Real-Time Self-Collision Avoidance in Joint Space for Humanoid Robots

Mikhail Koptev, Nadia Figueroa, and Aude Billard, *Member, IEEE*

***Abstract*—In this letter, we propose a real-time self-collision avoidance approach for whole-body humanoid robot control. To achieve this, we learn the feasible regions of control in the humanoid's joint space as smooth self-collision boundary functions. Collision-free motions are generated online by treating the learned boundary functions as constraints in a Quadratic Program based Inverse Kinematic solver. As the geometrical complexity of a humanoid robot joint space grows with the number of degrees-of-freedom (DoF), learning computationally efficient and accurate boundary functions is challenging. We address this by partitioning the robot model into multiple lower-dimensional submodels. We compare performance of several state-of-the-art machine learning techniques to learn such boundary functions. Our approach is validated on the 29-DoF iCub humanoid robot, demonstrating highly accurate real-time self-collision avoidance.**

***Index Terms*—Collision avoidance, humanoid robot systems, machine learning for robot control.**

I. Introduction

HUMANOID robots have a wide span of possible applications, potentially becoming universal assistants. Though they are difficult to control due to balancing, stepping, and whole-body control considerations [1]–[3], they have many advantages. They do not require flat floors to operate, are capable of bi-manual dexterous manipulation, and can extend their manipulation workspace by bending forward or crouching. Achieving such behaviors is challenging, as the control schemes must prevent collisions with the environment and their bodies (see Fig. 1). Robots with back-drivable joints [4] or equipped with soft skin [5], may allow self-collisions or exploit self-contacts and compliance to achieve a goal. However, for robots not equipped with such capabilities, self-collision prevention is necessary to avoid hardware damage.

In this paper, we focus on providing a real-time solution to the self-collision avoidance (SCA) problem for high-dimensional position-controlled humanoid robots. In the literature, the SCA problem is often tackled with *offline motion planning* algorithms that generate optimal collision-free paths in configuration space for robots in challenging and constrained environments [6]–[12]. Yet, as the robot's joint space dimensionality increases, so does the computational and geometrical complexity of the problem. While these specific methods have been capable of generating collision-free trajectories for high-DoF humanoid robots that can be dynamically-stable, efficient and optimal [8]–[10], their main shortcoming is computation time. It may take seconds to minutes to generate a single collision-free trajectory [9], [10]. Hence, their applicability to reactive and dynamic real-time control is limited.

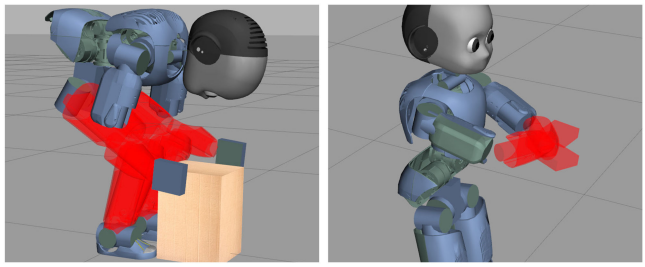


Fig. 1. Self-collided postures example. Links in the collided state are highlighted red. On the left – arms collide with legs in the crouching scenario, on the right – self-collisions between the arms.

To alleviate this, *online reactive approaches* for high-dimensional humanoid robots have been explored [13]–[20]. These approaches rely on distance computations between simplified geometrical representations of segments of the robot's body; i.e. spheres, swept-sphere volumes, capsules, convex-hulls, patch-based bounding volumes, etc. Self-collisions are avoided by using the distance functions as "repulsive potential" functions transformed to joint velocities for Jacobian-based inverse kinematics (IK) [13], [15], [17] and to joint torques for torque control schemes [14], [18]; or as constraints in optimization-based IK solvers [16], [19], [20].

Although providing better computational efficiency than the *offline* methods, this family of approaches have several issues. First, the accuracy and computation time of collision avoidance relies heavily on the precision of the geometric representation. A highly accurate representation of the segments will yield highly accurate distance functions but are more computationally taxing; and vice-versa. Further, the former approaches are susceptible to numerical issues induced by Jacobian inversions and are not robust to local minima [13]–[15], [17], [18]. This is handled by the latter approaches in which inequality constraints are defined as the distances between closest points on the robot body. Yet, in

Manuscript received October 15, 2020; accepted January 23, 2021. Date of publication February 3, 2021; date of current version February 18, 2021. This letter was recommended for publication by Associate Editor R. Cisneros Limon and Editor A. Kheddar upon evaluation of the reviewers' comments. This work was supported by the European Research Council (SAHR, Project ID: 741945). *(Corresponding author: Mikhail Koptev.)*

Mikhail Koptev and Aude Billard are with the Ecole Polytechnique Federale de Lausanne, Lausanne 1015, Switzerland (e-mail: mikhail.koptev@epfl.ch; aude.billard@epfl.ch).

Nadia Figueroa is with the Massachusetts Institute of Technology, Cambridge, Massachusetts 02139 USA (e-mail: nadiafig@mit.edu).

This letter has supplementary downloadable material available at https://doi.org/10.1109/LRA.2021.3057024, provided by the authors.

Digital Object Identifier 10.1109/LRA.2021.3057024

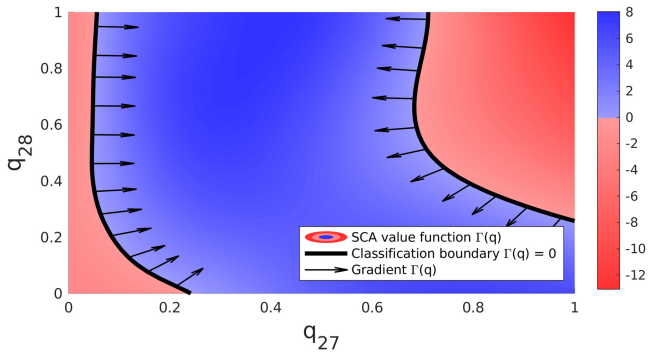


Fig. 2. SCA value function $\Gamma(\boldsymbol{q})$ projected to two selected dimensions. Red areas denote negative class ($\Gamma(\boldsymbol{q}) < 0$) and collided configurations, whereas blue area represent feasible configurations ($\Gamma(\boldsymbol{q}) > 0$). Arrows inside collision-free region stand for gradient $\nabla\Gamma(\boldsymbol{q})$ that is pointing in the direction away from collisions.

order for the optimizations to converge, the distance functions should be continuously differentiable [16]. This can be difficult to achieve. A distance function between two geometrical objects is continuously differentiable only if one object is convex and the other strictly convex [19]. To guarantee convergence, [16], [19] and [20] have proposed novel geometrical representation and distance functions that ensure differentiability. However, they rely on convex geometrical representations and approximations which can over-constrain the problem; i.e. SCA might be too conservative.

Insights from previous work on SCA for multiple fixed based robot arms [21], [22] show that self-collision regions of a multi-body robot are static and unique in joint space. Hence, a continuously differentiable function can be learned to model the self-collision boundary between collided and free joint configurations (see Fig. 2). In [21], [22] a sparse support vector machine (SVM) formulation [23] was used for fast ($\sim$2 ms) and accurate ($\sim$98%) collision detection and repulsive gradient computation. The SCA boundary functions and gradients were re-formulated as inequality constraints in a quadratic program (QP)-based optimization problem that is solved in real-time. This approach was validated on a 14-DoF dual-arm setup, however, it's scalability to higher-dimensional humanoid robots was not explored. Following this work, we propose to learn continuously differentiable boundary functions that can be approximated in real-time to prevent self-collisions while controlling the 29 DoF iCub humanoid.

Self-collision boundary functions can be learned by sampling the robot's workspace for collided and non-collided joint space configurations. With a sufficiently representative dataset, several function approximation approaches could learn the boundary functions with high prediction accuracy. Since the dataset is collected offline, we can use precise triangle mesh representation of the iCub robot body to acquire a highly accurate sampling of the boundary. The challenge we face is that such datasets are in the order of millions. Further, we seek to provide high SCA prediction accuracy and computationally efficiency while ensuring continuity requirements. We offer the former by decomposing the robot's 29-dimensional joint space into independent submodels of lower dimensionality, each covering a sub-combination of limbs and bodies. To learn the boundary functions, we explore and compare the sparse SVM used in [21], [22] known as Cutting-Plane Subspace Pursuit (CPSP) algorithm [23], GPU-accelerated feed-forward neural networks (NN) and GPU-accelerated SVM implementations.

Paper organization: In Section II we summarize related works that learn SCA boundaries. Sections III-IV describe the proposed dataset building technique and the analysis of boundary function approximation approaches. Section V describes the proposed IK optimization problem with the learned boundary functions as constraints. In Section VI,we show that our approach yields highly accurate real-time SCA for the 29 DoF iCub [24].

II. Related Works

Similar to our previous work [21], [22], the approach presented in [25] uses a SVM to detect self-collisions of the 31-DoF WALKMAN humanoid robot [25]. In this work, the SVM is used as a prediction tool to accelerate the online computation of the pairs of closest points. Given these predictions, the actual Euclidean distances are used to formulate constraints fed to a task-prioritized optimization problem as in [16]. While this approach is computationally efficient (10–20 ms), the misclassification rates of the learned SVMs are high and are dealt with an additional heuristic to improve prediction accuracy. Further, this approach may suffer from convergence issues due to the nondifferentiability of the constraints.

The idea of learning a model of a robot's workspace with SVM and using it to detect collision is also explored in [26]. An active learning scheme and kernel approximation are leveraged to efficiently detect collisions. They accelerate offline motion planning approaches such as rapidly exploring random trees (RRT) [27] variants to orders of magnitude above the state-of-the-art. This work clearly demonstrates the computational advantage of using a learned collision function over classical pairwise distance searches. However, its scalability to high-dimensional humanoid robots and applicability to real-time IK solvers is yet to be demonstrated.

Finally, none of the aforementioned works use the learned SCA models to directly formulate the avoidance constraints. In this work, we show that the learned SCA models can be used to actively avoid self-collisions within a *real-time* IK solver ($<$ 5 ms computation time).

III. Problem Formulation

We seek to learn an SCA model for the iCub humanoid robot [24] in joint space to predict and prevent self-collisions. Our motivation is to avoid repetitive geometrical sampling and minimal distances computations by learning a continuous and continuously differentiable function $\Gamma(\boldsymbol{q}) : \mathbb{R}^n \rightarrow \mathbb{R}$ from a dataset of sampled postures $\boldsymbol{q} \in \mathbb{R}^n$. This function should represent the cost for a self-collision constraint, describing collided and free robot configurations. While $\Gamma(\boldsymbol{q}) \in \mathbb{R}$ can help determine how close a given posture is to a collided state, the gradient of this function, $\nabla\Gamma(\boldsymbol{q}) \in \mathbb{R}^n$, can be used to navigate existing path-planning methods away from the collision border, see Fig. 2.

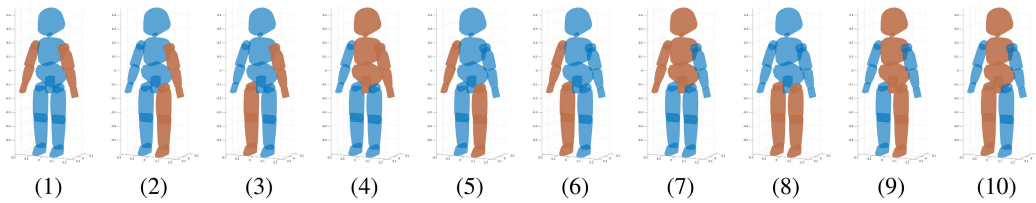


Fig. 3. Visualization of humanoid self-collision model separation into ten independent submodels. For each submodel 1–10 orange links are checked for collisions, while blue links' collisions are ignored. Neighbouring links collisions are also ignored. The features of each submodel correspond to the joints adjacent to highlighted links. For models 2, 3, 5 and 6 torso joints are also considered as features.

TABLE I
HUMANOID ROBOT MODEL SEPARATION (1–10). ENTRIES MARKED WITH (^{*x}) ARE SYMMETRICAL TO UNMARKED MODELS x. FULL-BODY IS PRESENT AS ZERO ENTRY FOR COMPARISON. VISUALISATION IS PRESENTED IN FIG. 3

N	Collided Links	Varied Joints	Features
0	Full body	Full body	29
1	Left arm, right arm	Left arm, right arm	14
2	Left arm, left leg	Left arm, left leg, torso	16
3	Left arm, right leg	Left arm, right leg, torso	16
4	Left arm, torso, head	Left arm, torso	10
5^{*3}	Right arm, left leg	Right arm, left leg, torso	16
6^{*2}	Right arm, right leg	Right arm, right leg, torso	16
7^{*4}	Right arm, torso, head	Right arm, torso	10
8	Left leg, right leg	Left leg, right leg	12
9	Left leg, torso, head	Left leg, torso	9
10^{*9}	Right leg, torso, head	Right leg, torso	9

To learn $\Gamma(.)$, any machine learning function approximation technique could in principle be used. Here, we set to compare three techniques: SVM, sparse SVM (CPSP), NN. In the case of SVM, the number of parameters (support vectors) grows linearly with the number of datapoints [28]. Hence, we evaluate the sparse SVM version (CPSP) [23]. We then contrast this to Neural Networks with a focus on finding the minimum number of neurons to achieve similar accuracy. We show that the resulting constraints from any of these techniques are continuously differentiable.

IV. SELF-COLLISION BOUNDARY LEARNING

This section describes our approach applied to the iCub humanoid robot, yet it is applicable to any humanoid.

A. Self-Collision Dataset

To learn a self-collision detection function, we generate a dataset containing examples of both collided and collision-free classes. The iCub has 53 actuated joints [24]. Excluding hands, eyes and the 3 joints associated with neck and head movements[1] reduces the number of actuated joints to 29. The resulting joint space of the iCub is equivalent to $\mathbb{R}^{29}$, as each joint q^i is revolute and has corresponding joint limits:

$$q_i^- < q_i < q_i^+, i = 1, \ldots, 29.$$

As noted by [25] and evidenced in our experiments, due to the complex joint space geometry, learning a single SCA model for a high-dimensional humanoid robot that is *both highly accurate and computationally efficient* is challenging. In this work, we simplify the problem by decomposing the 29 DoF humanoid model into ten submodels of lower dimensionality, that independently describe possible collisions between robot links. The links, joints and feature dimensionality for every submodel are specified in Table I and Fig. 3. Notably, certain pairs of submodels become symmetric if the torso joints are mirrored. By exploiting this symmetry, we reduce the number of models *to be learned* from ten to six. We indicate the symmetric submodel pairs in Table I.

While many works rely on geometrical simplifications of robot structure, we use precise triangle meshes. To detect collisions we use the Flexible Collision Library (FCL) [29]. The dataset is generated by a uniform random sampling with balancing heuristics. The dataset creation is structured so that 50% of samples are collided postures, 35% are feasible postures with a minimal distance across link pairs lower than a threshold (to have a better representation of collision boundary), and 15% of entries are uncollided postures without minimal distance requirements. For every submodel, we randomly sample robot configurations within the joint limits until the desired dataset size (for collided, within the threshold, and free postures) is reached.

The threshold value for 35% of close-to-collision configurations is set to 5 cm. Postures are considered collided if the minimal distance between any link pair is below 1 cm. Using joint limits q_i^-, q_i^+ all configurations are normalized, i.e. every feature belongs to (0, 1). As a result, for each submodel, we generate a *normalized* and *balanced* dataset, where 50% of samples are of collided configuration, and 50% are uncollided configurations including safe and close-to-collision postures. This procedure allows us to generate unlimited amounts of data, and datasets sizes are only constrained by the computation time.

B. Self-Collision Boundary Learning Via SVM

To learn a self-collision boundary from the collected data, we first follow the classic SVM formulation, where the kernel trick is employed to lift the data to higher dimension space and find separation hyperplane in feature space. We use the radial basis function (RBF) kernel $K(\boldsymbol{q}_1, \boldsymbol{q}_2) = e^{-\gamma||\boldsymbol{q}_1 - \boldsymbol{q}_2||^2}$, where parameter γ defines kernel width. For a given robot configuration $\boldsymbol{q}$, the SVM decision function $\Gamma(\boldsymbol{q})$ has the following form:

$$\begin{aligned}\Gamma(\boldsymbol{q}) &= \sum_{i=1}^{N_{sv}} \alpha_i y_i K(\boldsymbol{q}, \boldsymbol{q}_i) + b \\ &= \sum_{i=1}^{N_{sv}} \alpha_i y_i e^{-\gamma||\boldsymbol{q} - \boldsymbol{q}_i||^2} + b,\end{aligned} \tag{1}$$

[1]The collision space with the head moving is quasi equivalent to the collision space with the head static.

TABLE II
COMPARISON FOR VARIOUS LEARNING METHODS USED TO LEARN SELF-COLLISION DETECTION FUNCTION. ALL PARAMETERS REPRESENT SUM OR AVERAGE VALUE FOR SUCH PARAMETERS ACROSS TEN SUBMODELS (INCL. SYMMETRICAL) FROM TABLE I

Learning Method	SVM	CPSP	NN
Model size, # of doubles	2099422	138000	**15322**
Total training time, s	11250	4762	**2106**
Avg. accuracy	0.960	0.920	**0.990**
Avg. TPR	0.949	0.878	**0.984**
Avg. TNR	0.973	0.964	**0.995**
Evaluation time, ms	1.38(GPU)	0.42(CPU)	**0.11**(CPU)

TABLE III
SELF-COLLISION AVOIDANCE CONSTRAINTS PERFORMANCE FOR 1000 RANDOM OBJECT POSITIONS

IK method	r_c, %	d, cm	IK time, ms
No SCA (15 iter.)	63.26	6.56	3.41
Proposed method (15 iter.)	**4.86**	**5.78**	**3.61**
Jacobian-based SCA (15 iter.)	14.52	7.68	15.02
Jacobian-based SCA (30 iter.)	7.74	5.69	29.14

and the equation for $\nabla\Gamma$ is naturally derived as follows:

$$\nabla\Gamma(\boldsymbol{q}) = \sum_{i=1}^{N_{sv}} \alpha_i y_i \frac{\partial K(\boldsymbol{q}, \boldsymbol{q}_i)}{\partial \boldsymbol{q}} = \\ = -2\gamma \sum_{i=1}^{N_{sv}} \alpha_i y_i e^{-\gamma \|\boldsymbol{q}-\boldsymbol{q}_i\|^2} (\boldsymbol{q} - \boldsymbol{q}_i). \tag{2}$$

In (1), $\boldsymbol{q}_i$ $(i = 1, \ldots, N_{sv})$ are the support vectors from the training dataset, y_i are corresponding collision labels (-1 if posture is collided, $+1$ otherwise), $0 \leq \alpha_i \leq C$ are the weights for support vectors and $b \in \mathbb{R}$ is decision rule bias. Parameter $C \in \mathbb{R}$ is a penalty factor used to trade-off between errors minimization and margin maximization. Parameters α_i and b and the support vectors $\boldsymbol{q}_i$ are estimated by solving the optimization problem for the soft-margin kernel SVM [28]. For efficient learning of the SVM model from the sampled data, we use ThunderSVM – an open-source library that utilizes GPUs to speed-up the computations [30].

For each of the submodels (Table I) we must generate independent datasets and SVMs. Hence, the optimal hyperparameters C and γ are different for each submodel. To find C and γ we perform the procedure of grid-search in the following ranges: $C = [1, 5000]$ and $\gamma = [0.1, 10]$ for each model. For the grid-search, we use training and testing datasets with sizes of 100000 and 200000, respectively. This scheme offers faster parameter optimization with similar results to k-fold cross-validation. The grid-search results rely heavily on the sampling density and size of the cells in the grid. However, as we have multiple models to optimize, dense sampling on a large range would require excessive computations. Given that, for each model, we first localize optimal areas by performing two searches with the 5×5 grids and finally tune precisely with the 10×10 grid. Optimal hyperparameters for submodels are provided in Table IV.

After parameters C and γ are found for every submodel, we use 250000 sampled postures to train each SVM. We then estimate classification accuracy and the confusion matrix with another 150000 points previously unseen by the model. Performance validation is achieved by repeating the training five times, each time randomly selecting points for train and test datasets. To correctly classify collided and non-collided postures, we do not need to densely cover all state-space, but rather have enough points in the classification boundary's vicinity. As our sampling procedure is built to include close-to-collision configurations, we find that 250000 points are enough to learn a model with decent performance.

To evaluate the performance of the trained model, we use metrics such as classification *Accuracy* (A), *True Negative Rate* (TNR) and *True Positive Rate* (TPR). Those are computed using the elements of the confusion matrix:

$$\mathrm{TNR} = \frac{\mathrm{TN}}{(\mathrm{TN} + \mathrm{FP})}, \ \mathrm{TPR} = \frac{\mathrm{TP}}{(\mathrm{TP} + \mathrm{FN})}, \tag{3}$$

where TP stands for *True Positive*, TN – for *True Negative*, FP – for *False Positive* and FN – for *False Negative*. Negative class here is for the collided postures, and positive class means the robot configuration has no self-collisions. A high TPR (low FN) avoids classifying free configurations as collided and does not remove valid configurations from feasible space. On the other hand, a high TNR (low FP), will avoid labeling collided configurations as free. This is the most critical metric to be maximized, as we wish to avoid classifying self-collided configuration as feasible.

1) SVM Performance Evaluation: The trained models performance is provided in Table IV. Submodels distinguish collided and uncollided configurations with an average accuracy of 96.0%. TNR is above 94.5% for every submodel, while lowest value for TPR is 91%. At the same time, the full-body model with 29 DoF has relatively low accuracy (∼86%) even with a training dataset of one million entries.

Learned SVM models have total support vector count $N_{sv} = 236797$ (incl. symmetrical). To evaluate each model for a single query point, it is required to compute the squared euclidean distance from each support vector and calculate the sum of resulting RBF kernels (as in (1) and (2)). By means of optimized C and CUDA code, we can perform all computations efficiently and fast enough to be used online. The computations are easy to parallelize, so we compare the performance of single- and multithreaded CPU implementation, as well as GPU variant. The average time needed to evaluate one random query posture in three various ways – CPU sequential, CPU parallel, GPU – is 7.74 ms, 2.55 ms, and 1.38 ms respectively.

2) Sparse SVM (CPSP) Performance Evaluation: The previous sub-section demonstrated that it is possible to learn the self-collision boundary in the robot's 29-dimensional joint space with kernel-based SVM. We also showed that, in order to achieve real-time performance, it is required to utilize high-performance CPU or even GPU. Often, the computational capabilities of robots' onboard computers are limited, and we cannot assume access to GPU. It is hence important to investigate other learning methods, which may result in self-collision models with fewer parameters. We follow our previous approach [22] and apply the CPSP method [23] that reformulates the SVM optimization problem to put a strict upper bound on the number of support vectors. A key difference between CPSP and classical SVM is that support vectors $\boldsymbol{q}_i$ are not necessarily points that were part

TABLE IV
PERFORMANCE AND CONFUSION MATRIX ELEMENTS FOR SELF-COLLISION MODELS TRAINED WITH CLASSICAL KERNEL SVM. EACH SUBMODEL IS TRAINED ON 250000 POINTS DATASET AND TESTED ON 150000 UNSEEN DATA POINTS. FULL MODEL (0) IS TRAINED ON 1000000 (ONE MILLION) DATA SAMPLES. THE ELEMENTS OF THE CONFUSION MATRIX ARE PRESENTED RELATIVELY TO TESTING DATASET SIZE, AND IN THE PERFECT SCENARIO $\mathrm{TP} = \mathrm{TN} = 0.5$, WHILE $\mathrm{FP} = \mathrm{FN} = 0$. ALL VALUES (EXCEPT FOR COLUMNS 1-4) ARE AVERAGED BETWEEN FIVE TRAINING-VALIDATION RUNS. STANDARD DEVIATIONS σ FOR EACH CELL IN COLUMNS 7-14 DO NOT EXCEED 0.001

N	Dim	C	γ	N_{sv}	Training time, s	Training accuracy	Test accuracy	TPR	TNR	TN	TP	FN	FP
0	29	100	0.1	182548	15632	0.891	0.864	0.855	0.875	0.438	0.426	0.074	0.062
1	14	2000	0.5	25767	2781	0.989	0.943	0.936	0.951	0.476	0.468	0.032	0.024
2 and 6	16	400	0.4	42923	1756	0.976	0.926	0.910	0.945	0.473	0.453	0.047	0.027
3 and 5	16	1000	0.4	32788	2962	0.977	0.942	0.925	0.962	0.481	0.461	0.039	0.019
4 and 7	10	100	1.5	15886	287	0.999	0.979	0.971	0.987	0.494	0.486	0.014	0.006
8	12	5000	0.5	13988	3418	0.991	0.976	0.971	0.982	0.491	0.485	0.015	0.009
9 and 10	9	500	0.5	6924	46	0.992	0.991	0.985	0.997	0.499	0.493	0.007	0.001

TABLE V
PERFORMANCE AND CONFUSION MATRIX ELEMENTS FOR SELF-COLLISION MODELS TRAINED WITH CPSP METHOD. EACH SUBMODEL IS TRAINED ON 250000 POINTS DATASET AND TESTED ON 150000 UNSEEN DATA POINTS. FULL MODEL (0) IS TRAINED ON 1000000 (ONE MILLION) DATA SAMPLES. ALL VALUES (EXCEPT FOR COLUMNS 1-5) ARE AVERAGED BETWEEN FIVE TRAINING-VALIDATION RUNS. STANDARD DEVIATIONS σ FOR EACH CELL IN COLUMNS 7-14 DO NOT EXCEED 0.002

N	Dim	C	γ	N_{sv}	Training time, s	Training accuracy	Test accuracy	TPR	TNR	TN	TP	FN	FP
0	29	100	0.1	10000	117399	0.782	0.746	0.901	0.591	0.296	0.451	0.049	0.204
1	14	2000	0.5	1000	1052	0.916	0.906	0.869	0.943	0.476	0.434	0.066	0.028
2 and 6	16	400	0.4	1000	926	0.888	0.868	0.827	0.908	0.473	0.414	0.086	0.046
3 and 5	16	1000	0.4	1000	851	0.918	0.900	0.832	0.967	0.481	0.416	0.084	0.016
4 and 7	10	100	1.5	1000	676	0.938	0.936	0.891	0.981	0.494	0.446	0.054	0.009
8	12	5000	0.5	1000	826	0.965	0.964	0.939	0.989	0.491	0.470	0.030	0.006
9 and 10	9	500	0.5	1000	431	0.976	0.976	0.960	0.992	0.499	0.480	0.020	0.004

of the training dataset. However, the final expression for the functions $\Gamma(\boldsymbol{q}), \nabla\Gamma(\boldsymbol{q})$ ((1) and (2)), and the hyperparameters C and γ correspond to those of classical SVM. As in the SVM model learning, we repeat training-validation process five times, shuffling data between training and testing datasets.

To learn the sparse SVMs with CPSP optimization we use the SVM$^{\text{perf}}$ library [23]. We set the support vector budget to $N_{\max} = 1000$ for submodels, and $N_{\max} = 10000$ for the single full-dimensional self-collision model (for evaluation purposes). The resulting accuracy for self-collision detection in submodels is 92.1% on average. FP do not surpass 4.6%, but FN reach 10% for some models. Full metrics for the performance of learned models are provided in Table V. The sparse SVM method results in fast optimization and a reduced number of model parameters, yet with lower prediction accuracy compared to the classical SVM. This could be alleviated by setting a higher $N_{\max}$, however, at the expense of increased model complexity and its computation time.

C. Self-Collision Boundary Learning Via Neural Networks

Since we learn the SCA boundary functions as a classification problem, we also explore the use of Neural Networks (NN). By altering the depth (number of layers) and width (maximal number of nodes in a layer) of a NN, we can control the number of weight coefficients in the model, therefore controlling its size and evaluation time. The difficulty, however, lies in finding the optimal depth and width that will yield the best trade-off between *computational efficiency* and *prediction accuracy*. To ensure the necessary continuity properties for $\Gamma(.)$; i.e., that it should be $\mathcal{C}^1$, we can use continuous activation functions, such as sigmoid or hyperbolic tangent. Through empirical evaluation we found

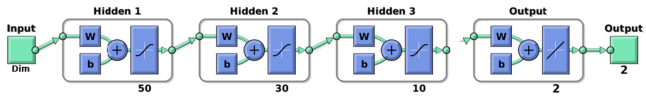


Fig. 4. NN layers used to learn self-collision boundary function.

that a NN with $50 - 30 - 10$ hidden layer structure and $\tanh$ activation function (see Fig. 4) is the optimal architecture to learn submodels. Several network architectures with various depths and widths were evaluated. The proposed architecture is the most computationally efficient that provides better prediction accuracy than the SVM counterparts. The NN parameters are weight matrices $\boldsymbol{W}_i$ and bias vectors $\boldsymbol{b}_i$, $i = 1, \ldots, 4$, their dimensions vary depending on the amount of neurons on i-th layer and dimensionality of state $\boldsymbol{q}$ across various submodels. For a given posture $\boldsymbol{q}$, the NN outputs a pair of real numbers (p_1, p_2):

$$[p_1(\boldsymbol{q}), p_2(\boldsymbol{q})]^T = \boldsymbol{b}_4 + \boldsymbol{W}_4 \tanh\left(\boldsymbol{b}_3 + \boldsymbol{W}_3 \cdot \tanh\left(\boldsymbol{b}_2 + \boldsymbol{W}_2 \tanh\left(\boldsymbol{b}_1 + \boldsymbol{W}_1 \boldsymbol{q}\right)\right)\right), \tag{4}$$

that represent respective probabilities of belonging to two classes (free or collided) after normalization. As we are interested in a single-valued output for our self-collision boundary function, we set $\Gamma(\boldsymbol{q}) = p_2 - p_1$, so that $\Gamma(\boldsymbol{q}) = 0$ when $p_1 = p_2$, this i.e. posture belongs to the self-collision boundary. For collided postures $\Gamma(\boldsymbol{q}) < 0$, and for feasible postures $\Gamma(\boldsymbol{q}) > 0$, as in previously learned SVM models. Without affecting boundary detection, we may also skip the normalization step, so that $\Gamma(\boldsymbol{q})$ is not bounded. The gradient of SCA function is derived as

TABLE VI
PERFORMANCE AND CONFUSION MATRIX ELEMENTS FOR SELF-COLLISION MODELS TRAINED WITH NNS. SUBMODELS ARE TRAINED ON 900000 POINTS DATASET AND TESTED ON 100000 UNSEEN DATA POINTS. FULL MODEL (0) IS TRAINED ON 2500000 POINTS, AND HAS THE SAME FEED-FORWARD ARCHITECTURE WITH THREE HIDDEN LAYERS, BUT WITH 250 NEURONS ON EACH LAYER. ALL VALUES (EXCEPT FOR COLUMNS 1-3) ARE AVERAGED BETWEEN FIVE TRAINING-VALIDATION RUNS. STANDARD DEVIATIONS σ FOR EACH CELL IN COLUMNS 5-12 DO NOT EXCEED 0.001

N	Dim	N_w	Training time, s	Training accuracy	Test accuracy	TPR	TNR	TN	TP	FN	FP
0	29	132750	2289	0.925	0.923	0.933	0.914	0.456	0.468	0.034	0.043
1	14	2612	348	0.999	0.998	0.997	1.000	0.500	0.499	0.002	0.000
2 and 6	16	2712	354	0.971	0.971	0.958	0.985	0.492	0.479	0.021	0.008
3 and 5	16	2712	350	0.989	0.988	0.980	0.996	0.498	0.490	0.010	0.002
4 and 7	10	2412	359	0.993	0.993	0.989	0.998	0.499	0.495	0.005	0.001
8	12	2512	343	0.994	0.994	0.989	0.998	0.499	0.495	0.006	0.001
9 and 10	9	2362	352	0.998	0.998	0.996	0.999	0.500	0.498	0.002	0.000

$$\nabla\Gamma(\boldsymbol{q}) = \frac{\partial p_2}{\partial \boldsymbol{q}} - \frac{\partial p_1}{\partial \boldsymbol{q}}, \text{ where}$$

$$\left[\frac{\partial p_1(\boldsymbol{q})}{\partial \boldsymbol{q}}, \frac{\partial p_2(\boldsymbol{q})}{\partial \boldsymbol{q}}\right]^T$$
$$= -\boldsymbol{W}_1\boldsymbol{W}_2\boldsymbol{W}_3\boldsymbol{W}_4 \left(\tanh\left(\boldsymbol{b}_1 + \boldsymbol{W}_1\boldsymbol{q}\right)^2 - 1\right)$$
$$\cdot \left(\tanh\left(\boldsymbol{b}_2 + \boldsymbol{W}_2 \tanh\left(\boldsymbol{b}_1 + \boldsymbol{W}_1\boldsymbol{q}\right)\right)^2 - 1\right)$$
$$\cdot \left(\tanh\left(\boldsymbol{b}_3 + \boldsymbol{W}_3 \tanh\left(\boldsymbol{b}_2 + \boldsymbol{W}_2 \tanh\left(\boldsymbol{b}_1 + \boldsymbol{W}_1\boldsymbol{q}\right)\right)\right)^2 - 1\right). \quad (5)$$

In (4) and (5) the hyperbolic tangent functions are applied element-wise for a vector input. We use the PyTorch library to learn the weights of the NNs by optimizing the negative log likelihood loss with RMSProp running for 2000 epochs. We train six NNs to cover self-collisions for all submodels from Table I. As training time is significantly lower than in SVM models, and the model size does not increase when a larger dataset of training points is used, we train each submodel with 900000 postures to achieve better accuracy, and validate using 100000 unseen points. We repeat train-test procedure five times, shuffling data between training and testing datasets. For the trained models, the mean accuracy is 99.0%, and the FNR is 0.08% on average. Full results for training and validation are provided in Table VI. NNs outperform SVMs in terms of accuracy, learning speed and evaluation time. Computation of values and gradients of function $\Gamma(.)$ learned via NNs requires $\sim$0.11 ms on 4.2 GHz Intel i7-7700 K. This includes 15 consecutive evaluations of ten submodels from Table I (our IK algorithm requires at least 15 iterations to converge). We can compare our method with [20], where the Vector Field Inequalities approach is used to compute the distance function and its gradient to avoid collisions. In their approach, the single evaluation of Jacobian constraint takes $\sim$5.2 ms with comparable CPU.

One might wonder if a single large NN is capable of modeling the SCA boundary as accurately as our decomposed SCA-NN approach. In the first row of Table VI we include the evaluation metrics of a single 29-dimensional NN model without submodel decomposition. To find the optimal NN structure we performed grid-search on combinations of depth and width sizes, with ranges of 3-9 hidden layers and 50-250 neurons per layer. By training the networks on a larger dataset (2.5 million) and running it for 5000 epochs, we found a 4-layer (including output) NN with 250 neurons on each layer to be the optimal choice.

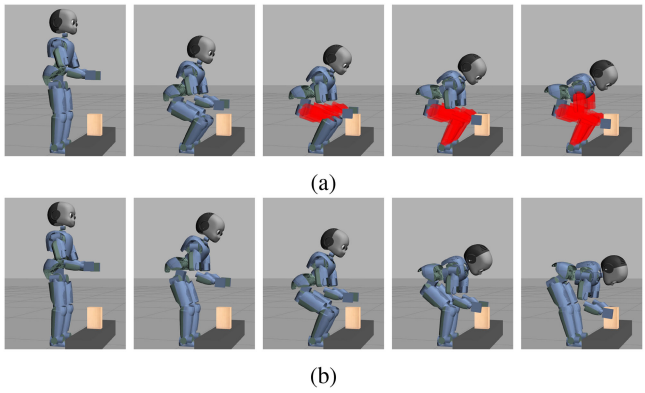


Fig. 5. The robot is trying the object close to the ground without (a) and with (b) proposed SCA constraints in the IK solver. Collided links are highlighted red. Refer to https://youtu.be/u3lTwFZFicY for the full experiment.

However, the prediction accuracy of this model reaches at most 92–93%, which is not sufficient to justify the increase in model complexity ($\sim$130 k weights) and computation time.

D. Comparison Across Methods

In Tables IV, V and VI we provide all the performance metrics for the three approaches, respectively. Further quantitative comparison of performance across is shown in Table II. As expected, SVM provides better average accuracy as well as better TPR and TNR than CPSP, whereas CPSP model is significantly smaller and offers faster evaluation. NN gives the overall best performance with even fewer parameters and better accuracy, as well as better TPR and TNR and the best training and evaluation time. Of course, such comparison depends heavily on the efforts spent on programming the techniques.[2] To provide a fair comparison, we used the most recent and efficient implementations offered for SVM (ThunderSVM), CPSP (SVM^{perf}), and NN (PyTorch). In the simulations and experiments reported next, we used the function $\Gamma(.)$ learned via NNs, as it offers the best combination of classification accuracy and evaluation times for querying postures.

V. APPLICATION TO ONLINE IK SOLVER

We propose to use the learned self-collision models as constraints for an online Inverse Kinematics solver. Taking

[2]The source code used in the paper is available at https://github.com/epfl-lasa/Joint-Space-SCA

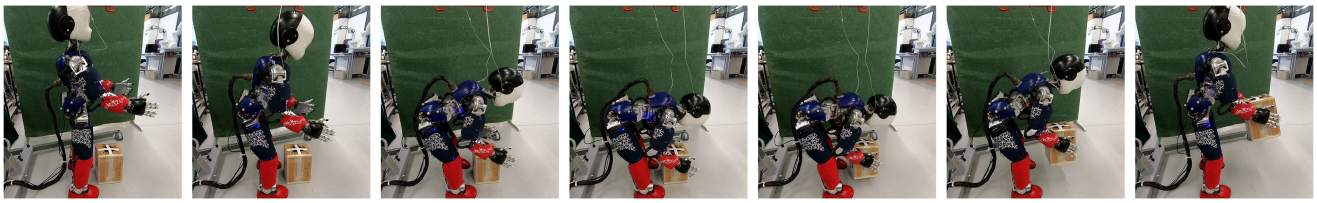


Fig. 6. iCub is picking the box from the ground with SCA constraints proposed in the paper. Refer to https://youtu.be/u3lTwFZFicY for the full experiment.

inspiration from the work [21], we model the self-collision boundary as a separation hyperplane and use it as a constraint for convex QP problem that can be solved in real-time. Compared to [21], our contribution is that we extend the method to the humanoid robot and demonstrate multiple collision constraints. We propose the following quadratic problem to solve the IK (adapted from [3] and [21]):

$$\min_{\Delta \boldsymbol{q}, \boldsymbol{\delta}} \boldsymbol{\delta}^T \boldsymbol{Q} \boldsymbol{\delta} + \Delta \boldsymbol{q}^T \boldsymbol{R} \Delta \boldsymbol{q}$$

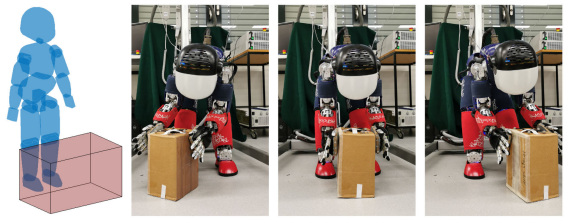
$$\text{s.t.} \quad \begin{cases} f(\boldsymbol{q}) + \dfrac{\partial f(\boldsymbol{q})}{\partial \boldsymbol{q}} \Delta \boldsymbol{q} = \boldsymbol{x} + \boldsymbol{\delta} \\ q_i^- < q_i + \Delta q_i < q_i^+, \quad i = 1..29 \\ -\nabla \Gamma_i(\boldsymbol{q}) \Delta \boldsymbol{q} \leq \ln(\Gamma_i(\boldsymbol{q}) + 1), \quad i = 1..10. \end{cases} \tag{6}$$

where $f(\boldsymbol{q})$ stands for the forward kinematics of the robot, $\boldsymbol{R}$ is equivalent to the well-known damping term in least-squares IK methods, and $\boldsymbol{Q}$ is a diagonal matrix that sets weights for the Cartesian tasks vector $\boldsymbol{x}$. Tasks include positions and orientations of the feet, hands, and the center of mass. Constraints in (6) represent a cartesian task $\boldsymbol{x}$ with slack $\boldsymbol{\delta}$, joint limits, and self-collision avoidance respectively. The last constraint forces the joint angles to move away from the self-collisions hyperplane as they approach it. When the robot is far from the boundary and $\Gamma_i(\boldsymbol{q}) > 0$, values $\ln(\Gamma_i(\boldsymbol{q}) + 1)$ are positive, which relaxes the set of self-collision inequality constraints, i.e., the robot accurately follows the desired end-effector trajectory. If i-th submodel is close to self-collision, then $\ln(\Gamma_i(\boldsymbol{q}) + 1)$ becomes negative, and the IK solver is forced to align joint motion $\Delta \boldsymbol{q}$ with corresponding gradient $\nabla \Gamma_i(\boldsymbol{q})$, thus moving the robot configuration away from the collision boundary and satisfying self-collision avoidance constraint. Recall that, due to symmetry, only six submodels (out of ten) are learned. To calculate Γ_i for symmetrical submodels ($i = 5, 6, 7, 10$), we mirror the torso pitch and yaw joints: $q_t^p \rightarrow 1 - q_t^p$ and $q_t^y \rightarrow 1 - q_t^y$, and invert the sign for the corresponding coordinates of $\nabla \Gamma_i$.

Equation (6) is a convex QP problem with equality and inequality constraints, hence, there is no closed-form solution for it. To solve it, we utilize CVXGEN solver, which generates C codes tailored for the specific formulation of the optimization problem [31]. The proposed control scheme is for the position-controlled robots, and after solving (6) the desired joint angles for the next control iteration are computed as $\boldsymbol{q}_{new} = \boldsymbol{q} + \Delta \boldsymbol{q}$.

VI. Experimental Validation

In our experiment, we consider the picking-up scenario. The robot is supposed to bend from the standing posture and reach the object located on the ground. Without SCA constraints in Inverse Kinematics, the robot tends to have a self-collision between the elbow and the knee (Fig. 5(a)). Not only this self-collision prevents the robot from reaching the target, but it is also harmful to the hardware, leading to torn cables and plates bending. However, with the proposed SCA constraints in the IK solver, the robot avoids that collision by shifting the hips and straightening the knees to maximize the distance between legs and arms links (Fig. 5(b)).



(a) (b) (c) (d)

Fig. 7. (a) - Area for random positions of the object in the benchmark. (b)–(d) - Grasping posture of the robot for various box positions.

Figs. 5(a) and 5(b) are from simulated environment, as it is easier to demonstrate self-collisions there. We also conducted the experiment with picking-up scenario on the real robot, as demonstrated in the video accompanying the paper. As expected, iCub is able to pick up arbitrary positioned box avoiding self-collisions. Snapshots of one full picking-up motion are demonstrated in Fig. 6, while grasping postures for three different box positions are shown in Figs. 7(b)–7(d).

To evaluate the proposed method's performance in a general picking-up scenario, we perform the following benchmark. The box object is placed randomly in front of the robot (red area in Fig. 7 a), and three different IK algorithms are used to generate a grasping posture from a fixed initial state. IK methods include (i) the proposed method (as in (6)), (ii) QP solver without SCA constraint ((6) without last line), and (iii) the QP solver where collision-avoidance is a task based on repulsion in Euclidean space. For (iii), the QP problem (6) is modified to additionally minimize $\boldsymbol{\sigma}^T \boldsymbol{S} \boldsymbol{\sigma}$, where $\boldsymbol{S}$ is a diagonal weight matrix, and $\boldsymbol{\sigma}$ is a slack variable:

$$J_i^{\text{SCA}}(\boldsymbol{q}) \Delta \boldsymbol{q} = s \boldsymbol{v}_i^{\text{rep}} + \boldsymbol{\sigma}. \tag{7}$$

$\boldsymbol{v}_i^{\text{rep}} \in \mathbb{R}^3$ is a vector connecting two closest points of the bodies in i-th submodel ($i = 1...10$) in task-space; i.e. $\boldsymbol{v}_i^{\text{rep}} = \frac{\boldsymbol{p}_1 - \boldsymbol{p}_2}{||\boldsymbol{p}_1 - \boldsymbol{p}_2||}$ where $\boldsymbol{p}_1, \boldsymbol{p}_2 \in \mathbb{R}^3$ are the closest points on the robot body. s is a scaling factor for $\boldsymbol{v}_i^{\text{rep}}$ to match the magnitude of $\Delta \boldsymbol{q}$. 7 replaces the proposed one (last line in (6)) with a traditional repulsive force and constraints approaches, similar to [13], [15], [17] and [16], [25], respectively. To speed-up the distance computations, the geometry is approximated with convex hulls

instead of precise meshes. We check the final posture for the collision and calculate the error $d = \frac{1}{2}(||p_l^d - p_l^s|| + ||p_r^d - p_r^s||)$ between the desired positions for left and right hands $p_{\{l,r\}}^d$, and positions $p_{\{l,r\}}^s$ given by the IK solvers. We run the simulation 1000 times and compare the collision rate r_c, average hands position error d, and evaluation time for IK; the results are provided in Table III. The collision rate r_c is lowest for the proposed method, while evaluation time is almost unaffected compared to basic IK without self-collision avoidance. At the same time, to achieve similar performance (in terms of mean task error d) with Jacobian-based SCA, it is necessary to perform more iterations; combined with repetitive distance calculations, that significantly slows down the computations.

VII. Conclusion

We propose a computationally efficient method to detect and avoid self-collision for high-DoF robots, such as humanoids. We treat self-collisions as constraints in the robot's joint space, allowing for direct use in the control scheme. Apart from the demonstrated use in IK solver, the proposed self-collision avoidance constraint can be used in other algorithms. For example, it may act as collision detection in RRT-navigation methods [6], additionally providing gradient to guide the sampling procedure. To some extent, it can replace distance field computations in [11]. Although our method may require extensive computations to learn a self-collision boundary, such computations are performed only once. Further, it provides a model for reliable and efficient self-collision detection, fitting the computation time into real-time limitations.

Our method's drawback is that it requires some heuristics, such as model separation, to reduce the overall dimensionality of a problem. However, the model separation strategy can be applied to any other humanoid robot. It is of interest though, to investigate how sophisticated sampling procedures (for example, balancing the number of collisions between every link pair) may help learn a single 29-dimensional model of feasible postures. Additionally, it is possible to include other constraints, such as static balance, in the classification, thus expanding unfeasible class.

References

[1] G. Nava *et al.*, "Modeling and control of humanoid robots in dynamic environments: ICub balancing on a seesaw," in *Proc. IEEE-RAS Int. Conf. Humanoid Robot.*, pp. 263–270.

[2] S. Faraji, S. Pouya, C. G. Atkeson, and A. J. Ijspeert, "Versatile and robust 3d walking with a simulated humanoid robot (Atlas): A. model predictive control approach," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2014, pp. 1943–1950.

[3] N. Figueroa, S. Faraji, M. Koptev, and A. Billard, "A dynamical system approach for adaptive grasping, navigation and co-manipulation with humanoid robots," in *Proc. IEEE Int. Conf. Robot. Automat*, 2020, pp. 7676–7682.

[4] A. Ajoudani *et al.*, "A manipulation framework for compliant humanoid coman: Application to a valve turning task," in *Proc. IEEE-RAS Int. Conf. Humanoid Robots*, 2014, pp. 664–670.

[5] J. R. Guadarrama-Olvera, E. Dean-Leon, F. Bergner, and G. Cheng, "Pressure-driven body compliance using robot skin," *IEEE Robot. Automat. Lett.*, vol. 4, no. 4, pp. 4418–4423, Oct. 2019.

[6] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Proc. IEEE Int. Conf. Robot. Automat.*, vol. 2, Apr. 2000, pp. 995–1001.

[7] L. E. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Automat.*, vol. 12, no. 4, pp. 566–580, Aug. 1996.

[8] D. Berenson, S. Srinivasa, and J. Kuffner, "Task space regions: A. framework for pose-constrained manipulation planning," *Int. J. Robot. Res.*, vol. 30, no. 12, pp. 1435–1460, 2011.

[9] J. J. Kuffner, S. Kagami, K. Nishiwaki, M. Inaba, and H. Inoue, "Dynamically-stable motion planning for humanoid robots," *Auton. Robots*, vol. 12, no. 1, pp. 105–118, Jan. 2002.

[10] R. Luna, M. Moll, J. Badger, and L. E. Kavraki, "A scalable motion planner for high-dimensional kinematic systems," *Int. J. Robot. Res.*, vol. 39, no. 4, pp. 361–388, 2020.

[11] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "CHOMP: Gradient optimization techniques for efficient motion planning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2009, pp. 4030–4035.

[12] J. Schulman *et al.*, "Motion planning with sequential convex optimization and convex collision checking," *Int. J. Robot. Res.*, vol. 33, no. 9, pp. 1251–1270, 2014.

[13] H. Sugiura, M. Gienger, H. Janssen, and C. Goerick, "Real-time self collision avoidance for humanoids by means of nullspace criteria and task intervals," in *Proc. 6th IEEE-RAS Int. Conf. Humanoid Robots*, Dec. 2006, pp. 575–580.

[14] 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 *Proc. IEEE/ASME Int. Conf. Adv. Intell. Mechatronics*, 2007, pp. 1–6.

[15] H. Sugiura, M. Gienger, H. Janssen, and C. Goerick, "Real-time collision avoidance with whole body motion control for humanoid robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2007, pp. 2053–2058.

[16] O. Stasse, A. Escande, N. Mansard, S. Miossec, P. Evrard, and A. Kheddar, "Real-time (self)-collision avoidance task on a HRP-2 humanoid robot," in *Proc. IEEE Int. Conf. on Robot. and Automat.*, 2008, pp. 3200–3205.

[17] M. Schwienbacher, T. Buschmann, S. Lohmeier, V. Favot, and H. Ulbrich, "Self-collision avoidance and angular momentum compensation for a biped humanoid robot," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2011, pp. 581–586.

[18] A. Dietrich, T. Wimbck, H. Tubig, A. Albu-Schffer, and G. Hirzinger, "Extensions to reactive self-collision avoidance for torque and position controlled humanoids," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2011, pp. 3455–3462.

[19] A. Escande, S. Miossec, M. Benallegue, and A. Kheddar, "A strictly convex hull for computing proximity distances with continuous gradients," *IEEE Trans. Robot.*, vol. 30, no. 3, pp. 666–678, Jun. 2014.

[20] J. J. Quiroz-Omaa and B. V. Adorno, "Whole-body control with (self) collision avoidance using vector field inequalities," *IEEE Robot. Automat. Lett.*, vol. 4, no. 4, pp. 4048–4053, Oct. 2019.

[21] S. S. M. Salehian, N. Figueroa, and A. Billard, "A unified framework for coordinated multi-arm motion planning," *Int. J. Robot. Res.*, vol. 37, no. 10, pp. 1205–1232, 2018.

[22] N. B. Figueroa Fernandez, S. S. Mirrazavi Salehian, and A. Billard, "Multi-arm self-collision avoidance: A sparse solution for a big data problem," in *Proc. 3rd Mach. Learn. Plan. Control Robot Motion Workshop.*, 2018, p. 6.

[23] T. Joachims and C.-N. Yu, "Sparse kernel SVMs via cutting-plane training," *Mach. Learn.*, vol. 76, pp. 179–193, 08 2009.

[24] G. Metta *et al.*, "The iCub humanoid robot: An open-systems platform for research in cognitive development," *Neural Netw.*, vol. 23, no. 8, pp. 1125 - 1134, 2010.

[25] 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 *Proc. IEEE-RAS 15th Int. Conf. Humanoid Robots*, 2015, pp. 1060–1066.

[26] N. Das and M. Yip, "Learning-based proxy collision detection for robot motion planning applications," *IEEE Trans. Robot.*, vol. 36, no. 4, pp. 1096–1114, Aug. 2020.

[27] S. M. LaValle, Planning Algorithms, Cambridge University Press, 2006.

[28] B. Schölkopf and A. Smola, *Learning With Kernels: Support Vector Machines, Regularization, Optimization, and Beyond, Ser. Adaptive Computation and Machine Learning*. Cambridge, MA, USA: MIT Press, Dec. 2002.

[29] J. Pan and D. Manocha, "Fast probabilistic collision checking for sampling-based motion planning using locality-sensitive hashing," *The Int. J. Robot. Res.*, vol. 35, no. 12, pp. 1477–1496, 2016.

[30] Z. Wen, J. Shi, Q. Li, B. He, and J. Chen, "ThunderSVM: A. fast SVM library on GPUs and CPUs," *J. Mach. Learn. Res.*, vol. 19, pp. 797–801, 2018.

[31] J. Mattingley and S. Boyd, "CVXGEN: A. code generator for embedded convex optimization," *Optim. Eng.*, vol. 13, pp. 1–27, Mar. 2012.