When Local Optimization is Bad: Learning What to (Not) Maximize in the Null-space for Redundant Robot Control

Giacomo Golluccio, Paolo Di Lillo, Alessandro Marino, Gianluca Antonelli

***Abstract*— Redundancy in robot structures allows the implementation of control algorithms in which it is possible to add secondary control objectives. Those are typically functions to be minimized/maximized and projected onto the null-space of the primary control objectives. As an example, typical metrics to maximize are the robot manipulability or the distance from its mechanical joint limits. Usually, designer's heuristics is used to decide which function eventually to optimize. This paper shows that heuristics may lead to counter-intuitive results such as, for example, reducing the dexterous workspace with respect to, e.g., avoiding optimization at all. A learning algorithm is proposed to allow the robot to dynamically select the function to optimize in a way to increase the overall dexterous workspace with respect to the static, heuristic choice. As a result, the robot will be able to increase its dexterous workspace by selecting the proper lower-priority task via the use of a neural network trained during a proper supervised learning process. A 3-link planar manipulator is used as numerical case study.**

I. Introduction

Nowadays robot manipulators are more and more involved in several fields, such as industrial, assistive and service robotics for solving many different tasks. In classical industrial environments, all the tasks are defined a-priori, but lately, a sort of intelligence is needed, due to the highly dynamic environments in which these robots are more and more commonly used. If robots have more Degrees of Freedom (DoFs) than the ones required by a task to perform, they are defined as redundant and different redundancy resolution strategies can be found in the literature to compute suitable joint motions [1]. Thus, by exploiting the additional DoFs, it is possible to perform multiple tasks at the same time, e.g. maximization of manipulability, maximizing the distance from the joint limits, and maximizing the distance from an obstacle. A possible approach to implement this behavior is seeing it as a cascade of Quadratic Programming (QP) problems solving them by resorting to the Hierarchical-Quadratic Programming (HQP) framework [2]. Most often the tasks are structured on defined priorities [3] and this approach is referred to as strict task priority [4], [5], [6]. More in detail, the aim of this approach is to minimize the task errors satisfying a set of constraints. Differently, in literature soft task priority approaches can also be found, where a weighted combination of tasks solutions is computed [7].

Authors are with the Department of Electrical and Information Engineering of the University of Cassino and Southern Lazio, via Di Biasio 43, 03043 Cassino (FR), Italy, {giacomo.golluccio, pa.dilillo, al.marino, antonelli}@unicas.it

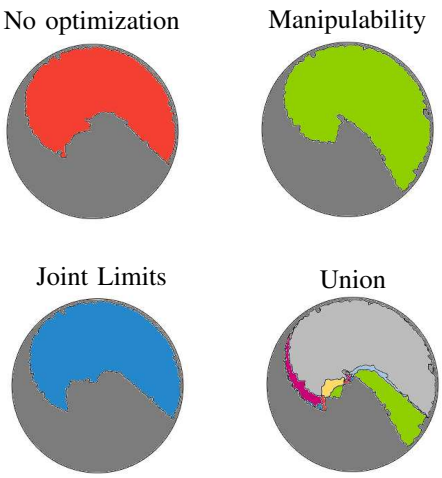

Fig. 1. Workspaces according to different optimization policies generated by a 3-link planar robot with initial configuration $[\pi/4 \quad \pi/3 \quad \pi/3]^T$. Top left: no optimization (red). Top right: optimization of the manipulability (green). Bottom left: optimization of the distance from the joint limits (blue). Bottom right: union of the workspaces.

Unfortunately, finding the best set of tasks for performing a certain high-level mission, with both strict priorities and soft priorities, pass through a time-consuming trial-and error tuning procedure [8]. A possible solution might be changing the tasks or their priorities dynamically during the robot motion, but, as reported in [5], it might cause a discontinuity in the control signals. For this reason, many efforts have been devoted to the development of control algorithms that allow inserting, removing, and swapping tasks without discontinuity, as e.g. in [9].

The authors in [10] present a framework that handles a set of tasks through stochastic parameters optimization using Gaussian kernels for computing weights avoiding conflicts between tasks, but there is no guarantee that the tasks will be accomplished. A mixture of controllers for whole-body motion generation via optimization of a derivative-free stochastic algorithm is proposed in [11], generalizing w.r.t new tasks through transfer policy learning. Again, in [12] a dynamically-consistent generalized hierarchical control is presented. For each pair of tasks, they choose between a soft or a hard priority with one task having a null effect on the other one. Otherwise, in [13] a method for learning task hierarchies is proposed, where the authors show robot potentiality to reproduce the learned priorities in new scenarios. Similarly, in [14] an iterative algorithm for identifying

a stack of tasks is presented. Starting from observation of the joint trajectories and predefined possible tasks executed in parallel, the method gradually removes the non-necessary tasks until a minimum set is identified. In [15], observing the movement, the kinematic constraints are exploited in order to learn the null space projection matrix.

Recently, the usage of Deep Learning in robotics is growing up, achieving impressive results, e.g. AlphaGo [16] or other complex and combinatorial problems [17], [18], [19]. In [8] a Deep Reinforcement Learning (DRL) algorithm for automatically assigning strict task priorities in a varying environment is proposed. More precisely, they showed capabilities such as adaptation to new situations, generalizing w.r.t unseen combinations of tasks, without retraining.

A classical approach to exploit the redundancy of a system is to maximize a function and project the resulting velocities onto the null space of the primary task. The choice of the function to optimize as secondary control objective affects the reachable workspace of the robot. As an illustrative example, Fig. 1 reports the three workspaces of a 3-link planar manipulator obtained by not maximizing any function and maximizing the distance from the joint limits and the manipulability (more details in the remainder of the paper). It is worth highlighting that the union of the three workspaces is actually larger than the three taken separately, proving that changing the function to optimize depending on the operational conditions increases the workspace dimension.

In this work we propose a Deep Learning approach capable of providing in output the function to optimize (if necessary), starting from an initial configuration in the joint space and assigning a desired configuration in the cartesian space for the end-effector. The architecture is validated with numerical simulations on a simple 3-link robot arm. It is shown that there is an enlargement of the workspace with respect to the optimization of a single function.

II. Mathematical Background

Let us consider a generic n joint manipulator with $\boldsymbol{q} \in \mathbb{R}^n$ representing the joint position vector. The end-effector pose is denoted by $\boldsymbol{\eta} = \begin{bmatrix} \boldsymbol{p}_{\text{ee}}^T & \boldsymbol{\mathcal{Q}}_{\text{ee}}^T \end{bmatrix}^T$, where $\boldsymbol{p}_{\text{ee}} \in \mathbb{R}^3$ is the position and $\boldsymbol{\mathcal{Q}}_{\text{ee}}$ the unit quaternion representing the orientation; the end-effector velocity is $\boldsymbol{v}_{\text{ee}} = \begin{bmatrix} \dot{\boldsymbol{p}}_{\text{ee}}^{\text{T}} & \boldsymbol{\omega}_{\text{ee}}^{\text{T}} \end{bmatrix}^{\text{T}} \in \mathbb{R}^6$, being $\dot{\boldsymbol{p}}_{\text{ee}} \in \mathbb{R}^3$ and $\boldsymbol{\omega}_{\text{ee}} \in \mathbb{R}^3$ the linear and angular velocities, respectively. The relationship between the joint velocities $\dot{\boldsymbol{q}} \in \mathbb{R}^n$ and the end-effector linear $\dot{\boldsymbol{p}}_{\text{ee}} \in \mathbb{R}^3$ and angular velocities $\boldsymbol{\omega}_{\text{ee}} \in \mathbb{R}^3$ can be expressed as:

$$\dot{\boldsymbol{p}}_{\text{ee}} = \boldsymbol{J}_{\text{P}}\, \dot{\boldsymbol{q}}, \tag{1}$$

$$\boldsymbol{\omega}_{\text{ee}} = \boldsymbol{J}_{\text{O}}\, \dot{\boldsymbol{q}}\ , \tag{2}$$

where $\boldsymbol{J}_P$ is the $3 \times n$ matrix that maps the joint velocities to the linear velocity of the end-effector, whereas $\boldsymbol{J}_O$ is the $3 \times n$ matrix that maps the joint velocities to the angular velocity of the end-effector. The full geometric Jacobian matrix $\boldsymbol{J} \in \mathbb{R}^{6\times n}$ is obtained by stacking the two matrices defined above, and Eq. (1) and Eq. (2) can be expressed in a more compact form as:

$$\boldsymbol{v}_{\text{ee}} = \begin{bmatrix} \dot{\boldsymbol{p}}_{\text{ee}} \\ \boldsymbol{\omega}_{\text{ee}} \end{bmatrix} = \boldsymbol{J}\dot{\boldsymbol{q}}\ . \tag{3}$$

Given a certain desired position $\boldsymbol{p}_d$ and desired quaternion $\boldsymbol{\mathcal{Q}}_{\text{d}}$ for the end-effector, the joint velocities that make the robot fulfill the end-effector pose task can be computed by resorting to the Closed-Loop Inverse Kinematics (CLIK) algorithm [20]:

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{\dagger} \left(\begin{bmatrix} \dot{\boldsymbol{p}}_{\text{d}} \\ \boldsymbol{\omega}_{\text{d}} \end{bmatrix} + \boldsymbol{K} \begin{bmatrix} \boldsymbol{e}_{\text{p}} \\ \boldsymbol{e}_{\text{o}} \end{bmatrix} \right), \tag{4}$$

where $\boldsymbol{e}_{\text{p}} = \boldsymbol{p}_{\text{d}} - \boldsymbol{p}$ is the end-effector position error, $\boldsymbol{e}_{\text{o}} = \boldsymbol{\mathcal{Q}}_{\text{d}} * \boldsymbol{\mathcal{Q}}^{-1}$ is the orientation error expressed in terms of quaternion error, $\dot{\boldsymbol{p}}_{\text{d}}$ is a feed-forward desired end-effector linear velocity, $\boldsymbol{\omega}_{\text{d}}$ is a feed-forward end-effector angular velocity, $\boldsymbol{K} \in \mathbb{R}^{6\times 6}$ is a positive-definite gain matrix and $\boldsymbol{J}^{\dagger}$ is the Moore-Pensore pseudoinverse of the Jacobian matrix $\boldsymbol{J}$. The differential kinematics can be characterized in terms of the range $\mathcal{R}(\boldsymbol{J})$ and null space $\mathcal{N}(\boldsymbol{J})$ of the Jacobian matrix. When n is greater than the number of DoFs strictly needed for the accomplishment of a certain task, the manipulator is said to be redundant with respect to that task, and the additional DoFs can be exploited to perform other control objectives simultaneously. In order to resolve the possible conflicts that might arise between the two tasks, a usual approach is to define a hierarchy, i.e. giving more importance to one task (which is called *primary*) over the other one (called *secondary*). The objective is to filter out the velocity contribution of the secondary task that would affect the execution of the primary one; in this way, the primary task is executed perfectly as long as it remains feasible, while the secondary one is executed at best. This is implemented by projecting the velocity contribution of the secondary task in the null space of the Jacobian matrix of the primary one, achieving the following solution:

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{\dagger} \left(\begin{bmatrix} \dot{\boldsymbol{p}}_{\text{d}} \\ \boldsymbol{\omega}_{\text{d}} \end{bmatrix} + \boldsymbol{K} \begin{bmatrix} \boldsymbol{e}_{\text{p}} \\ \boldsymbol{e}_{\text{o}} \end{bmatrix} \right) + \boldsymbol{N}\dot{\boldsymbol{q}}_{\text{s}}\ , \tag{5}$$

where $\boldsymbol{N}$ is the null-space projection matrix of Jacobian $\boldsymbol{J}$ computed as $\boldsymbol{N} = \boldsymbol{I} - \boldsymbol{J}^{\dagger}\boldsymbol{J}$, and $\dot{\boldsymbol{q}}_{\text{s}}$ is an additional term related to the secondary task that can be expressed as:

$$\dot{\boldsymbol{q}}_s = k_0 \left(\frac{\partial w(\boldsymbol{q})}{\partial \boldsymbol{q}} \right)^T\ , \tag{6}$$

where k_0 is a scalar gain and $w(\boldsymbol{q})$ is a function to optimize. This function is usually designed in order to maximize some metrics generating internal motions in the structure of the manipulator, i.e. without changing the end-effector trajectory.

III. Motivation and problem statement

Typical metrics to maximize in Eq. (6) are the distance from the joint limits and the manipulability of the arm. Indeed, on the one hand, it is desirable to minimize the

occurrence of joint limit violations, in order to guarantee the feasibility of the motion; on the other hand, it is desirable to maximize the manipulability of the arm to reduce the occurrence of singular configurations and the well-known undesirable effect that it has on the inverse kinematics solution [21]. It is worth noticing that optimizing these metrics employing Eq. (6) does not guarantee that the manipulator would not hit a joint limit or reach singular configurations. In order to do that, the control objectives should be considered at a higher priority with respect to the end-effector pose task, or, in other terms, as constraints in HQP formulations.

A possible expression of the function $w(\boldsymbol{q})$ that represents the measure of manipulability is [22]:

$$w_{\mathrm{m}}(\boldsymbol{q}) = \sqrt{\det\left(\boldsymbol{J}\boldsymbol{J}^T\right)}\,, \tag{7}$$

while for representing the distance from joint limits, the following expression can be adopted:

$$w_{\mathrm{jl}}(\boldsymbol{q}) = -\frac{1}{2n}\sum_{i=0}^{n}\left(\frac{q_i - \overline{q}_i}{q_{i_M} - q_{i_m}}\right)^2, \tag{8}$$

where q_{i_M} and q_{i_m} are the upper and lower limits of the i-th joint respectively, q_i is the i-th joint position and $\overline{q_i}$ is the midpoint between the maximum and the minimum limits.

The final goal of adding a secondary task is to increase the dexterous workspace of the robot, but depending on the chosen function to maximize, the resulting workspace might be different. Currently, the choice of the specific function to optimize is completely left to the designer, who usually employs some kind of heuristics to select a function to maximize that guarantees the best performance. Additionally, in traditional approaches, the function to optimize does not change during the execution, as it is chosen as a design parameter that remains static during the robot operations. This approach represents a huge limitation that decreases the performance of the robot: given a certain initial joint configuration and a certain desired end-effector pose, the function to optimize plays a key role in discriminating whether the robot will be able to eventually reach it or not. Indeed, if the end-effector trajectory would make some of the joints reach the proximity of a limit, it might be convenient to maximize the distance from the joint limits; on the other hand, if it would make the robot reach a close-to-singular joint configuration, the best choice would be to maximize the manipulability of the arm. Interestingly, there can be also situations in which maximizing the distance from the joint limits might make the robot reach a singular configuration during the trajectory, while maximizing the manipulability might make the robot hit a joint limit. In this case, as shown in the remainder of the paper, the best choice might even be to not optimize any function at all, that is employing Eq. (4) instead of Eq. (5).

For these reasons, the problem that we address in this paper is to develop a method that allows to automatically choose the function to optimize depending on the initial joint configuration of the manipulator and the desired end-effector pose. More in detail, we consider a regulation problem, in which the desired pose is constant thus $\dot{\boldsymbol{p}}_d$ and $\boldsymbol{\omega}_d$ are both equal to zero in Eqs. (4) and (5). In the remainder of the paper, as a demonstration example three different possibilities are considered: *1*) do not optimize anything, *2*) maximize the manipulability or *3*) maximize the distance from the joint limits. However, it is worth noticing that no constraint on the number of functions to optimize is set by the proposed approach.

The aim is to show that, using the proposed method, an enlargement of the workspace of the robot can be appreciated.

IV. Proposed Solution

Considering p possible control algorithms to employ $\mathcal{A}_1, \mathcal{A}_2 \ldots, \mathcal{A}_p \in \mathcal{A}$, and a set of constraints $\mathcal{C}$ expressed in terms of minimum and/or maximum values that functions $w_1(\boldsymbol{q}), w_2(\boldsymbol{q}) \ldots, w_h(\boldsymbol{q})$ should have, we define a label $\boldsymbol{l}$ as:

$$\boldsymbol{l} = \begin{bmatrix} l_1 & l_2 & \ldots & l_p \end{bmatrix}^T \quad i = 1, 2, \ldots, p\,, \tag{9}$$

where $l_1, l_2 \ldots l_p \in \{0, 1\}$ are values that represent the violation of at least one of the constraints in $\mathcal{C}$ employing the algorithms in $\mathcal{A}$: the components of the label assume the value 0 if there is a constraint violation in any instant during the trajectory execution and 1 if there is not. Given the definition of the labels, there are 2^p possible values that the labels can assume. From a learning perspective, it could be handled as a classification problem that maps these labels in 2^p classes c_i $(i = 1, 2, \ldots, 2^p)$.

For the case study taken into account in the validation section, we have considered three possible algorithms: $\mathcal{A}_1$ is represented by Eq. (4), that is without optimizing any function; $\mathcal{A}_2$ is as in Eq. (6) optimizing w_{m} as in Eq. (7); $\mathcal{A}_3$ is Eq. (6) optimizing w_{jl} as in Eq. (8). Regarding the constraints, we consider a minimum value for the manipulability, as well as minimum and maximum values for the joint positions. Taking into account the label $\boldsymbol{l}$, if $l_1 = 1$, it means that employing $\mathcal{A}_1$ none of the constraints would be violated; if $l_1 = 0$, it means that at least one of the two constraints would be violated during the trajectory. The same meaning has to be considered with the components l_2, associated with the employment of $\mathcal{A}_2$, and l_3, associated with $\mathcal{A}_3$. Table I resumes the 8 identified classes associating a color for the purpose of visualization.

TABLE I

Mapping for labelling: 0 is related to the violation of at least one of the constraints, 1 if there is no violation.

Class: c	$\mathcal{A}_1$: l_1	$\mathcal{A}_2$: l_2	$\mathcal{A}_3$: l_3	Color
c_1	0	0	0	
c_2	0	0	1	
c_3	0	1	0	
c_4	0	1	1	
c_5	1	0	0	
c_6	1	0	1	
c_7	1	1	0	
c_8	1	1	1	

The neural network that we have designed for classifying what function to optimize in the null space is a fully connected network, as shown in Fig. 2. The input is represented

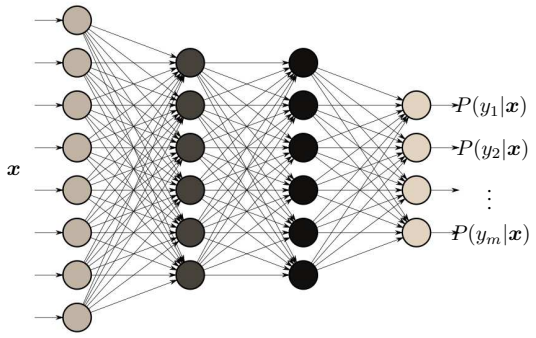

Fig. 2. Architecture of fully connected neural network: $\boldsymbol{x}$ is the input and $P(y|\boldsymbol{x})$ are the probabilities in output.

by the vector:

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{q}_0 \\ \boldsymbol{p}_{\text{ee,d}} \end{bmatrix} , \tag{10}$$

where $\boldsymbol{q}_0 \in \mathbb{R}^n$ is the vector of initial joint configuration and $\boldsymbol{p}_{\text{ee,d}} \in \mathbb{R}^3$ is the desired position of the end-effector. It is worth noticing that, in general, also the desired orientation $\boldsymbol{\mathcal{Q}_d}$ can be seamlessly included in the input vector. The neural network is composed of $j = 4$ layers with Xavier initialization that works on the $\tanh$ activation function. More in detail, the neurons of a network are composed of parameters named *weights* used to calculate a weighted sum of the inputs. The learning process of the neural network consists of the minimization of an error function through an optimization algorithm, namely, a Stochastic Gradient Descent (SGD), which modifies the network weights incrementally. The optimization starts from an initial point in the space of possible weight values. The above-mentioned Xavier approach computes the initial configuration W of random number as follows:

$$W^j \sim \mathcal{N}(\mu, \sigma^2 = \frac{1}{n^{j-1}}) , \tag{11}$$

where n^{l-1} is the number of neurons at layer $l-1$. In the last layer, an activation function called *softmax* is often applied in multi-class learning problems where a set of features can be related to one-of-K class. The output of the softmax describes the probability $P(y = c_i|\boldsymbol{x})$ of the neural network that a particular sample belongs to a certain class. From a mathematical perspective, it is defined as:

$$f_{\text{softmax}}(y_i) = \frac{e^{y_i}}{\sum_{k=1} e^{y_k}} . \tag{12}$$

In combination with the f_{softmax} activation function, the Negative-Log Likelihood (NLL) is used in this paper. It uses a negative connotation since the probabilities (or likelihoods) vary between zero and one, and the logarithms of values in this range are negative.

V. Validation

The proposed approach has been validated through numerical simulations taking into account a simple 3-link planar manipulator, with the length of the three links equal to 1m each. Upper and lower limits of $\pm 120°$ have been considered on all three joints, while a lower threshold for the manipulability of 0.1 has been set. The constraints are considered violated if the values of the joint limits and manipulability exceed the imposed thresholds during the motion of the robot. The input vector $\boldsymbol{x}$ of the DNN in Eq. (10) is composed of the vector containing the initial position of the three joints and the desired 2D end-effector position. The robot and its workspace without the joint limits are represented in Fig. 3. The entire architecture is validated in MATLAB environment for the dataset generation and Python, using the PyTorch framework for training the model using CUDA on a GPU. More in detail, all the software has been run on a desktop PC with CPU Intel(R) Core(TM) i9-9900KF 3.60GHz and GPU GeForce RTX 2070 Super equipped with Ubuntu 20.04 and MATLAB 2020b.

A. Dataset Generation

The dataset consists of a set of instances for each class obtained randomly selecting $\boldsymbol{x}$ in Eq. (10), such as $\boldsymbol{q}_0$ meets the joint limits, and the desired end-effector positions $\boldsymbol{p}_{\text{ee,d}}$ is in the circle of radius 3m.

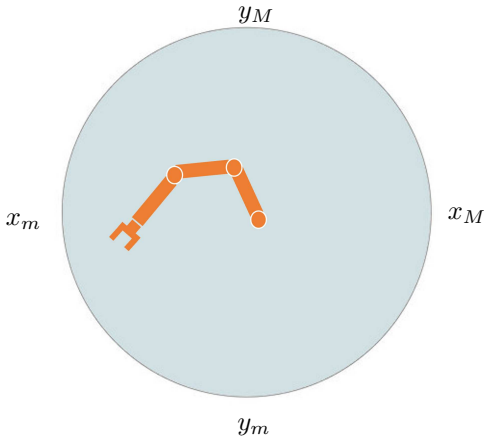

Fig. 3. 3-link planar robot and its workspace without the joint limits.

Furthermore, a common the dataset is split into three parts that are *training*, *validation* and *test*. The first ones are used to train the model, whereas the last one is a portion of data used for testing the performance.

B. Results

Figure 4 shows the entire robot workspace divided into regions with different colors (according to Table I), starting with a constant initial joint configuration $\boldsymbol{q}_0 = [\pi/4 \quad \pi/3 \quad \pi/3]^T$ rad. Each point of the figure is associated with a label that represents the functions to optimize to reach that point.

Looking at the figure, it is possible to notice that the distance among some of the regions is very small. This means that there might be a classification error. More in detail, in the range $x \approx [-0.05, 0.05]^T$ and $y \approx [0.0, 0.2]^T$ or $x \approx [-2.0, -1.55]^T$ and $y \approx [-0.8, -0.3]^T$ the class c_1 might be wrongly classified as c_2; in $x \approx [-0.05, 0.2]^T$

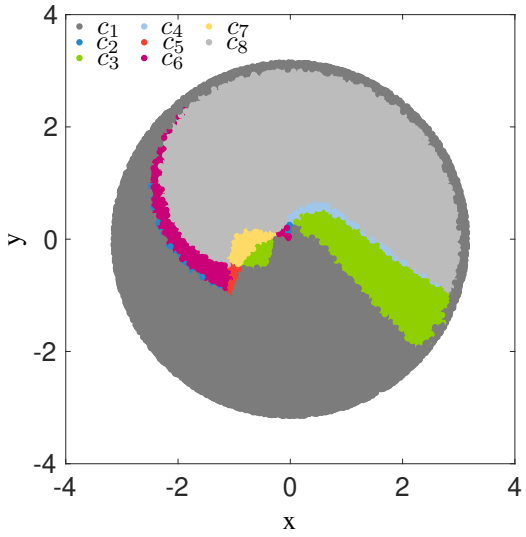

Fig. 4. Sampled workspace of the 3-link robot arm for the initial configuration $\boldsymbol{q}_0 = [\pi/4 \quad \pi/3 \quad \pi/3]^T$ with a robot base position in the origin.

and $y \approx [0.05, 0.3]^T$ or $x \approx [-0.9, -0.3]^T$ and $y \approx [-0.5, -0.1]^T$ the class c_3 might be classified as c_6 or c_7; in $x \approx [-1.4, -1.2]^T$ and $y \approx [-0.85, -0.75]^T$ the class c_5 might be classified as c_2; in $x \approx [-0.3, 0.1]^T$ and $y \approx [-0.3, -0.3]^T$ the class c_6 might be classified as c_3 or c_4. Finally, classes c_7 and c_8 might be wrongly classified in the range where $x \approx [-1.2, -0.3]^T$ and $y \approx [-0.5, 0.1]^T$. These considerations are consistent with the confusion matrix shown in Fig. 5; indeed, it is possible to notice that the model is capable of detecting and correctly classifying which algorithm to select among the ones in the set $\mathcal{A}$ in order to meet the constraints in $\mathcal{C}$ reaching the 94% on the test set.

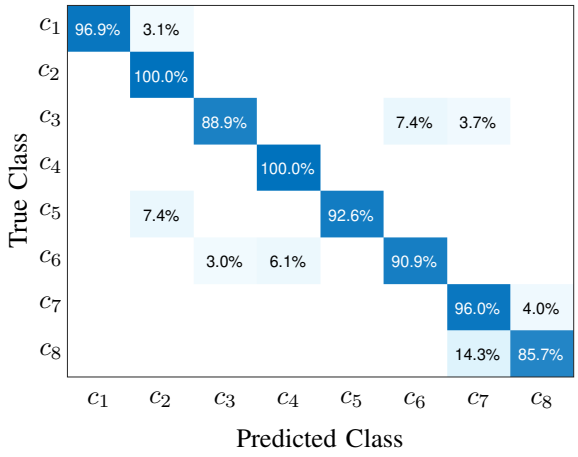

Fig. 5. Confusion Matrix normalized for row. The diagonal elements represent the correctly classified classes, whereas the other values represent the wrongly classified ones.

As an illustrative example of c_4, Fig. 6 shows the evolution of the joint positions and the manipulability with their thresholds employing the three abovementioned algorithms. It is worth noticing the violation of the upper limit of the second joint using $\mathcal{A}_1$, whereas $\mathcal{A}_2$ and $\mathcal{A}_3$ do not cause any constraint violation.

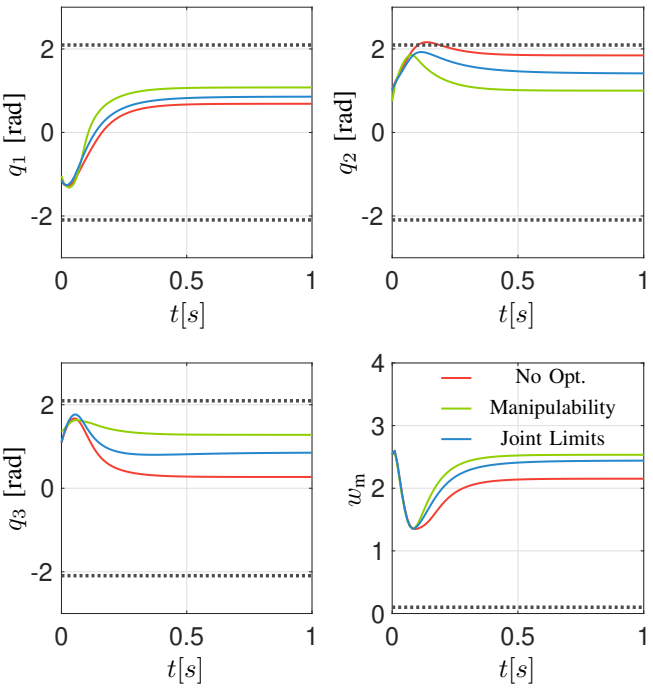

Fig. 6. Top-left: evolution of the first joint position using and their joint limits (dotted line). Top-right: evolution of the second joint position and their joint limits (dotted line). Bottom-right evolution of the third joint position and their joint limits (dotted line). Bottom-right: evolution of the manipulability functionals and their threshold (dotted line). The color of the plots are related to the algorithms $\mathcal{A}_1, \mathcal{A}_2$ and $\mathcal{A}_3$.

It is worth highlighting that the union of the three workspaces is actually larger than the three taken separately, proving that changing dynamically the function to optimize depending on the initial joint configuration and the desired and-effector pose increases the workspace dimension. This is quantified in Table II, which reports the areas of the workspaces normalized with respect to the union. Analyzing the results, it is possible to notice that using algorithm $\mathcal{A}_1$ (i.e., not optimizing any function) there is a loss of $\approx 20\%$ in terms of workspace, whereas by maximizing only the manipulability there is a loss of 7% and maximizing only the distance from the joint limits the workspace is reduced by 18%.

TABLE II

AREA OF THE WORKSPACES OBTAINED WITH THE THREE ALGORITHMS AND THEIR UNION

Region	Color	Classes	Area
No optimization		c_5, c_6, c_7, c_8	0.80
Manipulability		c_3, c_4, c_7, c_8	0.93
Joints limits		c_2, c_4, c_6, c_8	0.82
Union	+ +	$c_1, c_2 \ldots c_8$	1.00

It is important to observe that the difference in terms of workspace is strictly related to the initial joint configuration of the robot. As evidence, changing the sign of the last joint at the beginning of the simulation, i.e. by setting $\boldsymbol{q}_0 = [\pi/4 \quad \pi/3 \quad -\pi/3]^T$ rad, the obtained workspaces

are significantly different, and it causes a shift upwards of regions c_3, c_4, as it is shown in Fig. 7.

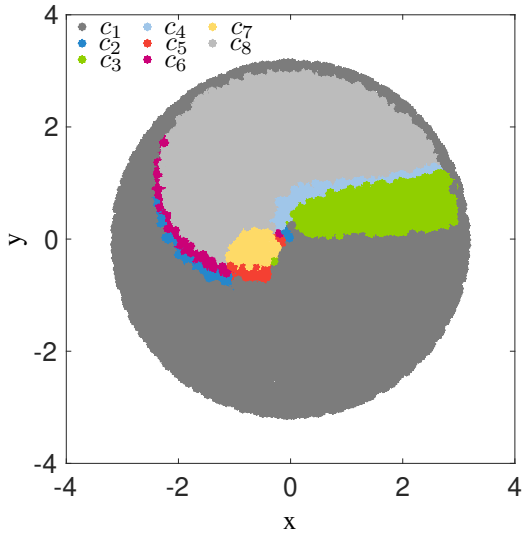

Fig. 7. Sampled workspace of the 3-link robot arm for the initial configuration $\boldsymbol{q}_0 = [\pi/4 \quad \pi/3 \quad -\pi/3]^T$ with a robot base position in the origin.

VI. Conclusion

A Supervised Learning process to properly handle optimization tasks has been investigated in this paper. A simple case study consisting of a 3-link planar manipulator already exhibits interesting properties and improvements under the proposed approach. Future research directions include: *(i)* investigation of the role of the optimization gains in the formation of the regions; *(ii)* introduction of a proper metric instead of the binary classification $0/1$ indicating (un)reachability; *(iii)* extension to full dimensional robotic structures; and *(iv)* a comparison using RGB images and 3D voxel data representation for the robot workspace.

VII. Acknowledgement

The authors declare that this work was supported by Dipartimento di Eccellenza granted to DIEI Department, University of Cassino and Southern Lazio, by H2020-ICT project CANOPIES (Grant Agreement N. 101016906), by POR FSE LAZIO 2014-2020, Project DE G06374/2021 and by Project "Ecosistema dell'innovazione - Rome Technopole" financed by EU in NextGenerationEU plan through MUR Decree n. 1051 23.06.2022 - CUP H33C22000420001.

References

[1] M. D. Fiore, G. Meli, A. Ziese, B. Siciliano, and C. Natale, "A general framework for hierarchical redundancy resolution under arbitrary constraints," *IEEE Transactions on Robotics*, pp. 1–20, 2023.

[2] A. Rocchi, E. M. Hoffman, D. G. Caldwell, and N. G. Tsagarakis, "Opensot: a whole-body control library for the compliant humanoid robot coman," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 6248–6253.

[3] Y. Nakamura, H. Hanafusa, and T. Yoshikawa, "Task-priority based redundancy control of robot manipulators," *The International Journal of Robotics Research*, vol. 6, no. 2, pp. 3–15, 1987.

[4] A. Del Prete, F. Nori, G. Metta, and L. Natale, "Prioritized motion–force control of constrained fully-actuated robots:"task space inverse dynamics"," *Robotics and Autonomous Systems*, vol. 63, pp. 150–157, 2015.

[5] A. Dietrich, C. Ott, and A. Albu-Schäffer, "An overview of null space projections for redundant, torque-controlled robots," *The International Journal of Robotics Research*, vol. 34, no. 11, pp. 1385–1400, 2015.

[6] L. Penco, E. M. Hoffman, V. Modugno, W. Gomes, J.-B. Mouret, and S. Ivaldi, "Learning robust task priorities and gains for control of redundant robots," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2626–2633, 2020.

[7] J. Salini, V. Padois, and P. Bidaud, "Synthesis of complex humanoid whole-body behavior: A focus on sequencing and tasks transitions," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 1283–1290.

[8] M. Karimi and M. Ahmadi, "A reinforcement learning approach in assignment of task priorities in kinematic control of redundant robots," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 850–857, 2021.

[9] S. Kim, K. Jang, S. Park, Y. Lee, S. Y. Lee, and J. Park, "Continuous task transition approach for robot controller based on hierarchical quadratic programming," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1603–1610, 2019.

[10] R. Lober, V. Padois, and O. Sigaud, "Variance modulated task prioritization in whole-body control," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 3944–3949.

[11] N. Dehio, R. F. Reinhart, and J. J. Steil, "Multiple task optimization with a mixture of controllers for motion generation," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 6416–6421.

[12] N. Dehio and J. J. Steil, "Dynamically-consistent generalized hierarchical control," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 1141–1147.

[13] J. Silvério, S. Calinon, L. Rozo, and D. G. Caldwell, "Learning task priorities from demonstrations," *IEEE Transactions on Robotics*, vol. 35, no. 1, pp. 78–94, 2018.

[14] S. Hak, N. Mansard, O. Stasse, and J. P. Laumond, "Reverse control for humanoid robot task recognition," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 42, no. 6, pp. 1524–1537, 2012.

[15] H.-C. Lin, M. Howard, and S. Vijayakumar, "Learning null space projections," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 2613–2619.

[16] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[17] G. Golluccio, D. Di Vito, A. Marino, A. Bria, and G. Antonelli, "Task-motion planning via tree-based q-learning approach for robotic object displacement in cluttered spaces," in *Proceedings of the 18th Int. Conf. on Informatics in Control, Automation and Robotics - ICINCO*, INSTICC. SciTePress, 2021, pp. 130–137.

[18] G. Golluccio, D. Di Vito, A. Marino, and G. Antonelli, "Robotic weight-based object relocation in clutter via tree-based q-learning approach using breadth and depth search techniques," in *2021 20th Int. Conf. on Advanced Robotics (ICAR)*. IEEE, 2021, pp. 676–681.

[19] G. Golluccio, P. Di Lillo, D. Di Vito, A. Marino, and G. Antonelli, "Objects relocation in clutter with robot manipulators via tree-based q-learning algorithm: Analysis and experiments," *Journal of Intelligent & Robotic Systems*, vol. 106, no. 2, pp. 1–20, 2022.

[20] S. Chiaverini, "Singularity-robust task-priority redundancy resolution for real-time kinematic control of robot manipulators," *IEEE Transactions on Robotics and Automation*, vol. 13, no. 3, pp. 398–410, 1997.

[21] D. Di Vito, C. Natale, and G. Antonelli, "A comparison of damped least squares algorithms for inverse kinematics of robot manipulators," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 6869–6874, 2017.

[22] T. Yoshikawa, "Manipulability of robotic mechanisms," *The International Journal of Robotics Research*, vol. 4, no. 2, pp. 3–9, 1985.