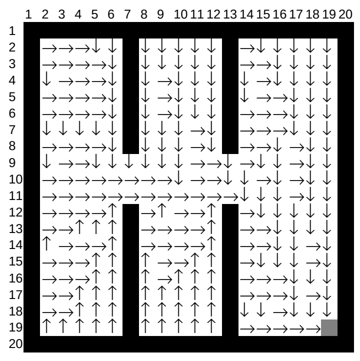
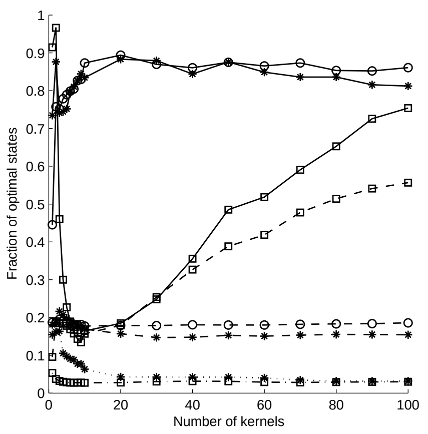
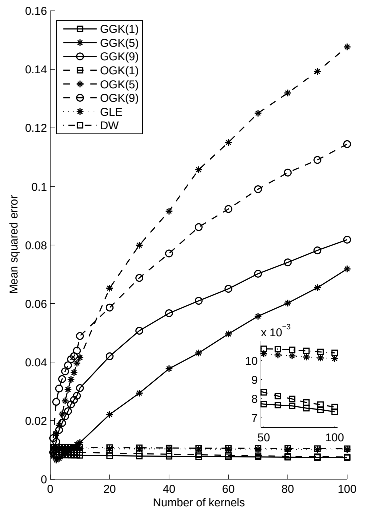

(b) Three-room maze

Fig. 2. Two intricated mazes used for simulation.


(a) Sutton's maze


(a) Sutton's maze

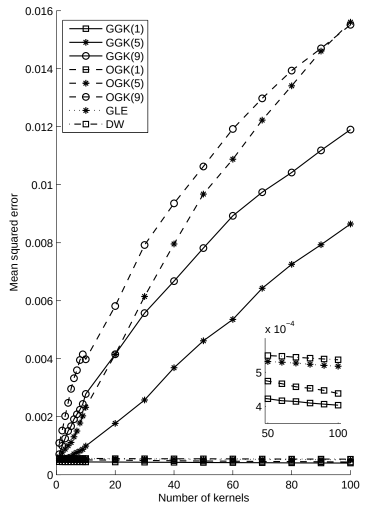

(b) Three-room maze

Fig. 3. Mean squared error of approximated value functions averaged over 100 trials for the Sutton and three room mazes. In the legend, the standard deviation σ of GGKs and OGKs is denoted in the bracket.

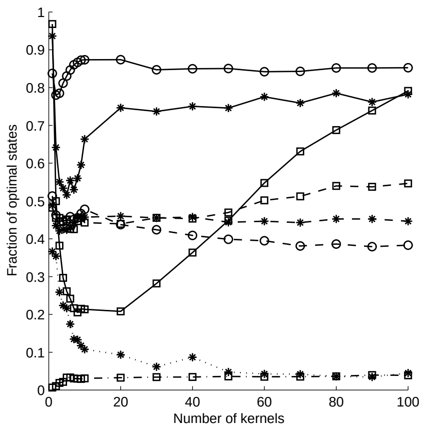

(b) Three-room maze

Fig. 4. Fraction of optimal states averaged over 100 trials for the Sutton and three room mazes.

IV. Experimental Comparison

In this section, we report the results of extensive and systematic experiments for illustrating the difference between GGKs and other basis functions. We employ two standard grid world problems illustrated in Fig.2, and evaluate the goodness of approximated *value functions* by computing the mean squared error (MSE) with respect to the optimal value function and the goodness of obtained *policies* by calculating the fraction of states from which the agent can get to the goal optimally (i.e., in the shortest number of steps). 20 series of random walk of length 300 are gathered as training samples, which are used for estimating the graph as well as the transition probability and expected reward. We set the edge weights in the graph to 1 (which is equivalent to the Euclidean distance between two nodes). We test GGKs, OGKs, *graph-Laplacian eigenfunctions* (GLEs) [5], and *diffusion wavelets* (DWs) [8].

This simulation is repeated 100 times for each maze and each method, randomly changing training samples in each run. The mean of the above scores as a function of the number of bases is plotted in Fig.4. Note that the actual number of bases is four times more because of the extension of basis functions over the action space (see Eq.(6) and Eq.(8)). GGKs and OGKs are tested with small/medium/large Gaussian widths.

Fig.3 depicts MSEs of the approximated value functions for each method. They show that MSEs of GGKs with width 1, OGKs with width 1, GLEs, and DWs are very small and decrease as the number of kernels increases. On the other hand, MSEs of GGKs and OGKs with medium/large width are large and increase as the number of kernels increases. Therefore, from the viewpoint of approximation quality of the value functions, the width of GGKs and OGKs should be small.

Fig.4 depicts the fraction of optimal states in the obtained policy. They show that overall GGKs with medium/large width give much better policies than OGKs, GLEs, and DWs. An interesting finding from the graphs is that GGKs tend to work better if the Gaussian width is large, while OGKs show the opposite trend; this may be explained as follows. Tails of OGKs extend across the wall. Therefore, OGKs with large width tend to produce undesired value function and erroneous policies around the partitions. This tail effect can be alleviated if the Gaussian width is made small.

However, this in turn makes the approximated value function fluctuating; so the resulting policies are still erroneous. The fluctuation problem with a small Gaussian width seems to be improved if the number of bases is increased, while the tail effect with a large Gaussian width still remains even when the number of bases is increased. On the other hand, GGKs do not suffer from the tail problem thanks to the geodesic construction. Therefore, GGKs allows us to make the width large without being affected by the discontinuity across the wall. Consequently, smooth value functions along the maze are produced and hence better policies can be obtained by GGKs with large widths. This result highlights a helpful property since it alleviates the practical issue of determining the values of the Gaussian width parameter.

V. Applications

As discussed in the previous section, the proposed GGKs bring a number of preferable properties for making value function approximation effective. In this section, we investigate the application of the GGK-based method to the challenging problems of a (simulated) robot arm control and mobile robot navigation and demonstrate its usefulness.

A. Robot Arm Control

We use a simulator of a two-joint robot arm (moving in a plane) illustrated in Fig.5(a). The task is to lead the end effector ('hand') of the arm to an object while avoiding the obstacles. Possible actions are to increase or decrease the angle of each joint ('shoulder' and 'elbow') by 5 degrees in the plane, simulating coarse stepper motor joints. Thus the state space $\mathcal{S}$ is the 2-dimensional discrete space consisting of two joint angles as illustrated in Fig.5(b). The black area in the middle corresponds to the obstacle in the joint angle state space. The action space $\mathcal{A}$ involves 4 actions: increase or decrease one of the joint angles. We give a positive immediate reward $+1$ when the robot's end effector touches the object; otherwise the robot receives no immediate reward. Note that actions which make the arm collide with obstacles are not allowed. The discount factor is set to $\gamma = 0.9$. In this environment, we can change the joint angle exactly by 5 degrees, so the environment is deterministic. However, because of the obstacles, it is difficult to explicitly compute an inverse kinematic model; furthermore, the obstacles introduce discontinuity in value functions. Therefore, this robot arm control task is an interesting test bed for investigating the behaviour of GGKs.

We collected training samples from 50 series of 1000 random arm movements, where the start state is chosen randomly in each trial. The graph induced by the above MDP consists of 1605 nodes and we assigned uniform weights to the edges. There are totally 16 goal states in this environment (see Fig.5(b)), so we put the first 16 Gaussian centers at the goals and the remaining centers are chosen randomly in the state space. For GGKs, kernel functions are extended over the action space using the shifting scheme (see Eq.(8)) since the transition is deterministic in this experiment.

Fig.6 illustrates the value functions approximated using GGKs and OGKs[3]. The graphs show that GGKs give a nice smooth surface with obstacle-induced discontinuity sharply preserved, while OGKs tend to smooth out the discontinuity. This makes a significant difference in avoiding the obstacle: from 'A' to 'B' in Fig.5(b), the GGK-based value function results in a trajectory that avoids the obstacle (see Fig.6(a)). On the other hand, the OGK-based value function yields a trajectory that tries to move the arm *through* the obstacle by following the gradient upward (see Fig.6(b)). The latter causes the arm to get stuck behind the obstacle.

Fig.7 summarizes the performance of GGKs and OGKs measured by the percentage of successful movements (i.e., the end effector reaches the target) averaged over 30 independent runs. More precisely, in each run, totally 50000 training samples are collected using a different random seed, a policy is then computed by the GGK- or OGK-based method using LSPI, and the obtained policy is tested. This graph shows that GGKs remarkably outperform OGKs since the arm can successfully avoid the obstacle. The performance of OGK does not go beyond 0.6 even when the number of kernels is increased. This is caused by the 'tail effect' of ordinary Gaussian functions; the OGK-based policy can not lead the end effector to the object if it starts from the bottom-left half of the state space

When the number of kernels is increased, the performance of both GGKs and OGKs once gets worse at around $k = 20$. This would be caused by our kernel center allocation strategy: the first 16 kernels are put at the goal states and the remaining kernel centers are chosen randomly. When k is less than or equal to 16, the approximated value function tends to have a unimodal profile since all kernels are put at the goal states. However, when k is larger than 16, this unimodality is broken and the surface of the approximated value function gets slightly fluctuated. This small fluctuation can cause an error in policies and therefore the performance is degraded at around $k = 20$. This performance degradation tends to be improved as the number of kernels is further increased.

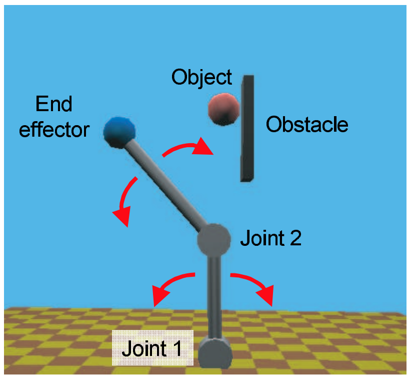
Overall, the above result shows that when GGKs are combined with our kernel center allocation strategy, almost perfect policies can be obtained with a very small number of kernels. Therefore, the proposed method is computationally very advantageous.

B. Robot Agent Navigation

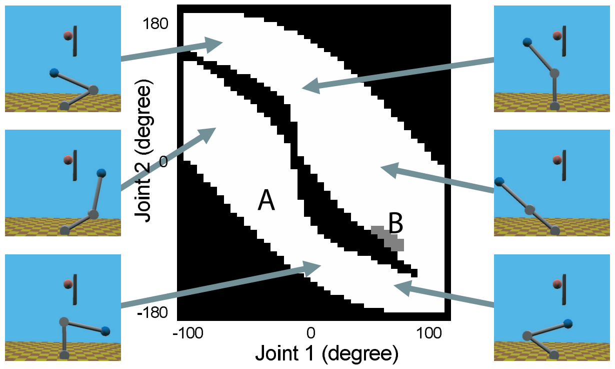
The above simple robot arm control simulation shows that the GGK method is promising. Here we apply GGKs to a more challenging task of a mobile robot navigation, which involves a *high-dimensional* and *continuous* state space.

We employ a *Khepera* robot illustrated in Fig.8(a) on a navigation task. A Khepera is equipped with 8 infra-red

[3]For illustration purposes, let us display the state value function $V^{\pi}(s) : \mathcal{S} \rightarrow \mathbb{R}$, which is the expected long-term discounted sum of rewards the agent receives when the agent takes actions following policy π from state s. From the definition, it can be confirmed that $V^{\pi}(s)$ is expressed $V^{\pi}(s) = Q^{\pi}(s, \pi(s))$.

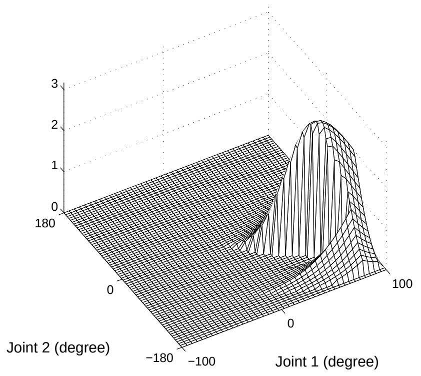


(a) A schematic

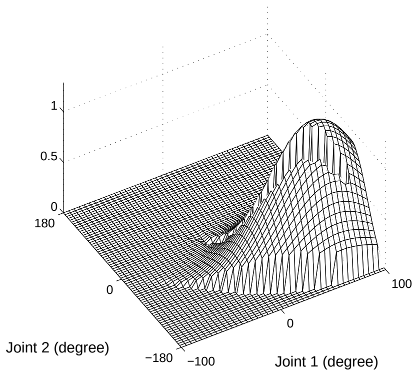


(b) State space

Fig. 5. A two-joint robot arm.



(a) Geodesic Gaussian kernels



(b) Ordinary Gaussian kernels

Fig. 6. Approximated value functions.

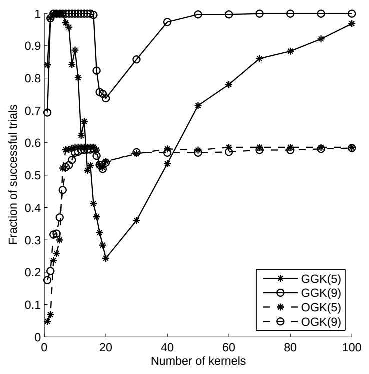


Fig. 7. Number of successful trials.

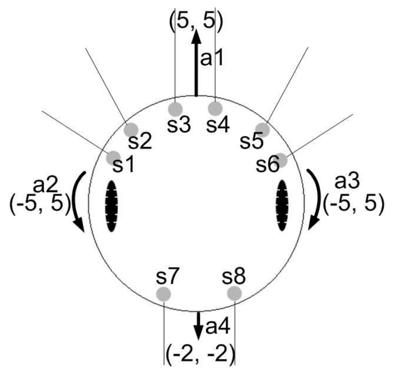
sensors (‘s1’ to ‘s8’ in the figure) which measure the strength of the reflected light returned from surrounding obstacles. Each sensor produces a scalar value between 0 and 1023 (which may be regarded as continuous): the sensor obtains the maximum value 1023 if an obstacle is just in front of the sensor and the value decreases as the obstacle gets farther till it reaches the minimum value 0. Therefore, the state space $\mathcal{S}$ is 8-dimensional and continuous. The Khepera has two wheels and takes the following 4 defined actions: forward, left-rotation, right-rotation and backward (i.e., the action space $\mathcal{A}$ contains 4 actions). The speed of the left and right wheels for each action is described in Fig.8(a) in the bracket (the unit is pulse per 10 milliseconds). Note that the sensor values and the wheel speed are highly stochastic due to the change of the ambient light, noise, the skid etc. Furthermore, perceptual aliasing occurs due to the limited range and resolution of sensors. Therefore, the state transition is highly stochastic. We set the discount factor to $\gamma = 0.9$.

The goal of the navigation task is to make the Khepera explore the environment as much as possible. To this end, we give a positive reward $+1$ when the Khepera moves forward and a negative reward -2 when the Khepera collides with an obstacle. We do not give any reward to the left/right rotation and backward actions. This reward design encourages the Khepera to go forward without hitting obstacles, through which extensive exploration in the environment could be achieved.

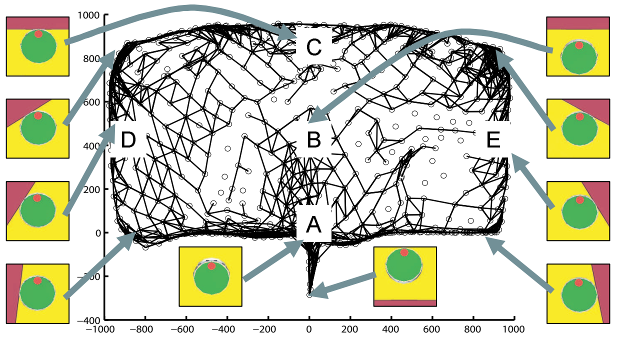
We collected training samples from 200 series of 100 random movements in a fixed environment with several obstacles (see Fig.9(a)). Then we constructed a graph from the gathered samples by discretizing the continuous state space using the *Self-Organizing Map* (SOM) [16]. The number of nodes (states) in the graph is set to 696 (equivalent with the SOM map size of 24×29); this value is computed by the standard rule-of-thumb formula $5\sqrt{n}$ [17], where n is the number of samples. The connectivity of the graph is determined by the state transition probability computed from the samples, i.e., if there is a state transition from one node to another in the samples, an edge is established between these two nodes and the edge weight is set according to the Euclidean distance between them. Fig.8(b) illustrates an example of the obtained graph structure—for visualization purposes, we projected the 8-dimensional state space onto a 2-dimensional subspace spanned by

$$\begin{array}{cccccccc}(-1 & -1 & 0 & 0 & 1 & 1 & 0 & 0), \\ (0 & 0 & 1 & 1 & 0 & 0 & -1 & -1).\end{array} \quad (9)$$

The i-th element in the above bases corresponds to the output of the i-th sensor (see Fig.8(a)). Therefore, the projection onto this subspace roughly means that the horizontal axis corresponds to the distance to the left/right obstacle, while the vertical axis corresponds to the distance to the front/back obstacle. For clear visibility, we only displayed the edges whose weight is less than 250. This graph has a notable feature: the nodes around the region ‘B’ in the figure are

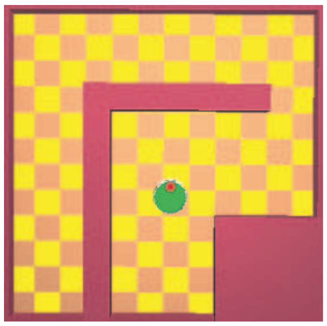


(a) A schematic

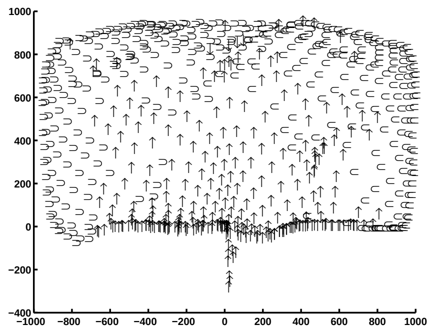


(b) State space projected onto a 2-dimensional subspace for visualization.

Fig. 8. Khepera robot.



(a) Training



(a) Geodesic Gaussian kernels

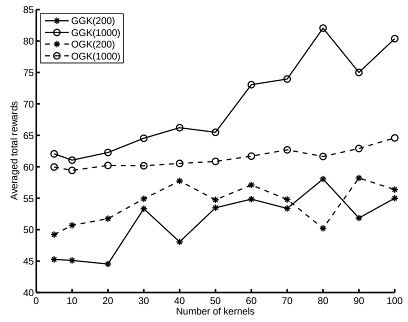
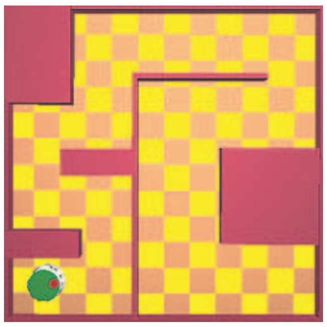
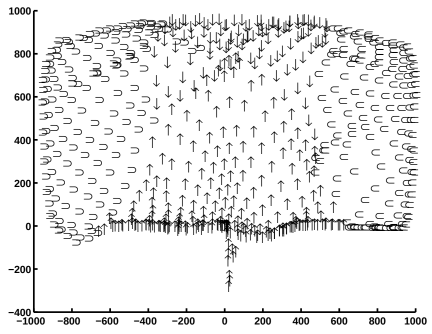


Fig. 11. Average amount of exploration.



(b) Test

Fig. 9. Simulation environment



(b) Ordinary Gaussian kernels

Fig. 10. Examples of obtained policy.

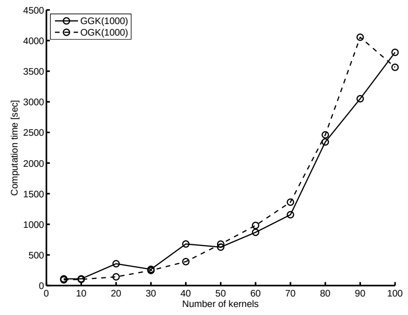


Fig. 12. Computation time.

directly connected to the nodes at 'A', but are not directly connected to the nodes at 'C', 'D', and 'E'. This implies that the geodesic distance from 'B' to 'C', 'D', or 'E' is large, although the Euclidean distance is small.

Since the transition from one state to another is highly stochastic in the current experiment, we decided to simply duplicate the GGK function over the action space (see Eq.(6)). For obtaining continuous GGKs, GGK functions need to be interpolated (see Sec.III-D). We may employ a simple linear interpolation method in general. However, the current experiment has unique characteristics—at least one of the sensor values is always zero since the Khepera is never completely surrounded by obstacles. Therefore, samples are always on the surface of the 8-dimensional hypercube-shaped state space. On the other hand, the node centers determined by the SOM are not generally on the surface. This means that any sample is not included in the convex hull of its nearest nodes and we need to *extrapolate* the function value. Here, we simply add the Euclidean distance between the sample and its nearest node when computing kernel values; more precisely, for a state s that is not generally located on a node center, the GGK-based basis function is defined as

$$\phi_{i+(j-1)m}(s,a) = I(a = a^{(i)}) \exp\left(-\frac{(\mathrm{ED}(s,\tilde{s}) + \mathrm{SP}(\tilde{s},c^{(j)}))^2}{2\sigma^2}\right), \quad (10)$$

where $\tilde{s}$ is the node closest to s in the Euclidean distance.

Fig.10 illustrates an example of actions selected at each node by the GGK-based and OGK-based policies. We used 100 kernels and set the width to 1000. The symbols '↑', '↓', '⊂', and '⊃' in the figure indicates forward, backward, left rotation, and right rotation actions. This shows that there is a clear difference in the obtained policies at the state ‘C’; the backward action is most likely to be taken by the OGK-based policy while the left/right rotation are most likely to be taken by the GGK-based policy. This causes a significant difference in the performance. To explain this, let us assume that the Khepera is at the state ‘C’, i.e., it faces the wall. The GGK-based policy guides the Khepera from ‘C’ to ‘A’ via ‘D’ or ‘E’ by taking left/right rotation actions and it can avoid the obstacle successfully. On the other hand, the OGK-based policy leads Khepera from ‘C’ to ‘A’ via ‘B’ by taking backward actions; then the forward action is taken at ‘B’. Thus, the Khepera returns to ‘C’ again and ends up moving back and forth between ‘C’ and ‘B’ (see also the attached video).

For the performance evaluation, we use a more complicated environment than the one used for gathering training samples (see Fig.9). Thus we are evaluating how well the obtained policies can be *generalized* to an unknown environment. In this test environment, we let the Khepera run from a fixed starting position (see Fig.9(b)) and take 150 steps following the obtained policy. We compute the sum of rewards, i.e., $+1$ for the forward action. If the Khepera collides with an obstacle before 150 steps, we stop the evaluation. The mean test performance over 20 independent runs is depicted in Fig.11 as a function of the number of kernels. More precisely, in each run, we construct a graph based on the training samples taken from the training environment and put the specified number of kernels randomly on the graph. Then, a policy is learned by the GGK or OGK-based LSPI method using the training samples. The test performance is measured 5 times for each policy and the average is output. Fig.11 shows that GGKs significantly outperform OGKs, demonstrating that GGKs are promising even in the challenging setting with a high-dimensional continuous state space.

Fig. 12 depicts the computation time of each method as a function of the number of kernels. This shows that the computation time monotonically increases as the number of kernels increases and the GGK-based and OGK-based methods have comparable computation time. This implies that the computation time of the GGK functions is negligible. Given that the GGK-based method works much better than the OGK-based method with a smaller number of kernels, the proposed method could be regarded as a computationally efficient alternative to the standard OGK-based method.

VI. Conclusions and Outlook

We proposed a new basis construction method for value function approximation. The proposed *geodesic Gaussian kernels* (GGKs) have several preferable properties such as the smoothness along the graph and easy computability. We demonstrated the practical usefulness of the proposed method for challenging applications: both the robot arm reaching with obstacles and the Khepera exploration experiments showed quantitative improvements as well as intuitive, interpretable behavioral advantages evident from the experiments.

Experiments in Sec.IV showed that GGKs with large width has larger MSEs than that with smaller width, but GGKs with large width gave better policies than that with smaller width. We conjecture that the GGKs with large width give smoother value functions and they result in stable policies. Although this explanation would be intuitively reasonable, it needs to be elucidated in a more rigorous way.

It is shown that the policies obtained by GGKs are not so sensitive to the choice of the width of the Gaussian kernels, i.e., a reasonable large width works very well. This is a very useful property in practice. Also, the heuristics of putting Gaussian centers on goal states is shown to work quite well. Even so, it is an important future direction to develop a method for optimally tuning the width as well as the location parameters, e.g., based on the statistical machine learning theory [9].

We defined the Gaussian kernels on the state space, and then extended them over the action space. If we define basis functions directly on the state-action space, the quality of value function approximation and the computational efficiency could be further improved. Our future research will focus on this topic.

References

[1] R. S. Sutton and G. A. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.
[2] M. G. Lagoudakis and R. Parr, “Least-squares policy iteration,” *Journal of Machine Learning Research*, vol. 4, no. Dec, pp. 1107–1149, 2003.
[3] F. Girosi, M. Jones, and T. Poggio, “Regularization theory and neural networks architectures,” *Neural Computation*, vol. 7, no. 2, pp. 219–269, 1995.
[4] I. Daubechies, *Ten Lectures on Wavelets*. Philadelphia and Pennsylvania: Society for Industrial and Applied Mathematics, 1992.
[5] S. Mahadevan, “Proto-value functions: Developmental reinforcement learning,” in *Proceedings of International Conference on Machine Learning*, Bonn, Germany, 2005.
[6] F. R. K. Chung, *Spectral Graph Theory*. Providence, R.I.: American Mathematical Society, 1997.
[7] R. Coifman and M. Maggioni, “Diffusion wavelets,” *Applied and Computational Harmonic Analysis*, vol. 21, no. 1, pp. 53–94, 2006.
[8] S. Mahadevan and M. Maggioni, “Value function approximation with diffusion wavelets and Laplacian eigenfunctions,” in *Advances in Neural Information Processing Systems 18*, Y. Weiss, B. Schölkopf, and J. Platt, Eds. Cambridge, MA: MIT Press, 2006, pp. 843–850.
[9] V. N. Vapnik, *Statistical Learning Theory*. New York: Wiley, 1998.
[10] B. Schölkopf and A. J. Smola, *Learning with Kernels*. Cambridge, MA: MIT Press, 2002.
[11] Y. Engel, S. Mannor, and R. Meir, “Reinforcement learning with Gaussian processes,” in *Proceedings of International Conference on Machine Learning*, Bonn, Germany, 2005.
[12] E. W. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.
[13] M. L. Fredman and R. E. Tarjan, “Fibonacci heaps and their uses in improved network optimization algorithms,” *Journal of the ACM*, vol. 34, no. 3, pp. 569–615, 1987.
[14] M. Sugiyama, H. Hachiya, C. Towell, and S. Vijayakumar, “Geodesic Gaussian kernels for value function approximation,” in *Proceedings of 2006 Workshop on Information-Based Induction Sciences*, Osaka, Japan, Oct. 31–Nov. 2 2006, pp. 316–321.
[15] A. V. Goldberg and C. Harrelson, “Computing the shortest path: A* search meets graph theory,” in *16th Annual ACM-SIAM Symposium on Discrete Algorithms*, Vancouver, Canada, 2005, pp. 156–165.
[16] T. Kohonen, *Self-Organizing Maps*. Berlin: Springer, 1995.
[17] J. Vesanto, J. Himberg, E. Alhoniemi, and J. Parhankangas, “SOM toolbox for Matlab 5,” Helsinki University of Technology, Tech. Rep. A57, 2000. [Online]. Available: http://www.cis.hut.fi/projects/somtoolbox/package/papers/techrep.pdf