Functional Gradient Motion Planning in Reproducing Kernel Hilbert Spaces

Zita Marinho*¶, Byron Boots†, Anca Dragan‡, Arunkumar Byravan§, Geoffrey J. Gordon*, Siddhartha Srinivasa*
zmarinho@cmu.edu, bboots@cc.gatech.edu, adragan@berkeley.edu, barun@uw.edu, ggordon@cs.cmu.edu, siddh@cs.cmu.edu

* Robotics Institute, Carnegie Mellon University, USA
† Interactive Computing, Georgia Institute of Technology, USA
§ CSE, University of Washington, USA
¶ Instituto Superior Tecnico, Portugal
‡ EECS, University of California, Berkeley, USA

***Abstract*—We introduce a functional gradient descent trajectory optimization algorithm for robot motion planning in Reproducing Kernel Hilbert Spaces (RKHSs). Functional gradient algorithms are a popular choice for motion planning in complex many-degree-of-freedom robots, since they (in theory) work by directly optimizing within a space of continuous trajectories to avoid obstacles while maintaining geometric properties such as smoothness. However, in practice, implementations such as CHOMP and TrajOpt typically *commit to a fixed, finite parametrization* of trajectories, often as a sequence of waypoints. Such a parameterization can lose much of the benefit of reasoning in a continuous trajectory space: e.g., it can require taking an inconveniently small step size and large number of iterations to maintain smoothness. Our work generalizes functional gradient trajectory optimization by formulating it as minimization of a cost functional in an RKHS. This generalization lets us represent trajectories as linear combinations of kernel functions. As a result, we are able to take larger steps and achieve a locally optimal trajectory in just a few iterations. Depending on the selection of kernel, we can directly optimize in spaces of trajectories that are *inherently smooth* in velocity, jerk, curvature, etc., and that have a *low-dimensional*, adaptively chosen parameterization. Our experiments illustrate the effectiveness of the planner for different kernels, including Gaussian RBFs with independent and coupled interactions among robot joints, Laplacian RBFs, and B-splines, as compared to the standard discretized waypoint representation.**

I. Introduction & Related Work

Motion planning is an important component of robotics: it ensures that robots are able to safely move from a start to a goal configuration without colliding with obstacles. *Trajectory optimizers* for motion planning focus on finding feasible configuration-space trajectories that are also efficient—e.g., approximately locally optimal for some cost function. Many trajectory optimizers have demonstrated great success in a number of high-dimensional real-world problems [13, 19–21]. Often, they work by defining a cost functional over an infinite-dimensional Hilbert space of trajectories, then taking steps down the functional gradient of cost to search for smooth, collision-free trajectories [14, 26].

In this work we exploit the same functional gradient approach, but with a novel approach to trajectory representation. While previous algorithms are derived for trajectories in Hilbert spaces in theory, *in practice* they commit to a finite parametrization of trajectories in order to instantiate a gradient update [6, 11, 26]—typically a large but finite list of discretized waypoints. The number of waypoints is a parameter that trades off between computational complexity and trajectory expressiveness. Our work frees the optimizer from a discrete parametrization, enabling it to perform gradient descent on a much more general trajectory parametrization: reproducing-kernel Hilbert spaces (RKHSs) [2, 7, 18], of which waypoint parametrizations are merely one instance. RKHSs impose just enough structure on generic Hilbert spaces to enable a concrete and implementable gradient update rule, while leaving the choice of parametrization flexible: different kernels lead to different geometries (Section II).

Our contribution is two-fold. Our *theoretical* contribution is the formulation of functional gradient descent motion planning in RKHSs, as the minimization of a cost functional regularized by the RKHS norm (Section III). Regularizing by the RKHS norm is a common way to ensure smoothness in function approximation [5], and we apply the same idea to trajectory parametrization. By choosing the RKHS appropriately, the trajectory norm can quantify different forms of smoothness or efficiency, such as preferring small values for any n-th order derivative [23]. So, RKHS norm regularization can be tuned to prefer trajectories that are smooth with, for example, low velocity, acceleration, or jerk (Section IV) [17].

Our *practical* contribution is an algorithm for very efficient motion planning in inherently smooth trajectory space with low-dimensional parametrizations. Unlike discretized parametrizations, which require many waypoints to produce smooth trajectories, our algorithm can represent and search for smooth trajectories with only a few point evaluations. The inherent smoothness of our trajectory space also increases efficiency; our parametrization allows the optimizer to take large steps at every iteration without violating trajectory smoothness, therefore converging to a collision-free and high-quality trajectory faster than competing approaches. Our experiments demonstrate the effectiveness of planning in RKHSs using synthetic 2D environment, with a 3-DOF planar arm, and using more complex scenarios, with a 7-DOF robotic arm. We show how different choices of kernels yield different preferences over trajectories. We further introduce reproducing kernels that represent interactions among joints. Sections V and VI

illustrate these advantages of RKHSs, and compare different choices of kernels.

II. Trajectories in an RKHS

A trajectory is a function $\boldsymbol{\xi} \in \mathcal{H} : [0,1] \to \mathcal{C}$ mapping time $t \in [0,1]$ to robot configurations $\boldsymbol{\xi}(t) \in \mathcal{C} \equiv \mathbb{R}^D$.[1] We can treat a set of trajectories as a Hilbert space by defining vector-space operations such as addition and scalar multiplication of trajectories [8]. In this paper, we restrict to trajectories in *Reproducing Kernel Hilbert Spaces*. We can upgrade our Hilbert space to an RKHS $\mathcal{H}$ by assuming additional structure: for any $\boldsymbol{y} \in \mathcal{C}$ and $t \in [0,1]$, the functional $\boldsymbol{\xi} \mapsto \boldsymbol{y}^\top \boldsymbol{\xi}(t)$ must be continuous [16, 18, 22]. Note that, since configuration space is typically multidimensional ($D > 1$), our trajectories form an RKHS of *vector-valued* functions [9], defined by the above property. The *reproducing kernel* associated with a vector valued RKHS becomes a matrix valued kernel $K : [0,1] \times [0,1] \to \mathcal{C} \times \mathcal{C}$. Eq. 1 represents the kernel matrix for two different time instances:

$$K(t,t') = \begin{bmatrix} k_{1,1}(t,t') & k_{1,2}(t,t') & \dots & k_{1,D}(t,t') \\ k_{2,1}(t,t') & k_{2,2}(t,t') & \dots & k_{2,D}(t,t') \\ \vdots & & \ddots & \vdots \\ k_{D,1}(t,t') & k_{D,2}(t,t') & \dots & k_{D,D}(t,t') \end{bmatrix} \quad (1)$$

This matrix has a very intuitive physical interpretation. Each element in (1), $k_{d,d'}(t,t')$ tells us how joint $[\xi(t)]_d$ at time t affects the motion of joint $[\xi(t')]_{d'}$ at t', *i.e.* its degree of correlation or similarity between the two (joint,time) pairs. In practice, off-diagonal terms of (1) will not be zero, hence perturbations of a given joint d propagate through time, as well as through the rest of the joints. The norm and inner product defined in a coupled RKHS can be written in terms of the kernel matrix, via the reproducing property: trajectory evaluation can be represented as an inner product of the vector valued functions in the RKHS, as described in (2) below. For any configuration $\boldsymbol{y} \in \mathcal{C}$, and time $t \in [0,1]$, we get the inner product of $\boldsymbol{y}$ with the trajectory in the vector-valued RKHS evaluated at time t:

$$\boldsymbol{y}^\top \boldsymbol{\xi}(t) = \langle \boldsymbol{\xi}, K(t,\cdot)\boldsymbol{y} \rangle_{\mathcal{H}}, \quad \forall \boldsymbol{y} \in \mathcal{C} \quad (2)$$

In our planning algorithm we will represent a trajectory in the RKHS in terms of some finite support $\{t_i\}_{i=1}^N \in \mathcal{T}$. This set grows adaptively as we pick more points to represent the final trajectory. At each step our trajectory will be a linear combination of functions in the RKHS, each indexed by a time-point in $\mathcal{T}$.

$$\boldsymbol{y}^\top \boldsymbol{\xi}(t) = \sum_{t_i \in \mathcal{T}} \boldsymbol{y}^\top K(t,t_i)\boldsymbol{a_i} \quad (3)$$

for $t, t_i \in [0,1]$, and $\boldsymbol{a_i} \in \mathcal{C}$. If we consider the configuration vector $\boldsymbol{y} \equiv \boldsymbol{e}_d$ to be the indicator of joint d, then we can capture its evolution over time as: $[\xi(t)]_d = \sum_i \boldsymbol{e_d}^\top K(t,t_i)\boldsymbol{a_i}$, taking into account the effect of all other joints. The inner product in $\mathcal{H}$ of functions $\boldsymbol{y}^\top \boldsymbol{\xi}^1(t) = \sum_i \boldsymbol{y}^\top K(t,t_i)\boldsymbol{a}_i$ and $\boldsymbol{y}^\top \boldsymbol{\xi}^2(t) = \sum_j \boldsymbol{y}^\top K(t,t_j)\boldsymbol{b}_j$, for $\boldsymbol{y}, \boldsymbol{a}_i, \boldsymbol{b}_j \in \mathcal{C}$ is defined as:

$$\langle \boldsymbol{\xi}^1, \boldsymbol{\xi}^2 \rangle_{\mathcal{H}} = \sum_{i,j} \boldsymbol{a}_i^\top K(t_i,t_j)\boldsymbol{b}_j \quad (4)$$

$$\|\boldsymbol{\xi}\|_{\mathcal{H}}^2 = \langle \boldsymbol{\xi}, \boldsymbol{\xi} \rangle = \sum_{i,j} \boldsymbol{a}_i^\top K(t_i,t_j)\boldsymbol{a}_j \quad (5)$$

For example, in the Gaussian RBF RKHS (with kernel $k_{d,d'}(t,t') = \exp(\|t-t'\|^2/2\sigma^2)$, when $d' = d$ and 0 otherwise), a trajectory is a weighted sum of radial basis functions:

$$\boldsymbol{\xi}(t) = \sum_{d,i} a_{i,d} \exp\left(\frac{\|t-t_i\|^2}{2\sigma^2}\right) \boldsymbol{e}_d, \quad a_{i,d} \in \mathbb{R} \quad (6)$$

The coefficients $a_{i,d}$ assess how important a particular joint d at time t_i is to the overall trajectory. They can be interpreted as weights of local perturbations to the motions of different joints centered at different times. Interactions among joints, as described in (1), can be represented for instance using a kernel matrix of the form :

$$K(t,t') = \exp\left(\frac{\|t-t'\|^2}{2\sigma^2}\right) \boldsymbol{J}^\top \boldsymbol{J} \ \in \mathbb{R}^{D \times D} \quad (7)$$

where $\boldsymbol{J} \in \mathbb{R}^{3 \times D}$ represents the workspace Jacobian matrix at a fixed configuration. This strategy changes the RKHS metric in configuration space according to the robot Jacobian in workspace. This norm can be interpreted as an approximation of the velocity of the robot in workspace [15]. The trajectory norm measures the size of the perturbations, and the correlation among them, quantifying how complex the trajectory is in the RKHS, see Section IV. Different norms can be considered for representing the RKHS; this can leverage more problem-specific information, which could reduce the number of iterations required to find a low cost trajectory.

III. Motion Planning in an RKHS

In this section we describe how trajectory optimization can be achieved by functional gradient descent in an RKHS of trajectories.

1) Cost Functional: We introduce a cost functional $\mathcal{U} : \mathcal{H} \to \mathbb{R}$ that maps each trajectory to a scalar cost. This functional quantifies the quality of a given a trajectory (function in the RKHS). $\mathcal{U}$ trades off between a regularization term that measures the efficiency of the trajectory, and an obstacle term that measures its proximity to obstacles:

$$\mathcal{U}[\boldsymbol{\xi}] = \mathcal{U}_{obs}[\boldsymbol{\xi}] + \frac{\beta}{2}\|\boldsymbol{\xi}\|_{\mathcal{H}}^2 \quad (8)$$

As described in Section IV, we choose our RKHS so that the regularization term encodes our desired notion of smoothness or trajectory efficiency—*e.g.*, minimum length, velocity, acceleration, jerk. The obstacle cost functional is defined on trajectories in configuration space, but obstacles are defined in the robot's workspace $\mathcal{W} \equiv \mathbb{R}^3$. So, we connect configuration space to workspace via a *forward kinematics* map x: if $\mathcal{B}$ is the set of body points of the robot, then $x : \mathcal{C} \times \mathcal{B} \to \mathcal{W}$ tells

[1] Note that $\boldsymbol{\xi} \in \mathcal{H}$ is a trajectory, a vector valued function in the RKHS, while $\boldsymbol{\xi(t)} \in \mathcal{C}$ is a trajectory evaluation corresponding to a robot configuration.

us the workspace coordinates of a given body point when the robot is in a given configuration. We can then decompose the obstacle cost functional as:

$$\mathcal{U}_{obs}[\boldsymbol{\xi}] \equiv \underset{t,u}{\text{reduce}}\, c\left(x(\boldsymbol{\xi}(t), u)\right) \tag{9}$$

where reduce is an operator that aggregates costs over the entire trajectory and robot body—e.g., a maximum or an integral, see Section V. We assume that the reduce operator takes (at least approximately) the form of a sum over some finite set of (time, body point) pairs $\boldsymbol{\mathcal{T}}(\boldsymbol{\xi})$:

$$\mathcal{U}_{obs}[\boldsymbol{\xi}] \approx \sum_{(t,u)\in\boldsymbol{\mathcal{T}}(\boldsymbol{\xi})} c\left(x(\boldsymbol{\xi}(t), u)\right) \tag{10}$$

For example, the maximum operator takes this form: if (t, u) achieves the maximum, then $\boldsymbol{\mathcal{T}}(\boldsymbol{\xi})$ is the singleton set $\{(t, u)\}$. Integral operators do not take this form, but they can be well approximated in this form using quadrature rules, see Section V-2.

2) Optimization: We can derive the functional gradient update by minimizing a local linear approximation of $\mathcal{U}$ in (8):

$$\boldsymbol{\xi}^{n+1} = \arg\min_{\boldsymbol{\xi}} \ \langle \boldsymbol{\xi} - \boldsymbol{\xi}^n, \nabla\mathcal{U}[\boldsymbol{\xi}^n]\rangle_{\mathcal{H}} + \frac{\lambda}{2}\|\boldsymbol{\xi} - \boldsymbol{\xi}^n\|^2_{\mathcal{H}} \tag{11}$$

The quadratic term is based on the RKHS norm, meaning that we prefer "smooth" updates, analogous to Zucker et al. [26].

This minimization admits a solution in closed form:

$$\boldsymbol{\xi}^{n+1}(\cdot) = \left(1 - \frac{\beta}{\lambda}\right)\boldsymbol{\xi}^n(\cdot) - \frac{1}{\lambda}\nabla\mathcal{U}_{obs}[\boldsymbol{\xi}^n](\cdot) \tag{12}$$

Since we have assumed that the cost functional $\mathcal{U}_{obs}[\boldsymbol{\xi}]$ depends only on a finite set of points $\boldsymbol{\mathcal{T}}(\boldsymbol{\xi})$ (10), it is straightforward to show that the functional gradient update has a finite representation (so that the overall trajectory, which is a sum of such updates, also has a finite representation). In particular, assume the workspace cost field c and the forward kinematics function x are differentiable; then we can obtain the cost functional gradient by the chain rule [16, 18]:

$$\nabla\mathcal{U}_{obs}(\cdot) = \sum_{(t,u)\in\boldsymbol{\mathcal{T}}} K(\cdot, t)\ \mathbf{J}^\top(t, u)\nabla c(x(\boldsymbol{\xi}(t), u)) \tag{13}$$

where $\mathbf{J}(t, u) = \frac{\partial}{\partial\boldsymbol{\xi}(t)}x(\boldsymbol{\xi}(t), u) \in \mathbb{R}^{3\times D}$ is the workspace Jacobian matrix at time t for body point u, and the kernel function $K(\cdot, t)$ is the gradient of $\boldsymbol{\xi}(t)$ with respect to $\boldsymbol{\xi}$. The kernel matrix is defined in Equation (1).

This solution is a generic form of functional gradient optimization with a *directly instantiable* obstacle gradient that does not depend on a predetermined set of waypoints, offering a more expressive representation with fewer parameters. We derive a constrained optimization update rule, by solving the KKT conditions for a vector of Lagrange multipliers, see Section III-3. The full method is summarized as Algorithm 1.

3) Constrained optimization: Consider equality constraints (fixed start and goal configurations) and inequality constraints (joint limits) on the trajectory $h(\boldsymbol{\xi}(t)) = 0,\ g(\boldsymbol{\xi}(t)) \leq 0$, respectively. We write them as inner products with kernel functions in the RKHS (14). For any $\boldsymbol{y} \in \mathcal{C}$:

$$h(\cdot)^\top\boldsymbol{y} \leftarrow \langle\boldsymbol{\xi}, K(t_o, \cdot)\boldsymbol{y}\rangle_{\mathcal{H}} - \boldsymbol{q_o}^\top\boldsymbol{y} = 0, \tag{14}$$
$$\boldsymbol{q_o} \in \mathcal{C},\ \text{for } t_o = \{0, 1\}$$
$$g(\cdot)^\top\boldsymbol{y} \leftarrow \langle\boldsymbol{\xi}, K(t_p, \cdot)\boldsymbol{y}\rangle_{\mathcal{H}} - \boldsymbol{q_p}^\top\boldsymbol{y} \leq 0, \tag{15}$$
$$\boldsymbol{q_p} \in \mathcal{C},\ \text{for } t_p = [0, 1]$$

Let, $\boldsymbol{\gamma}^o,\ \boldsymbol{\mu}^p \in \mathbb{R}^D$ be the concatenation of all equality (γ^o) and inequality (μ^p) Lagrange multipliers. We rewrite the objective function in (11) including joint constraints:

$$\boldsymbol{\xi}^{n+1}(\cdot) = \arg\min_{\boldsymbol{\xi}} \ \langle\boldsymbol{\xi} - \boldsymbol{\xi}^n, \nabla\mathcal{U}[\boldsymbol{\xi}^n]\rangle_{\mathcal{H}} + \frac{\lambda}{2}\|\boldsymbol{\xi} - \boldsymbol{\xi}^n\|^2_{\mathcal{H}} \tag{16}$$
$$+ \boldsymbol{\gamma^o}^\top h[\boldsymbol{\xi}] + \boldsymbol{\mu^p}^\top g[\boldsymbol{\xi}]$$

Solving the KKT system for the stationary point of (16) $(\boldsymbol{\xi}, \gamma^o, \mu^p)$ with $\mu^p \geq 0$, we obtain the constrained solution (17). Let $\mathrm{d}c_j \equiv \boldsymbol{J}^\top(t_j, u_j)\nabla c\left(x(\boldsymbol{\xi}^n(t_j), u_j)\right) \in \mathbb{R}^D$. The full update rule becomes:

$$\boldsymbol{\xi}^*(\cdot) = \left(1 - \frac{\beta}{\lambda}\right)\boldsymbol{\xi}^n(\cdot) - \frac{1}{\lambda}\left(\sum_{t_j\in\boldsymbol{\mathcal{T}}} K(\cdot, t_j)\mathrm{d}c_j\right. \tag{17}$$
$$\left. + K(\cdot, t_o)\boldsymbol{\gamma}o + K(\cdot, t_p)\boldsymbol{\mu}^p\right)$$

This constrained solution, ends up augmenting the finite support set $(\mathcal{T})$ with points that are in constraint violation, weighted by the respective Lagrange multipliers. Each of the multipliers can be interpreted as a quantification of how much the points t_o or t_p affect the overall trajectory over time and joint space.

IV. Trajectory Efficiency as Norm Encoding in RKHS

In different applications it is useful to consider different notions of trajectory efficiency or smoothness. We can do so by choosing appropriate kernel functions that have the desired property, and consequently desired induced norm/metric. For instance, we can choose a kernel according to the topology of the obstacle field, or we can learn a kernel from demonstrations or user input, bringing problem specific information into the planning problem. This can help improve efficiency of the planner. Another possibility is to tune the resolution of the kernel via its width. We could build a kernel with adaptive width according to the environment, *i.e.*, higher sensitivity (lower width) in cluttered scenarios.

Additionally, it is often desirable to penalize the velocity, acceleration, jerk, or other derivatives of a trajectory instead of (or in addition to) its magnitude. To do so, we can take advantage of a *derivative reproducing property*: let $\mathcal{H}_1$ be one of the coordinate RKHSs from our trajectory representation, with kernel k. If k has sufficiently many continuous derivatives, then for each partial derivative operator D^α, there exist representers $(D^\alpha k)_t \in \mathcal{H}_1$ such that, for all $f \in \mathcal{H}_1$,

Algorithm 1 — Trajectory optimization in RKHSs $\left(N, c, \nabla c, \boldsymbol{\xi}^{(n)}(0), \boldsymbol{\xi}^{(n)}(1)\right)$

for each joint angle $d \in D$ **do**
  Initialize to a straight line trajectory $\xi_d^0(t) = \xi_d(0) + (\xi_d(1) - \xi_d(0))t$.
end for
while ($\mathcal{U}[\boldsymbol{\xi}^n] > \epsilon$ and $n < N_{\text{MAX}}$) **do**
  Compute $\mathcal{U}_{obs}[\boldsymbol{\xi}^n]$ (13).
  Find the support of time/body points $\boldsymbol{\mathcal{T}}(\boldsymbol{\xi}) = \{t_i, u_i\}, i = 1, \ldots, N$ (10).
  for $(t_i, u_i)_{i=1}^N \in \boldsymbol{\mathcal{T}}(\boldsymbol{\xi})$ **do**
    Evaluate the cost gradient $\nabla c(\boldsymbol{\xi}(t_i), u_i)$ and Jacobian $\mathbf{J}(t_i, u_i)$
  end for
  Update trajectory: $\boldsymbol{\xi}^{n+1} = (1 - \frac{\beta}{\lambda})\boldsymbol{\xi}^n - \frac{1}{\lambda} \sum_{(t,u)\in\boldsymbol{\mathcal{T}}} K(\cdot, t)\ \mathbf{J}^\top(t, u)\nabla c(x(\boldsymbol{\xi}(t), u))$
  If constraints are present, project onto constraint set (16).
end while
Return: Final trajectory $\boldsymbol{\xi}^*$ and costs $\|\boldsymbol{\xi}^*\|^2_{\mathcal{H}}, \mathcal{U}_{obs}[\xi^*]$.

$(D^\alpha f)(t) = \langle (D^\alpha k)_t, f \rangle$ [25, Theorem 1]. (Here α is a multi-set of indices, indicating which partial derivative we are referring to.) We can, therefore, define a new RKHS with a norm that penalizes the partial derivative D^α: the kernel is $k^\alpha(t, t') = \langle (D^\alpha k)_t, (D^\alpha k)_{t'} \rangle$. If we use this RKHS norm as the smoothness penalty for the corresponding coordinate of our trajectories, then our optimizer will automatically seek out trajectories with low velocity, acceleration, or jerk in this coordinate.

For example, consider an RBF kernel with a reproducing first order derivative: $D^1 k(t, t_i) = D^1 k_{t_i}[t] = \frac{(t-t_i)}{2\sigma^2} k(t, t_i)$ is the reproducing kernel for the velocity profile of a trajectory defined in an RBF kernel space $k(t, t_i) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\|t - t_i\|^2/2\sigma^2)$. The velocity profile of a trajectory $\boldsymbol{\xi}(t) = \sum_i \beta_i k(t, t_i)$ can be written as $D^1\boldsymbol{\xi}(t) = \sum_i \beta_i D^1 k(t, t_i)$. The trajectory can be found by integrating $D^1\boldsymbol{\xi}(t)$ once and using the constraint $\boldsymbol{\xi}(0) = \boldsymbol{q}_i$.

$$\begin{aligned} \boldsymbol{\xi}(T) &= \int_0^T D^1\boldsymbol{\xi}(t)dt = \sum_i \beta_i \int_0^T \frac{(t - t_i)}{2\sigma^2} k(t, t_i)dt \\ &= \sum_i \beta_i \left[k(T, t_i) - k(0, t_i)\right] + \boldsymbol{q}_i \end{aligned} \tag{18}$$

The initial condition is verified automatically. The endpoint condition can be written as $\boldsymbol{q}_f = \sum_i \beta_i \left[k(1, t_i) - k(0, t_i)\right] + \boldsymbol{q}_i$; this imposes additional information over the coefficients $\beta_i \in \mathcal{C}$, which we can enforce during optimisation. Here we explicitly consider only a space of first derivatives, but extensions to higher order derivatives can be derived similarly integrating p times to obtain the trajectory profile. Constraints over higher derivatives (up to order n), can be enforced using any constraint projection method, inside our gradient descent iteration. The update rule in this setting can be derived using the natural gradient in the space, where the new obstacle gradient becomes:

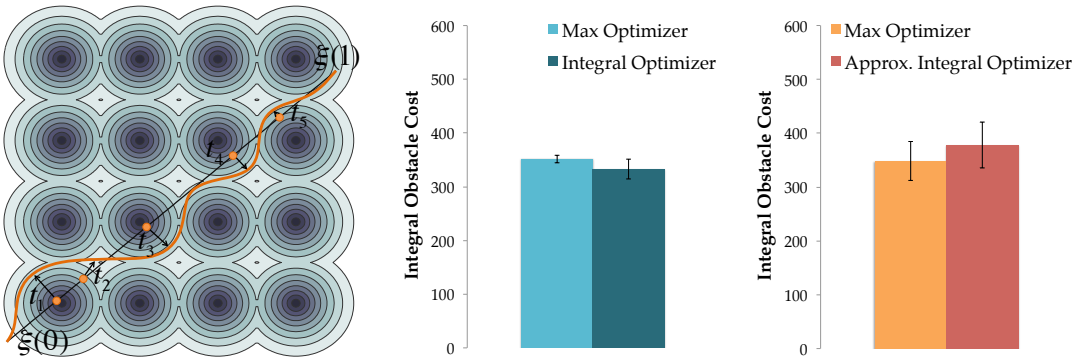
$$\nabla\mathcal{U}_{obs}(\cdot) = \sum_\alpha^n \sum_{(t,u)\in\mathcal{T}} D^\alpha K(\cdot, t)\ \mathbf{J}^\top(t, u)\nabla c(\ x(\boldsymbol{\xi}(t), u)\ ) \tag{19}$$

Although Gaussian RBFs are a default choice of kernel in many kernel methods, RKHSs can also easily represent other types of kernel functions. For example, B-splines are a popular parametrization of smooth functions [3, 10, 24], that are able to express smooth trajectories while avoiding obstacles, even though they are finite-dimensional kernels. Regularization schemes in different RKHSs may encode different forms of trajectory efficiency. Here we proposed a different norm using derivative penalization. The choice of kernel should be application driven, and any reproducing kernel can easily be considered under the optimization framework presented in this paper. Other RKHS norms may be defined as sums, products, tensor product of kernels, or any closed kernel operation.

V. Cost Functional Analysis

Next we analyze how the cost functional (different forms of the reduce operation in Section III-1) affects obstacle avoidance performance and the resulting trajectory (Section V). In this paper, we adopt a maximum cost version (Section V-1), and an approximate integral cost version of the obstacle cost functional (Section V-2). Other variants could be considered, providing the trajectory support remains finite, but we leave this as future work. Additionally, in Section V-3, we compare the two forms against a more commonly used cost functional, the path integral cost [14], and we show our formulations do not perform worse, while being faster to compute. Based on these experiments, in the remaining sections of the paper we consider only the max cost formulation, which we believe represents a good tradeoff between speed and performance.

1) Max Cost Formulation: The maximum obstacle cost penalizes body points that pass too close to obstacles, *i.e.* high cost regions in workspace (regions inside/near obstacles). This maximum cost version of the reduce operation, considered in Eq. (9), can be described as picking time and body points (sampling) deepest inside or closest to obstacles, see Figure 1(a). The sampling strategy for picking time points to represent the trajectory cost can be chosen arbitrarily, and further improved for time efficiency. In this paper, we consider



(a) reduce operation via max cost points (b) $\mathcal{U}_{obs}$, Integral vs Max cost (c) $\mathcal{U}_{obs}$, Approx integral vs Max cost

Fig. 1: a) At each iteration, the optimizer takes the current trajectory (black) and identifies the point of maximum obstacle cost t_i (orange points). It then updates the trajectory by a point evaluation function centered around t_i. Grey regions depict isocontours of the obstacle cost field (darker means closest to obstacles, higher cost).b) The integral costs after 5 large steps comparing using Gaussian RBG kernels vs. using the integral formulation (with waypoints). c) Gaussian RBF kernel integral cost using our max formulation vs. the approximate quadrature cost (20 points, 10 iterations).

a simple version, where we sample points uniformly along sections of the trajectory, and choose N maximum violating points, one per section. This max cost strategy allows us to represent trajectories in terms of a few points, rather then a set of finely discretized waypoints. This is a simplified version of the obstacle cost functional that yields a more compact representation [6, 11, 14].

2) Integral Cost Formulation: Instead of scoring a trajectory by the maximum of obstacle cost over time and body points, it is common to integrate cost over the entire trajectory and body, with the trajectory integral weighted by arc length to avoid velocity dependence [26]. While this path integral depends on all time and body points, we can approximate it to high accuracy from a finite number of point evaluations using numerical quadrature [12]. $\mathcal{T}(\boldsymbol{\xi})$ then becomes the set of abscissas of the quadrature method, which can be adaptively chosen on each time step (e.g., to bracket the top few local optima of obstacle cost), see Section A. In our experiments, we have observed good results with Gauss-Legendre quadrature.

3) Integral vs. Max Cost Formulation: We show that the max cost does not hinder the optimization— that it leads to practically equivalent results as an integral over time and body points [26]. To do so, we manipulate the cost functional formulation, and measure the resulting trajectories' cost in terms of the integral formulation. Figure 1(b) shows the comparison: the integral cost decreased by only 5% when optimizing for the max. Additionally we tested the max cost formulation against the approximate integral cost using a Gauss-Legendre quadrature method. We performed tests over 100 randomly sampled scenarios and measured the final obstacle cost after 10 iterations. We used 20 points to represent the trajectory in both cases. Figure 1(c) shows the approximate integral cost formulation is only 8% above the max approach.

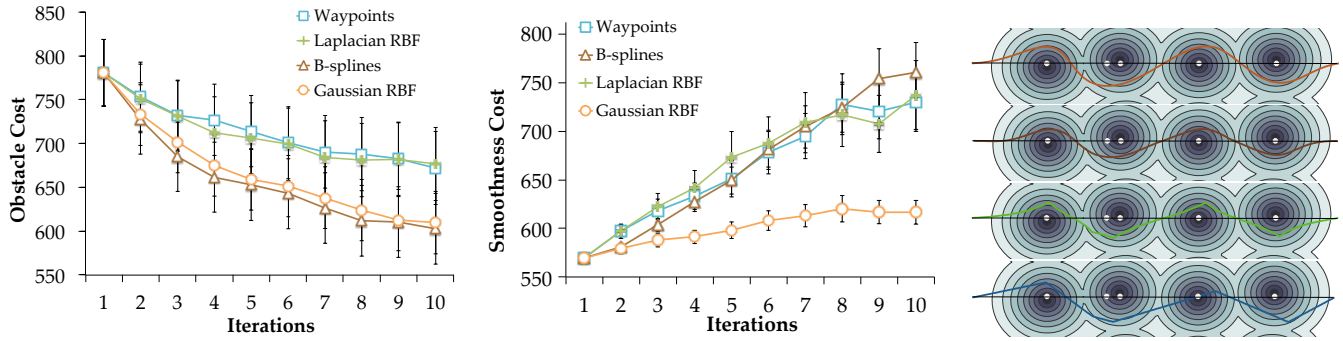
VI. Experimental Results

In what follows, we compare the performance of RKHS trajectory optimization vs. a discretized version (CHOMP) on a set of motion planning problems in a 2D world for a 3 DOF link planar arm as in Figure 3, and how different kernels with different norms affect the performance of the algorithm (Section VI-A). We then introduce a series of experiments that illustrate why RKHSs improve optimization (Section VI-B).

A. RKHS with various Kernels vs. Waypoints

For our main experiment, we systematically evaluate different parametrizations across a series of planning problems. We manipulate the parametrization (waypoints vs different kernels) as well as the number of iterations (which we use as a covariate in the analysis).We qualitatively optimize stepsize for all methods over 10 iterations. We also select a stepsize that best performs in 10 iterations and we keep this parameter fixed and constant between methods $\sigma = 0.9$. To control for the cost functional as a confound, we use the max formulation for both parameterizations. We use iterations as our covariate because they are a natural unit in optimization, and because the amount of time per iteration is similar: the computational bottleneck in all methods is computing the maximum penetration points. We measure the obstacle and smoothness cost of the resulting trajectories. For the smoothness cost, we use the norm in the waypoint parametrization as opposed to the norm in the RKHS as the common metric, to avoid favoring our new methods.

We use 100 different random obstacle placements and keep the start and goal configurations fixed. We compare the effectiveness of obstacle avoidance over 10 iterations, in 100 trials, of 12 randomly placed obstacles in a 2D environment, see Figure 3. The trajectory is represented with 4 maximum-violation points over time and robot body points at each iteration. The RKHS parametrization results in comparable obstacle cost and lower smoothness cost for the same number of iterations. We performed a t-test using the last iteration samples, and showed that the Gaussian RBF RKHS representation resulted in significantly lower obstacle cost ($t(99) = -2.63, p < .01$) and smoothness cost ($t(99) = -3.53, p < .001$). We expect this to be true because with the Gaussian RBF parametrization, the algorithm can take larger steps without breaking smoothness, see Section VI-B. We observe that waypoints and Laplacian RBF kernels (with large widths) have similar behaviour,



(a) Obstacle cost vs. iterations for different kernel choices

(b) Smoothness cost vs. iterations for different kernel choices

(c) 2D trajectory with large steps (1 iter.)

Fig. 2: (a,b): Cost over iterations for a 3DoF robot in 2D. Error bars show the standard error over 100 samples. (c) Trajectory profile using different kernels (5 time points in white) in order: Gaussian RBF, B-splines, Laplacian RBF kernels, and waypoints.

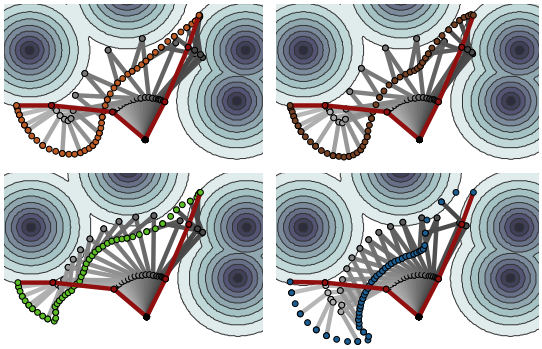


Fig. 3: Robot 3DoF. Trajectory after 10 iterations: top-left: Gaussian RBF kernel, top-right: B-splines kernel, bottom-left: Laplacian RBF kernel, bottom-right: waypoints.

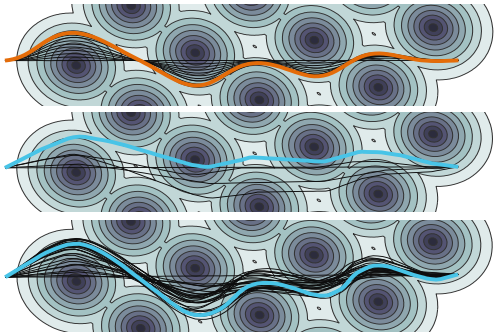


Fig. 4: 1DOF 2D trajectory in a maze environment (obstacle shaded in grey). top: Gaussian RBF, large steps (5 it.); middle: waypoints, large steps (5 it.); bottom: waypoints, small steps (25 it.)

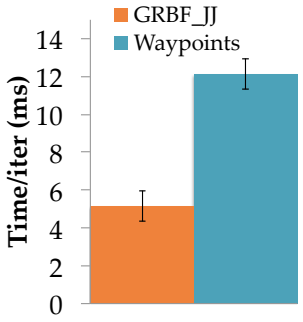
while Gaussian RBF and B-spline kernels provide a smooth parametrization that allows the algorithm to take larger steps at each iteration. These kernels provide the additional benefit of controlling the motion amplitude, which is an important consideration for an adaptive motion planner. We compare the effectiveness of obstacle avoidance over 10 iterations, in 100 trials, of 12 randomly placed obstacles in a 2D environment, see Figure 3.

B. RKHSs Allow Larger Steps than Waypoints

One practical advantage of using a RKHS instead of the waypoint parametrization is the ability to take large steps during the optimization. Figure 4 compares the two approaches, while taking large steps: it takes 5 Gaussian RBF iterations to solve the problem, but would take 28 iterations with smaller steps for the waypoint parametrization — otherwise, large steps cause oscillation and break smoothness. The resulting obstacle cost is always lower with Gaussian RBFs ($t(99) = 5.32, p < .0001$). The smoothness cost is lower ($t(99) = 8.86, p < .0001$), as we saw in the previous experiment as well. Qualitatively, however, as Figure 2(c) shows, the Gaussian RBF trajectories appear smoother: even after just one iteration, as they do not break differential continuity. We represented the discretized trajectory with 100 waypoints, but only used 5 kernel evaluation sets of points for the RKHS. We also tested the waypoint parametrization with only 5 waypoints, in order to evaluate an equivalent low dimensional representation, but this resulted in a much poorer trajectory with regard to smoothness.

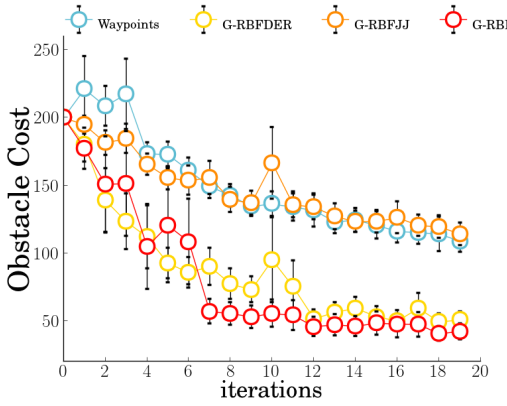
C. Experiments on a 7-DOF Manipulator

This section describes a comparison of the waypoint parametrization (CHOMP) and the RKHS Gaussian RBF (GRBF) on a 7-DOF simple manipulation task. We qualitatively optimized the obstacle cost weight λ and smoothness weight β for all methods after 25 iterations (including the Waypoint method). The kernel width is kept constant over all 7-DOF experiments (σ=0.9 same as previous experiments). 7 (a and b) illustrate both methods after 10 and 25 iterations, respectively. Figure 7(a) shows the end-effector traces after 10 iterations. The path for CHOMP (blue) is very non-smooth and collides with the cabinet, while the Gaussian RBF optimization is able to find a smoother path (orange) that is not in collision. Note that we only use a single max-point for the RKHS version, which leads to much less computation per iteration as compared to CHOMP. Figure 7(b) shows the results from both methods after 25 iterations of optimization. CHOMP is now able to find a collision-free path, but the path is still not very smooth as compared to the RKHS-optimized path. These results echo our findings from the robot simulation and planar

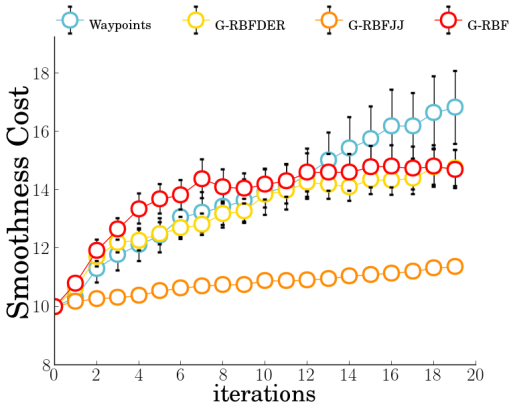


(a) Avg. time per iteration

Fig. 5: (a) Avg. time per iteration for GRBF RKHS with 1 max point (50 iter., $\lambda = 8, \beta = 1$) vs. waypoints (50 iter., $\lambda = 40$)

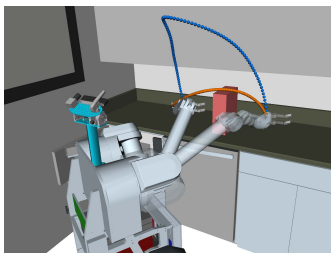


(a) Obstacle cost

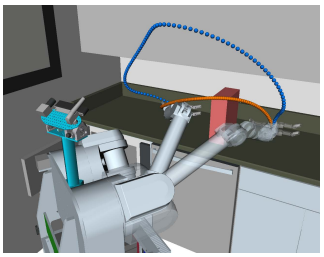


(b) Smoothness cost

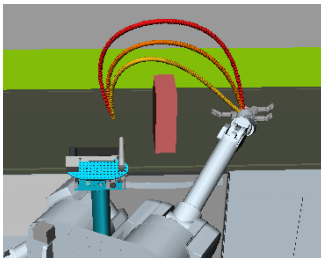
Fig. 6: (a) Obstacle and (b) smoothness costs for (GRBF-JJ) GRBF with joint interactions in orange, (GRBF-der) GRBF derivative with independent joints in yellow, (GRBF) GRBF independent joints in red, waypoints in blue, 20 iter.



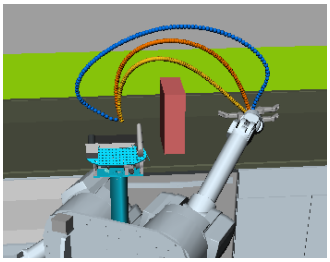
(a) Gaussian RBF (orange) vs. Waypoints (blue), 10 iter.



(b) Gaussian RBF (orange) vs. Waypoints (blue), 25 iter.



(c) coupled vs. indep. RKHSs, 50 iter.



(d) RKHS vs. waypoints, 50 iter.

Fig. 7: 7-dof experiment, plotting end-effector position from start to goal. (a) GRBF with 1 max point (10 iter., $\lambda = 20, \beta = 0.5$) vs. Waypoints (10 iterations, $\lambda = 200$). (b) GRBF with 1 max point (25 iterations, $\lambda = 20, \beta = 0.5$) vs. waypoints (25 iterations, $\lambda = 200$)(c) in orange GRBF with joint interactions (GRBF-JJ), in yellow GRBF derivative with joint interactions (GRBF-der) (1 max point,λ=8,β=1.0), in red GRBF with independent joints (GRBF) in red, (1 max point, λ=16,β=1.0). (d) in orange GRBF with joint interactions (GRBF-JJ), in yellow GRBF derivative with joint interactions (GRBF-der) (1 max point,λ=8,β=1.0), waypoints in blue, (λ=40).

arm experiments.

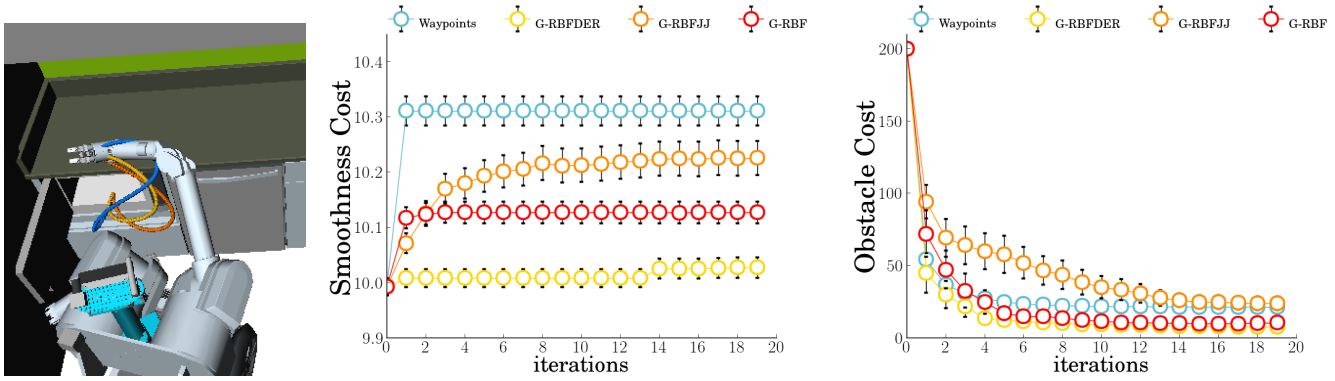
D. Optimization under Different RKHS Norms

In this section we experiment with different RKHS norms, where each one expresses a distinct notion of trajectory efficiency or smoothness. Here we have optimized obstacle cost weight (λ) after 5 iterations. Figure 7(c) shows three different Gaussian RBF kernel based trajectories. We show the end-effector traces after 50 iterations for a Gaussian RBF kernel with independent joints (1) (red) , and a Gaussian RBF derivative with independent joints (yellow). We also consider interactions among joints (orange), with a vector-valued RKHS with a kernel composed of a 1-dim. time-kernel, combined with a $D \times D$ kernel matrix that models interactions across joints. We build this matrix based on the robot Jacobian at the start configuration (see Equation (7)).

The Gaussian derivative kernel takes the form of a sum over a Gaussian RBF kernel and its first order derivative. This kernel can be interpreted as an infinitely differentiable function with increased emphasis on its first derivative (penalizes trajectory velocity more). We can observe that the derivative kernel achieves a smoother path, when compared with the ordinary Gaussian RBF RKHS. The Gaussian RBF kernel with joint interactions yields trajectories that are also smoother than the independent Gaussian RBF path. Figure 7(d) shows the end-effector traces of the GRBF derivative and the joint interaction kernel, together with a CHOMP trajectory with waypoints (blue). This result shows that Gaussian RBF RKHSs achieve a smoother, low-cost trajectory faster. Figure 5(a) shows a time comparison between the Gaussian RBF with join interactions and CHOMP (waypoints) method. This shows that the kernel method is less time consuming than CHOMP (t(49), p<.001) over 10 iterations.

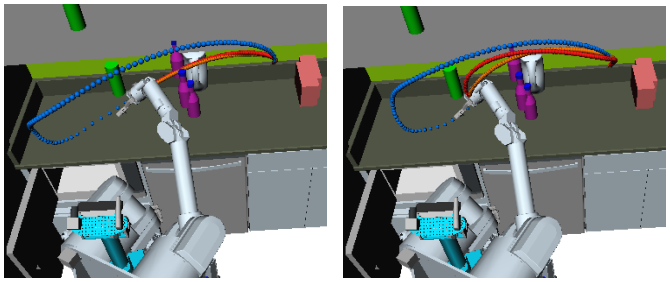
E. 7-DOF Experiments in a Cluttered Environment

Next, we test our method in more cluttered scenarios. We create random environments, populated with different shaped objects to increase planning complexity (see Figure 9(a)). We place 8 objects (2 boxes, 3 bottles, 1 kettle, 2 cylinders) in the area above the kitchen table randomly in x, y, z positions. We plan with random collision-free initial and final configurations. Figures 6(a) and 6(b) show obstacle and smoothness cost per iteration, respectively. We compare all results according to CHOMP cost functions. As above, smoothness is measured in terms of total velocity (waypoint metric), and obstacle cost is given by distance to obstacles of the full trajectory (not only max-points). The RKHS trajectory with joint interactions (orange) achieves a smoother trajectory than the



(a) 30 iterations (b) Obstacle cost (c) Smoothness cost

Fig. 8: (a)7-DOF robot experiment, plotting the end-effector position from start to goal. waypoints (λ=40) (blue), Gaussian RBF-der (1 max point,λ=45,β=1.0) (yellow), Gaussian RBF JJ (1 max point,λ=40,β=1.0) (orange) (b) Obstacle and (c) Smoothness cost in constrained environment after 20 iter.



(a) 6 iterations (b) 20 iterations

Fig. 9: 7-DOF robot experiment, plotting the end-effector position from start to goal. Waypoints (λ=40) (blue), GRBF (1 max point,λ=45,β=1.0) (orange), GRBF-der (1 max point,λ=25,β=1.0) (yellow)(a) after 6 iter. (left). (b) after 20 iter.

waypoint representation (blue). We perform a Wilcoxon t-test ($t(40) = -4.4, p < .001$), and obtains comparable obstacle costs. The method performs updates more conservatively and ensures very smooth trajectories. The other two RKHS variants consider independent joints, GRBF (red) and GRBF derivative (yellow). Both GRBF and GRBF derivative kernels achieve lower cost trajectories than the waypoint parametrization ($t(40) = -3.7, p < .001$, $t(40) = -4.04, p < .001$), and converge in fewer iterations (approx. 12it.). The smoothness costs are not statistically significantly different that the waypoint representation, after 20 iterations. We show an example scenario with three variants of RKHSs (GRBF-JJ orange ($\lambda = 75$), GRB yellow ($\lambda = 45$), Waypoints blue ($\lambda = 100$)) after 6 iterations (Figure 9(a)), and after 20 iterations (Figure 9(b)). The smoothness and obstacle weights are kept fixed in all scenarios (β=1.0).

F. 7-DOF Experiments in a Constrained Environment

Next, we measured performance in a more constrained task. We placed the robot closer to the kitchen counter, and planned to a fixed goal configuration inside the microwave. We ran over 40 different random initial configurations. Figure 8(a) shows an example of Gaussian RBF with joint interactions (GRBF-JJ orange), independent joints (GRBF-der yellow), and waypoints (blue), after 30 iterations. We report smoothness and obstacle costs per iteration, see Figures 8(c) and 8(b) respectively. In more constrained scenarios, the RKHS variants achieve smoother costs than the waypoint representation ($p < .0.01$, GRBF $t(40) = -3.04$, GRBF-JJ $t(40) = 3.12$, GRBF-der $t(40) = -3.8$), for approximately the equivalent obstacle cost after 12 iterations. The joint interactions are smooth but take longer to converge (12 iter.), while the GRBF kernels with independent joints converged to a collision free trajectory approximately in the same number of iterations as the waypoints experiments (4 iter.). In Figure 8(a) we observe the end effector traces after 30 iterations. We optimized the model parameters for this scenario ($\beta = 1.0$, GRBF ($\lambda = 45$), GRBF-JJ ($\lambda = 40$), GRBF-der ($\lambda = 45$), Waypoints ($\lambda = 100$)). In Section F we compare against a non-optimized model.

VII. Discussion and Future Work

We present a kernel approach to trajectory optimization: we represent smooth trajectories as vector valued functions in an RKHS. Different kernels lead to different notions of smoothness, including commonly-used variants as special cases (velocity, acceleration penalization). We introduced a novel functional gradient trajectory optimization method based on RKHS representations, and demonstrated its efficiency compared with another optimization algorithm (CHOMP). This method benefits from a low-dimensional trajectory parametrization that is fast to compute. In the future, we are interested in extending the model to a multi-resolution planner, by learning model parameters dynamically, such as obstacle weight or kernel width. Furthermore, RKHSs enable us to plan with kernels learned from user demonstrations, leading to spaces in which more predictable motions have lower norm, and ultimately fostering better human-robot interaction [4]. Our work is an important step in exploring RKHSs for motion planning. It describes trajectory optimization under the light of reproducing kernels, which we hope can leverage the knowledge used in kernel based machine learning to develop better motion planning methods.

Acknowledgments

Support for this research was provided by FCT Portuguese Foundation for Science and Technology through the Carnegie Mellon Portugal Program under Grant SFRH/BD/52015/2012.

REFERENCES

[1] S. Amari. Natural gradient works efficiently in learning. *Neural Comput.*, 10(2):251–276, 1998.

[2] N. Aronszajn. Theory of reproducing kernels. In *Transactions of the American Mathematical Society*, 1950.

[3] A. Blake and M. Isard. *Active Contours*. Springer-Verlag New York, Inc., 1998.

[4] A. Dragan and S. Srinivasa. Familiarization to robot motion. *International Conference on Human-Robot Interaction (HRI)*, 2014.

[5] T. Hofmann, B. Schölkopf, and A. J. Smola. Kernel methods in machine learning. *Annals of Statistics*, 2008.

[6] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal. Stochastic trajectory optimization for motion planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[7] G. S. Kimeldorf and G. Wahba. Some results on tchebycheffian spline functions. In *Journal of Mathematical Analysis and Applications*, 1971.

[8] E. Kreyszig. *Introductory Functional Analysis with Applications*. Krieger Publishing Company, 1978.

[9] Charles A. Micchelli and Massimiliano A. Pontil. On learning vector-valued functions. *Neural Comput.*, 2005.

[10] J. Pan, L. Zhang, and D. Manocha. Collision-free and smooth trajectory computation in cluttered environments. In *International Journal of Robotics Research (IJRR)*, 1995.

[11] C. Park, J. Pan, and D. Manocha. Itomp: Incremental trajectory optimization for real-time replanning in dynamic environments. In *in Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2012.

[12] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C*. Cambridge University Press, 1992.

[13] S. Quinlan and O. Khatib. Elastic bands: Connecting path planning and control. In *IEEE International Conference on Robotics and Automation (ICRA)*, 1993.

[14] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa. CHOMP: Gradient optimization techniques for efficient motion planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2009.

[15] N. Ratliff, M. Toussaint, and S. Schaal. Understanding the geometry of workspace obstacles in motion optimization. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[16] N.D. Ratliff and J. A. Bagnell. Kernel conjugate gradient for fast kernel machines. 2007.

[17] K. Rawlik, M. Toussaint, and S. Vijayakumar. Path integral control by reproducing kernel hilbert space embedding. In *International Joint Conference on Artificial Intelligence*, IJCAI, pages 1628–1634, 2013.

[18] B. Scholkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2001.

[19] J. Schulman, J. Ho, A. Lee, I. Awwal, H. Bradlow, and P. Abbeel. Finding locally optimal, collision-free trajectories with sequential convex optimization. In *in Proc. of Robotics: Science and Systems (RSS)*, 2013.

[20] E. Todorov and W. Li. A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *in Proc. of the American Control Conference (ACC)*, 2005.

[21] J. van den Berg, P. Abbeel, and K. Goldberg. Lqg-mp: Optimized path planning for robots with motion uncertainty and imperfect state information. *International Journal of Robotics Research (IJRR)*, 2011.

[22] G. Wahba. *Advances in Kernel Methods*. MIT Press, 1999.

[23] M. Yuan and T. Cai. A reproducing kernel hilbert space approach to functional linear regression. *Annals of Statististics*, 2010.

[24] J. Zhang and A. Knoll. An enhanced optimization approach for generating smooth robot trajectories in the presence of obstacles. In *in Proc. of the European Chinese Automation Conference*, 1995.

[25] D. Zhou. Derivative reproducing properties for kernel methods in learning theory. *Journal of Computational and Applied Mathematics*, 2008.

[26] M. Zucker, N. Ratliff, A. Dragan, M. Pivtoraiko, M. Klingensmith, C. Dellin, J. A. Bagnell, and S. Srinivasa. Chomp: Covariant hamiltonian optimization for motion planning. In *International Journal of Robotics Research (IJRR)*, 2013.

APPENDIX

A. Finite approximation of Path Integral Cost

Trajectory optimization in RKHSs can be derived for different types of obstacle cost functionals, provided that trajectories have a finite representation. Previous work defines an obstacle cost in terms of the arc-length integral of the trajectory [26]. We approximate the path integral cost functional, with a finite representation using integral approximation methods, such as quadrature methods [12]. Consider a set of finite time points $t_i \in \boldsymbol{\mathcal{T}}$ to be the abscissas of an integral approximation method. We use a Gauss-Legendre quadrature method, and represent t_i as roots of the Legendre polynomial $P_n(t)$ of degree n. Let w_i be the respective weights on each cost sample:

$$\mathcal{U}_{obs}[\boldsymbol{\xi}] = \int_0^1 c\left[\boldsymbol{\xi}(t)\right] \left\| D^1\boldsymbol{\xi}(t) \right\| dt \approx \sum_{t_i \in \mathcal{T}} \omega_i \; c\left[\boldsymbol{\xi}(t_i)\right] \left\| D^1\boldsymbol{\xi}(t_i) \right\| \tag{20}$$

with coefficients, and the Legendre polynomials obtained recursively from the Rodriguez Formula:

$$P_n = 2^n \sum_{j=0}^{n} t^j \binom{n}{j} \binom{\frac{n+j-1}{2}}{n}$$

$$w_i = \frac{2}{(1-t_i^2)\left[D^1 P_n(t_i)\right]^2}$$

We denote $D^1 \equiv \frac{d}{dt}$ the first order time derivative. Using this notation, we are able to work with integral functionals, using still a finite set of time points to represent the full trajectory.

B. Waypoint Parametrization as an Instance of RKHS

Consider a general Hilbert space of trajectories $\xi \in \Xi$, (not necessarily an RKHS) equipped with an inner product $\langle \xi_1, \xi_2 \rangle_\Xi = \xi_1^T A \xi_2$. In the waypoint representation [26], A is typically the Hessian matrix over points in the trajectory, which makes the norm in ξ penalize unsmooth and inefficient trajectories, in the sense of high acceleration trajectories. The minimization under this norm $\|\xi\|_A = \sqrt{\xi^T A \xi}$ performs a line search over the negative gradient direction, where A dictates the shape of the manifold over trajectories. This paper generalizes the waypoint parameterization, we can represent waypoints by representing the trajectory in terms of delta Dirac basis functions $\langle \xi, \delta(t,\cdot) \rangle = \xi(t)$ with an additional smoothness metric A. Without A, each individual point is allowed to change without affecting points in the vicinity. Previous work, overcome this caveat by introducing a new metric that propagates changes of a single point in the trajectory to all the other points. Kernel evaluations in this case become $k(t_i,\cdot) = A^{-1}\delta(t_i,\cdot)$, where $\xi(t) = \sum_i a_i A^{-1}\delta(t_i,\cdot)$. The inner product of two functions is defined as $\langle \xi_1, \xi_2 \rangle_A = \sum_{i,j} a_i b_i A^{-1}\delta(t_i,t_i)$.
Here $\delta(t_i,\cdot)$ represents the finite dimensional delta function which is one for point t_i and zero for all the other points. A trajectory in the waypoint representation becomes a linear combination of the columns of A^{-1}. Columns of A^{-1} dictate how the corresponding point will affect the full trajectory. For an arbitrary kernel representation the behaviour of these points over the full trajectory are associated with the kernel functions associated with the space. For radial basis functions the trajectory is represented as Gaussian functions centered at a set of chosen time points (fewer in practice) instead of the full trajectory waypoints. In this sense, we have a more compact trajectory representation using RKHSs.

C. Kernel Metric in RKHS

The norm provides a form of quantifying how complex a trajectory is in the space associated with the RKHS kernel metric K. The kernel metric is determined by the kernel functions we choose for the RKHS, as we have seen before (Section IV). Likewise, the set of time points $\mathcal{T}$ that support the trajectory contribute to the design of the kernel metric:

$$\begin{aligned} \|\boldsymbol{\xi}\|_{\mathcal{H}}^2 &= \sum_{d,d'} \sum_{t_i,t_j \in \mathcal{T}} a_{d,i} k_{d,d'}(t_i,t_j) a_{d',j} \\ &= \sum_{t_i,t_j \in \mathcal{T}} \boldsymbol{a}_i^\top K(t_i,t_j)\boldsymbol{a}_{j'}, \quad \boldsymbol{a}_i, \boldsymbol{a}_j \in \mathbb{R}^D \\ &= \boldsymbol{a}^\top \boldsymbol{K}(\mathcal{T},\mathcal{T})\boldsymbol{a}, \boldsymbol{a} \in \mathbb{R}^{DN} \end{aligned} \tag{21}$$

Here $\boldsymbol{a}$ is the concatenation of all coefficients $\boldsymbol{a}_i$ over $\mathcal{T}$, $|\mathcal{T}| = N$. $\boldsymbol{K}(\mathcal{T},\mathcal{T}) \in \mathbb{R}^{DN \times DN}$ is the *Gram matrix* for all time points in the support of $\boldsymbol{\xi}$, and all joint angles of the robot. This matrix expresses the degree of correlation or similarity among different joints throughout the time points in $\mathcal{T}$. It can be interpreted, alternatively, as a tensor metric in a Riemannian manifold [1, 15]. Its inverse is the key element that bridges the gradient of functional cost $\nabla\mathcal{U}$ (gradient in the RKHS, Eq.13), and its conventional gradient (Euclidean gradient).This makes the process covariant (invariant to reparametrization).

$$\nabla\mathcal{U} = \boldsymbol{K}^{-1}(\mathcal{T},\mathcal{T})\nabla_E \, \mathcal{U} \tag{22}$$

The minimizer of the full functional cost $\mathcal{U}$ has a closed form solution in (12). Where the gradient $\nabla\mathcal{U}$, is the natural gradient in the RKHS. This can be seen as a warped version of the obstacle cost gradient according to the RKHS metric.

D. Efficient Constraint Update

We consider the sets of max points in the support set $\mathcal{T}$, equality constraint violation points (endpoints) T^o and inequality violation points (joint limits) T^p. We recover the Lagrange multipliers in (17) by solving (23).

$$\begin{aligned} &\begin{bmatrix} K(T^o,T^o) & K(T^o,T^p) \\ K(T^o,T^p)^\top & K(T^p,T^p) \end{bmatrix} \begin{bmatrix} \boldsymbol{\gamma^o} \\ \boldsymbol{\mu^p} \end{bmatrix} \\ &= \begin{bmatrix} (\lambda-\beta)\,\boldsymbol{\xi}^n(T^o) + \mathrm{dc}_J^\top K(\mathcal{T},T^o) - \boldsymbol{q_o} \\ (\lambda-\beta)\,\boldsymbol{\xi}^n(T^p) + \mathrm{dc}_j^\top K(\mathcal{T},T^p) - \boldsymbol{q_p} \end{bmatrix} \equiv \begin{bmatrix} h(T^o) \\ g(T^p) \end{bmatrix} \end{aligned} \tag{23}$$

To solve the system efficiently we make use of the reproducing property $K(T^o,T^o) = K(T^o,T^p)K(T^p,T^p)^{-1}K(T^o,T^p)^\top$, and the fact that we use use a separable kernel $K(t_i,t_j) =$

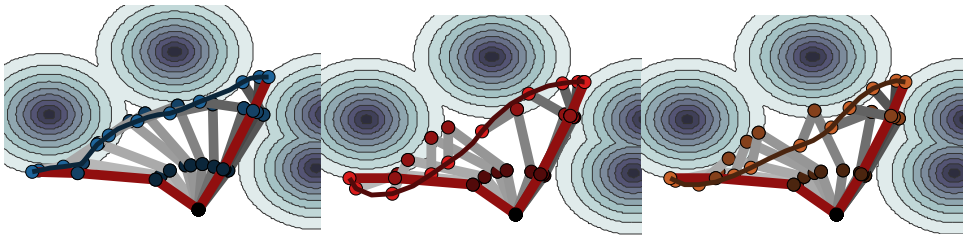
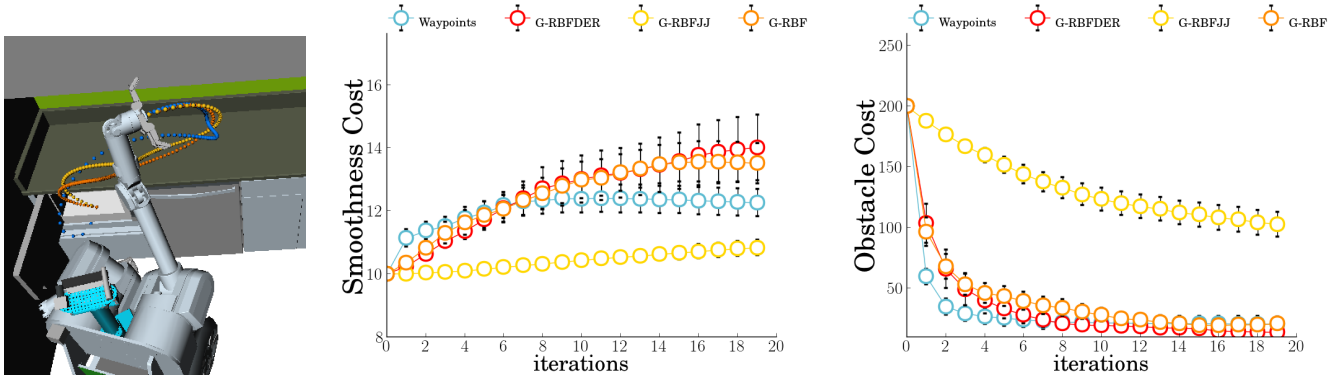


Fig. 10: 3DoF robot in 2D trajectory profile using different kernels. left: waypoints (blue), middle: Gaussian RBF with waypoints (red), right: Gaussian RBF (brown).



(a) 30 iterations (b) Obstacle cost (c) Smoothness cost

Fig. 11: (a)7-DOF robot experiment, plotting the end-effector position from start to goal. Waypoints (λ=40) (blue), Gaussian RBF (1 max point,λ=45,β=1.0) (orange), Gaussian RBF JJ (1 max point,λ=45,β=1.0) (yellow) (b) Obstacle and (c) Smoothness cost of random trajectories in constrained environment after 20 iterations

$k(t_i, t_j) \otimes J^\top J$.

$$
\begin{aligned}
K(T^p, T^p)^{-1} &= k(T^p, T^p)^{-1} \otimes (J^\top J)^{-1} \\
\boldsymbol{\mu^p} &= K(T^p, T^p)^{-1} g(T^p) \\
\boldsymbol{\gamma^o} &= (T^o, T^o)^{-1} \left(h(T^o) - K(T^o, T^p)\boldsymbol{\mu}^p\right)
\end{aligned} \tag{24}
$$

E. RBF norm vs. Waypoint parametrization

In this section, we study the effect of using an RKHS norm vs. the CHOMP norm. We perform an additional experiment, where we compare against an RKHS norm with waypoints. The trajectory becomes as weighted combinations of kernel functions $\xi(t) = \sum_i w_i k(t_i, \cdot) \forall_{t_i \in \mathcal{T}}$, evaluated at equally spaced time points (waypoints) . We performed qualitative experiments in a 2D setting with a 3-DOF planar arm. Figure 10 shows the end-effector traces (coloured line) after 10 iterations. We kept the RBF RKHS parameters fixed and optimized the obstacle weight, in order to achieve similar convergence speeds. We compare the waypoint CHOMP (blue) trajectory for the same number of waypoints (20) and only 4 max cost time points for the GRBF RKHS (brown). The GRBF RKHS with equally spaced points (red) has a smooth behaviour, similar to the GRBF RKHS norm with max points. This qualitative example suggests that optimizing in the RKHS norm yields smoother trajectories, independently of using waypoints or max cost time points.

F. Parameter selection sensitivity

Here we show the results for the 7-DOF simulation in constrained environments, without performing hyper-parameter optimization of the model (obstacle cost weight), §VI-F. We use the same model parameters as in the high clutter experiments §VI-E without optimizing for this particular task. We measure performance over 40 different random initial configurations. Figure 11(a) shows an example of Gaussian RBF with joint interactions in yellow (GRBF-JJ), independent joints in orange (GRBF), and waypoints in blue, after 30 iterations. We show smoothness and obstacle costs per iteration, see Figures 11(c) and 11(b) respectively. Without weight cost tuning the RKHSs perform worse, but still achieve low cost and smooth solutions. The updates with joint interactions are smoother but take longer to converge, while the GRBF kernels with independent joints converged approximately in the same number of iterations as the waypoints experiments (12it.). In Figure 11(a) we observe the end effector traces after 30 iterations. The joint interactions kernel is smoother compared with the waypoint parametrization, however the full trajectory smoothness is not statistically different when measured in the waypoint smoothness metric. We observe that optimizing the model parameters has a significant impact on the overall method performance, see Section VI-F. An interesting research extension could leverage this variability to produce planners with multi-resolution capabilities according to the surrounding environment (large motion in uncluttered scenes with high obstacle cost and high kernel widths vs. localized changes lower obstacle cost and with low kernel widths).