Multi-Arm Self-Collision Avoidance: A Sparse Solution for a Big Data Problem

Nadia Figueroa[1] Seyed Sina Mirrazavi Salehian[1] and Aude Billard[1]

***Abstract*— In this work, we propose a data-driven approach for real-time self-collision avoidance in multi-arm systems. The approach consists of modeling the regions in joint-space that lead to collisions via a Self-Collision Avoidance (SCA) boundary and use it as a constraint for a centralized Inverse Kinematics (IK) solver. This problem is particularly challenging as the dimensionality of the joint-configurations is in the order of millions (for a dual-arm system), while the IK solver must run within a control loop of 2ms. Hence, an extremely sparse solution is needed for this big data problem. The SCA region is modeled through a sparse non-linear kernel classification method that yields a runtime of less than 2ms (on a single thread CPU process) and has a False Positive Rate (FPR)=1.5%. Code for generating multi-arm datasets and learning the sparse SCA boundary are available at: `https://github.com/nbfigueroa/SCA-Boundary-Learning`**

I. INTRODUCTION

Self-collision avoidance is one of the main challenges in multi-arm manipulation. It is particularly relevant in the humanoid robot community and hence, has been extensively studied. Throughout the years, the approaches for solving collision avoidance for manipulation or locomotion in humanoids can be categorized into two types: (i) *planning* methods which generate feasible collision-free trajectories of known/quasi-static environments [27], [5], [4] and (ii) *reactive* approaches which solve collision-avoidance via the Inverse Kinematics (IK) problem online [8], [22], [26], [7]. For a comprehensive review on collision avoidance strategies for bi-manual systems refer to [16]. Most approaches, be it (i) *planning-based* or (ii) *reactive*, have a common methodology: they rely on computing *minimum distances* between links/joints/segments/objects (represented as sphere/swept-spheres/polygons) to detect/avoid collisions.

The use of *minimum distances* for collision avoidance inherently introduces non-linear and non-convex constraints to an otherwise convex optimization problem. This, in fact, is the main reason (i) *planning* algorithms rely on *computationally inefficient* global optimization or trajectory optimization methods and that (ii) *reactive* methods tend to get stuck in local minima. To this end, approaches based on signed distance fields have been successful in encoding proximity of obstacles as continuous costs in local trajectory optimization frameworks; by either providing explicit cost gradients [29], [6] or through derivative-free stochastic optimization methods [13]. Such approaches, however, require multiple initial-

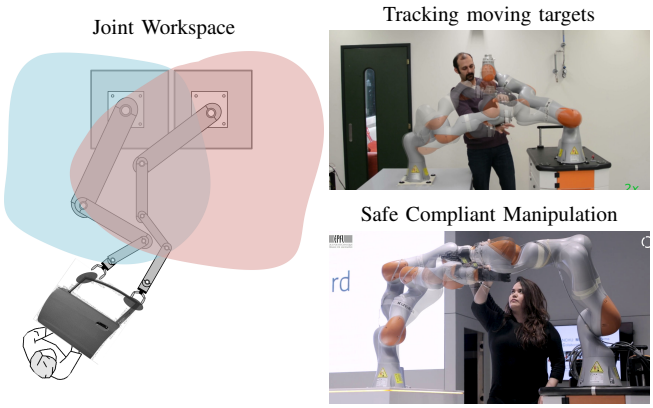


Fig. 1: Scenarios where real-time multi-arm Self-Collision Avoidance (SCA) resolution is necessary. Shaded areas denote workspace of each robot arm. The main idea in this work is to model the regions in joint-space which lead to collisions and use them as constraints in a centralized IK Solver.

izations as they fail to recover when solving for optimization problems that become ill-defined due to particular shapes of obstacles which might induce many local minima [18]. One solution to this is to exploit the Riemannian geometry of the workspace to represent costs for obstacle avoidance, kinematic limits, etc, by warping the workspace via Riemannian metrics and their gradients. While the previously mentioned trajectory optimization approaches' computation times range from 0.3-6s, the approach presented in [18] reports a computation time of 0.5s for a dual-arm platform. One of the draw-backs of these methods is the fact that they generate the collision-free trajectory a priori, hence, they are not robust to uncertainties in the workspace, the environment or the measurements that lead to collision detection. To tackle this issue, probabilistic approaches such as Monte Carlo Motion Planning (MCMP) [23], [10], have been introduced to compute low-cost paths that fulfill probabilistic collision avoidance constraints via importance sampling. In [23] a computation time of 2s for a 13s trajectory duration was reported for trajectory estimation under uncertainty of robots described by linear dynamics with control policies derived as LQG controllers tracking nominal trajectories. Though promising, it is hard to predict how this approach scales to a multi-arm robot configuration with non-linear dynamics. Needless to say, all of these *trajectory optimization* methods, though *efficient* in their own domain, are not sufficiently fast for applications that require tracking fast moving targets or safe compliant manipulation for human-robot-interaction, which rely on fast adaptive, typically closed-loop, control strategies requiring control-rates of 1-2ms [20], [21]. Such is the typical control-loop rate of KUKA 7DOF arms; e.g., the LWR and IIWA.

*Work supported by the EU project Cogimon H2020-ICT-23-2014

†Nadia Figueroa, Seyed Sina Mirrazavi Salehian and Aude Billard are both with the Learning Algorithms and Systems Laboratory (LASA), École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland - 1015. {nadia.figueroafernandez}@epfl.ch

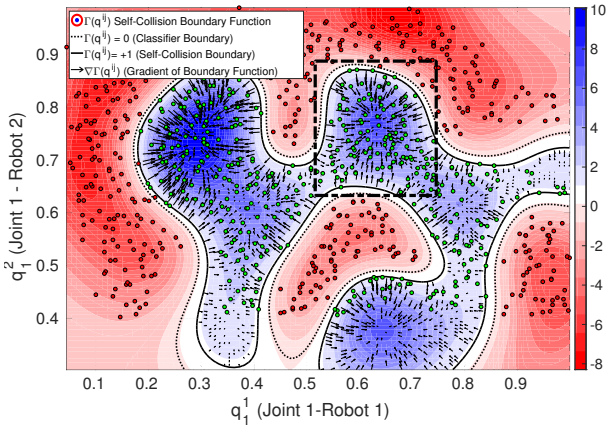


Fig. 2: $\Gamma(q^{ij})$ function for a toy 2D example. Assume two robots with 1-DOF each corresponding to each axis; i.e., $q^{ij} = [q_1^1, q_2^1]$. The green data-points represent "collision-free" robot configurations ($y = +1$), while the red data-points represent "collided" robot configurations ($y = -1$).

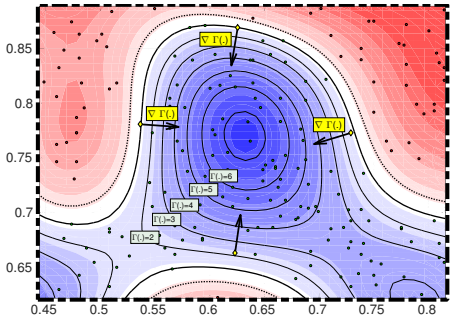


Fig. 3: Zoomed-in illustration of Fig. 2. The **Arrows** inside the *collision-free* region denote $\nabla\Gamma(q^{ij})$.

In this paper we focus on solving the self-collision avoidance problem for this particular type of applications *efficiently*; i.e., in $< 2ms$. Rather than explicitly computing distances or collision probabilities, we propose to learn a continuous and continuously differentiable function $\Gamma(.)$ from a dataset of "collided" and "non-collided" multi-arm joint configurations. $\Gamma(.)$ represents the region of feasible and infeasible robot configurations, implicitly encoding a distance in *feature space*, of the current robot configuration to a "collided" configuration. By formulating $\Gamma(.)$ as the prediction rule of a kernel Support Vector Machine (SVM) [24] we can compute a continuous $\nabla\Gamma(.)$ on-line. We propose a centralized IK solver as a convex QP optimization problem subject to *linear* inequality constraints imposed by $\Gamma(.)$ and $\nabla\Gamma(.)$, avoiding (i) the computation of $\min(||.||_2)$ pair-wise distances between joints/links/segments and (ii) the need for trajectory optimization. The main challenge is thus, to learn $\Gamma(.)$ from a simulated dataset of joint configurations in the order of millions, while predicting at $< 2ms$. Hence, we need to solve a big data problem with a sparse solution. By using a sparse SVM approximation [11] we learn an efficient representation of $\Gamma(.)$ that enables us to solve IK problem in less than $2ms$, on a single threaded CPU.

II. Problem Statement

To avoid collisions between the joints of the arms in a multi-arm system, we propose to use an Inverse Kinematic (IK) solver that not only considers the kinematic constraints of each robot, but also self-collision constraints. Assuming the robots' bases are fixed wrt. each other, we can explore the joint workspace of the robots, in order to model the regions that may lead to collision. Since the space of joint configurations is continuous, we must approximate the regions of collisions by building a continuous map of the feasible (safe) and infeasible (collided) configurations. Assuming that the infeasible regions can be bounded through a continuous and continuously differentiable function $\Gamma(q^{ij}) : \mathbb{R}^{d_{q_i}+d_{q_j}} \rightarrow \mathbb{R}$, where $q^{ij} = [q^i, q^j]^T \in \mathbb{R}^{d_{q_i}+d_{q_j}}$ are the joint angles of the i^{th} and j^{th} robots, respectively. We define $\Gamma(.)$ such that:

$$\begin{aligned} &\text{Collided configurations:} && \Gamma(q^{ij}) < 1 \\ &\text{Boundary configurations:} && \Gamma(q^{ij}) = 1 \\ &\text{Free configurations:} && \Gamma(q^{ij}) > 1 \end{aligned} \tag{1}$$

(1) provides constraints that must be met when solving the IK problem, proposed as the following quadratic program):

$$\underbrace{\underset{\dot{\mathbf{q}}}{\text{argmin}} \frac{\dot{\mathbf{q}}^T W \dot{\mathbf{q}}}{2}}_{\text{Minimize Expenditure}}$$

subject to

$$\begin{cases} (a) & \underbrace{\mathbf{J}(\mathbf{q})\dot{\mathbf{q}} = \dot{\boldsymbol{\xi}}^R}_{\text{Satisfy the desired end-effector motion}} \\ (b) & \underbrace{\dot{\boldsymbol{\theta}}^- \leq \dot{\mathbf{q}} \leq \dot{\boldsymbol{\theta}}^+}_{\text{Satisfy the kinematic constraints}} \\ (c) & -\nabla\Gamma^{ij}(q^{ij})^T \dot{q}^{ij} \leq \log(\Gamma^{ij}(q^{ij}) - 1) \\ & \underbrace{\forall (i,j) \in \{(1,2),(1,3),\ldots,(N_R-1,N_R)\}}_{\text{Do not penetrate the collision boundary}} \end{cases} \tag{2}$$

Where, $\mathbf{q} = [q^1, \ldots, q^{N_R}]^T \in \mathbb{R}^{d_\mathbf{q}}$, $d_\mathbf{q} = \sum_{i=1}^{N_R} d_{q_i}$ for N_R robots. W is a block diagonal matrix of positive definite matrices. $\mathbf{J} = diag(J_1, \ldots, J_{N_R})$ is block diagonal matrix of the Jacobian matrices. $\dot{\boldsymbol{\xi}}^R = \begin{bmatrix} \dot{\xi}_1^R & \ldots & \dot{\xi}_{N_R}^R \end{bmatrix}^T \in \mathbb{R}^{d_\mathbf{n}}$, $d_\mathbf{n} = N_R d_n$ is the desired velocity given by a task-space motion generator. $\dot{\boldsymbol{\theta}}^i = \begin{bmatrix} \dot{\theta}^i{}_1 & \ldots & \dot{\theta}^i_{N_R} \end{bmatrix}$ $\forall i \in \{-,+\}$ and $\dot{\theta}_i^+ \in \mathbb{R}^m$ and $\dot{\theta}_i^- \in \mathbb{R}^m$ are conservative lower and upper bounds of the joint limits. We follow the approach proposed in [28] to compute the joint limits at the velocity level.

In Fig. 2, we illustrate a $\Gamma(.)$ for a toy $2D$ example to highlight its role in the quadratic program. While the robots are far from the boundary configurations, the value of $\log(\Gamma^{ij}(q^{ij}) - 1)$ is positive which relaxes the inequality constraints; i.e., the robots accurately follow the desired end-effector trajectory. When they are near the boundary configurations, the value of $\log(\Gamma^{ij}(q^{ij}) - 1)$ is negative. Therefore, constraint (2)(c) forces the joint angles to move away from the boundary as they approach it. (2) is a convex quadratic program (QP) with equality/inequality constraints. This can be solved via a standard nonlinear programming solver such as Nlopt [12] or a constrained convex optimization solver such as CVXGEN [15]. For a comparison, refer to [21].

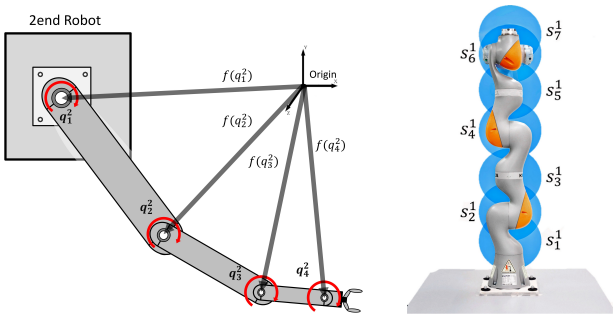


Fig. 4: Illustration of **(left)** 3D Cartesian positions (e.g. $f(q_1^2) : S^1 \rightarrow \mathbb{R}^3$) of the individual joint angles in the multi-arm system reference frame used to learn the SCA Boundary function $\Gamma(f(q^2))$ **(right)** the geometric representation of i-th robot with a set of spheres corresponding to each joint used to construct the SCA dataset.

III. Learning a Self-Collision Avoidance (SCA) Boundary

As per (1), the SCA boundary function $\Gamma(q^{ij})$ should be of class $\mathcal{C}^0$ and $\mathcal{C}^1$. Interestingly, the problem can be formulated as a binary classification problem, $y \leftarrow \text{sgn}\left(\Gamma(q^{ij})\right)$ for $y \in \{+1, -1\}$, where "collided" joint configurations belong to the negative class (i.e. $y = -1$) and "non-collided" configurations belong to the positive class (i.e., $y = +1$). When $\Gamma(q^{ij}) = 1$, q^{ij} is at the boundary of the positive class; i.e., the self-collision boundary (black line in Fig. 2).

To recall, the configuration space of a multi-arm system is an n-dimensional torus ($S^1 \times S^1 \times \cdots \times S^1 (n-times) = \mathcal{T}^N$) for n DOF [14]. Given two robots with 7DOF each, the manifold in which the multi-arm joint-angle vector lies in is $q^{ij} \in \mathcal{T}^{14}$. Employing q^{ij} as the feature vector for a classification problem can be problematic for several reasons. Many machine learning algorithms rely on computing distances/norms in Euclidean space, assuming the features are i.i.d. from an underlying distribution in $\mathbb{R}^N$. Hence, a Euclidean norm applied on $q^{ij} \in \mathcal{T}^N$, is merely an approximation of the actual distance in the $\mathcal{T}^N$ manifold. In fact, a proper distance metric for joint-angles, i.e., $d(q_1^{ij}, q_2^{ij})$ where $q^{ij} \in \mathcal{T}^N$, is non-existent. For this reason, most trajectory optimization planning algorithms rely on mapping joint-space configurations to task-space via forward kinematics [9], where distances are well-established.

For such reasons, and in-line with the trajectory optimization literature, instead of learning our self-collision avoidance (SCA) decision boundary function $\Gamma(.)$ on the joint-angle data q^{ij}, we learn $\Gamma(.)$ on the 3D Cartesian representation of the joint-angles $f(q^{ij})$. As illustrated in Fig. 4, $f(q^{ij})$ is a vector composed of the 3D Cartesian positions of all joints for the i-th and j-th robot, computed via forward kinematics. The feature vector for a dual-arm robotic system is thus $f(q^{ij}) \in \mathbb{R}^{3d_{q_i}+3d_{q_j}}$. We posit that, by using $f(q^{ij})$ instead of q^{ij}, we can achieve a better trade-off between model complexity and error rate. Moreover, since the output of $\Gamma(.)$ is expected to be a scalar (1), no extra computation is necessary, as $\Gamma(q^{ij}) \equiv \Gamma(f(q^{ij}))$. The linear inequality constraints in (2) require $\nabla\Gamma(.)$ to be $\mathcal{C}$, this be provided by either Support Vector Machines (SVM) or Neural Networks (NN). In this work, we favor the use of SVMs, for two main reasons: (i) Learning a SVM is a convex optimization problem; hence, we can always reach a *global optimum*. (ii) SVMs yield sparser models for high-dimensional non-linear classification problems, leading to better runtimes.

A. $\mathcal{C}^0$ and $\mathcal{C}^1$ Self-Collision Avoidance (SCA) Boundary via Support Vector Machines

We follow the kernel Support Vector Machine (SVM) formulation and propose to encode $\Gamma\left(f(q^{ij})\right)$ as the SVM decision rule. By omitting the sign function and using the RBF Kernel $k\left(f(q^{ij}), f(q_n^{ij})\right) = e^{\left(-\frac{1}{2\sigma^2}||f(q^{ij}) - f(q_n^{ij})||^2\right)}$, for a kernel width σ; $\Gamma\left(f(q^{ij})\right)$ (from herein the superscripts (ij) of are dropped) has the following form,

$$\begin{aligned}\Gamma\left(f(q^{ij})\right) &= \sum_{n=1}^{N_{sv}} \alpha_n y_n k\left(f(q^{ij}), f(q_n^{ij})\right) + b \\ &= \sum_{n=1}^{N_{sv}} \alpha_n y_n e^{\left(-\frac{1}{2\sigma^2}||f(q^{ij}) - f(q_n^{ij})||^2\right)} + b,\end{aligned} \tag{3}$$

for N_{sv} support vectors, where $y_i \in \{-1, +1\}$ are the positive/negative labels corresponding to non-collided/collided configurations, $0 \leq \alpha_i \leq C$ are the weights for the support vectors which must yield $\sum_{n=1}^{N_{sv}} \alpha_n y_n = 0$ and $b \in \mathbb{R}$ is the bias for the decision rule. $C \in \mathbb{R}$ is a penalty factor used to trade-off between maximizing the margin and minimizing classification errors. Given C and σ, α_i's and b are estimated by solving the dual optimization problem for the *soft-margin* kernel SVM [24]. (3) naturally yields a continuous gradient,

$$\begin{aligned}\nabla\Gamma\left(f(q^{ij})\right) &= \sum_{n=1}^{N_{sv}} \alpha_n y_n \frac{\partial k\left(f(q^{ij}), f(q_n^{ij})\right)}{\partial f(q^{ij})} \\ &= \sum_{n=1}^{N_{sv}} -\frac{1}{\sigma^2} \alpha_n y_n e^{\left(-\frac{1}{2\sigma^2}||f(q^{ij}) - f(q_n^{ij})||^2\right)} \left(f(q^{ij}) - f(q_n^{ij})\right).\end{aligned} \tag{4}$$

Although $\nabla\Gamma(f(q^{ij}))$ already satisfies the constraints imposed by (2), it lives in a $3d_{q_i} + 3d_{q_j}$-dimensional space, $\nabla\Gamma\left(f(q^{ij})\right) \in \mathbb{R}^{3d_{q_i}+3d_{q_j}}$. We must then project this gradient onto its corresponding $\mathbb{R}^{d_{q_i}+d_{q_j}}$ joint-space; i.e., $\nabla\Gamma(q^{ij}) \in \mathbb{R}^{d_{q_i}+d_{q_j}}$ with the following expansion:

$$\nabla\Gamma\left(q^{ij}\right) = \frac{\partial\Gamma\left(f(q^{ij})\right)}{\partial f(q^{ij})} \cdot \frac{\partial f(q^{ij})}{\partial q^{ij}}, \tag{5}$$

the first term is equivalent to (4) and the second term is the Jacobian of each 3D joint position wrt. each joint angle $J(q^{ij}) = \frac{\partial f(q^{ij})}{\partial q^{ij}}$ for which we have a closed-form solution. Given the feature vector $f(q^{ij})$, the self-collision avoidance constraint (2) becomes,

$$-\nabla\Gamma^{ij}(q^{ij})^T \dot{q}^{ij} \leq \log(\Gamma^{ij}(f(q^{ij})) - 1). \tag{6}$$

B. Self-Collision Avoidance (SCA) Dataset Construction

In order to learn $\Gamma\left(f(q^{ij})\right)$, we must initially generate a dataset capable of identifying the so-called *self-collision boundary*. We begin by describing our simplified geometric representation of the robot's kinematic configuration used to identify "collided" and "non-collided" configurations. For simplicity, let's assume a dual-arm setting, with each arm

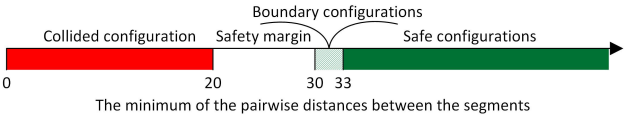
being a KUKA 7DOF (Fig. 5). Similar to [29], we simplify the representation of the robot's structure by fitting spheres to each joint and its adjoining physical structure. We thus generate a discrete representation of the multi-arm robotic system as a set of spheres $S^{ij} = \{s_1^i, \dots, s_7^i, s_1^j, \dots, s_7^j\}$. By using spheres as a geometric representation of a joint, we simplify the distance computation between joints, as the distance from any point in a sphere to the nearest obstacle is lower-bounded by $d(c) - r$; where c is the center of the sphere and r its corresponding radius [19]. Further, the lower-bound between two spheres is the distance between their centers (c_k^i) minus the sum of their respective radii (r_k^i), for the k-th spheres of the i-th robot. For example, given s_5^1 and s_7^2 the lower-bounded distance between them is $d(s_5^1, s_7^2) = d(c_5^1, c_7^2) - (r_5^1 + r_7^2)$.

To identify collision in the dual-arm system, we compute the pairwise distances of the centers of the set of spheres of the i-th robot (S^i) wrt. the set of spheres of the j-th robot (S^j) and find the minimum distance $\min[d(c_{k^*}^1, c_{k^*}^2)]$, as shown in Fig. 4. We then define a label for each robot configuration S^{ij} as,

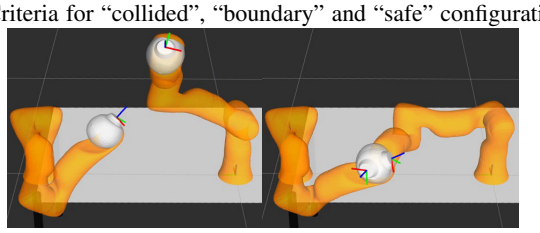
$$y(S^{ij}) = \begin{cases} -1 & \text{if} \quad \min[d(c_{k^*}^1, c_{k^*}^2)] < (r_{k^*}^1 + r_{k^*}^2) \\ +1 & \text{if} \quad b_- \leq \min[d(c_{k^*}^1, c_{k^*}^2)] \leq b_+ \\ \emptyset & \text{if} \quad \min[d(c_{k^*}^1, c_{k^*}^2)] > b_+ \end{cases} \tag{7}$$

where $r_{k^*}^i$ corresponds to the radius of the k-th sphere, and b_-,b_+ correspond to minimum/maximum distances of the "safe" boundary. Specifically, a joint configuration is "collided", i.e. labeled as $y = -1$, when the $\min[d(c_{k^*}^1, c_{k^*}^2)]$ between the centers of the closest spheres is less than the sum of the radii of the corresponding spheres, i.e., $(r_{k^*}^1 + r_{k^*}^2)$. In practice, we set the spheres to a fixed radius of $10cm$, hence $(r_{k^*}^1 + r_{k^*}^2) = 20cm$. Given that virtually any robot configuration where $\min[d(c_{k^*}^1, c_{k^*}^2)] > (r_{k^*}^1 + r_{k^*}^2)$ can be considered "non-collided" configurations, we would end up with a heavily un-balanced dataset of "collided"/"non-collided" data-points. We thus, introduce a decomposition of the "non-collided" robot configurations into "boundary", labeled as $y = +1$, and "safe" configurations, which are not labeled $y = \emptyset$. If $\min[d(c_{k^*}^1, c_{k^*}^2)]$ lies within a safety margin, denoted by b_- and b_+, the robots are very close to each other but still safe, see Fig. 5. We empirically found $b_- = 30cm$ and $b_+ = 33cm$ to be safe boundaries for our dual-arm setting. Hence, a "non-collided" configuration is in fact a "boundary" configuration, as all of the "safe" configurations are filtered out. This has a geometric meaning, rather than finding the margin between "collided" and "safe" configurations, our boundary function will model the tighter margin between "non-collided" and "boundary" configurations. From herein, we consider "boundary" configurations as the "non-collided" configurations.

To generate the positive ($y(S^{ij}) = +1$) and negative samples ($y(S^{ij}) = -1$) for our SCA dataset, we sample from all the possible motions of the robots in their respective workspaces and apply (7) to each configuration. To explore all possible joint configurations q^{ij}, we systematically displace all of the joints of both robots by $20\,\text{deg}$ each. Joints



(a) Criteria for "collided", "boundary" and "safe" configurations.



(b) "boundary configuration" (c) "collided configuration"

Fig. 5: Examples of the collided/boundary configurations of a *dual-arm setting* with an offset of $X_{off} = [0.0, 1.3, 0.34]m$ between their bases.

q_1^i, q_3^i, q_5^i, q_7^i have a range of $\pm 170\,\text{deg}$, whereas joints q_2^i, q_4^i and q_6^i have a range of $\pm 120\,\text{deg}$. Given the $20\,\text{deg}$ sampling resolution, this leads to 18 samples for the former group and 12 for the latter. Hence, the total number of possible configurations for one arm is $18^3 * 12^3$, which would lead to $\approx 1e14$ possible joint configuration for a dual-arm setting. However, using our systematic sampling of "collided" and "boundary" robot configurations, we gathered a balanced dataset of approximately ≈ 5.4 million data-points, ≈ 2.4 million belonging to the "collided" configuration class $y = -1$ and the rest to the "non-collided" configurations $y = +1$.

C. Sparse SVMs for Large Datasets

Training time of a kernel SVM has a complexity of $\approx \mathcal{O}({N_M}^2 D)$, where N_M is the number of samples and D is the dimension of the data-points. Prediction time, on the other hand, depends on the number of support vectors N_{sv} learned through training. In practice, the N_{sv} tends to increase linearly with the amount of training data N_M [1]. More specifically, for a kernel SVM $N_{sv}/N_M \to \mathcal{B}_k$, where $\mathcal{B}_k$ is the smallest achievable classification error by the kernel k [25]; i.e. in a non-separable classification scenario, to achieve 5% error, at least 5% of the training points must become support vectors. This comes as a nuisance when large training sets are involved, as is the case for our application. A $N_{sv} \gg$ signifies a dense solution for representing the hyper-plane of the classifier margin $\mathbf{w} = \sum_{i=1}^{N_{sv}} \alpha_i y_i \Phi(f(q_i^{ij}))$. Naturally, the denser the solutions, the more computationally expensive they are at run-time. This makes dense SVMs infeasible for real-time robot control. In order to achieve fast adaptation for both the desired end-effector positions and self-collisions, the IK solver must run *(at most)* at a rate of $2ms$. Note that during this cycle, prior to solving (2), both (3) and (5) must be evaluated.

Given the desired control rate ($2ms$), the specific hardware used to control the robots (i.e., 3.4-GHz i7 PC with 8GB RAM) and the kinematic specifications of each robot, we can define a *computational budget* for our Self-Collision Avoidance (SCA) Boundary function. This *budget* translates to, defining a limit of the maximum allowable N_{sv} for our SVM representation of $\Gamma(f(q^{ij}))$. In Fig. 6, we show a plot

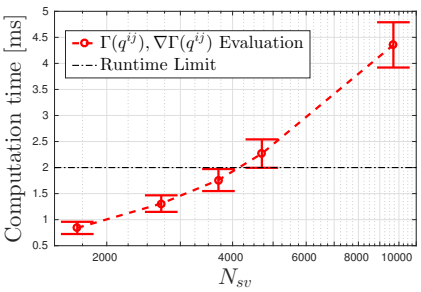


Fig. 6: Comparison of runtime computational cost for evaluating $\Gamma(f(q^{ij}))$ and $\nabla\Gamma\left(f(q^{ij})\right)$ on the specified hardware for a *dual-arm setting*. The presented runtimes are of $\approx 2k$ control loop cycles of a self-collision avoidance test. The maximum allowable $N_{sv} \leq 3k$ in order to comply with the $2ms$ limit.

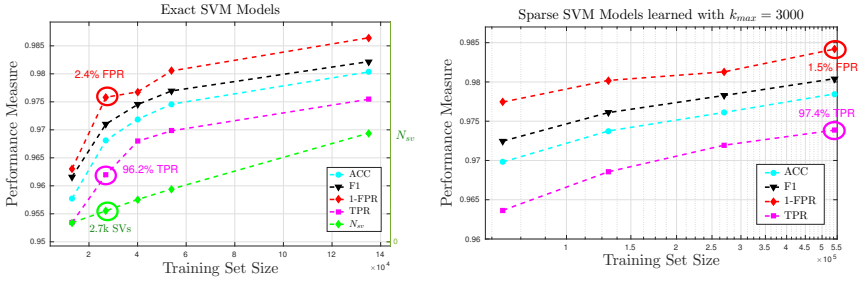


Fig. 7: Performance Comparison of learning exact SVM models on *randomly sub-sampled* datasets vs sparse SVM models on larger chunks of the dataset. Each model was evaluated on the test set, which contains 2.7 million unseen sample robot configurations. (left) With the random sub-sampling method, using the 2nd model ($N_{sv} = 2,7k$), one can achieved $FPR \approx 2.4\%$ and $TPR \approx 96.19\%$ within the desired $2ms$ runtime limit. (right) With a sparse SVM model trained on $540k$ points we can achieve $FPR \approx 1.45\%$ and $TPR \approx 97.4\%$ $\mathbf{k_{max} = 3000}$.

of different computation times. We omit computation time of the IK solver as this is presented in detail in [21].

In order to comply with the $2ms$ runtime requirement, we have a *computational budget* of $N_{sv} \leq 3k$. Given the size of our dataset, it is not feasible to train SVM models that typically optimize for the dual through a variant of Sequential Minimal Optimization (SMO) [17] or SMO-type decomposition methods [2], [3]. The fact that SVM learning algorithms tend to produce dense solutions has been recognized as one of its main weaknesses. To this end, several approaches have been proposed in order to solve the problem of finding sparser solutions to $\mathbf{w}$. These can be categorized into: i) post-processing approximations and ii) objective function or optimization strategy modification. The *former* approaches rely on approximating a sparse solution to an initially dense SVM (through the exact solution). The *latter* approaches either modify the SVM objective function by imposing sparsity constraints or propose a modified optimization algorithm for with sparsity considerations.In this work, we choose one of the prevailing approaches which reformulates the SVM optimization problem, namely the Cutting Plane Subspace Pursuit (CPSP) method introduced by [11]; as it directly estimates a solution to the hyper-plane with a strict bound on the number of support vectors $\mathbf{k_{max}}$. In short, the CPSP method approximates a sparse hyper-plane by expressing it in terms of a set $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_{\mathbf{k_{max}}}\}$ of basis vectors $\mathbf{b}_i \in \mathbb{R}^{3d_{q_i}+3d_{q_j}}$ (not necessarily training points) as $\mathbf{w} = \sum_{i=1}^{\mathbf{k_{max}}} \alpha_i y_i \Phi(\mathbf{b}_i)$. The optimization algorithm to estimate this new $\mathbf{w}$ then focuses on *pursuing* such a subspace through the fixed-point iteration approach for RBF kernels [24]. The learned basis vectors $\mathbf{B}$ and α_i's can be directly used in (3) and (4). We direct the interested reader to [11] for theoretical equivalence proofs and implementation details.

IV. Results

A. Evaluation of Learning Performance

We begin our $\Gamma\left(f(q^{ij})\right)$ learning performance analysis by presenting results from learning exact SVMs from small sub-samples of the $5.4m$ point dataset. To generate such comparison, the libSVM library [3] was used for learning the SVM models, cross-validation routines to find the optimal hyper-parameters are provided in the cited code. A

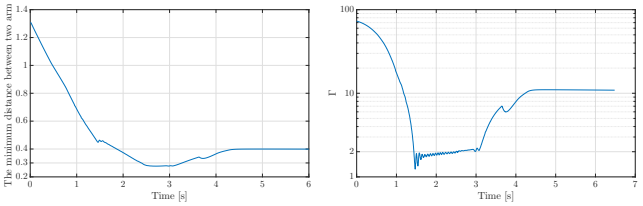


Fig. 8: Self-Collision Avoidance Test.(left) and (right) show the minimum distance between the arms and the value of Γ during the motion execution. $\Gamma(\cdot)$ never reaches < 1 indicating that the motion of the arms are safe. Refer to Video Link 2 Experiment 1 for execution.

50% split for training+validation/testing datasets was used to generate such evaluations. We evaluate 5 models with increasing complexity $N_{sv} = \{1.7k, 2.7k, 3.7k, 4.7k, 9.7k\}$. These were learned from using $\{0.5\%, 1\%, 1.5\%, 2\%, 5\%\}$ of the training set (i.e., 2.7 million data-points). For each model, a 10-fold Cross-validation was performed to find the optimal hyper-parameters C and σ which yield the best trade-off between N_{sv}/N_M and classification accuracy. The search space of each hyper-parameter was log-spaced in the following ranges $C = [10^{-1}, 10^4]$ and $\sigma = [0.2, 2]$.

In our application, we care about correctly classifying the negative class (i.e., "collided" configurations), for this we have two objectives: (i) *Minimize False Positive Rate* ($FPR = \frac{FP}{(FP+TN)}$)*:* FPR quantifies the probability of negative samples ($y = -1$) being classified as positive ($y = +1$). Classifying "collided" configurations ($y = -1$) as "non-collided" configurations ($y = +1$) yields a False Positive (FP). This error is critical as it would cause the IK solver to move the robots into an infeasible region, leading to collision and possibly permanent damage. (ii) *Maximize True Positive Rate* ($TPR = \frac{TP}{(TP+FN)}$)*:* TPR quantifies the probability of positive samples ($y = +1$) being classified as positive ($y = +1$). Classifying "collided" configurations ($y = +1$) as "non-collided" ($y = -1$) yields a False Negative (FN). This error is not as critical as the former, but it has an effect on the performance of the IK solver, as classifying "non-collided" configuration as "collided" would restrict the IK solver to move the robots into regions that are feasible.

As can be seen from Fig. 7, we can achieve optimal error rates on the testing set of $FPR \approx 1.3\%$ and $TPR \approx 97.54\%$, with 5% of the training dataset, albeit surpassing the N_{SV} limit. One might argue that, with such high performance of models trained on a minuscule amount of

data (relative to the complete dataset), perhaps such a large dataset is not necessary. This is related to the SCA dataset construction procedure (Section III-B), where we set the joint sampling interval to $20\,\mathrm{deg}$. We can see that with the 2nd model (i.e., 1% of training data), we achieve error rates of: $FPR \approx 2.4\%$ and $TPR \approx 96.19\%$ withing the *computational budget*. This is quite acceptable performance, however, due to the delicacy of our application we seek to achieve the best solution possible, i.e. at least $FPR \approx 1\%$. In Fig. 7, we present the results of using the CPSP SVM learning approach on different sub-sets of our training data limited to a support vector budget of $k_{max} = 3000$, specifically $\{2.5\%, 5\%, 10\%, 20\%\}$. As can be seen, for the models learned on datasets with the same size as the exact SVM solutions, the results are marginally lower. However, as the number of training-points increases the error rates improve as much as $FPR = 1.5\%$ and $TPR = 97.4\%$ for a training set of $540k$ points. By using this sparse learning method we have proven that optimal error rates can be achieved with minimal model complexity.

B. Experimental Validation

The proposed approach has been successfully used to control multi-arm reach-to-grab motions as reported in [21] and shown in Video Link 1. We have also evaluated the approach in tasks where the robots must track *fast* moving targets that go inside each other's workspaces and in tasks where the robots are perturbed by humans leading to possible collision as shown in Video Link 2. In Fig. 8 we report the evolution of $\Gamma(.)$ and the real min distance between the robots for Experiment 1 in Video Link 2. As can be seen, when $\Gamma(.) < 2$, the robots are pushed away from each other.

V. Discussion and Future Work

In this paper, we proposed an efficient data-driven technique to successfully solve the self-collision avoidance problem in a multi-robot arm setting. We report experiments on the 7DOF KUKA LWR and IIWA platforms, yet the approach is platform agnostic. An evident limitation is the fact that each trained SCA boundary is dependent on the relative distance between the robots. Given a change in the robot workspace, the boundary must be re-trained with a new dataset. This can be tackled in a more principled approach via incremental boundary learning or using ensemble methods.

Furthermore, we focused on the problem of learning a sparse model with a prediction time of $<$ 2ms, due to our hardware limitations. Given a GPU with multiple cores, our computational budget $\mathbf{k}_{\max}$, will surely increase. This would allow us to use more data to learn the SCA boundary, potentially reaching better error rates. The implementation and comparison of such an approach is of interest and is being planned as the next steps of this work.

References

[1] Gökhan H. Bakir, Léon Bottou, and Jason Weston. Breaking svm complexity with cross-training. In *Proceedings of the 17th NIPS*, NIPS'04, pages 81–88, Cambridge, MA, USA, 2004. MIT Press.

[2] Antoine et al. Bordes. Fast kernel classifiers with online and active learning. *JMLR*, 6:1579–1619, September 2005.

[3] Chih-Chung Chang and Chih-Jen Lin. Libsvm: A library for support vector machines. *ACM TIST*, 2(3):27:1–27:27, May 2011.

[4] B. Chrètien, A. Escande, and A. Kheddar. Gpu robot motion planning using semi-infinite nonlinear programming. *IEEE Transactions on Parallel and Distributed Systems*, 27(10):2926–2939, Oct 2016.

[5] A. Escande, S. Miossec, and A. Kheddar. Continuous gradient proximity distance for humanoids free-collision optimized-postures. In *2007 7th IEEE-RAS Humanoids*, pages 188–195, Nov 2007.

[6] John Schulman et al. Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research*, 33(9):1251–1270, 2014.

[7] C. Fang, A. Rocchi, E. M. Hoffman, N. G. Tsagarakis, and D. G. Caldwell. Efficient self-collision avoidance based on focus of interest for humanoid robots. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pages 1060–1066, Nov 2015.

[8] S. S. Ge and Y. J. Cui. New potential functions for mobile robot path planning. *IEEE Trans. on Rob. and Autom.*, 16(5):615–620, Oct 2000.

[9] Rachel Holladay and Siddhartha Srinivasa. Distance metrics and algorithms for task space path optimization. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, October 2016.

[10] Lucas Janson, Edward Schmerling, and Marco Pavone. *Monte Carlo Motion Planning for Robot Trajectory Optimization Under Uncertainty*, pages 343–361. Springer International Publishing, Cham, 2018.

[11] Thorsten Joachims and Chun-Nam John Yu. Sparse kernel svms via cutting-plane training. *Mach. Learn.*, 76(2-3):179–193, 2009.

[12] Steven G Johnson. The NLopt nonlinear-optimization package, 2016.

[13] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal. Stomp: Stochastic trajectory optimization for motion planning. In *IEEE ICRA*, Shanghai, China, May 9-13, 2011.

[14] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, New York, NY, USA, 2006.

[15] Jacob Mattingley and Stephen Boyd. Cvxgen. *Optimization and Engineering*, 13(1):1–27, 2012.

[16] T. Petrič, A. Gams, N. Likar, and L. Žlajpah. *Motion and Operation Planning of Robotic Systems: Background and Practical Approaches, Part II, Chapter Obstacle Avoidance with Industrial Robots*, pages 113–145. Springer International Publishing, Cham, 2015.

[17] John C. Platt. Advances in kernel methods. chapter Fast Training of Support Vector Machines Using Sequential Minimal Optimization, pages 185–208. MIT Press, Cambridge, MA, USA, 1999.

[18] N. Ratliff, M. Toussaint, and S. Schaal. Understanding the geometry of workspace obstacles in motion optimization. In *2015 ICRA*, pages 4202–4209, May 2015.

[19] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *2009 IEEE ICRA*, pages 489–494, May 2009.

[20] Seyed Sina Mirrazavi Salehian, Nadia Figueroa, and Aude Billard. Coordinated multi-arm motion planning: Reaching for moving objects in the face of uncertainty. In *Proceedings of Robotics: Science and Systems*, AnnArbor, Michigan, June 2016.

[21] Seyed Sina Mirrazavi Salehian, Nadia Figueroa, and Aude Billard. A unified framework for multi-arm motion planning. *International Journal of Robotics Research [In press]*, 2018.

[22] A. De Santis, A. Albu-Schaffer, C. Ott, B. Siciliano, and G. Hirzinger. The skeleton algorithm for self-collision avoidance of a humanoid manipulator. In *2007 IEEE/ASME AIM*, pages 1–6, Sept 2007.

[23] E. Schmerling and M. Pavone. Evaluating Trajectory Collision Probability through Adaptive Importance Sampling for Safe Motion Planning. *ArXiv e-prints*, September 2016.

[24] Bernhard Scholkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001.

[25] Ingo Steinwart. Sparseness of support vector machines. *J. Mach. Learn. Res.*, 4:1071–1105, December 2003.

[26] H. Sugiura, M. Gienger, H. Janssen, and C. Goerick. Real-time collision avoidance with whole body motion control for humanoid robots. In *2007 IEEE/RSJ IROS*, pages 2053–2058, Oct 2007.

[27] N. Vahrenkamp, T. Asfour, and R. Dillmann. Simultaneous grasp and motion planning: Humanoid robot armar-iii. *IEEE Robotics Automation Magazine*, 19(2):43–57, 2012.

[28] Y. Xia and J. Wang. A recurrent neural network for solving linear projection equations. *Neural Networks*, 13(3):337–350, 2000.

[29] Matthew Zucker, Nathan Ratliff, and et al. Chomp: Covariant hamiltonian optimization for motion planning. *IJRR*, May 2013.