University of Cassino and Southern Lazio

Doctoral Course in
Methods, Models and Technologies for Engineering
Curriculum in Computer Engineering
Cycle XXXV

Robot Planning and Control combined with Machine Learning Techniques

---

SSD: ING-INF/04 AUTOMATICA

Supervisors:
Prof. Alessandro Marino
Prof. Gianluca Antonelli

Coordinator:
Prof. Fabrizio Marignetti

Author:
Giacomo Golluccio

"*The Three Laws of Robotics:*

- *A robot may not injure a human being or, through inaction, allow a human being to come to harm*
- *A robot must obey orders given it by human beings except where such orders would conflict with the First Law*
- *A robot must protect its own existence as long as such protection does not conflict with the First or Second Law*

I. Asimov"

Abstract

Robots and Artificial Intelligence (AI) have been growing in various contexts, such as automotive, healthcare and manufacturing. In this thesis, the adoption of Machine Learning (ML) and control techniques is explored to solve relevant tasks in the robotics field.

The focus is on the task of retrieving an object target from cluttered environment; as it is well-known, the problem is combinatorial and hard to solve in reasonable time. Here, it is solved by using a two layers architecture characterized from an high level Task Planner (TP) made of a Reinforcement Learning (RL) agent, combined to a low level based on Motion Planner (MP) and Inverse Kinematics (IK) Control. The architecture is validated via simulation and using the KINOVA Jaco^2 7-DoFs robot manipulator. During the computation of the path from the Motion Planner, it could be useful to check the collision quickly. Thus, the collision detection problem in unstructured environment by using a learning-based approach is considered. The idea is to present an architecture that could reduce the computational time spent by the planner. Going into detail, it is addressed employing depth images and point clouds for adapted neural networks, i.e. CNN and PointNet. The proposed approach is validated with an industrial robot at the Technology & Innovation Center (TIC) of KUKA Deutschland GmbH in Augsburg (Germany).

Furthermore, also an optimization in the control law is considered. When a robot manipulator is redundant, it is possible to exploit the additional degree of freedoms for maximizing different functionals into the null space of the Jacobian matrix, e.g. maximization of manipulability and distance from joint limits. The idea is to show that, through a Supervised Learning (SL) approach, it is possible to enlarge the dextrous workspace of the robot.

Contents

Contents **v**

List of Figures **ix**

List of Tables **xvii**

1 Introduction **1**
1.1 Motivation 1
1.2 State of the Art 3
1.2.1 Thesis Outline and Contribution 11
1.2.2 Publications 14
Other Contributions 15

2 Background **17**
2.1 Robot Modelling and Identification 17
2.1.1 Kinematics 17
Position and Orientation of a Rigid Body 17
Direct Kinematics 19
Inverse Kinematics 19
2.1.2 Dynamics 23
Lagrange Formulation 24
Kinetic Energy 24
Potential Energy 25
Equations of Motion 26
2.1.3 Dynamic Parameters 26
2.1.4 Identification of Dynamic Parameters 27
2.1.5 Constraints on the Robot Model 29

Physical Feasibility 29
Physical Consistency 30
2.1.6 Methods for Estimating Dynamic Parameters 31
CAD 32
Unconstrained Least Square (ULS) 32
Constrained Least Square (CLS) - Technique 1 (CLS-1) 33
Constrained Least Square (CLS) - Technique 2 (CLS-2) 34
Constrained Least Square (CLS) - Technique 3 (CLS-3) 35
2.1.7 Identification Methods: A Comparison 36
2.1.8 Validation methodology 37
2.1.9 Trajectories 39
2.1.10 Results 39
2.2 Machine Learning 47
2.2.1 Reinforcement Learning 48
Rooted Trees 50
2.2.2 Supervised Learning 51
Neural Networks 51
Feed-Forward Neural Networks 51
Convolutional Neural Network 53

3 Task-Motion Planning via Reinforcement Learning 57
3.1 Retrieving Objects from Clutter 57
3.1.1 System Architecture 58
Low-level: Motion Planner 59
High-level: RL-Task Planner 62
3.1.2 Exploration Policies 63
Tree-search Methods 65
Q-Tree Learning Algorithm 66
Optimality Analysis 66
3.2 Simulation and Experiments 69
Case 1: Identical Objects 69
Case 2: Different Objects 71

4 Learning-based Robot Collision Detection 85
4.1 Data Representation and Camera 85

4.2 Problem Description 87
4.3 Data Generation 87
4.3.1 Depth Images 87
4.3.2 Point Clouds 89
4.4 Collision Checkers 90
4.4.1 Geometric-based 90
4.4.2 CNN-based 91
FCNN - 1 Depth Image 92
FCNN - 2 Depth Images 93
ResNet18 93
4.4.3 PointNet-based 94
Standard PointNet 94
Fast-PointNet 95
4.4.4 Hybrid-based 96
MixNet 96
4.5 Simulation & Results 96

5 Deep Learning for Task Priority Inverse Kinematics 103
5.1 Multi-class Problem 104
5.1.1 Learning Model 105
5.2 Validation 107
5.2.1 Dataset Generation 107
5.2.2 Results 108

6 Conclusions and Future Works 113

7 Appendix 117
7.1 Notes on the positive definiteness of dynamic matrix $\boldsymbol{M}(\boldsymbol{q})$ . . 119
7.2 User Guide 121
Code Ocean version 122
IEEE DataPort 123
7.2.1 Running the code 124

Bibliography 129

List of Figures

1.1 Overview of different contexts: top-left an healthcare application; top-right an assistive one application; at bottom-left an industrial application in automotive and bottom-right an application with cobots. 2

1.2 The proposed architecture for this work is composed of several blocks. The perception one provides the objects pose estimation and generates the depth image or point cloud of the scene. The outputs are then used by the task planner and collision detection systems. The task planner system uses a combination of reinforcement learning and motion planner to devise an object relocation procedure. The collision detection system exchanges information with the task planner system to generate a collision-free trajectory. Finally, the control block computes joint velocities or torques for the robot. 12

2.1 Example of robot manipulator in the space. 20

2.2 Mapping between the velocities in joint and cartesian space. . . 22

2.3 The KINOVA Jaco^2 robot is being considered as a test case. On the left, an orange external ellipsoid is shown, bounding the 4 th link, while a red internal ellipsoid containing the center of mass is also displayied. On the right, the reference frames of the KINOVA Jaco^2 robot are illustred according to the DH convention. 31

2.4 Joint position, velocity and acceleration zoom representation of the first 20 seconds of the 3rd trajectory. The full version of trajectory is used as identification dataset. 40

2.5 Low pass filter effect on joint torque, example for the second joint of 3rd trajectory with a zoom-in plot in the time interval $[4.4, 5]$ s. 42
2.6 CAD reconstruction errors along the 3rd trajectory. 43
2.7 ULS reconstruction errors along the 3rd trajectory. 43
2.8 CLS-1 reconstruction errors along the 3rd trajectory. 44
2.9 CLS-2 reconstruction errors along the 3rd trajectory. 44
2.10 CLS-3 reconstruction errors along the 3rd trajectory. 46
2.11 Machine Learning Algorithms: Supervised, Unsupervised and Reinforcement. 47
2.12 Scheme of Reinforcement Learning approach: An agent in a state s_k applies an action a_k interacting with the environements, which gives a reward r_k to the agent that updates its state with s_{k+1}. 48
2.13 Architecture of Feed-Forward Neural Network. 51
2.14 Architecture of VGGNet16. 55

3.1 Representation of the proposed architecture: the RL-Task Planner chooses the action a_k with an appropriate policy, while the Motion Planner provides information about the feasibility g_k of the chosen action a_k in terms of fulfillment of kinematic constraints. In case of a feasible action, the joint velocities vector $\dot{\boldsymbol{q}}(t)$ is sent to the Robot that actually performs it, relocating the object. The environment elaborates the information related to the feasibility g_k of the action and generates a reward signal r_k and the new state s_k, which are used to update the RL agent. 59
3.2 Motion planner architecture, designed as three blocks: the Action-Objects Mapping translates the actions in constant desired end-effector poses; the Sampling Algorithm computes obstacle-free trajectories for the end-effector; the STPIK (Set-based Task-Priority Inverse Kinematics) checks the feasibility of the trajectories in terms of joint-level kinematic constraints (joint limits and self-hits). 60

3.3 Example of complete Q-Tree: The node root s_H represents the initial configuration where all objects are in the initial location. The last nodes contain the target T. 64
3.4 Different exploration method: Breadth prefers to explore at the same levels, whereas the Depth one prefers to move forward. . 65
3.5 An example of cluttered environment. The target is represented by the cylinder in green, while obstacles are in red. Starting from the scenario on the top, the manipulator is required to relocate some obstacles in order to grasp the target (bottom scenario). . 69
3.6 Comparison of three learning policies defined for each scenario. The average episodes is calculated on 50 training and represent the first time that the target T is reached through the optimal sequence. 71
3.7 Comparison of convergence between learning algorithms defined above for each scenario. The average episodes number is calculated on 50 training. 72
3.8 The target is represented by a green cylinder, while the obstacles are indicated with a different color, based on weights, which models concepts as fragility or energetic-related metrics. The robot, a KINOVA Jaco^2 in the case study, needs to eventually relocate objects in order to reach the target. 74
3.9 Plots represent the sum of all tree edge values $\overline{Q}_h$ normalized with respect to the steady state value $\overline{Q}_\infty$ for different values of α and γ in the case of H-LRND$_b$ approach and Scenario 3. 75
3.10 Steady State Parametric Analysis: This bar graph is relative to the reaching of optimal solution varying α and γ for the Scenario 1. . . . 79
3.11 Steady State Parametric Analysis: This bar graph is relative to the reaching of optimal solution varying α and γ for the Scenario 2. 79
3.12 Steady State Parametric Analysis: This bar graph is relative to the reaching of optimal solution varying α and γ for the Scenario 3. 80

3.13 Comparison between Breadth (left) and Depth (right) approaches, considering the proposed tree exploration strategies for the Scenario 3, with learning rate $\alpha = 0.5$ and discount factor $\gamma = 0.9$. 80
3.14 Three learning policies comparison: Average, minimum and maximum episode relative at the first time that the agent reaches the target T through the optimal sequence considering Breadth search approach. The statistics are based on 50 training with $\alpha = 0.5$ and $\gamma = 0.9$. 81
3.15 Three learning policies comparison: Average, minimum and maximum episode relative at the first time that the agent reaches the target T through the optimal sequence considering Depth search approach. The statistics are based on 50 training with $\alpha = 0.5$ and $\gamma = 0.9$. 81
3.16 Left. The robotic setup adopted to demonstrate the devised approach. Right. A top view representation of target (in green) and obstacles in their initial configuration. 82
3.17 Software-Hardware Architecture: The perception module receives the scene from the camera and provides information on the objects pose to the RL-Task Planner (MATLAB). This latter selects an action according to its policy and sends the action to the Motion Planner (C++). Finally, if there is a free-obstacle path that satisfies all the joint constraints, the robot receives the computed joint velocities to perform the task. 82
3.18 Robot manipulator during the validation phase: the KINOVA Jaco^2 is moving the objects in the real world scenario. 83

4.1 The mesh of the robot end-effector M_g is projected into the a desired configuration $\boldsymbol{\eta}_{\mathrm{ee,d}}$ for detecting the collision. 86
4.2 Scenario with robot manipulator and a full bin with objects; the camera is top-mounted in order to have a complete view of the bin. 88
4.3 Example of real bin from the camera point of view. The bin contains some different industrial objects. 89

4.4 Generation dataset $\mathcal{D}_{\text{depth}}$: a camera acquires the scene and the gripper is projected into the scene. The output consists of three depth images: gripper, scene (without gripper) and complete scene (with gripper). 90
4.5 Application of the crop filter Δ_c on the complete depth image $\boldsymbol{I}_c$. It reduces the size of the images from (640, 480) to (156, 156). 91
4.6 Generation dataset $\mathcal{D}_{\text{pointcloud}}$: a real camera acquires the scene providing the depth image of the scene. The PointCloud Generator G_{PC} block creates the pointclouds of the scene and the gripper. The concatenation block U_{PC} combines them generating the complete point cloud. 92
4.7 In the left part the complete (scene and gripper) depth image is reported, whereas in the right part the complete pointcloud is reported. 92
4.8 Architecture of Fully Convolutional Neural Network with one depth image $(156, 156, 1)$ as input. 94
4.9 Architecture of Fully Convolutional Neural Networks for two depth images $(156, 156, 2)$ as input. 95
4.10 Architecture of ResNet18, which receives two depth images $(156, 156, 2)$ as input. 96
4.11 Architecture of PointNet receives a complete point cloud, which contains scene and gripper as input. 98
4.12 Architecture of MixNet, which receives a depth image $(156, 156, 1)$ and a point cloud as input. 99
4.13 Software Architecture: the main block receives the depth image by the camera and, building the data (depth image or point cloud) and sends them to a neural network allowed to detect collisions with the environment; the collision checker returns a feedback that compute the control signal for the KUKA Agilus, moving it in the desired configuration. 100
4.14 Experimental validation of KUKA Agilus. 101

5.1 Architecture of fully connected neural network: $\boldsymbol{u}$ is the input and $P(y|\boldsymbol{u})$ are the probabilities of the output. 106

5.2 3-link planar Robot and its workspace without joint limits. . . 108
5.3 Sampled workspace of the 3-link robot arm for the initial configuration $[\pi/4 \quad \pi/3 \quad \pi/3]^T$ with a robot base position in $(0, 0)$. 109
5.4 Confusion Matrix normalized for row. The diagonal represents the classes correctly classified, whereas the others value represent the classes wrongly classified. 110
5.5 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 thresholds (dotted line). The color of the plots are related to the algorithms $\mathcal{F}_1, \mathcal{F}_2$ and $\mathcal{F}_3$. 111
5.6 Sampled workspace of the 3-link robot arm for the initial configuration $[\pi/4 \quad \pi/3 \quad -\pi/3]^T$ with a robot base position in $(0, 0)$. 112

7.1 Interface of GUI Matlab necessary to reproduce experiments. . 119
7.2 Software architecture. 122
7.3 Code Ocean platform. 123
7.4 CVX download page. 124
7.5 CVX folder extracted in the identification toolbox folder. . . . 125
7.6 MATLAB Environment with initialized working directory. . . . 125
7.7 GUI how it appears after the file *main.m* has been launched. . 126
7.8 Trajectory selection window after either *load ID* or *load VAL* buttons are push. 126
7.9 Building regressor phase with a progress bar showing the elapsed and remaining time. 127
7.10 *Identification* button enabled after the identification and validation trajectories have been selected and the relative regressors computed. 127
7.11 Progress bar showing the elapsed and remaining time once the *Identification* button is pushed. 128
7.12 Validation button enabled with *CLS3* method selected. 128

7.13 Numerical results displayed at the end of the validation process. 128

List of Tables

2.1 Denavit-Hartenberg table for the KINOVA Jaco^2 28
2.2 Identifiability of the parameters (red cells: not identifiable; blue cells: identifiable in linear combination; white cells: identifiable alone. 36
2.3 Dynamic parameters in linear combinations. 37
2.4 Numerical values for base dynamic parameters of the algorithms on the 3rd validation trajectory. 41
2.5 Summary of the results. With red background the largest error for a specific trajectory, with green the smallest. 45
2.6 Constraints satisfaction of the different identification methods. 46

3.1 Analysis considering $\alpha = 0.5$, and $\gamma = 0.9$ and as evaluation criteria: the number of episodes E_{ss} necessary to converge, the number of motion planning queries MP_q and the the episode number $E_{1_{st}}$ in which the optimal solution is reached for the first time. This case is for the Scenario 1 with obstacles number $N_o = 5$. 70
3.2 Analysis considering $\alpha = 0.5$, and $\gamma = 0.9$ and as evaluation criteria: the number of episodes E_{ss} necessary to converge, the number of motion planning queries MP_q and the the episode number $E_{1_{st}}$ in which the optimal solution is reached for the first time. This case is for the Scenario 2 with obstacles number $N_o = 10$. 73

3.3 Analysis considering $\alpha = 0.5$, and $\gamma = 0.9$ and as evaluation criteria: the number of episodes E_{ss} necessary to converge, the number of motion planning queries MP_q and the the episode number $E_{1_{st}}$ in which the optimal solution is reached for the first time. This case is for the Scenario 3 with obstacles number $N_o = 15$. 73
3.4 Parametric analysis considering (α, γ) parameters and as evaluation criteria: the number of episodes E_{ss} necessary to converge, the number of motion planning queries MP_q and the the episode number $E_{1_{st}}$ in which the optimal solution is reached for the first time. This case is for the Scenario 1 with obstacles number $N_o = 5$. 76
3.5 Parametric analysis considering (α, γ) parameters and as evaluation criteria: the number of episodes E_{ss} necessary to converge, the number of motion planning queries MP_q and the the episode number $E_{1_{st}}$ in which the optimal solution is reached for the first time. This case is for the Scenario 2 with obstacles number $N_o = 10$. 77
3.6 Parametric analysis considering (α, γ) parameters and as evaluation criteria: the number of episodes E_{ss} necessary to converge, the number of motion planning queries MP_q and the the episode number $E_{1_{st}}$ in which the optimal solution is reached for the first time. This case is for the Scenario 3 with obstacles number $N_o = 15$. 78

4.1 Analysis of learning-based methods for collision detection on training, validation and test set. 98
4.2 Times for feeding a sample into the models using CPU-GPU. . 100

5.1 Mapping for labelling: 0 is related to the violation of constraints, 1 if there is no violation. 105
5.2 Area of the workspaces obtained with the three algorithms and their union. 110

List of Abbreviations

IFR	International Federation of Robotics
DLR	German Aerospace Center
TIC	Technology & Innovation Center
APC	Amazon Picking Challenge
MP	Motion Planning
TP	Task Planning
TAMP	Task-Motion Planning
AI	Artificial Intelligence
ML	Machine Learning
RL	Reinforcement Learning
SL	Supervised Learning
CNN	Convolutional Neural Network
FCL	Flexible Collision Library
BVH	Bounding Volume Hierarchies
RRT	Rapidly Exploring Random Tree
DH	Denavit-Hartenberg
CLIK	Closed Loop Inverse Kinematics
ULS	Unconstrained Least Square
CLS	Constrained Least Square
DOF	Degree Of Freedom
GUI	Graphical User Interface
SBTPIK	Set-Based Task-Priority Inverse Kinematics
MDP	Markov Decision Process
DAG	Directed Acyclic Graph
BFS	Breadth First Search
DFS	Depth First Search
YOLO	You Only Look Once

List of Symbols

$\boldsymbol{x}$	pose
$\boldsymbol{p}$	position
$\boldsymbol{\varphi}$	orientation
$\boldsymbol{R}$	rotation matrix
$\boldsymbol{\Sigma}_I$	inertial frame
$\boldsymbol{\Sigma}_B$	base frame
$\boldsymbol{\Sigma}_E$	end-effector reference frame
$\mathcal{Q}$	unit quaternion
$\boldsymbol{x}_{ee}$	pose end-effector
$\boldsymbol{q}$	joint position
$\dot{\boldsymbol{p}}_{ee}$	linear velocity
$\boldsymbol{\omega}_{ee}$	angular velocity
$\boldsymbol{J}_P$	Jacobian of position
$\boldsymbol{J}_O$	Jacobian of orientation
$\boldsymbol{J}$	Jacobian
$\boldsymbol{v}_{ee}$	end-effector velocities
$\dot{\boldsymbol{q}}$	joint velocities
$g(\dot{\boldsymbol{q}}, \boldsymbol{\lambda})$	Lagrange multipliers
$\mathcal{R}(\boldsymbol{J})$	range of Jacobian matrix $\boldsymbol{J}$
$\mathcal{N}(\boldsymbol{J})$	null of Jacobian matrix $\boldsymbol{J}$
$(\cdot)^\dagger$	pseudoinverse operator
$w(\boldsymbol{q})$	functional to maximize in the null space
q_{i_m}	lower joint limit
q_{i_M}	upper joint limit
$\boldsymbol{\sigma}_x$	variable task
$\mathcal{L}$	Lagrangian function
$\mathcal{T}$	kinetic energy
$\mathcal{U}$	potential energy
$\boldsymbol{\xi}$	nonconservative forces
$\boldsymbol{M}(\boldsymbol{q})$	inertia matrix
$\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$	Coriolis forces
$\boldsymbol{F}_v$	viscous friction coefficients

$\boldsymbol{F}_s \text{sgn}(\dot{\boldsymbol{q}})$	Coulomb friction torques
$\boldsymbol{g}(\boldsymbol{q})$	gravity torques
$\boldsymbol{\tau}$	torques
$\boldsymbol{h}_e$	force and moment exerted by end-effector
$\boldsymbol{Y}_{\text{full}}$	full regressor
$\boldsymbol{L}_i$	inertia tensor
$\boldsymbol{\pi}$	vector of dynamic parameters
m_i	mass of link i
$\boldsymbol{m}_{c_i}^i$	center of mass
l_i	element i of inertia tensor
$\boldsymbol{Y}_{\text{b}}$	base regressor
$\boldsymbol{C}_i$	inertia tensor on center of mass for link i
$\boldsymbol{S}(\cdot)$	skew-symmetric matrix operator
$\boldsymbol{Z}_i$	matrix for physical feasilibity
λ	eigenvalue
γ_r	regularization factor
$\text{tr}(\cdot)$	trace operator
$\boldsymbol{m}_{c_i,\text{LB}}^i$	lower bound on center of mass of link i
$\boldsymbol{m}_{c_i,\text{UB}}^i$	upper bound on center of mass of link i
$\mathcal{E}_i$	ellipsoid
$\boldsymbol{x}_{c_i}^i$	center of ellipsoid
$\bar{\boldsymbol{\tau}}$	stacked torques
$\boldsymbol{\pi}_{\text{full}}$	full vector of the dynamic parameters
$\boldsymbol{\pi}_{\text{b}}$	base vector of the dynamic parameters
$\boldsymbol{\pi}_{\text{LB}}$	lower bounds on the dynamic parameters
$\boldsymbol{\pi}_{\text{UB}}$	upper bounds on the dynamic parameters
$\mathcal{S}_i$	ellipsoid in homogeneous form
$\boldsymbol{Q}_i$	matrix that define an ellipsoid
ς	percentage w.r.t the number of samples
ς_t	percentage w.r.t the torques
N_s	number of samples
χ^2	reconstruction error
$\bar{\boldsymbol{Y}}_{\text{full}}$	stacked full regressor
$\bar{\boldsymbol{Y}}_b$	stacked base regressor
$\hat{\boldsymbol{\pi}}_b$	estimate vector of base dynamic parameters
$\hat{\boldsymbol{\pi}}_{\text{full}}$	estimate vector of full dynamic parameters
$\boldsymbol{I}_3$	identity matrix 3×3
ν	statistical degrees of freedom
σ_π	standard deviation
$\hat{\boldsymbol{H}}$	covariance of the parameter estimates
$\mathcal{S}$	state space

$\mathcal{A}$	action space
p	transistion probability
r	reward
γ	discount factor
π	policy
α	learning rate
δ	temporal difference error
$Q_\pi(s,a)$	action-value function
$Q_*(s,a)$	optimal action-value function
$\mathcal{G}$	directed graph
$\mathcal{V}$	set of nodes
$\mathcal{X}$	set of vertices
u_j	relationship between input and hidden
$w_{i,j}$	weight between neurons i and j
$h(\cdot)$	activation function
z_j	output hidden layer
$\boldsymbol{w}$	weights of neural network
E	error function
y_k	ground truth output
$\hat{y}_k$	predicted output
$\boldsymbol{p}_{t,0}$	initial position
$\boldsymbol{p}_{t,f}$	final position
$\mathcal{O}$	set of obstacles
N_o	number of obstacles
T	target object
$\mathcal{S}_u$	sequence of objects
$\boldsymbol{\eta}_{\mathrm{ee,d}}(t)$	time-varying trajectory
$\boldsymbol{\eta}_{\mathrm{ee},0}$	end-effector initial position
$\boldsymbol{\eta}_{\mathrm{ee,d}}$	end-effector desired position
g_k	feasibility action signal
$\mathcal{E}_{\max}$	maximum number of episode
E_h	episode
$\mathcal{S}_{s_k}$	sequence of relocated objects
χ_k	edge between s_k and s_{k+1}
ξ_t	number of possible sequences
ε	epsilon value
ε_0	initial epsilon value
$\varepsilon_{\min}$	minimum epsilon value
s_H	root node
$\overline{Q}_h$	sum of all edge values
β	threshold on steady state condition

$G(z)$	transfer function in $\mathcal{Z}$ domain
g	static gain
$a_k^\star$	optimal action
$\Phi_{\mathcal{S}_u}$	cost function to minimize
Φ_i	weigth of object
$\overline{Q}_\infty$	steady state value
f_x	focal length x
f_y	focal length y
c_x	optical center x
c_y	optical center y
d_s	depth scale
M_s	mesh scene
M_g	mesh gripper
$d_{\min}$	minimum distance
$\mathit{\Pi}$	minimum function
Δ_c	crop filter
γ_c	feedback on collision
$\mathcal{D}_{\text{depth}}$	dataset made of depth images
$\mathcal{D}_{\text{pointcloud}}$	dataset made of point clouds
$\boldsymbol{I}_s$	depth image of scene $(640, 480)$
$\boldsymbol{I}_g$	depth image of gripper $(640, 480)$
$\boldsymbol{I}_c$	complete depth image $(640, 480)$
$\boldsymbol{I}_s^*$	cropped depth image of scene $(156, 156)$
$\boldsymbol{I}_g^*$	cropped depth image of gripper $(156, 156)$
$\boldsymbol{I}_c^*$	cropped complete depth image $(156, 156)$
W	weight size image
H	heigth size image
$\boldsymbol{PC}_s$	point cloud of the scene
$\boldsymbol{PC}_g$	point cloud of the gripper
$\boldsymbol{PC}_c$	complete point cloud
$\boldsymbol{p}_d$	desired position
$\dot{\boldsymbol{p}}_d$	feed-forward desired linear velocity
$\dot{\boldsymbol{\omega}}_d$	feed-forward desired angular velocity
$\boldsymbol{e}_p$	position error
$\boldsymbol{e}_o$	orientation error
$\mathcal{F}$	set of algorithms
$\mathcal{C}$	set of constraints
$\boldsymbol{\zeta}$	labels
$\boldsymbol{q}_0$	initial joint configuration
c_i	class i
w_{m}	manipulability functional

w_{jl}	joint limits functional
$\boldsymbol{u}$	data input
$P(y\|\boldsymbol{u})$	probability output given input
$\mathcal{N}$	normal distribution
μ	mean value
W	set of neural network weights
$f_{\text{softmax}}(\cdot)$	softmax operator
$\mathcal{L}_{\text{NLL}}$	Negative-Log Likelihood function

Chapter 1

Introduction

1.1 Motivation

According to the report of World Robotics 2022 provided by the *International Federation of Robotics* (IFR) a record of 517,385 new industrial robots were installed in factories around the world in 2021. This represents an increase of 31% with respect to the previous record (reached in pre-covid era, 2018) of 22%. Italy is the second biggest marker in Europe back to Germany, followed by France. On the other hand, Asia continues to be the largest market in the world for industrial robots; the 74% of all newly implemented robots in 2021 were installed in Asia.

The high demand for robotic systems is due to their extensive use in medicine and healthcare, aerospace, and industrial fields. In recent years, hospitals have become increasingly dangerous for nurses because of COVID-19. At the same time, people infected by the virus need to receive treatments for their pathologies, causing different risks for hospital specialists. This has lead to a new and important re-definition of assistants into the hospital areas, increasing the usage of robots. Hospitals have been using robots for disinfecting internal rooms, moving autonomously and killing the virus. The Danish company *UVD Robots* has made robots capable to disinfect patient rooms in hospitals.

Other cases of interest include the assistive robotics. Nowadays, people with mobility impairments need to have a costant and helpful caregivers for daily life operations such as drinking, dressing or feeding; therefore, robots can support

users in a wide range of applications. In support of this, there is a necessity to develop assistive robotics architectures aimed to handle these complex situations, performing basic life tasks [1].

Furthermore, pre-programmed routines such as drilling, fastening, and metal parts welding, in which precision and rigidity are required and where robots can outperform humans in terms of time and cost, justify their wide use. In 2004 the company *KUKA Deutschland GmbH* introduced the first *Cobot*, a lightweight robot born from a collaboration with the *German Aerospace Center* (DLR). Cobots are similar to the traditional industrial robots, mechanical arms that can be programmed to perform various tasks in the factory setting. They are designed to share the workspace with humans.

Each of the tasks mentioned above focuses on the interaction between robot and the real world. From an industrial perspective, of particular interest is the robotic picking and packaging, where it is necessary to increase the accuracy, repetibility and speed along with lowering production costs. Examples are reported in Fig. 1.1.

Figure 1.1: Overview of different contexts: top-left an healthcare application; top-right an assistive one application; at bottom-left an industrial application in automotive and bottom-right an application with cobots.

This thesis work focuses on the combination of learning and control algorithms,

respectively. Specially, first of all, an architecture made of a Task Planner (based on Reinforcement Learning) and a Motion Planner to handle the planning of trajectories for retrieving an object from clutter is proposed; then, a tool for detecting collisions between robot and environment during the plan of a trajectory is addressed. Finally, an intelligent control based on Supervised Learning approach for decide the tasks to optimize in the null-space of redundat robot is proposed.

1.2 State of the Art

In recent years, there have been important changes in production chains in the industry, requiring robots capable of adapting to dynamic and unstructured environments. Indeed, the research with focus on hybrid architectures made of model-based control and machine learning decision making systems has grown in the latest years. A complete review on the state of the art is reported in [2]. Nowadays, robotics systems are involved always more in different and complex tasks, solving them in autonomous and/or semi-autonomous way. It means that, in scenarios with unpredictable changes (e.g. sudden moves of human in a shared workspace with the robot, objects with a wrong configuration in the pick and place problem), intelligent and adaptive behaviours are needed.

For years, the aim has been to define a specif robots-behavior, but it is not achievable directly. One possibility to plan the behavior of a robot was proposed in the 1979 and it is called *Motion Planning* (MP) problem [3]. Its goal is to find an optimal path for the robot starting from an initial to a final configuration avoiding obstacles. Typically, this problem is solved considering constrained optimization [4] and sampling-based algorithms [5]. As it is known in literature, a planning algorithm can be divided in classes: *complete*, where if there is solution it is capable to find it, otherwise the algorithm fails reporting the result; *semicomplete*, where if there is no solution, the algorithm can continue to search it endlessly; *resolution complete*, where if there is a solution it finds one, whereas, if there is no one solution, it terminates returning that there is no solution within a specified resolution exists; finally, *probabilistically complete*, where if a solution exists, the associated probability that the algorithm finds it tends to one as the number of iterations tends to infinity.

Since isolated motion planning typically assumes a fixed configuration space, an integrated description that orchestrates an high level representation is proposed referred to as the *Task Planning* (TP).

Its aim is to compute sequences of actions to guide the agent in solving a complex desired task. The actions can either be discrete or continuous: the first ones contain a finite set of options which the agent can choose, e.g. move left, right, up or down, whereas the second ones are represented with a value, such as, move left of 1 cm, and the number of actions can be very large.

To solve various complex tasks, there is a way to combine the two previous approaches into a final one called *Task-Motion Planning* (TAMP) [6]. This architecture is highly useful because it gives the possibility to plan the sequence of actions for solving the problem. To make it work, it is necessary a middle layer that maps information between two different worlds.

Particularly interesting is the robotic objects manipulation [7] because of the high number of application contexts, e.g. in manufacturing industry driven by Industry 4.0 requirements. Currently, due to the high dexterity and reasoning necessary to solve the task, it is performed mainly by human workers [8].

The real challenge is based on the unstructured nature of the involved environments combined to the $\mathcal{NP}$-hard complexity of the task [9, 10], providing the task intractable [11, 12, 13, 14, 15]. Based on this, Artificial Intelligence (AI) can provide a great help for handling combinatorial problem in a large state space, focusing the interests of many researchers [16]. A brach of AI is *Machine Learning* which uses structures similar to human brain, capable to select the best actions to act in particular configurations, reducing the computational burden for solving the task.

Early exploration in robotic grasping primarily relied on analytical methods and 3D reasoning to anticipate the ideal locations and configurations for grasping objects [17, 18, 19]. They assumed complete knowledge of the objects, e.g., their 3D models, surface friction properties, and mass distribution.

In the years 2015 and 2016, *Amazon Picking Challenge* (APC) took place; it is a competition where the ability of the robots for retrieving objects from

cluttered environments are tested. In this competion, the task consists of picking 25 objects from a warehouse shelf and placing them into a storage container with a time constraint set to 20 minutes [20]. Grasping from the top was forbidden, making the the task more challenging by using higher shelves.

Then, the robots knew the type of objects, but the arrangment was unknown. Initially, the teams in the challenge proposed solutions that focused on conventional perception and robotics problems. In subsequent years, the challenge received several modifications, changing the environments and leading to the task being more adaptable to the robotic systems [21].

Neverthless, grasping objects in a cluttered environment is also extremely challenging due to the lack of collision free grasp affordances. Hitting and shifting objects during the motion could change the state of the entire system, making the desired object unreachable, resulting the task unresolvable. Additionally, the grasp can sometimes fail if the desired object is obstructed or if it assumes singular configuration (e.g. extremity of the bin during a pick and place task).

Robotic grasping phase can be divided in three sub-phases: *objects pose estimation, grasp pose* and *planning* toward the desired object [22]. The first part relies on the perception module, which is made of sensors, such as camera that uses RGB, RGB-D or Point Cloud and it is capable to extract information about the environment. Despite the high-quality devices, it is still an open challenge, as the encoding of information is strictly related to light conditions and occlusion present in real scenarios. For the second aspect of the grasping phase, the pose estimation is foundamental for a good score of the executed action. Several techniques have been proposed to estimate the optimal point for grasping known objects, such as exploiting geometric features extracted from the CAD model or using fine-tuning approaches in the real world. Lastly, the planning for object grasping involves computing a collision-free trajectory for the robot.

In the state of the art, the problem of retrieving an object target from clutter has been mainly addressed by relying on geometric methods that potentially make use of specific heuristics to reach effective solutions while reducing the computation complexity. In [11], the authors propose a geometric method

with the aim of minimizing the number of obstacles to be relocated, and consequently the time (or the energy) spent to the scope. Their algorithm is shown to be complete and efficient, outperforming other methods in terms of execution time, but the optimality of the solution is not guaranteed. Similarly, the planner in [23] solves the rearrangement problem exploiting dynamic nonprehensile actions guaranteeing only the feasibility of the plan. In [24] a probabilistic solution, scalable with respect to the number of objects, is analyzed. In the proposed method, the authors choose constraints to take into consideration depending on the motion feasibility. In case of an unfeasible motion, it is necessary to remove some of the constraints or to increase the motion planning timeouts in order to make the algorithm complete.

In [25], the authors provide a geometric method for multiple objects reorganization in clutter, minimizing the number of objects to move. Differently from the works mentioned above, here, the optimal solution is obtained by splitting a continuous 2D plane into discrete cells, that are then used in a hybrid planner for generating the motion plan. However, the elementary motions that compose the plan are determined not considering the reachability of the objects and the robot kinematic constraints; for this reason, the plan might not be feasible for real robots. Alternatively, in [26] the authors consider a novel approach based on TAMP for unknown object rearrangements, relying on graphs that are built online in order to retrieve the target object. Their approach does not make use of heuristics in the exploration, potentially resulting in a large graph and not computationally efficient solution.

Recently the TAMP problem, including object manipulation and grasping in a cluttered environment, have made use of machine and deep learning [16, 27] techniques. For instance, in [28] a neural approach is proposed, with a particular focus on considering unknown objects. In detail, they describe the *Neural Rearrangement Planning*, an approach for rearranging unknown objects from perceptual data in the real world. It is capable of rearranging previously unseen objects, exploiting segmented point clouds coming from a RGB-D sensors.

Among the Machine Learning techniques, Reinforcement Learning (RL) has been considered by many researchers as a base for solutions of this kind of problems [29]. It is a technique based on a figure named *agent*, which interacts

with a space named *environment* choosing a possible *action* that provides a feedback named *reward*. In particular, the agent objective is to collect the maximum reward values over time. The aim of this learning technique is to provide robots abilities like learning, improving over time, adapting and reproducing tasks [30]. The approach proposed in [16] combines the action planning with a goal-independent reinforcement learning approach considering a sparse reward. The problem is to find high-level actions to send to a low-level layer as trajectories for robots to solve random puzzles. The obtained results prove that the proposed approach is able to solve the task if the considered solution space is not too large. In [31], the authors propose a data-driven method to be employed in case of an occluded target. They assign a probability distribution to the target object pose considering partial observations and an occlusion-aware heuristic, and then they exploit a receding horizon approach. They present an architecture that allows learning a generative model used to update the target pose probability distribution in a continuous action space.

There is a large amount of approaches that exploit a visual input, e.g. RGB or RGB-D images, mapping it into feasible actions to bring the agent towards the goal [32, 33]; to the scope, they make use of Convolutional Neural Networks (CNNs) combined with a policy-based Reinforcement Learning [34, 35, 36].

While in highly-structured environment the robot motion might be offline planned, this approach is deemed to fail when coping with unstructured and dynamic scenarios such as assistive, medical or industrial, to name a few. In these settings, it is of the utmost importance to equip the robot with the ability to perceive the environment and detect collision in the shortest time in order to have the robot re-plan its trajectory accordingly.

Unfortunately, as it is known in literature, the time necessary for detecting a collision is the planners *bottleneck* [37]. It could be very slow, and it depends from the complexity of the scenario. According to [38], a generic motion planner could be not capable to adopt a re-plan strategy in short time if the environment changes during the motion of the robot. Indeed, in [39, 40, 41], the authors solve an objects relocation in cluttered environment and, according to the literature, the planner spent different milliseconds to find a free-path (if it exists), using the classical geometrical approaches for the collision detection.

The authors in [42] present a sampling-based motion planner capable to improve the performance of classical optimal motion planning that use RRT* algorithm. They show that the proposed planner is able to find a fast initial path and, then, decreases the cost of this path in an iterative way. Even if they overcome limitations of the classic motion planning in high-dimensional space, there are some heuristics to define.

One of the most geometric-based collision detection used in the motion planning context is the *Flexible Collision Library* (FCL) [43], which combines different techniques for a fast and accurate collision checking computation. Knowledge of the mesh for objects present in a scene ensures fast and accurate methods for collision checking among objects [44].

The majority and prevalent approaches regard the polyhedral models and a large part of the commonly used techniques are based on *Bounding Volume Hierarchies* (BVH). The complexity of the BVHs approaches is related to the involved polyhedra. In contrast, other methods based on *Nearest-neighbor* technique are used, even if the queries suffer in high-dimensional space, making it unfeasible in some scenarios [45]. Thus, these kinds of modeling through approximation is not applicable in situations where high precision is required [46].

In the recent years, Machine Learning has been employed to solve different complex-problems. It has long been dominated by approaches mainly based on images employing the well-known Convolutional Neural Networks [47], even if, several tasks can be solved handling 3D point clouds and 3D voxel data representation. Indeed, since 2017, a challenge problem for computer vision using 3D data is solved in [48], where the authors proposed an architecture based on point cloud capable of classifying and segmenting 40 classes of objects on the dataset ModelNet40 [49], which requires high computation and memory resources. In [50], an end-to-end trainable architecture for point cloud for the 3D detection called VoxelNet is proposed; it is capable to operate directly on sparse points, outperforming the state-of-the-art LiDAR based 3D detection; whereas the authors in [51] introduced VoxNet, a network that uses volumetric representation to process the 3D data for a robust object recognition.

Approaches like these could help model-based systems, accelerating the resolution of multiple tasks, as well as grasping and collision checking; these approaches have often proved to outperform the classic ones, partially or totally based on a model description [52]. Promising results are arriving from data-driven algorithms based on depth images for robotic grasping and state estimation based on classification. It is very usual to decide a-priori a list of possible grasps considering the geometric and physical models of the objects [53, 19]; the authors in [54] demonstrate that, using convolutional neural networks on RGBD images, it is possible to find the optimal grasp for an object on the Cornell Grasping Dataset. In [55], given the task of learning robotic grasping based on depth images and gripper force feedback, the authors implement an algorithm that reduces the quantity of the data for training the model. In [56] an end-to-end network that generates a distribution of 6-DoF parallel-jaw grasps from a depth is addressed; they obtained 90% of accuracy on objects never seen before, using ≈ 17 million of simulated grasp.

Tasks related to the objects in cluttered scenes are a real challenge because of necessity to consider, simultaneously, the correct grasp-phase and any collision with the rest of the environment. Computer vision researchers have considered Deep Learning techniques enabled to work well with 3D object geometries, suffering about efficiency for handling large number of collision queries, whose are fundamental for optimization and control in robotics [57]. In [58] the authors demonstrate that tools for collision detection based on robotics motion planning may be accelerated by performing collision checks considering Machine Learning model; their model named Fastron is capable to model the configuration space of robot manipulators, which can be used as a proxy collision detector, by replacing the standard geometric collision approaches. The results show that in a simulated environment, their method outperform FCL, but it requires a number of samples strictly related to the dimensionality of the space of interest. Researchers from NVIDIA and Berkley [59], proposed a neural network named SceneCollisionNet that consider raw point clouds for objects and scene, adding the pose of the final one as input and returns the likelihood about the object collision as output, exploiting a voxel representation. They obtained 93% of accuracy on 2 million of collision queries investing $10\mu s$, thus

to be $10x$ faster than the classic FCL library.

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 [60].

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 [61]. Most often the tasks are structured on defined priorities [62] and this approach is referred to as strict task priority [63, 64, 65]. 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 [66].

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 [67]. A possible solution might be changing the tasks or their priorities dynamically during the robot motion, but, as reported in [64], 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 in [68].

The authors in [69] 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 [70], generalizing w.r.t new tasks through transfer policy learning. Again, in [71] a

dynamically-consistent generalized hierarchical control is presented. For each pair of tasks, they choose between a soft or hard priority with one task having a null effect on the other one. Otherwise, in [72] a method for learning task hierarchies is proposed, where the authors show robot potentiality to reproduce the learned priorities in new scenarios. Similarly, in [73] 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 [74], 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 [75] or other complex and combinatorial problems [76, 39, 41]. In [67] 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.

1.2.1 Thesis Outline and Contribution

The contribution of this thesis is manifold. The first one is an overview on kinematic and dynamic of robot manipulators, with a particular focus on dynamic identification. It lays the foundation for the robot control.

Even so, the main one is related to the retrieving of a target object from cluttered environment, which result to be $\mathcal{NP}$-hard with respect to the number of the objects. The proposed architecture to solve the problem mentioned above is made of layers, that are composed by machine learning decision making and motion planner system, respectively. It is shown that the proposed architecture is able to find the optimal sequence of objects to relocate and for checking the collision that could speed up the time spent by the motion planner. Finally, the control part is addressed, exploiting a learning-based approach to enlarge the dextrerous workspace in case of redundant robot control. A full architecture that could consider all the modules proposed in the chapters of this thesis is reported in Fig.1.2.

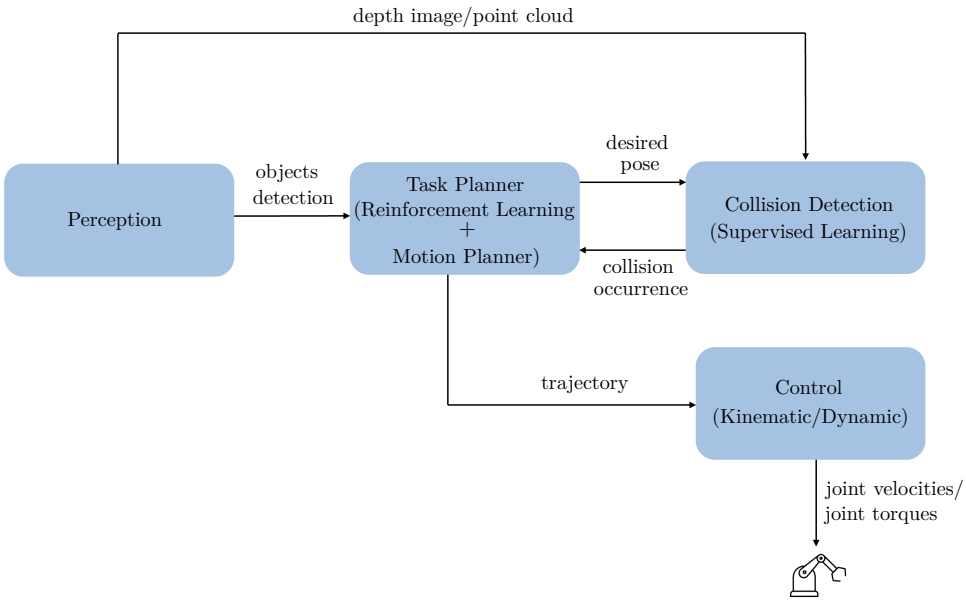

Figure 1.2: The proposed architecture for this work is composed of several blocks. The perception one provides the objects pose estimation and generates the depth image or point cloud of the scene. The outputs are then used by the task planner and collision detection systems. The task planner system uses a combination of reinforcement learning and motion planner to devise an object relocation procedure. The collision detection system exchanges information with the task planner system to generate a collision-free trajectory. Finally, the control block computes joint velocities or torques for the robot.

Particularly, this thesis is divided in the following chapters:

- **Chapter 2 - Background**
  This chapter provides mathematical details on kinematics and dynamics of robot manipulators, starting from basic concepts. Specially, there is a focus on the identification of a dynamic model for robot manipulators, handling a comparison among the main identification procedures, sharing with the community the first consistent dynamic model for the KINOVA Jaco^2 7-DoFs robot manipulator.

- **Chapter 3 - Task-Motion Planning via Reinforcement Learning**
  The second chapter is focused on a Task-Motion Planning architecture made of Motion Planner and a Reinforcement Learning agent to solve the

problem of retrieving an object from clutter. In this regard, an algorithm based on Q-Learning, named Q-Tree is proposed in literature, which reduces the computational load of the problem.

The architecture is validated experimentally way on a KINOVA Jaco^2 7-DoFs robot manipulator present in the Industrial Automation Laboratory at University of Cassino and Southern Lazio, Italy.

- **Chapter 4 - Deep Learning-based Robot Collision Detection**
  Another important aspect of this thesis is a proposal of a tool based on Deep Learning for the robot collision detection.

  It is a collision checker based on the main data representation such as depth images and point clouds is proposed; these approaches are based on deep neural networks capable to handle depth images and point clouds. This activity is carried out in collaboration with the Technology & Innovation Center of KUKA Deutschtland GmbH and the entire architecture is validated on KUKA Agilus, an industrial robot manipulator of their laboratory in Augsburg, Germany.

- **Chapter 5 - Deep Learning for Task Priority**
  The choice of the function to optimize as secondary control objective affects the reachable workspace of the robot. In this work 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 is addressed. The architecture is validated with numerical simulations on a simple 3-link robot arm.

- **Chapter 6 - Conclusion and Future Work**
  This chapter concludes the thesis work, emphasing once again its contributions. Each chapter is summarized, stressing the main concept and showing the limitations of the proposed methods to deepen in possible future works.

1.2.2 Publications

a) **G. Golluccio, G. Gillini, A. Marino, G. Antonelli**, *Robot Dynamics Identification: A Reproducible Comparison with Experiments on the Kinova Jaco*[2], IEEE Robotics & Automation Magazine, 2020

   – Selected for Workshop and Tutorial presentation at the IEEE International Conference on Robotics and Automation (ICRA), May 23 - May 27, 2022, Philadelphia, USA

   – Selected for Workshop presentation at the IEEE International Conference on Intelligent Robots and Systems (IROS), September 27 - October 1, 2021, Prague, Czech Republic

   – Selected for presentation at the IEEE International Conference on Robotics and Automation (ICRA), May 30 - June 5, 2021, Xi'an, China

b) **G. Golluccio, D. Di Vito, A. Marino, G. Antonelli**, *Task-motion Planning via Tree-based Q-learning Approach for Robotic Object Displacement in Cluttered Spaces*, International Conference on Informatics in Control, Automation and Robotics (ICINCO), 2021

c) **G. Golluccio, D. Di Vito, A. Marino, G. Antonelli**, *Robotic Weight-based Object Relocation in Clutter via Tree-based Q-learning Approach using Breadth and Depth Search Techniques*, IEEE International Conference on Advanced Robotics (ICAR), 2021

d) **G. Golluccio, P. Di Lillo, D. Di Vito, A. Marino, G. Antonelli**, *Objects Relocation in Clutter with Robot Manipulators via Tree-based Q-Learning Algorithm: Analysis and Experiments*, Journal of Intelligent & Robotic Systems, Springer, 2022

e) **G. Golluccio, P. Di Lillo, A. Marino, G. Antonelli**, *When Local Optimization is Bad: Learning What to (Not) Maximize in the Null-space for Redundant Robot Control*, IEEE International Conference on Control, Decision and Information Technologies (CODIT), 2023

f) (to be submitted) **G. Golluccio, A. Marino, G. Antonelli**, *Learning-based Robot Collision Detection in Cluttered Environments*

Other Contributions

g) **C. Carissimo, G. Cerro, L. Ferrigno, G. Golluccio,A. Marino**, *Development and Assessment of a Movement Disorder Simulator Based on Inertial Data*, Sensors, 2022

h) **C. Carissimo, L. Ferrigno, G. Golluccio, A. Marino, G. Cerro** *Parkinson's Disease aided Diagnosis: Online Symptoms Detection by a Low-Cost Wearable Inertial Measurement Unit*, IEEE International Symposium on Medical Measurements and Applications (MeMeA), 2022

i) **C. Carissimo, G. Cerro, L. Ferrigno, G. Golluccio, A. Marino**, *Realization and Validation of Data IMU Simulator for Detection and Classification of Tremor in Parkinson's disease*, Conference of Italian Association of Telemedicine and Medical Informatics (AITIM), 2021

Chapter 2

Background

This chapter provides an overview of kinematics and dynamics concepts for robot manipulators, including the main identification techniques. Additionally, it explains the concepts of Machine and Deep Learning, covering a range of approaches from Reinforcement to Supervised Learning.

2.1 Robot Modelling and Identification

2.1.1 Kinematics

Position and Orientation of a Rigid Body

The rigid body is an ideal representation of a body that does not deform or change its shape. In general, it is defined as particles with fixed distances among them during a motions of the body. It is completely described in the space by its pose (position and orientation) with respect to a frame $\boldsymbol{\Sigma}_I$ as

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{p} & \boldsymbol{\varphi} \end{bmatrix}^{\mathrm{T}} \in \mathbb{R}^6 \,, \tag{2.1}$$

where $\boldsymbol{p} = \begin{bmatrix} p_x & p_y & p_z \end{bmatrix}^{\mathrm{T}} \in \mathbb{R}^3$ and $\boldsymbol{\varphi} \in \mathbb{R}^3$ represent position and orientation of the rigid body.

Regarding the orientation, it is possible to use different representations. A very known and used representation for the orientation are the *Euler Angles*, descripted as following

$$\boldsymbol{\varphi} = \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^{\mathrm{T}} \in \mathbb{R}^3 \,, \tag{2.2}$$

where the ϕ, θ and ψ are called angles of *roll*, *pitch* and *yaw*. It is a minimal representation of the orientation, made of three independent parameters.

Starting from this representation, given a base frame $\boldsymbol{\Sigma}_B$ it is possible to obtain the related *rotation matrix* using the ZYX-convention in current frame as

$$\boldsymbol{R}_B^I = \begin{bmatrix} c_\psi c_\theta & -s_\psi c_\phi + c_\psi s_\theta s_\phi & s_\psi s_\phi + c_\psi c_\phi s_\theta \\ s_\psi c_\theta & c_\psi c_\phi + s_\phi s_\theta s_\psi & -c_\psi s_\phi + s_\theta s_\psi c_\phi \\ -s_\theta & c_\theta s_\phi & c_\theta c_\phi \end{bmatrix} \tag{2.3}$$

where $c_* = \cos(*)$ and $s_* = \sin(*)$.

Althought it is a minimal representation, different problem can be shown. In particular, this type of representation is affected by an issue known as *singularity representation*. More in detail, it means that the conversion from the rotation matrix to the euler angles representation is not defined when θ assumes values as $\pm\frac{\pi}{2}$ rad. For this reason, in some cases this representation cannot be used and it could be necessary to use a non-minimal one: *Unit Quaternion*.

The latter is a non-minimal due to the usage of four parameters for the representation of a rigid body orientation. Specially, considering the orientation between two frames with same origin as following

$$\boldsymbol{R}_e(\alpha) = \cos\alpha \boldsymbol{I}_{3\times 3} + (1 - \cos\alpha)\boldsymbol{e}\boldsymbol{e}^{\mathrm{T}} - \sin\alpha \boldsymbol{S}(\boldsymbol{e}) \tag{2.4}$$

with $\boldsymbol{e}$ unit vector to identify the axis in which has to be rotated of the α angle to align the two frames, $\boldsymbol{I}_{3\times 3}$ is the identity matrix and $\boldsymbol{S}$ is the skew matrix performing the cross product between two (3×1) vectors, the unit quaternion is defined as

$$\mathcal{Q}(\eta, \boldsymbol{\varepsilon}) \tag{2.5}$$

where

$$\eta = \cos\frac{\alpha}{2}, \quad \boldsymbol{\varepsilon} = \boldsymbol{e}\sin\frac{\alpha}{2} \tag{2.6}$$

are the scalar and vector part, respectively, satisfying the following condition:

$$\eta^2 + \boldsymbol{\varepsilon}^{\mathrm{T}}\boldsymbol{\varepsilon} = 1\ . \tag{2.7}$$

Furthermore, since the two quaternions $\mathcal{Q}(\eta, \boldsymbol{\varepsilon})$ and $\mathcal{Q}(-\eta, -\boldsymbol{\varepsilon})$ represent the same orientation, aimed at guaranteeing the representation uniqueness, in Eq. (2.6) it is assumed $\eta \geq 0$ that corresponds to $\alpha \in [-\pi; \pi]$.

Direct Kinematics

A manipulator is made of rigid bodies series (links) connected by mechanical pairs or joints, which can be of two types: revolute and prismatic. The entire structure of the robot forms a kinematic chain. On one end of the chain, there is a constrained base, while on the other side an end-effector (gripper) is mounted to allow the manipulation of objects in space.

The robots structure is characterized by a number of degrees of freedom (DOFs) which determine its posture. Each degree of fredoom is related to a joint to which it is associated a variable.

The direct kinematics determines the relationship between the end-effector pose $\boldsymbol{x}_{ee}$ of the robot manipulator, which is expressed in the *operational space*, with the joint variables $\boldsymbol{q}$ in the *joint space*, as following

$$\boldsymbol{x}_{ee} = \boldsymbol{k}(\boldsymbol{q}) \ , \tag{2.8}$$

where $\boldsymbol{x}_{ee}$ is a vector $m \times 1$, $\boldsymbol{q}$ is a vector $n \times 1$ and $\boldsymbol{k}$ is a non-linear function that allows computation of the operational space variables from the knowledge of the joint space variables.

An example of robot manipulator is reported in Fig. 2.1.

Inverse Kinematics

Computing the joint configuration given $\boldsymbol{x}_{ee}$ is not trivial problem. For simple robot structures, there are several geometric approaches that can be exploited. However, when there are a large number of DoFs, there can be infinite solution. For this reason, a numerical method named *differential inverse kinematics* is used. Therefore, the aim of differential kinematics is to find the relationship

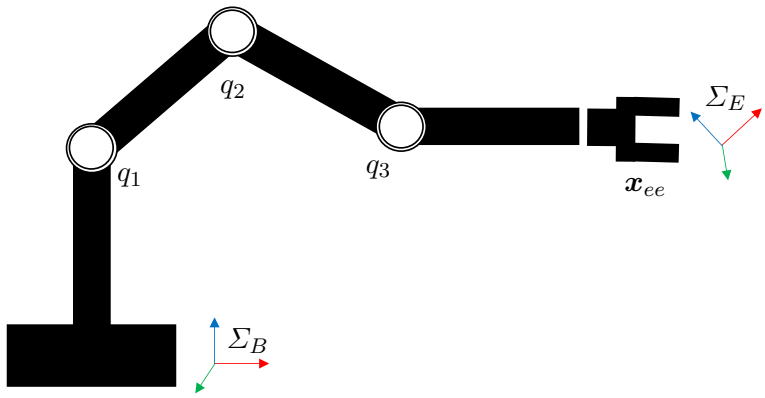

Figure 2.1: Example of robot manipulator in the space.

between the joint velocities $\dot{\boldsymbol{q}}$ and the end-effector linear $\dot{\boldsymbol{p}}_{ee}$ and angular velocities $\boldsymbol{\omega}_{ee}$ as

$$\dot{\boldsymbol{p}}_{ee} = \boldsymbol{J}_P(\boldsymbol{q})\dot{\boldsymbol{q}} \tag{2.9}$$

$$\boldsymbol{\omega}_{ee} = \boldsymbol{J}_O(\boldsymbol{q})\dot{\boldsymbol{q}} \tag{2.10}$$

where $\boldsymbol{J}_P$ is the $3 \times n$ matrix relative to the contribution in the joint space for the linear velocity of the end-effector, whereas $\boldsymbol{J}_O$ is the $3 \times n$ for the angular velocity. The *geometric jacobian* matrix $\boldsymbol{J}$ is obtained as

$$\boldsymbol{J} = \begin{bmatrix} \boldsymbol{J}_P \\ \boldsymbol{J}_O \end{bmatrix} . \tag{2.11}$$

Combining the Eq. (2.9), Eq. (2.10) and Eq. (2.11), a compact form is

$$\boldsymbol{v}_{ee} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} . \tag{2.12}$$

Equation (2.12) is linear in the velocities and it is necessary to invert it in order to compute $\dot{\boldsymbol{q}}$ and $\boldsymbol{q}$. If the the rank is full the relationship becomes

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{-1}\boldsymbol{v}_{ee} , \tag{2.13}$$

where it is possible to compute the joint configuration as

$$\boldsymbol{q}(t) = \int_0^t \dot{\boldsymbol{q}}(\rho)d\rho + \boldsymbol{q}(0) , \tag{2.14}$$

with the related numerical *Euler integration*

$$\boldsymbol{q}(t_{k+1}) = \boldsymbol{q}(t_k) + \dot{\boldsymbol{q}}(t_k)\Delta t\,. \tag{2.15}$$

Indeed, when a robot manipulator has more DoFs than necessary to perform a task, it is called *redundant* and the Eq. (2.12) admits infinite solutions. Thus, an optimization problem is defined as following

$$\min_{\dot{\boldsymbol{q}}} \quad g(\dot{\boldsymbol{q}}) = \frac{1}{2}\dot{\boldsymbol{q}}^{\mathrm{T}}\dot{\boldsymbol{q}}\,. \tag{2.16}$$

The optimization problem in Eq. (2.16) can be solved using the *Lagrange multipliers*

$$g(\dot{\boldsymbol{q}}, \boldsymbol{\lambda}) = \frac{1}{2}\dot{\boldsymbol{q}}^{\mathrm{T}}\dot{\boldsymbol{q}} + \boldsymbol{\lambda}^{\mathrm{T}}(\boldsymbol{v}_{ee} - \boldsymbol{J}\dot{\boldsymbol{q}})\,, \tag{2.17}$$

where $\boldsymbol{\lambda} \in \mathbb{R}^r$ is the vector of Lagrange multipliers. At this point, by imposing

$$\left(\frac{\partial g}{\partial \dot{\boldsymbol{q}}}\right)^{\mathrm{T}} = \mathbf{0} \quad \left(\frac{\partial \mathbf{g}}{\partial \dot{\boldsymbol{\lambda}}}\right)^{\mathrm{T}} = \mathbf{0} \tag{2.18}$$

the optimal solution that locally minimizes the joint velocities is

$$\dot{\boldsymbol{q}} = \underbrace{\boldsymbol{J}^{\mathrm{T}}(\boldsymbol{J}\boldsymbol{J}^{\mathrm{T}}))^{-1}}_{\boldsymbol{J}^{\dagger}}\boldsymbol{v}_{ee} \tag{2.19}$$

with $\boldsymbol{J}^{\dagger}$ right pseudo-inverse of the Jacobian $\boldsymbol{J}$.

The differential kinematics equation in Eq. (2.12) can be characterized in terms of the range $\mathcal{R}(\boldsymbol{J})$ and null $\mathcal{N}(\boldsymbol{J})$ spaces

$$\dim(\mathcal{R}(\boldsymbol{J})) = r \quad \dim(\mathcal{N}(\boldsymbol{J})) = n - r \tag{2.20}$$

mantaining the following relationship without dependency from the rank of the Jacobian $\boldsymbol{J}$

$$\dim(\mathcal{R}(\boldsymbol{J})) + \dim(\mathcal{N}(\boldsymbol{J})) = n\,. \tag{2.21}$$

A scheme of this relationship is reported in Fig. 2.2

Therefore, in the redundant robots the additional DoFs can be exploited to

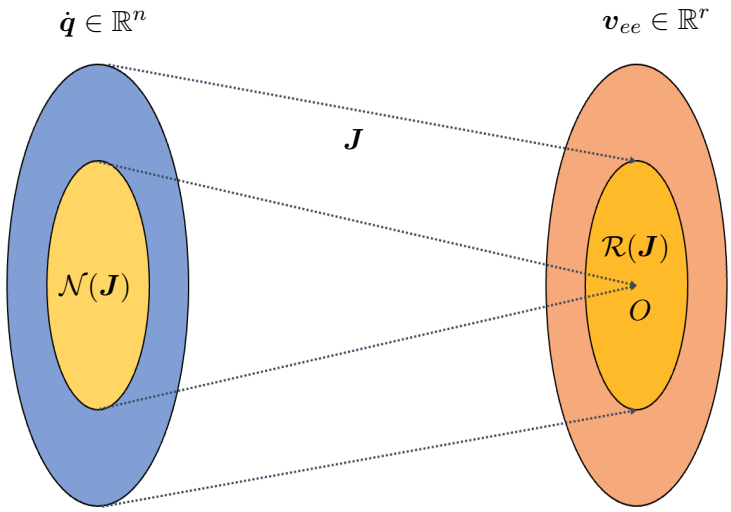

Figure 2.2: Mapping between the velocities in joint and cartesian space.

solve parallel tasks due to the existance of $\mathcal{N}(\boldsymbol{J})$. Thus, the Eq. (2.19) can be rewritten as

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{\dagger}\boldsymbol{v}_{ee} + (\boldsymbol{I}_n - \boldsymbol{J}^{\dagger}\boldsymbol{J})\dot{\boldsymbol{q}}_a \ , \tag{2.22}$$

where the first term is the solution as mentioned above whereas the second term is the homogeneous solution for satisfying additional constraints. More in detail, the term $\dot{\boldsymbol{q}}_a$ assumes the form of

$$\dot{\boldsymbol{q}}_a = k_0 \left(\frac{\partial w(\boldsymbol{q})}{\partial \boldsymbol{q}} \right)^{\mathrm{T}} \ , \tag{2.23}$$

with $w(\boldsymbol{q})$ secondary objective function to consider that can be

- Maximization of manipulability

$$w(\boldsymbol{q}) = \sqrt{\det(\boldsymbol{J}(\boldsymbol{q})\boldsymbol{J}(\boldsymbol{q})^T)} \ , \tag{2.24}$$

- Maximization of distance from joint limits, mantaining the joint value q_i at the average value $\overline{q}_i$ between limits $[q_{i_m} q_{i_M}]$

$$w(\boldsymbol{q}) = -\frac{1}{2n}\sum_{i=0}^{n}\left(\frac{q_i - \overline{q}_i}{q_{i_M} - q_{i_m}}\right)^2 . \tag{2.25}$$

Equation (2.19) can lead to numerical drift during the computation of the position reported in Eq. (2.15).

Recalling the *Closed-Loop Inverse Kinematics* (CLIK) algorithm [77], given a certain desired position $\boldsymbol{p}_d$ and desired quaternion $\boldsymbol{\mathcal{Q}}_d$ for the end-effector, the joint velocities that make the robot fulfill the end-effector pose task can be computed as

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{\dagger}\left(\begin{bmatrix}\dot{\boldsymbol{p}}_d \\ \boldsymbol{\omega}_d\end{bmatrix} + \boldsymbol{K}\begin{bmatrix}\boldsymbol{e}_p \\ \boldsymbol{e}_o\end{bmatrix}\right), \tag{2.26}$$

where $\boldsymbol{e}_p = \boldsymbol{p}_d - \boldsymbol{p}$ is the end-effector position error, $\boldsymbol{e}_o = \boldsymbol{\mathcal{Q}}_d \ * \boldsymbol{\mathcal{Q}}^{-1}$ is the orientation error in terms of quaternion error, $\dot{\boldsymbol{p}}_d$ is a feed-forward desired end-effector linear velocity, $\boldsymbol{\omega}_d$ is a feed-forward end-effector angular velocity.

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

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{\dagger}\left(\begin{bmatrix}\dot{\boldsymbol{p}}_d \\ \boldsymbol{\omega}_d\end{bmatrix} + \boldsymbol{K}\begin{bmatrix}\boldsymbol{e}_p \\ \boldsymbol{e}_o\end{bmatrix}\right) + \boldsymbol{N}\dot{\boldsymbol{q}}_a , \tag{2.27}$$

with $\boldsymbol{N}$ null space of the Jacobian matrix and $\dot{\boldsymbol{q}}_a$ the functional to maximize (see Eq. 2.24 or Eq. 2.25).

2.1.2 Dynamics

The dynamic model of a robot manipulator describes the relationship between joint torques and motion of the structure. It is foundamental for realistic simulation of motion, analysis of manipulator structures, and design of control algorithms, i.e. inverse dynamics control.

There are two ways to compute the model: *Lagrange* and *Newton-Euler* formulations. The first method is conceptually simple and systematic, whereas the second one is possible throught recursive approaches. The latter is more efficient from a computational perspective. Furthermore, an additional way to compute the dynamic model throught the *identification of dynamic parameters* method is presented in this chapter, showing the important contribution obtained in this thesis on this topic.

Lagrange Formulation

The *Lagrangian* of n-DOF manipulator system can be defined as

$$\mathcal{L} = \mathcal{T} - \mathcal{U} \ , \tag{2.28}$$

where $\mathcal{T}$ is the total kinetic energy and $\mathcal{U}$ the potential energy of the system.

The Lagrange equations are expressed by

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{\boldsymbol{q}}}\right)^T - \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{q}}\right)^T = \boldsymbol{\xi} \ , \tag{2.29}$$

where $\boldsymbol{\xi}$ represents the nonconservative forces, i.e., the joint actuator torques and the joint friction torques.

The relationship between the forces applied to the manipulator and the positions, velocities and accelerations of the joints is represented by the Eq. (2.29).

Kinetic Energy

The kinetic energy is obtained by the sum of the contributions given from the motion of each link and from the motion of each joint actuator

$$\boldsymbol{\mathcal{T}} = \sum_{i=1}^{n} \left(\boldsymbol{\mathcal{T}}_{\ell_i} + \boldsymbol{\mathcal{T}}_{m_i}\right) \ . \tag{2.30}$$

The kinetic energy associated with link i is expressed in terms of its contribution as follows:

$$\boldsymbol{\mathcal{T}}_{\ell_i} = \frac{1}{2}\int_{V_{\ell_i}} \dot{\boldsymbol{p}}_i^{*T} \dot{\boldsymbol{p}}_i^{*} \rho dV \tag{2.31}$$

where $\dot{\boldsymbol{p}}_i^{*T}$ denotes the linear velocity vector and ρ is the density of the elementary particle of volume dV. V_{ℓ_i} is the volume of the link i. The kinetic energy associated with each link is composed of both translational and rotational contributions. The kinetic energy of Rotor i can be written as

$$\mathcal{T}_{m_i} = \frac{1}{2} m_{m_i} \dot{\boldsymbol{p}}_{m_i}^T \dot{\boldsymbol{p}}_{m_i} + \boldsymbol{\omega}_{m_i}^T \boldsymbol{L}_{m_i} \boldsymbol{\omega}_{m_i} \tag{2.32}$$

where m_{m_i} is the mass of the rotor, $\dot{\boldsymbol{p}}_{m_i}$ is the linear velocity of the center of mass of the rotor, $\boldsymbol{L}_{m_i}$ is the inertia tensor of the rotor relative to its center of mass, and $\boldsymbol{\omega}_{m_i}$ denotes the angular velocity of the rotor.

The total kinetic energy of the manipulator is determined by summing the individual contributions from each link and rotor.

$$\boldsymbol{\mathcal{T}} = \frac{1}{2} \dot{\boldsymbol{q}}^T \boldsymbol{M}(\boldsymbol{q}) \dot{\boldsymbol{q}} \,, \tag{2.33}$$

where $\boldsymbol{M}(\boldsymbol{q})$ is the inertia matrix, which is symmetric, positive definite, and (in general) configuration-dependent.

Potential Energy

The potential energy stored in the manipulator can be determined by considering the contributions from each individual link and rotor as

$$\boldsymbol{\mathcal{U}} = \sum_{i=1}^{n} \left(\boldsymbol{\mathcal{U}}_{\ell_i} + \boldsymbol{\mathcal{U}}_{m_i} \right) \,. \tag{2.34}$$

In the case of rigid links, the contribution is solely due to gravitational forces and it is given by

$$\boldsymbol{\mathcal{U}}_{\ell_i} = -m_{\ell_i} \boldsymbol{g}_0^T \boldsymbol{p}_{\ell_i} \,, \tag{2.35}$$

where m_{ℓ_i} is the mass of the link, $\boldsymbol{g}_0$ is the gravity acceleration vector in the base frame and $\boldsymbol{p}_{\ell_i}$ is the position of the link.

The contribution of each rotor is

$$\boldsymbol{\mathcal{U}}_{m_i} = -m_{m_i} \boldsymbol{g}_0^T \boldsymbol{p}_{m_i}. \tag{2.36}$$

where $\boldsymbol{p}_{m_i}$ is the position of the rotor. Hence, the potential energy is formalized as

$$\boldsymbol{\mathcal{U}} = -\sum_{i=1}^{n} \left(m_{\ell_i} \boldsymbol{g}_0^T \boldsymbol{p}_{\ell_i} + m_{m_i} \boldsymbol{g}_0^T \boldsymbol{p}_{m_i} \right) . \tag{2.37}$$

Equations of Motion

The Lagrangian from Eq. (2.28) can be rewritten as

$$\boldsymbol{\mathcal{L}}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{\mathcal{T}}(\boldsymbol{q}, \dot{\boldsymbol{q}}) - \boldsymbol{\mathcal{U}}(\boldsymbol{q}) . \tag{2.38}$$

Using the Eq. (2.29), the following result is achieved

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{\xi} , \tag{2.39}$$

where

$$\boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \dot{\boldsymbol{M}}(\boldsymbol{q})\dot{\boldsymbol{q}} - \frac{1}{2} \left(\frac{\partial}{\partial \boldsymbol{q}} \dot{\boldsymbol{q}}^T \boldsymbol{M}(\boldsymbol{q}) \dot{\boldsymbol{q}} \right)^T + \left(\frac{\partial \boldsymbol{\mathcal{U}}(\boldsymbol{q})}{\partial \boldsymbol{q}} \right)^T . \tag{2.40}$$

Finally, it is possible to rewrite the Eq. (2.39) as

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + (\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{F}_v)\, \dot{\boldsymbol{q}} + \boldsymbol{F}_s \mathrm{sgn}(\dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{\tau} - \boldsymbol{J}^{\mathbf{T}}(\boldsymbol{q}) \boldsymbol{h}_e , \tag{2.41}$$

where $\boldsymbol{M}(\boldsymbol{q})$ is the *inertia matrix*, $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ denotes the *Coriolis* forces, $\boldsymbol{F}_v$ is the diagonal matrix of viscous friction coefficients, $\boldsymbol{F}_s$sgn denotes the *Coulomb* friction torques, $\boldsymbol{g}(\boldsymbol{q})$ denotes the *gravity* torques, $\boldsymbol{\tau}$ are the *actuation* torques and $\boldsymbol{h}_e$ is the vector of force and moment exerted by the end-effector on the environment.

2.1.3 Dynamic Parameters

It is of the utmost importance to derive the dynamic model and, then, to have an estimate of its parameters [78]. However, physically infeasible estimates might lead to non-positive inertia matrices at some joint configurations or, in the worst case, in the overall joint space. Such dynamic models would lead to unrealistic simulations and would negatively affect model-based control since the use of a dynamic model with non-positive inertial matrix might lead to an

unstable system [79]. In general, robot dynamic parameters, such as mass and inertia tensors, exhibit physical restrictions that need to be properly addressed to obtain meaningful estimates. However, several identification solutions, being based on regression techniques [80] (with and without considering constraints), generate nonphysical estimates due, for instance, to unavoidable modeling errors, incorrect setup of the identification experiment and to incorrect choice of parameter constraints.

2.1.4 Identification of Dynamic Parameters

For open-chain manipulators consisting of n rigid links connected by n rotational or prismatic joints, the equations of motion can be determined using methods such as the recursive Newton-Euler or Lagrange formulation [81].

In this chapter, the superscript i denotes that the corresponding quantity is expressed with respect to frame i, while no superscript means that it is expressed in the world reference frame.

The mathematical model of a robot manipulator can be written in compact form as

$$\boldsymbol{\tau} = \boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{Y}_{\text{full}}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}})\boldsymbol{\pi}_{\text{full}} \tag{2.42}$$

where $\boldsymbol{M}(\boldsymbol{q}) \in \mathbb{R}^{n\times n}$ is the symmetric and positive definite inertia matrix, $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} \in \mathbb{R}^n$ denotes the Coriolis and centrifugal vector, $\boldsymbol{g}(\boldsymbol{q}) \in \mathbb{R}^n$ is the gravity vector, and $\boldsymbol{\tau} \in \mathbb{R}^n$ is the vector of collected joint torques.

Moreover, the Eq. (2.42) also highlights that the model can be rewritten in alternative form by taking into consideration that it is linear with respect to the dynamic parameters [82] being $\boldsymbol{Y}_{\text{full}} \in \mathbb{R}^{n\times 10n}$ an upper triangular matrix and $\boldsymbol{\pi}_{\text{full}} \in \mathbb{R}^{10n}$ the vector collecting dynamic parameters of each link.

By referring to Table 2.1, considering as case study the KINOVA Jaco^2 robot, the ith link is characterized by 10 parameters [83] which are mass $m_i \in \mathbb{R}$, center of mass $\boldsymbol{m}_{c_i} \in \mathbb{R}^3$ and the inertia matrix $\boldsymbol{L}_i \in \mathbb{R}^{3\times 3}$ referred to the link frame and of which only 6 parameters are considered due to its symmetry, that is $\boldsymbol{l}_i = \begin{bmatrix} l_{i,xx} & l_{i,xy} & l_{i,xz} & l_{i,yy} & l_{i,yz} & l_{i,zz} \end{bmatrix}^{\mathrm{T}} \in \mathbb{R}^6$.

These parameters are stacked into the vector $\boldsymbol{\pi}_i = \begin{bmatrix} m_i & \boldsymbol{m}_{c_i}^{i\mathrm{T}} & \boldsymbol{l}_i^{\mathrm{T}} \end{bmatrix}^{\mathrm{T}} \in \mathbb{R}^{10}$ which represents the vector of dynamic parameters relative to link i, while vector $\boldsymbol{\pi}_{\mathrm{full}}$ in Eq. (2.42) is such as $\boldsymbol{\pi}_{\mathrm{full}} = \begin{bmatrix} \boldsymbol{\pi}_1^{\mathrm{T}} & \boldsymbol{\pi}_2^{\mathrm{T}} & \dots & \boldsymbol{\pi}_n^{\mathrm{T}} \end{bmatrix}^{\mathrm{T}}$.

joint	$\boldsymbol{a}$ [m]	$\boldsymbol{\alpha}$ [rad]	$\boldsymbol{d}$ [m]	$\boldsymbol{\theta}$
1	0	$\pi/2$	0.2755	θ_1
2	0	$\pi/2$	0	θ_2
3	0	$\pi/2$	-0.410	θ_3
4	0	$\pi/2$	-0.0098	θ_4
5	0	$\pi/2$	-0.3111	θ_5
6	0	$\pi/2$	0	θ_6
7	0	0	0.2638	θ_7

Table 2.1: Denavit-Hartenberg table for the KINOVA Jaco^2.

It is worth noticing that the model above is written *at the link side*, i.e., by ignoring the motor inertia and the motor friction. This is rather common for the latest generation of robots defined as *lightweight*, which embeds a torque sensor in each of the joints at the link side [84, 85].

It is well known that, in general, not all the dynamic parameters provide a dynamic contribution [86]. Indeed, by resorting to, for instance, the numerical procedure described in [87] they can be clustered in 3 groups, namely (i) *identifiable*, (ii) *not identifiable* and (iii) *identifiable in linear combination*. By ignoring the not identifiable parameters since they do not contribute to the robot dynamics (e.g., the mass of the base link of fixed base manipulators with a first rotational joint), by removing the corresponding column from $\boldsymbol{Y}_{\mathrm{full}}$ and by properly merging together the columns of the regressor $\boldsymbol{Y}_{\mathrm{full}}$ corresponding to parameters identifiable in linear combination, the regressor-based model can be rewritten as

$$\boldsymbol{\tau} = \boldsymbol{Y}_{\mathrm{b}}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}})\boldsymbol{\pi}_{\mathrm{b}} \,, \tag{2.43}$$

with $\boldsymbol{Y}_{\mathrm{b}} \in \mathbb{R}^{n \times n_b}$ and $\boldsymbol{\pi}_{\mathrm{b}} \in \mathbb{R}^{n_b}$, and where the dimension n_b ($n_b \leq 10n$) depends on the specific robot kinematics and will be specified for the KINOVA Jaco^2 robot in next sections.

The set of all the dynamic parameters, defined as *full* above, is also defined as *standard* by part of the literature; the set of the dynamic parameters providing

a dynamic contribution is defined as *base* parameters or *dynamic coefficients*.

2.1.5 Constraints on the Robot Model

Since robots are physical systems, the vector of dynamics parameters $\boldsymbol{\pi}_{\text{full}}$ (and $\boldsymbol{\pi}_b$) is constrained to account for physical properties of the system, e.g. the well-known inertia matrix $\boldsymbol{M}(\boldsymbol{q})$ is positive definite. More in detail, the main physical constraints considered in the literature are here briefly reviewed.

Physical Feasibility

First of all, mass m_i and inertia matrix $\boldsymbol{L}_i$ are constrained such as

$$\begin{cases} m_i > & 0 \\ \boldsymbol{L}_i \succ & 0 \end{cases} \tag{2.44}$$

However, these constraints are not sufficient to guarantee that matrix $\boldsymbol{M}(\boldsymbol{q})$ is positive definite since $\boldsymbol{L}_i$ is the inertia matrix about the ith link frame. The property $\boldsymbol{M}(\boldsymbol{q}) \succ 0$, $\forall \boldsymbol{q}$, is ensured if the following constraints are jointly considered

$$\begin{cases} m_i > & 0 \\ \boldsymbol{C}_i = \boldsymbol{L}_i - \dfrac{1}{m_i}\boldsymbol{S}(\boldsymbol{m}_{c_i}^i)^{\mathrm{T}}\boldsymbol{S}(\boldsymbol{m}_{c_i}^i) \succ & 0 \end{cases} \tag{2.45}$$

where $\boldsymbol{S}(\cdot)$ is the skew-symmetric matrix operator and $\boldsymbol{C}_i$ is the inertia matrix about the center of mass of link i which, based on Huygens-Steiner theorem is related to the inertia matrix $\boldsymbol{L}_i$; thus, by resorting to the *Schur* complement condition for positive definite matrices, it is equivalent to the following Linear Matrix Inequality [88] as

$$\boldsymbol{Z}_i = \begin{bmatrix} m_i\boldsymbol{I}_3 & \boldsymbol{S}(\boldsymbol{m}_{c_i}^i)^{\mathrm{T}} \\ \boldsymbol{S}(\boldsymbol{m}_{c_i}^i) & \boldsymbol{L}_i \end{bmatrix} \succ 0 \tag{2.46}$$

where $\boldsymbol{I}_3$ is the 3×3 identity matrix. It is worth highlighting that condition in Eq. (2.46) on matrix $\boldsymbol{Z}_i$ is denoted as *physical feasibility* in [86] and as *physical semi-consistency* in [88].

Physical Consistency

Authors in [89] highlighted that an additional condition to Eq. (2.46) needs to be considered (namely the triangle inequality condition) on the eigenvalues λ_j of $\boldsymbol{C}_i$ due to the non-negative mass density in order to achieve the *physical consistency*. Such a condition is taken into consideration in [88, 90] and it is formulated as

$$\begin{cases} 0 < \lambda_3 < \lambda_1 + \lambda_2 \\ 0 < \lambda_2 < \lambda_1 + \lambda_3 \\ 0 < \lambda_1 < \lambda_2 + \lambda_3 \end{cases} \tag{2.47}$$

$$\boldsymbol{E}_i = \begin{bmatrix} \frac{\mathrm{tr}(\boldsymbol{L}_i)}{2}\boldsymbol{I}_3 - \boldsymbol{L}_i & \boldsymbol{m}^i_{c_i} \\ \boldsymbol{m}^{i\,\mathrm{T}}_{c_i} & m_i \end{bmatrix} \succ 0 \tag{2.48}$$

where $\mathrm{tr}(\cdot)$ is the *trace* operator of its matrix argument.

Additional constraints might exploit the knowledge of the geometric structure of the robot by introducing bounded-volume limits [88, 91]. The latter mainly relate to the position of the center of mass of each link which might lay in a cuboid expressed in the link frame [84, 86] as

$$\boldsymbol{m}^i_{c_i,LB} \leq \boldsymbol{m}^i_{c_i} \leq \boldsymbol{m}^i_{c_i,UB} \tag{2.49}$$

with $\boldsymbol{m}_{c_i,LB}$ and $\boldsymbol{m}_{c_i,UB}$ the lower and the upper bounds, respectively, and where the inequalities are to be intended component-wise.

Alternatively, by following the approach in [88] it is possible to require the center of mass of link i to lay inside an ellipsoid $\mathcal{E}_i$ with center $\boldsymbol{x}_{c_i} \in \mathbb{R}^3$ described by

$$\mathcal{E}_i = \{\boldsymbol{x} \in \mathbb{R}^3 \,|\, (\boldsymbol{x} - \boldsymbol{x}^i_{c_i})^{\mathrm{T}} \boldsymbol{Q}_{s_i} (\boldsymbol{x} - \boldsymbol{x}^i_{c_i}) \leq 1\} \tag{2.50}$$

where $\boldsymbol{Q}_{s_i} \in \mathbb{R}^{3\times3}$ is a positive definite matrix defining the shape and orientation of the ellipsoid in the link frame. Finally, as additional constraints considered in [88], the overall mass of a link might be required to lay inside a region $\mathcal{S}_i$ (which could be an ellipsoid as well, see Section 2.1.6).

This constraint, together with the physical consistency, is addressed as $\mathcal{S}$-*density realizability* in [88]. As an example, Fig. 2.3 (left) reports the ellipsoid $\mathcal{S}_4$ (in orange) containing the 4 th link and the ellipsoid $\mathcal{E}_4$ (in red) in which the center of mass $\boldsymbol{m}_{c_4}^4$ is constraint to lay.

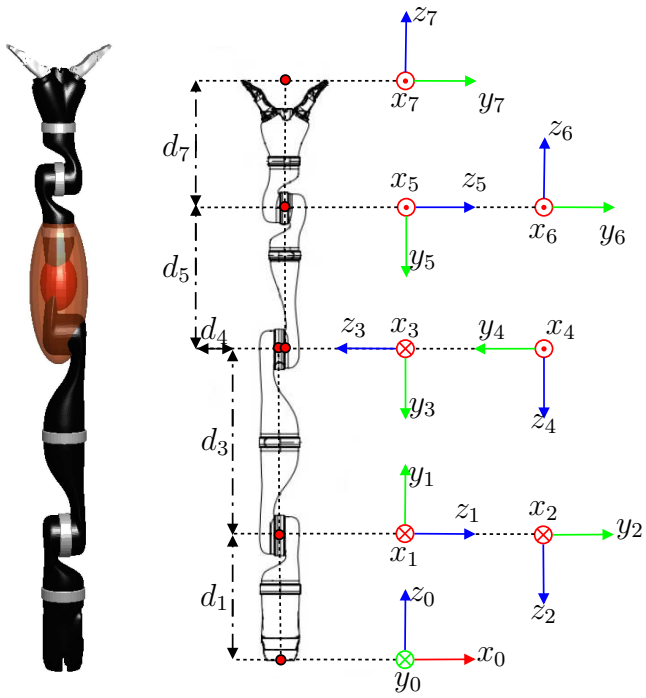

Figure 2.3: The KINOVA Jaco^2 robot is being considered as a test case. On the left, an orange external ellipsoid is shown, bounding the 4 th link, while a red internal ellipsoid containing the center of mass is also displayied. On the right, the reference frames of the KINOVA Jaco^2 robot are illustred according to the DH convention.

2.1.6 Methods for Estimating Dynamic Parameters

In this Section a review and comparison of the main identification methods is presented. It is worth considering that, apart from the CAD method, the other four identification approaches require to either acquire data or to reconstruct them from the robot on-board sensors consisting in N joint configurations $(\boldsymbol{q}(t_i), \dot{\boldsymbol{q}}(t_i), \ddot{\boldsymbol{q}}(t_i))$ together with the corresponding torque vector $\boldsymbol{\tau}(t_i)$ being t_i the generic time instant and $i \in \{1, 2, \ldots, N\}$.

These data, together with Eq. (2.42) or Eq. (2.43) are generally used to build an over-determined system of linear equations in the unknown vector of dynamic parameters $\boldsymbol{\pi}_{\mathrm{b}}$ in the case of ULS method and $\boldsymbol{\pi}_{\mathrm{full}}$ in the case of CLS-1, CLS-2, CLS-3 methods in the form of

$$\bar{\boldsymbol{\tau}} = \begin{bmatrix} \boldsymbol{\tau}(t_1) \\ \boldsymbol{\tau}(t_2) \\ \vdots \\ \boldsymbol{\tau}(t_N) \end{bmatrix} = \begin{bmatrix} \boldsymbol{Y}_{(\cdot)}(t_1) \\ \boldsymbol{Y}_{(\cdot)}(t_2) \\ \vdots \\ \boldsymbol{Y}_{(\cdot)}(t_N) \end{bmatrix} \boldsymbol{\pi}_{(\cdot)} = \bar{\boldsymbol{Y}}_{(\cdot)} \boldsymbol{\pi}_{(\cdot)} \tag{2.51}$$

where $\boldsymbol{Y}_{(\cdot)}$ and $\boldsymbol{\pi}_{(\cdot)}$ are either the base or full regressor computed in the ith configuration and the base or full vector of dynamic parameters depending on the identification method adopted.

CAD

The simplest way to obtain the vector of dynamic parameters $\boldsymbol{\pi}_{\mathrm{full}}$ is to retrieve it from CAD data, which many robot manufacturers make available to the community. The link parameters can be automatically extracted by CAD software once, for instance, properties like the density of each component are specified. However, due to the complexity of robot structure, the many components involved and the uniform density assumption which is usually made, the extracted data might be significantly different than the real one. Moreover, parameters related to friction, when relevant, are unavailable with a CAD approach. On the other hand, the physical constraints are met by construction.

Unconstrained Least Square (ULS)

The ULS method [82] is a popular method for systems identification, that in the case considered and in virtue of Eq. (2.51) consists in solving the following minimisation problem in the unknown vector $\boldsymbol{\pi}_{\mathrm{b}}$

$$\min_{\boldsymbol{\pi}_{\mathrm{b}}} \quad (\bar{\boldsymbol{\tau}} - \bar{\boldsymbol{Y}}_{\mathrm{b}} \boldsymbol{\pi}_{\mathrm{b}})^{\mathrm{T}} (\bar{\boldsymbol{\tau}} - \bar{\boldsymbol{Y}}_{\mathrm{b}} \boldsymbol{\pi}_{\mathrm{b}}) \, . \tag{2.52}$$

In particular, identification requires to properly design the experiments and it is important to design exciting trajectories to provide accurate and fast parameter

estimation even in presence of measurement noise, unmodeled dynamics and external disturbances. Most of the works in the field of robot identification relate the condition number of regressor $\bar{\boldsymbol{Y}}_{\mathrm{b}}$ to the reliability of the data [82] and design the identification trajectory minimizing this condition number [80].

However, despite its simplicity the ULS approach leads to an estimate of $\boldsymbol{\pi}_{\mathrm{b}}$ for which the physical properties mentioned in Section 2.1.5 are not guaranteed and that might suffer of over-fitting [88, 90]. For these reasons, in the last decades many researchers put effort in the constrained identification of robot manipulators which basically differ on the number and type of constraints considered (see Section 2.1.5) and in the methods adopted to solve these problems.

Constrained Least Square (CLS) - Technique 1 (CLS-1)

The goal of constrained identification is to extract a dynamic model maintaining as much as possible the physically meaning of the parameters. As seen in [92, 84], constraints on the numerical value of dynamic parameters in terms of bounds on the CAD values could be added. Inspired by the approach presented in [84], the method consists in estimating the vector of full parameters $\boldsymbol{\pi}_{\mathrm{full}}$ is formalized as

$$\begin{aligned} \min_{\boldsymbol{\pi}_{\mathrm{full}}} \quad & \left(\boldsymbol{\pi}_{\mathrm{b}}(\boldsymbol{\pi}_{\mathrm{full}}) - \hat{\boldsymbol{\pi}}_{\mathrm{b}}\right)^{\mathrm{T}} \left(\boldsymbol{\pi}_{\mathrm{b}}(\boldsymbol{\pi}_{\mathrm{full}}) - \hat{\boldsymbol{\pi}}_{\mathrm{b}}\right) \\ \text{s.t.} \quad & \boldsymbol{\pi}_{\mathrm{LB}} \leq \boldsymbol{\pi}_{\mathrm{full}} \leq \boldsymbol{\pi}_{\mathrm{UB}} \end{aligned} \tag{2.53}$$

More in detail, it exhibits the following characteristics:

- The dependency of $\boldsymbol{\pi}_{\mathrm{b}}$ on the full vector $\boldsymbol{\pi}_{\mathrm{full}}$ is required, that is $\boldsymbol{\pi}_{\mathrm{b}} = \boldsymbol{\pi}_{\mathrm{b}}(\boldsymbol{\pi}_{\mathrm{full}})$. This relationship can be either linear or non-linear depending on the parametrization assumed;
- This dependency is necessary since constraints are set on the vector of full parameters and are in the form $\boldsymbol{\pi}_{\mathrm{LB}} \leq \boldsymbol{\pi}_{\mathrm{full}} \leq \boldsymbol{\pi}_{\mathrm{UB}}$ where $\boldsymbol{\pi}_{\mathrm{LB}}$ and $\boldsymbol{\pi}_{\mathrm{UB}}$ are the lower and upper bounds on the full vector of dynamic parameters, respectively, mainly obtained by CAD and heuristic considerations (e.g., masses are positive);

- The solution sought is the one that, taken into account the acquired data, is closest to a vector $\hat{\boldsymbol{\pi}}_{\mathrm{b}}$, which may be exactly the unconstrained solution in Section 2.1.6 and that meets the bounds on $\boldsymbol{\pi}_{\mathrm{full}}$.

Moreover, it is important to highlight here that the problem formulation is slightly changed with respect to [84] but the overall approach is kept. Finally, it is worth noticing that this approach does not guarantee that neither the physical feasibility nor the physical consistency are guaranteed. Indeed, this property can only be verified a-posteriori and the bounds may be eventually modified according to a trial and error approach.

Constrained Least Square (CLS) - Technique 2 (CLS-2)

Differently from the previously mentioned techniques, the approach in [86] properly takes into account the constraints in Eq. (2.45).

In particular, the considered constrained identification aims to estimate the full vector of parameters $\boldsymbol{\pi}_{\mathrm{full}}$ by minimizing the reconstruction error like in Eq. (2.52) and where the *base* quantities are substituted by their *full* counterparts; the constrained identification is formulated as

$$\begin{aligned} \min_{\boldsymbol{\pi}_{\mathrm{full}}} \quad & (\bar{\boldsymbol{\tau}} - \bar{\boldsymbol{Y}}_{\mathrm{full}}\boldsymbol{\pi}_{\mathrm{full}})^{\mathrm{T}}(\bar{\boldsymbol{\tau}} - \bar{\boldsymbol{Y}}_{\mathrm{full}}\boldsymbol{\pi}_{\mathrm{full}}) \\ \text{s.t.} \quad & \boldsymbol{Z}_i \succ 0 \quad \text{for } i \in \{1, \dots, n\} \\ & m_{i,LB} \leq m_i \leq m_{i,UB} \\ & \boldsymbol{m}^i_{c_i,LB} \leq \underbrace{m_i \boldsymbol{r}^i_{i,c_i}}_{\boldsymbol{m}^i_{c_i}} \leq \boldsymbol{m}^i_{c_i,UB} \end{aligned} \tag{2.54}$$

Explaining, the considered constraints for each link are:

- Physical feasibility described in Section 2.1.5;
- Lower and upper bounds are considered for link mass m_i which are $m_{i,LB}$ and $m_{i,UB}$, respectively;
- Lower and upper bounds are considered for first moment of mass $\boldsymbol{m}^i_{c_i}$ which are $\boldsymbol{m}^i_{c_i,LB}$ and $\boldsymbol{m}^i_{c_i,UB}$, respectively.

The overall constrained problem is globally solved in the framework of Linear-Matrix-Inequality and Semi-Definite-Programming (LMI-SDP). Since the full vector of parameters is identified, the full regressor is used that in general is intrinsically numerically bad conditioned, i.e., the matrix $\bar{\boldsymbol{Y}}_{\text{full}}$ is never full rank no matter the data collected. Therefore, it is necessary to provide the algorithm a *regularization* coefficient.

Constrained Least Square (CLS) - Technique 3 (CLS-3)

The last method we consider is presented in the version [88] (a similar approach is presented in [90]) and is the most complete one considering all the constraints presented above. Similarly to the CLS-2, the CLS-3 technique aims to estimate the full vector of parameters $\boldsymbol{\pi}_{\text{full}}$ by minimizing the reconstruction error while considering the following constraints as

$$\begin{aligned} \min_{\boldsymbol{\pi}_{\text{full}}} \quad & (\bar{\boldsymbol{\tau}} - \bar{\boldsymbol{Y}}_{\text{full}}\boldsymbol{\pi}_{\text{full}})^{\text{T}}(\bar{\boldsymbol{\tau}} - \bar{\boldsymbol{Y}}_{\text{full}}\boldsymbol{\pi}_{\text{full}}) \\ & + \gamma_r(\boldsymbol{\pi}_{\text{CAD}} - \boldsymbol{\pi}_{\text{full}})^{\text{T}}(\boldsymbol{\pi}_{\text{CAD}} - \boldsymbol{\pi}_{\text{full}}) \\ \text{s.t.} \quad & \boldsymbol{E}_i \succ 0 \quad \text{for } i \in \{1, \dots, n\} \\ & \boldsymbol{V}_i \succ 0 \quad \text{for } i \in \{1, \dots, n\} \\ & \text{tr}(\boldsymbol{E}_i\boldsymbol{Q}_i) \geq 0 \quad \text{for } i \in \{1, \dots, n\} \end{aligned} \tag{2.55}$$

with $0 < \gamma_r \ll 1$ a regularization factor; as reported in [88], the second condition is equivalent to require that a matrix $\boldsymbol{V}_i \succ 0$ where

$$\boldsymbol{V}_i = \begin{bmatrix} m_{c_i} & \boldsymbol{m}_{c_i}^{i\text{T}} - m_{c_i}\boldsymbol{x}_{c_i}^{i\text{T}} \\ \boldsymbol{m}_{c_i}^{i} - m_i\boldsymbol{x}_{c_i}^{i} & m_{c_i}\boldsymbol{Q}_{s_i} \end{bmatrix} . \tag{2.56}$$

The latter constraint does not address the case the entire mass of link i is contained within a given region $\mathcal{S}_i$ (this condition is addressed in [88] as $\mathcal{S}$-*density*). In the case $\mathcal{S}_i$ is an ellipsoid expressed in homogeneous form

$$\mathcal{S}_i = \left\{ \boldsymbol{x} \in \mathbb{R}^3 \mid \begin{bmatrix} \boldsymbol{x} \\ 1 \end{bmatrix}^{\text{T}} \boldsymbol{Q}_i \begin{bmatrix} \boldsymbol{x} \\ 1 \end{bmatrix} \geq 0 \right\} \tag{2.57}$$

with $\boldsymbol{Q}_i \in \mathbb{R}^{4\times4}$ a positive definite matrix defining the center and the principal axis of the ellipsoid. In [88], it is shown that $\mathcal{S}$-*density* condition holds if and only if

$$\boldsymbol{E}_i \succ 0 \ \wedge \ \mathrm{tr}(\boldsymbol{E}_i\boldsymbol{Q}_i) \geq 0 \, . \tag{2.58}$$

In summary, the constraints for this method are:

- Physical consistency described in Section 2.1.5;
- The position of the first moment of mass $\boldsymbol{m}_{c_i}$ is forced to reside in an ellipsoid $\mathcal{E}_i$ described by matrix $\boldsymbol{Q}_{s_i}$ and its center $\boldsymbol{x}_{c_i}$ as in Eq. (2.50).
- The mass of link i is contained within a given region $\mathcal{S}_i$ represented as an ellipsoid as well.

From the mathematical perspective, solution of CLS-3 is identical to the CLS-2 and thus it lays in the framework of Linear-Matrix-Inequality and Semi-Definite-Programming (LMI-SDP). A regularization factor is needed as well [88].

2.1.7 Identification Methods: A Comparison

By applying the numerical procedure described in [87], it is possible to cluster the KINOVA Jaco^2 dynamic parameters as shown in Table 2.2 with the linear combination reported in Table 2.3.

	1	2	3	4	5	6	7
m							
$m_{c,x}$							
$m_{c,y}$							
$m_{c,z}$							
l_{xx}							
l_{yy}							
l_{zz}							
l_{yz}							
l_{xy}							
l_{xz}							

Table 2.2: Identifiability of the parameters (red cells: not identifiable; blue cells: identifiable in linear combination; white cells: identifiable alone.

Parameters
$\beta_1 = l_{1,yy} + l_{2,zz}$
$\beta_2 = m_{c_3,y} - d_3(m_4 + m_5 + m_6 + m_7) + m_{c_2,z}$
$\beta_3 = l_{2,xx} - l_{2,zz} + l_{3,zz} + d_3^2(m_4 + m_5 + m_6 + m_7)$
$\beta_4 = l_{2,yy} + l_{3,zz} + d_3^2(m_4 + m_5 + m_6 + m_7)$
$\beta_5 = m_{c_4,y} + d_4(m_6 + m_7 + m_5) + m_{c_3,z}$
$\beta_6 = l_{3,xx} - l_{3,zz} + l_{4,zz} + d_4^2(m_5 + m_6 + m_7)$
$\beta_7 = l_{3,yy} + l_{4,zz} + d_4^2(m_5 + m_6 + m_7)$
$\beta_8 = l_{3,yz} - d_3 d_4(m_5 + m_6 + m_7) - d_3 m_{c_4,y}$
$\beta_9 = m_{c_5,y} + d_5(m_6 + m_7) + m_{c_4,z}$
$\beta_{10} = l_{4,xx} - l_{4,zz} + l_{5,zz} + d_5^2(m_6 + m_7)$
$\beta_{11} = l_{4,yy} + l_{5,zz} + d_5^2(m_6 + m_7)$
$\beta_{12} = l_{4,yz} - d_4 d_5(m_6 + m_7) - d_4 m_{c_5,y}$
$\beta_{13} = m_{c_6,y} + m_{c_5,z}$
$\beta_{14} = l_{5,xx} - l_{5,zz} + l_{6,zz}$
$\beta_{15} = l_{5,yy} + l_{6,zz}$
$\beta_{16} = l_{5,yz} - d_5 m_{c6,y}$
$\beta_{17} = m_{c_6,z} + m_{c_7,z}$
$\beta_{18} = l_{6,xx} + l_{7,yy} - l_{6,zz}$
$\beta_{19} = l_{6,yy} + l_{7,yy}$
$\beta_{20} = l_{7,xx} - l_{7,yy}$

Table 2.3: Dynamic parameters in linear combinations.

The robot is equipped with joint torques mounted after the gear, as common to several lightweight arms such as, for example, the hardware described in [84, 85]. In addition to the joint torques, only the joint positions can be measured at a sampling frequency $f_s = 100\,\text{Hz}$ with an Ethernet connection resorting to the library developed by the manufacturer under ROS (Robot Operating System)[1].

2.1.8 Validation methodology

There are two main requirements concerning validation that need to be satisfied. On one hand the identification *nature* asks for a minimization of the reconstruction error along a set of data (the identification set) different from the one used for the identification itself (the validation set). On the other

[1] https://github.com/Kinovarobotics/kinova-ros

hand, it is required that constraints introduced above are met which have, as main consequence, that the joint-space inertia matrix is strictly positive definite ($\boldsymbol{M}(\boldsymbol{q}) \succ 0$).

The first requirement is usually verified by resorting to a proper metric, typically the reconstruction error and eventually relative information such as the percentage error. In this work, the following errors are considered

$$\varsigma = \frac{\sqrt{\chi^2}}{N_s}, \quad \varsigma_r = \frac{\sqrt{\chi^2}}{\|\bar{\boldsymbol{\tau}}\|} \tag{2.59}$$

where N_s is the number of samples of the dataset at hand and χ^2 is defined as

$$\chi^2 = \begin{cases} (\bar{\boldsymbol{\tau}} - \bar{\boldsymbol{Y}}_{\mathrm{b}}\hat{\boldsymbol{\pi}}_{\mathrm{b}})^{\mathrm{T}}(\bar{\boldsymbol{\tau}} - \bar{\boldsymbol{Y}}_{\mathrm{b}}\hat{\boldsymbol{\pi}}_{\mathrm{b}}) & \text{for ULS} \\ (\bar{\boldsymbol{\tau}} - \bar{\boldsymbol{Y}}_{\mathrm{full}}\hat{\boldsymbol{\pi}}_{\mathrm{full}})^{\mathrm{T}}(\bar{\boldsymbol{\tau}} - \bar{\boldsymbol{Y}}_{\mathrm{full}}\hat{\boldsymbol{\pi}}_{\mathrm{full}}) & \text{for CAD, CLS-1, 2, 3} \end{cases}$$

By defining $\sigma_\pi^2 = \frac{\chi^2}{\nu}$ with ν the difference between number of samples and the number of dynamic parameters identified, an estimate of the covariance matrix $\hat{\boldsymbol{H}}$ of the estimated parameters is [82]

$$\hat{\boldsymbol{H}} = \sigma_\pi^2 (\bar{\boldsymbol{Y}}_{\mathrm{b}}^{\mathrm{T}} \bar{\boldsymbol{Y}}_{\mathrm{b}})^{-1} , \tag{2.60}$$

for all the techniques. For CLS-1, CLS-2 and CLS-3, which identifies the full vector of dynamic parameters, the covariance is actually computed on the *base* representation of the obtained estimation.

The requirement on the positive definiteness of $\boldsymbol{M}(\boldsymbol{q})$ is actually a binary constraint and is met by construction by the CAD estimates and algorithms CLS-2 and CLS-3; however, the ULS and CLS-1 methods may or may not meet it. Since the method CLS-1 outputs the full vector of dynamic parameters, it is possible to check on their value the fulfillment of the constraints, while this is not possible in the ULS case; in addition, since symbolic computation of the positive definiteness of $\boldsymbol{M}(\boldsymbol{q})$ is computationally intractable, numerical methods need to be implemented to check this property. In this case, the numerical validation of the condition $\boldsymbol{M}(\boldsymbol{q}) \succ 0$ is valid for a specific joint configuration $\boldsymbol{q}$, a sample-based approach is needed to check it against the

joint space. Obviously, one single counterexample is sufficient to invalidate the property while the opposite is true only in a probabilistic fashion for *large* number of samples.

2.1.9 Trajectories

Numerical conditioning on the various optimization problems to be solved needs to be properly guaranteed [82, 93, 94]. Considering the stacked regressor matrices $\bar{\boldsymbol{Y}}_{\mathrm{b}}$ in Eq. (2.51), it is required to keep its condition number as small as possible while increasing the minimum singular value. In the case of $\bar{\boldsymbol{Y}}_{\mathrm{full}}$ in Eq. (2.51), it is instead required that the smallest nonzero singular value is as large as possible. In practice, this means to properly span the collected data within the allowed range of joint positions, velocities and accelerations characterizing the robot under study. In these cases, 5 different trajectories have been considered and generated as in [80], and each of them plays the role of identification trajectory and the remaining ones of validation trajectories.

As an example, one exciting trajectory is shown in Fig. 2.4. The corresponding base regressor is characterized by a condition number of approximately 50 with a minimal singular value of 34.

Since the data are spoiled from noise, it is necessary to carry out low pass filtering on the data. A Butterworth 2nd order low pass filter with a cut-off frequency equal to 5 Hz has been implemented. In the literature, a similar choice of cut-off frequency is adopted; for example in [84] the value of 1 Hz has been used in the identification of the KUKA LWR IV.

In Fig. 2.5, an example of the effect of filtering on torque data of joint 2 in one of trajectories is reported.

2.1.10 Results

The CAD parameter values have been derived from the official repository of the manufacturer [95] properly referred to the adopted DH-compliant frames by resorting to the Huygens-Steiner theorem. Moreover, the identification algorithms described above have been run. The identified parameters are reported

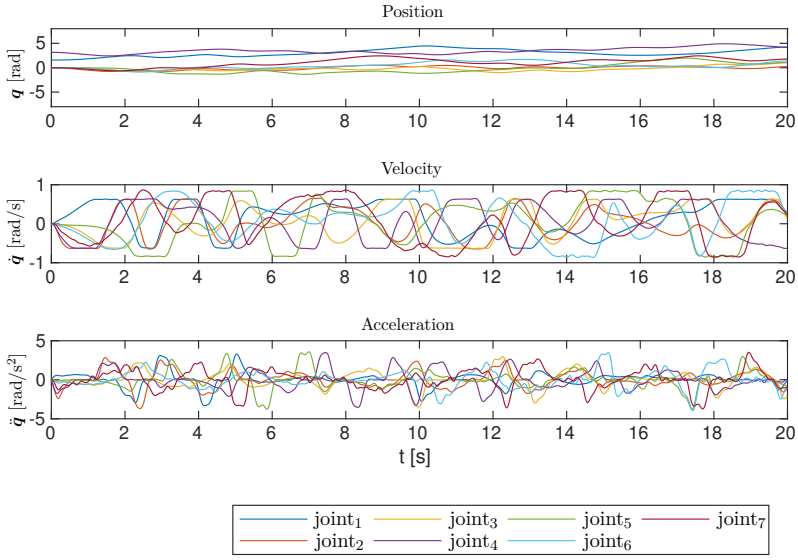

Figure 2.4: Joint position, velocity and acceleration zoom representation of the first 20 seconds of the 3rd trajectory. The full version of trajectory is used as identification dataset.

in Table 2.4 together with the corresponding variance of the estimate (for instance, the 4th trajectory is adopted to the scope of identification and the 3rd one for the validation). As already mentioned above, since ULS only identifies the base vector $\boldsymbol{\pi}_{\mathrm{b}}$, while the other algorithms identify the full vector $\boldsymbol{\pi}_{\mathrm{full}}$, the linear combinations of the latter have been considered in order to compare them with the former.

Since a graphical visualization of the errors is always meaningful, Figs. 2.6-2.10 report the reconstruction errors as time history of measured versus estimated torques for the different algorithms and by using trajectory 3 for validation.

Tables 2.5-2.6 report an overall comparison of the various techniques among different identification and validation trajectories.

The ULS exhibits the smallest error along the identification trajectory. This superior performance, however, is not confirmed along the validation trajectories. This confirms the observation made by [90], i.e., that the ULS somehow overfits the identification trajectory, while other techniques, taking into account

Params	CAD	ULS		CLS-1		CLS-2		CLS-3	
		Value	σ	Value	σ	Value	σ	Value	σ
β_1	0.003715	-0.027334	3.9e-03	0.000001	1.7e-02	0.003664	8.3e-03	0.000824	3.9e-03
β_2	-0.956111	-1.391333	6.9e-04	-1.076000	2.9e-03	-1.383625	1.5e-03	-1.390848	6.9e-04
β_3	0.413706	0.535243	9.5e-03	0.477060	4.0e-02	0.579379	2.0e-02	0.594658	9.5e-03
β_4	0.414086	0.585506	4.5e-03	0.457060	1.9e-02	0.605004	9.6e-03	0.596306	4.5e-03
β_5	-0.016199	-0.035576	4.7e-04	-0.017640	2.0e-03	-0.033551	1.0e-03	-0.035730	4.7e-04
β_6	0.000463	0.031621	6.5e-03	-0.016427	2.8e-02	0.054993	1.4e-02	0.004703	6.6e-03
β_7	0.000843	0.024736	2.8e-03	0.000573	1.2e-02	0.004193	6.0e-03	0.006755	2.8e-03
β_8	-0.006642	-0.017786	2.3e-03	-0.007232	9.8e-03	-0.013756	4.9e-03	-0.008831	2.3e-03
β_9	-0.370088	-0.572468	5.0e-04	-0.373640	2.1e-03	-0.678596	1.1e-03	-0.572292	5.1e-04
β_{10}	0.116272	0.165417	4.1e-03	0.118846	1.8e-02	0.210676	8.8e-03	0.173466	4.2e-03
β_{11}	0.116576	0.188398	2.7e-03	0.115246	1.2e-02	0.210918	5.7e-03	0.177551	2.7e-03
β_{12}	-0.003570	-0.015456	1.6e-03	-0.003603	6.6e-03	-0.006591	3.3e-03	-0.004862	1.6e-03
β_{13}	0	-0.002278	3.4e-04	0	1.5e-03	-0.000001	7.2e-04	-0.002054	3.4e-04
β_{14}	0.002554	-0.002358	3.4e-03	0.001000	1.4e-02	0.013544	7.2e-03	0.000973	3.4e-03
β_{15}	0.002647	-0.006334	2.0e-03	0.001000	8.4e-03	0.003730	4.2e-03	0.001497	2.0e-03
β_{16}	0	0.000519	1.3e-03	0	5.5e-03	0	2.7e-03	-0.001571	1.3e-03
β_{17}	-0.042324	0.142087	3.5e-04	-0.043600	1.5e-03	-0.043599	7.5e-04	0.142759	3.5e-04
β_{18}	0.003917	0.036578	2.4e-03	-0.001000	1.0e-02	0.114899	5.0e-03	0.017273	2.4e-03
β_{19}	0.004010	0.021991	1.5e-03	0	6.5e-03	0.105452	3.3e-03	0.017606	1.5e-03
β_{20}	0	-0.018514	1.6e-03	0	6.7e-03	-0.008520	3.3e-03	0.000107	1.6e-03
$m_{c_2,x}$	0	0.006347	4.0e-04	0	1.7e-03	0	8.4e-04	0.006854	4.0e-04
$l_{2,xy}$	0	0.000908	4.1e-03	0	1.7e-02	0	8.7e-03	0	4.1e-03
$l_{2,xz}$	0	0.041782	4.1e-03	0	1.8e-02	0	8.8e-03	0	4.2e-03
l_{2yz}	0	-0.006675	2.4e-03	0	1.0e-02	0	5.0e-03	0	2.4e-03
$m_{c_3,x}$	0	0.013107	5.3e-04	0	2.3e-03	-0.000001	1.1e-03	0.012370	5.3e-04
$l_{3,xy}$	0	-0.003376	2.1e-03	0	9.0e-03	0	4.5e-03	0	2.1e-03
$l_{3,xz}$	0	-0.022552	3.1e-03	0	1.3e-02	0	6.5e-03	0	3.1e-03
$m_{c_4,x}$	0	0.028568	3.8e-04	0	1.6e-03	0	8.1e-04	0.028330	3.8e-04
$l_{4,xy}$	0	0.016842	1.6e-03	0	6.9e-03	0	3.4e-03	0	1.6e-03
$l_{4,xz}$	0	0.008521	1.8e-03	0	7.8e-03	0	3.9e-03	0	1.8e-03
$m_{c_5,x}$	0	0.002895	4.2e-04	0	1.8e-03	0.000001	9.0e-04	0.003247	4.3e-04
$l_{5,xy}$	0	0.003167	1.1e-03	0	4.7e-03	0	2.3e-03	0	1.1e-03
$l_{5,xz}$	0	-0.017816	1.5e-03	0	6.3e-03	0	3.1e-03	0	1.5e-03
$m_{c_x,6}$	0	0.005368	3.6e-04	0	1.5e-03	0.000001	7.7e-04	0.004566	3.6e-04
$l_{6,xy}$	0	0.006955	9.9e-04	0	4.2e-03	0	2.1e-03	0	9.9e-04
$l_{6,xz}$	0	0.003348	9.6e-04	0	4.1e-03	0	2.0e-03	0	9.6e-04
$l_{6,yz}$	0	0.010184	1.0e-03	0	4.4e-03	0	2.2e-03	0	1.0e-03
$m_{c_x,7}$	0	0.000949	2.9e-04	0	1.2e-03	0	6.2e-04	0	2.9e-04
$m_{c_y,7}$	0	-0.001331	2.8e-04	0	1.2e-03	0	6.0e-04	-0.000298	2.9e-04
$l_{7,xy}$	0	0.004124	7.6e-04	0	3.3e-03	0	1.6e-03	0	7.7e-04
$l_{7,xz}$	0	0.002020	7.2e-04	0	3.1e-03	0	1.5e-03	0	7.3e-04
$l_{7,yz}$	0	0.003968	7.4e-04	0	3.2e-03	0	1.6e-03	0	7.4e-04
$l_{7,zz}$	0.000582	0.000470	1.1e-03	0.000470	4.6e-03	0.000004	2.3e-03	0.000107	1.1e-03

Table 2.4: Numerical values for base dynamic parameters of the algorithms on the 3rd validation trajectory.

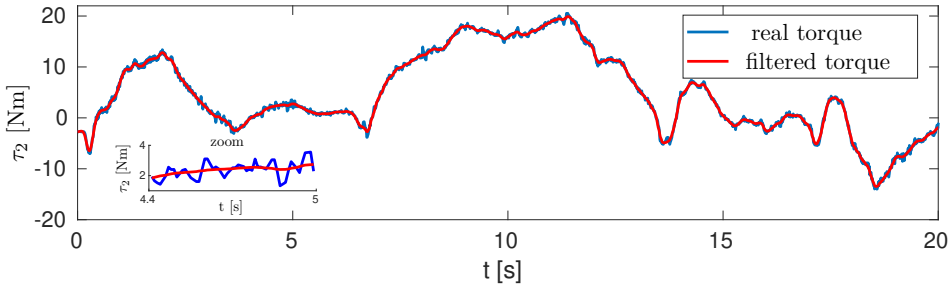

Figure 2.5: Low pass filter effect on joint torque, example for the second joint of 3rd trajectory with a zoom-in plot in the time interval $[4.4, 5]$ s.

the physical constraints, finally better perform on the validation trajectories.

The reconstruction made by resorting to the CAD values is consistently found to have the worst performance among the various techniques, confirming the importance of the identification process.

The CLS-3 method consistently exhibits the smallest error, although it is outperformed by ULS in a few samples and by CLS-1 in one case. It is worth noticing that the displacement of the first moment of inertia is numerically small and that the errors are numerically very close one to each other.

The last two lines of the table show the binary conditions resulting in the requirement to satisfy the physical constraints: the sole CAD and CLS-3 satisfy all of them. However, if the reconstruction error is considered, the performance of CLS-3 among the 5 methods is clearly superior.

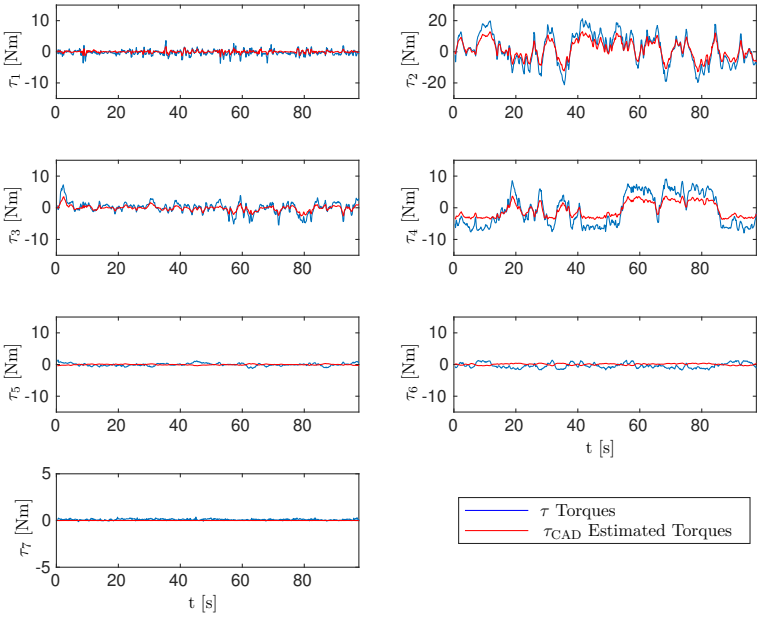

Figure 2.6: CAD reconstruction errors along the 3rd trajectory.

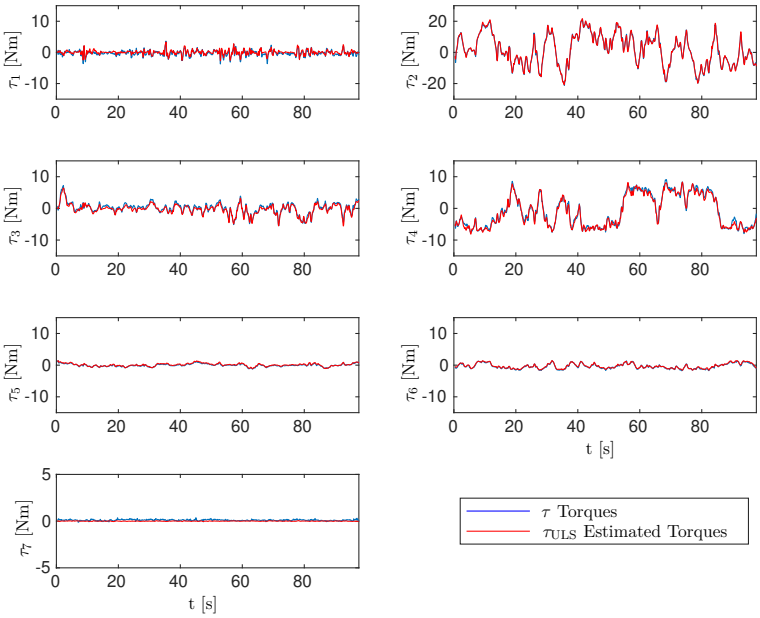

Figure 2.7: ULS reconstruction errors along the 3rd trajectory.

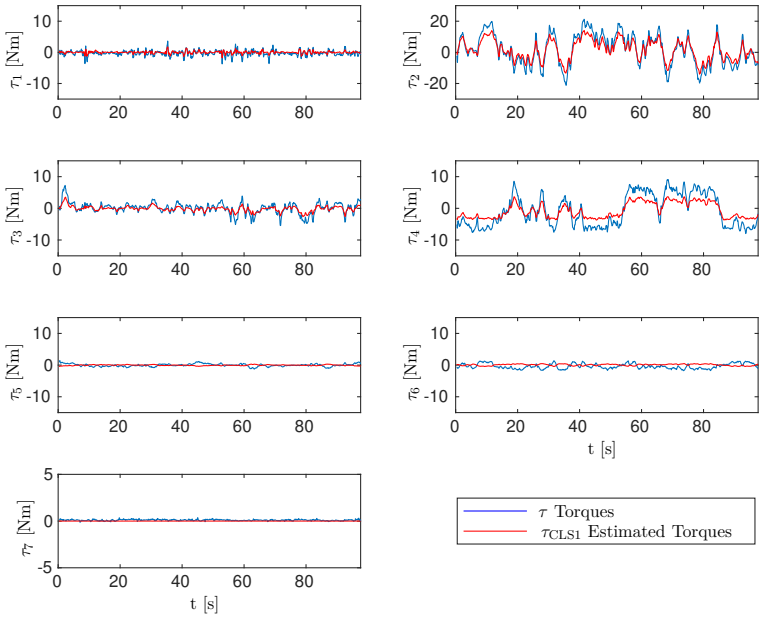

Figure 2.8: CLS-1 reconstruction errors along the 3rd trajectory.

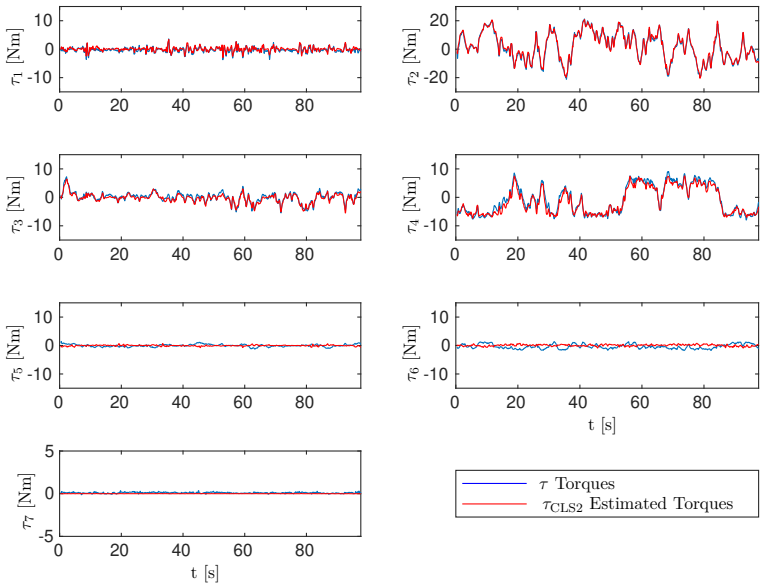

Figure 2.9: CLS-2 reconstruction errors along the 3rd trajectory.

Traj	CAD		ULS		CLS-1		CLS-2		CLS-3	
	ς [Nm]	ς_r	ς [Nm]	ς_r	ς [Nm]	ς_r	ς [Nm]	ς_r	ς [Nm]	ς_r
Id: Traj 1	0.01694	0.39842	0.00302	0.07097	0.01465	0.34454	0.00724	0.17023	0.00310	0.07297
Val: Traj 2	0.01947	0.39431	0.00537	0.10872	0.01690	0.34224	0.00665	0.13477	0.00526	0.10647
Val: Traj 3	0.00700	0.43991	0.00213	0.13407	0.00648	0.40725	0.00298	0.18740	0.00203	0.12791
Val: Traj 4	0.00556	0.42289	0.00166	0.12647	0.00505	0.38421	0.00265	0.20172	0.00159	0.12060
Val: Traj 5	0.00348	0.41734	0.00090	0.10735	0.00310	0.37147	0.00133	0.15957	0.00089	0.10710
Id: Traj 2	0.01947	0.39431	0.00292	0.05909	0.01689	0.34222	0.00585	0.11841	0.00307	0.06210
Val: Traj 1	0.01694	0.39842	0.00468	0.11017	0.01464	0.34430	0.00797	0.18741	0.00408	0.09599
Val: Traj 3	0.00700	0.43991	0.00166	0.10448	0.00648	0.40728	0.00287	0.18031	0.00154	0.09691
Val: Traj 4	0.00556	0.42289	0.00150	0.11404	0.00505	0.38420	0.00256	0.19432	0.00129	0.09794
Val: Traj 5	0.00348	0.41734	0.00078	0.09305	0.00310	0.37148	0.00134	0.16051	0.00071	0.08475
Id: Traj 3	0.00700	0.43991	0.00143	0.08963	0.00647	0.40677	0.00277	0.17403	0.00144	0.09030
Val: Traj 1	0.01694	0.39842	0.00378	0.08884	0.01462	0.34390	0.00781	0.18363	0.00380	0.08928
Val: Traj 2	0.01947	0.39431	0.00328	0.06651	0.01688	0.34183	0.00620	0.12555	0.00326	0.06609
Val: Traj 4	0.00556	0.42289	0.00123	0.09381	0.00505	0.38369	0.00254	0.19269	0.00121	0.09225
Val: Traj 5	0.00348	0.41734	0.00065	0.07827	0.00065	0.07827	0.00133	0.15934	0.00066	0.07852
Id: Traj 4	0.00556	0.42289	0.00118	0.08982	0.00505	0.38382	0.00251	0.19105	0.00119	0.09044
Val: Traj 1	0.01694	0.39842	0.00476	0.08842	0.01464	0.34425	0.00783	0.18409	0.00373	0.08766
Val: Traj 2	0.01947	0.39431	0.00329	0.06663	0.01688	0.34195	0.00607	0.12303	0.00330	0.06678
Val: Traj 3	0.00700	0.43991	0.00148	0.09291	0.00647	0.40691	0.00281	0.17648	0.00146	0.09193
Val: Traj 5	0.00348	0.41734	0.00066	0.07911	0.00310	0.37138	0.00134	0.16069	0.00066	0.07902
Id: Traj 5	0.00480	0.41734	0.00062	0.07412	0.00310	0.37152	0.00132	0.15820	0.00064	0.07676
Val: Traj 1	0.01694	0.39842	0.00530	0.12469	0.01465	0.34457	0.00744	0.17503	0.00372	0.08743
Val: Traj 2	0.01947	0.39431	0.00429	0.08688	0.01690	0.34238	0.00636	0.12886	0.00323	0.06550
Val: Traj 3	0.00700	0.43991	0.00198	0.12472	0.00648	0.40740	0.00286	0.18003	0.00148	0.09326
Val: Traj 4	0.00556	0.42289	0.00180	0.13693	0.00506	0.38436	0.00259	0.19691	0.00123	0.09333

Table 2.5: Summary of the results. With red background the largest error for a specific trajectory, with green the smallest.

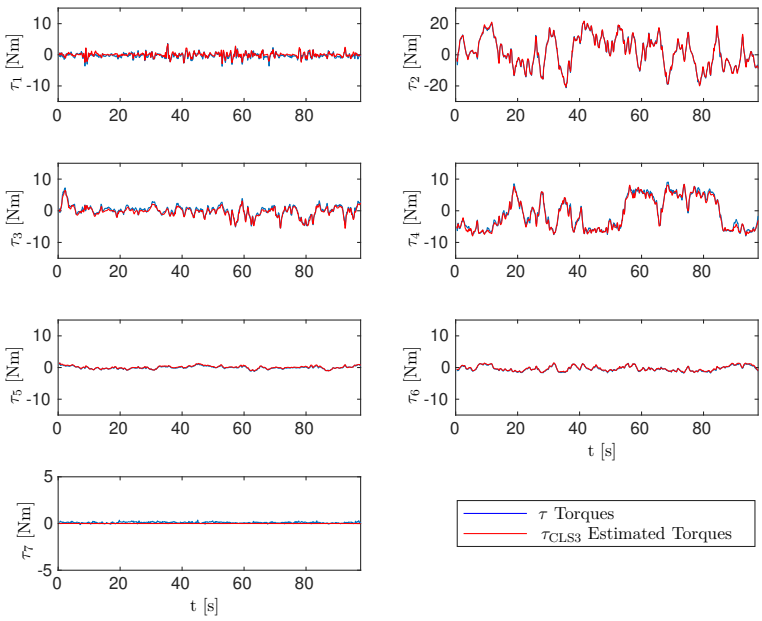

Figure 2.10: CLS-3 reconstruction errors along the 3rd trajectory.

Constraints	CAD	ULS	CLS-1	CLS-2	CLS-3
Eq. (2.45)	yes	no	no	yes	yes
Eq. (2.47)	yes	no	no	no	yes
Eq. (2.49) or Eq. (2.50)	yes	no	yes	yes	yes

Table 2.6: Constraints satisfaction of the different identification methods.

As a final consideration, since the robot at hand is equipped with sensors at the link side, motor inertia and joint friction parameters at motor side are not taken into considerations being the torque measures insensitive to these parameters. The identifications discussed have been run as well including the friction terms at the link side and, consistently with the literature, the result was that the errors were almost invariant while the variance corresponding to those parameters, computed as in Eq. (2.60), were much higher than the rest of the parameters.

2.2 Machine Learning

Machine Learning is commonly divided in three subgroups reported in Fig. 2.11. However, in the following, a short explaination for each of them is done.

- **Supervised Learning:** algorithm that learns w.r.t a training set that includes the correct output for each entry in the set. In general, this approach would learn to approximate a function that best represents the training data, with the aim of to give accurate output on unseen examples (useful in the classification problems);
- **Unsupervised Learning:** the goal of this approach is to learn something about given datasets based on the data alone, without labels or training examples. More in detail, it uses algorithms to analyze and cluster unlabeled datasets, discovering hidden patterns or data groupings without the need for human intervention;
- **Reinforcement Learning:** these algorithms aim to teach an agent the behavior to assume in an environment through rewards. More in detail, they allow agents to learn a policy with the goal of maximizing the reward that receives each time that interacts with the environment.

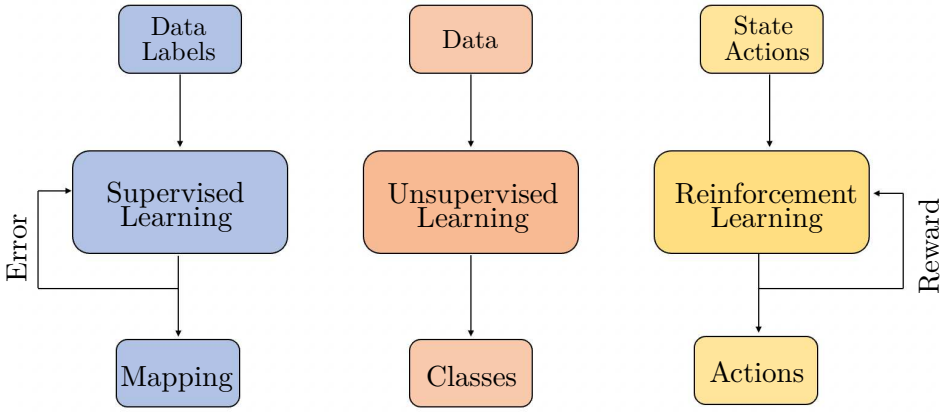

Figure 2.11: Machine Learning Algorithms: Supervised, Unsupervised and Reinforcement.

In this thesis a focus on reinforcement and supervised learning is put. Regarding the first one, specially, it is a trial-and-error method based on an agent that takes actions based on the observations and collected feedbacks from the environment. In particular, at each time step k the agent in a state s_k selects an action a_k receiving a scalar value named reward r_k as feedback (see Fig.2.12). The aim is to maximise the expected reward over time.

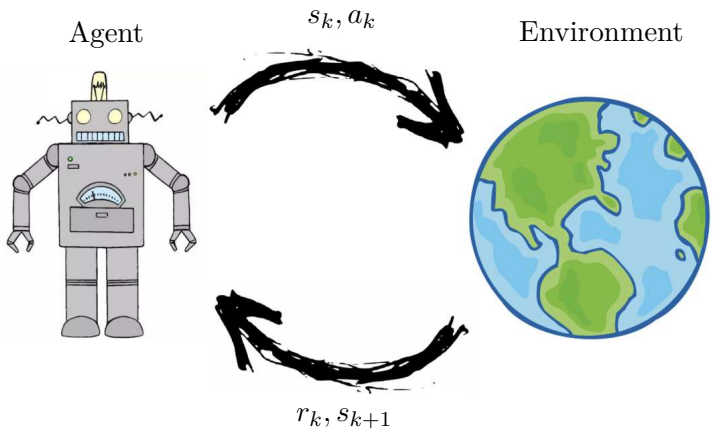

Figure 2.12: Scheme of Reinforcement Learning approach: An agent in a state s_k applies an action a_k interacting with the environements, which gives a reward r_k to the agent that updates its state with s_{k+1}.

2.2.1 Reinforcement Learning

The decision-making problem is usually modelled as a *Markov Decision Process* (MDP), that is defined as a tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$, where

- $\mathcal{S}$ denotes the state space;
- $\mathcal{A}$ denotes the action space;
- $p = P(s_{k+1} = s' | s_k = s, a_k = a)$ is the transition probability;
- r is the reward function;
- $\gamma \in [0, 1]$ is the discount factor.

The mapping between states and actions is called policy $\pi(a|s)$. The agent estimates the goodness of a state in order to decide which action to perform at a particular time-step. To the scope, a possible method is to consider the *action-value* function, also known as the *Q-function*, defined as the expected sum of rewards after performing action a_k in the state s_k following a policy π

$$Q_\pi(s,a) = \mathbb{E}_\pi(\sum_{l=0}^{\infty} \gamma^l r_{k+l+1} | s_k = s, a_k = a) \ . \tag{2.61}$$

The optimal action-value function is given by

$$Q_*(s,a) = \max_\pi Q_\pi(s,a), \forall s \in S, a \in A \ , \tag{2.62}$$

which, combined with Eq. (2.61), leads to the *Bellman optimality equation*

$$Q_*(s,a) = \sum_{s'} p(s'|s,a)[r_{k+1} + \gamma \max_{a'} Q_*(s',a')] \ , \tag{2.63}$$

where $s_k = s, s_{k+1} = s', a_k = a, a_{k+1} = a'$.

Equation (2.63) assumes the knowledge of the state transition probability $p(s'|s,a)$, i.e., it is a *model-based* problem.

The same equation can be applied in a *model-free* problem ignoring the state transition probability. This is achieved by setting $p(s'|s,a) = 1$ and, therefore, obtaining

$$Q_*(s,a) = r_{k+1} + \gamma \max_{a'} Q_*(s',a') \ . \tag{2.64}$$

Equation (2.64) is a recursive nonlinear function without a closed-form solution. The following iterative update law can be adopted

$$Q_{k+1}(s_k,a_k) = Q_k(s_k,a_k) - \alpha\delta_k \ , \tag{2.65}$$

where $\alpha \in [0,1]$ is known as *learning rate* and δ_k is the temporal difference error, defined as

$$\delta_k = Q_k(s_k,a_k) - r_{k+1} - \gamma \max_{a \in \mathcal{A}} Q_k(s_{k+1},a) \ . \tag{2.66}$$

Finally, by combining Eq. (2.66) and Eq. (2.65), one obtains

$$Q_{k+1}(s_k, a_k) = Q_k(s_k, a_k) + \alpha(r_{k+1} + \gamma \max_{a \in \mathcal{A}} Q_k(s_{k+1}, a) - Q_k(s_k, a_k)) \,, \tag{2.67}$$

which is an iterative approach named *Q-Learning* [96].

Rooted Trees

A directed graph is an ordered pair $\mathcal{G} = \{\mathcal{V}, \mathcal{X}\}$, where $\mathcal{V} = \{\nu_1, \ldots, \nu_l\}$ is the set of nodes, or vertices, and $\mathcal{X} = \{\chi_1, \ldots, \chi_m\}$ is the set of oriented pairwise edges from node ν_i to ν_j. A scalar value might be assigned to edges; in this case the edge value is assumed to be 1 unless specified otherwise. A path between two nodes ν_i and ν_j is the set of directed edges through which a node ν_j can be reached from node ν_i. A graph $\mathcal{G}$ is defined cyclic if it contains a cycle, i.e., there is a subset of the edge set that forms a path such that the first node of the path corresponds to the last. On the opposite, if no cycle exists a graph is defined acyclic.

A tree is an undirected acyclic graph such that there is a unique path between every pairs of vertices; this implies that in a tree of l nodes, $m = l - 1$ edges exist. A directed tree is a *Directed Acyclic Graph* (DAG) whose underlying undirected graph is a tree. In particular, in a directed rooted tree, given a node ν_i, there is exactly one edge from another node ν_j, called *parent*, to ν_i that is, then, a *child* of ν_j; then, every node has a unique parent except the root which has no parent and from which exactly one path exists to any other node of the tree; furthermore, a node with no child is a terminal node.

Moreover, the *depth* or *level* of a node ν_i is its distance from the root, i.e., the length of the unique path from the root to ν_i computed by summing the weights associated to the path. Thus, the root has depth 0.

2.2.2 Supervised Learning

Neural Networks

The origin of the name comes from biological studies for modelling the mathematical structure of the nervous system. Even if the first official mention was in the 1943, only almost 20 years later, Frank Rosenblatt proposed the perceptron (or feed forward neural network) [97, 98].

Feed-Forward Neural Networks

This kind of network is the most commonly used and it is made of many connected units called *neurons*, where each of them produces a real value as result of applying a non linear activation function to a linear combination of weights multiplying the values of other neurons.

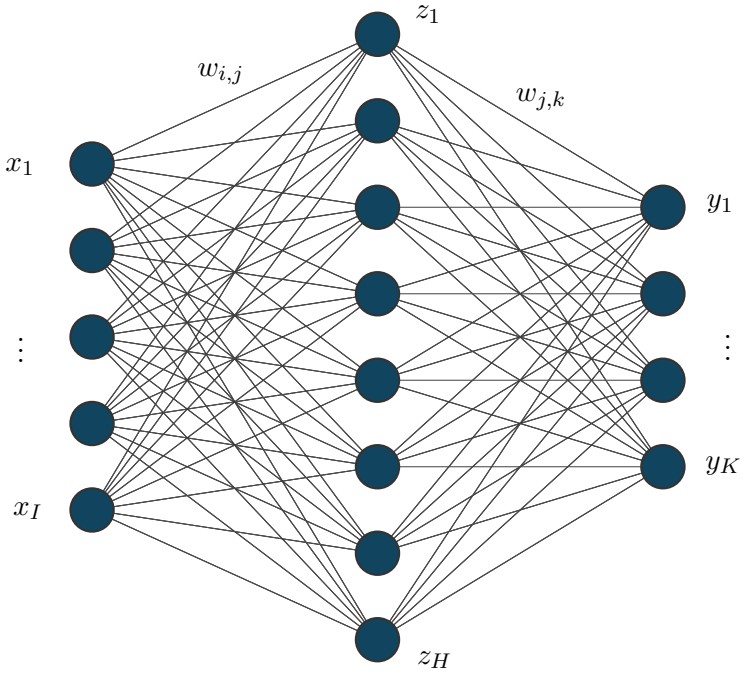

Figure 2.13: Architecture of Feed-Forward Neural Network.

In Fig. 5.1 an example of feed-forward neural network is reported. It consists of three layer named input, hidden and output, respectively.

The arrows represent connections, and the term feed-forward means that each connection is associated with a value that multiplies a neuron of the input layer. The result is considered in the value of a neuron of the output layer. Usually, this is called full connection, because each unit receives connections from the neuros present in the previous layer with an additional bias term.

Here it is possible to consider the mathematical model. Suppose that the input layer is made of I values with H dimension of the intermediate layer (hidden). The relationship between input and hidden is the following

$$u_j = \sum_{i=1}^{H} w_{i,j} x_i + b_{j,0} \,, \tag{2.68}$$

where $j = 1, \ldots, H$, $w_{i,j}$ are the weights and $b_{j,0}$ is the bias.

At this point, it is possible to apply on these values a non-linear activation function h as

$$z_j = h(u_j) \,. \tag{2.69}$$

According to the Eq. (2.68) and Eq. (2.69), the value of the k-th neuron present in the output layer is $h(u_k)$ and it is known as *forward pass*.

Differently, there is another operation called *backward pass* and it is based on the gradient descent approach for minimizing an error function between real and estimate values $E(\boldsymbol{w})$. Thus, the weights $w_{i,j}$ of the network need to be updated and it is done through the *back propagation* rule as

$$w_{j,i}^{\text{new}} = w_{j,i}^{\text{old}} - \alpha \frac{\partial E(\boldsymbol{w})}{w_{j,i}} \,, \tag{2.70}$$

with α learning rate and $\boldsymbol{w}$ vector that contains all the weights of the network.

As reference is considered supervised learning mentioned above for a multi-class problem, where values of the ground truth are availables for each class, the error function is defined as

$$E = \frac{1}{2} \sum_{k=1}^{K} (y_k - \hat{y_k}) \,, \tag{2.71}$$

where $\hat{y_k}$ is the prediction of the model. By setting $y_k = u_k$ it is possible to compute the derivative w.r.t the weights of the network

$$\frac{\partial E}{\partial w_{i,j}} = \frac{\partial E}{\partial u_j}\frac{\partial u_j}{\partial w_{i,j}} = (y_j - \hat{y_j})z_i \ . \tag{2.72}$$

Going to compact the form, and particularizing at the output neurons, it is possible to obtain

$$\delta_k = \frac{\partial E_n}{\partial u_k} = y_k - \hat{y_k} \ , \tag{2.73}$$

whereas for hidden layers

$$\delta_j = \frac{\partial E_n}{\partial u_j} = \sum_{k=1}^{K} \frac{\partial E}{\partial u_k}\frac{\partial u_k}{\partial u_j} \ . \tag{2.74}$$

Thus, it is possible to obtain the backpropagation rule as

$$\delta_j = h^{'}(u_j) \sum_{k=1}^{K} w_{l,j}\delta_k \ . \tag{2.75}$$

A generalization of neural network is the deep neural network, where the word deep means that there are more than one hidden layer. Specifically, this structure is capable of extracting more information than the previous one.

Convolutional Neural Network

A widely used architecture is the *Convolutional Neural Network* (CNN), that can receive images in input and it is allowed to encode some properties into the model. The most important component is the *convolutional layer*, which consists of a collection of different convolutional filters, also known as *kernels*. The input data, in this case the images, are expressed as matrices and they are convolved with the filters for generating the output named *feature map*.

More in detail, the kernel is made of discrete numbers that are adjusted during the training phase. There is also a pooling layer that aims to sample the feature maps while mantaining the dominant information, an activation function that maps inputs to outputs in a non-linear way, and a fully connected layer where

each neuron of a layer is connected to all the other neurons of the previous layer.

An example of CNN called VGGNet16 is reported in Fig. 2.14.

- **Kernel**: it is represented by a grid of discrete numbers, where each value is called kernel weight. These values are initialized at the beginning of the training phase, and then, they are adjusted during the remaining part of the training, where the kernel learns to extract important features;
- **Pooling Layer**: the aim is to sample the feature maps, maintaining a large part of the dominant information in every step of the pooling stage. A significant limitation of the feature map provided by the convolutional layers is related to the recording of the precise position of features in the input. As a result, small variations in the feature position of the input image can result in a different feature map. To overcome this problem a common approach is to apply a sampling, considering a low-resolution version, without loss of the most important information;
- **Activation Function**: it is a function that maps inputs to outputs in a non-linear way. In particular, it decides whether to fire a neuron with reference to a specific input by creating the corresponding output;
- **Fully connected layer**: the fully connected layer is present at the end of the CNN architecture. Each neuron of a layer is connected to all the other neurons of the previous layer. The input of this layer is the output of the last pooling/convolutional layer and it is a vector, which is created from the feature maps after the flattening operation. Finally, the output of the fully connected layer represents the final output of the CNN;
- **Loss Function**: it is used into the output layer in order to calculate the predicted error on the training samples in the CNN model, which represents the error between the real and predicted value. Furthermore, it is optimized throughout the learning process.

In the neural network functioning there are two kinds of propagation for the data: forward and backward. For the first one, if there is the assumption that being to have $n \times n$ neuron layers followed by a convolutional layer, considering

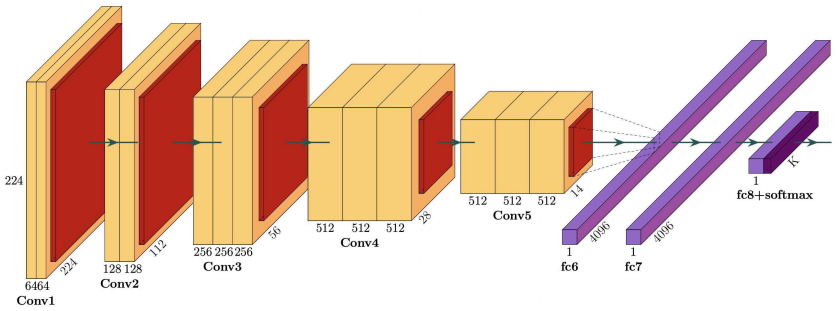

Figure 2.14: Architecture of VGGNet16.

$m \times m$ filters $\boldsymbol{w}$, the output has dimensionality as $(n - m + 1) \times (n - m + 1)$. More in detail, considering the input $x^l_{i,j}$ with l number of the considered layer, it is expressed as

$$x^l_{i,j} = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} w_{a,b} y^{l-1}_{i+a,j+b} \, , \tag{2.76}$$

and the relative output is

$$y_{i,j} = h(x^l_{i,j}) \, , \tag{2.77}$$

where h is the activation function.

Usually, the most common activation function in this context is the $ReLU$, and it is mathematically defined as following

$$h(x) = max(0, x) = \begin{cases} 0 & \text{if} \quad x < 0 \\ x & \text{if} \quad x \geq 0 \, . \end{cases} \tag{2.78}$$

Instead, taking into account the Eq. (2.76) and Eq. (2.77) and regarding the backward part, if E is the error at a certain layer, the gradient is computed at the previous layer such as

$$\frac{\partial E}{\partial w_{a,b}} = \sum_{i=0}^{N-m} \sum_{j=0}^{N-m} \frac{\partial E}{\partial x^l_{i,j}} \frac{\partial x^l_{i,j}}{\partial w_{a,b}} = \sum_{i=0}^{N-m} \sum_{j=0}^{N-m} \frac{\partial E}{\partial x^l_{i,j}} y^{l-1}_{i+a,j+b} \, , \tag{2.79}$$

where for the compute of the gradient it is necessary to calculate $\frac{\partial E}{\partial x_{i,j}^l}$ as

$$\frac{\partial E}{\partial x_{i,j}^l} = \frac{\partial E}{\partial y_{i,j}^l}\frac{\partial y_{i,j}^l}{\partial x_{i,j}^l} = \frac{\partial E}{\partial y_{i,j}^l}\frac{\partial}{\partial x_{i,j}^l}(\sigma(x_{i,j}^l)) = \frac{\partial E}{\partial y_{i,j}^l}\sigma'(x_{i,j}^l)\ . \tag{2.80}$$

Furthermore, exploiting the previous relationships in Eq. (2.79) and Eq. (2.80), it is possible to compute the weights of the model, exploiting another time the chain-rule such as

$$\begin{aligned}\frac{\partial E}{\partial y_{i,j}^{l-1}} &= \sum_{a=0}^{m-1}\sum_{b=0}^{m-1}\frac{\partial E}{\partial x_{(i-a)(j-b)}^l}\frac{\partial x_{(i-a)(j-b)}^l}{\partial y_{i,j}^{l-1}} \\ &= \sum_{a=0}^{m-1}\sum_{b=0}^{m-1}\frac{\partial E}{\partial x_{(i-a)(j-b)}}w_{a,b}\ ,\end{aligned} \tag{2.81}$$

where it is possible to noting that

$$\frac{\partial x_{(i-a)(j-b)}^l}{\partial y_{i,j}^{l-1}} = w_{a,b}\ . \tag{2.82}$$

Chapter 3

Task-Motion Planning via Reinforcement Learning

This chapter addresses the problem of retrieving a target object from cluttered environment using a robot manipulator. In the proposed solution relies on a Task and Motion Planning approach based on a two-level architecture: the high-level is a Task Planner aimed at finding the optimal objects sequence to relocate, according to a metric based on the objects weight; the low-level is a Motion Planner in charge of planning the end-effector path for reaching the specific objects taking into account the robot physical constraints. The high-level task planner is a Reinforcement Learning agent, trained using the information coming from the low-level Motion Planner. The *Q-Tree* algorithm, which is based on a dynamic tree structure inspired by the Q-Learning technique. Three different RL-policies with two kinds of tree exploration techniques (Breadth and Depth) are compared in simulation scenarios with different complexity. Moreover, the proposed learning methods are experimentally validated in a real scenario by adopting a KINOVA Jaco^2 7-DoFs robot manipulator.

3.1 Retrieving Objects from Clutter

The aim of this work is to design a learning agent for solving the task of retrieving a target object T in clutter and moving it from an initial position $\boldsymbol{p}_{t,0} \in \mathbb{R}^3$ to a final position $\boldsymbol{p}_{t,f} \in \mathbb{R}^3$, relying on a Reinforcement Learning approach. Given the presence of N_o obstacles $\mathcal{O} = \{O_1, \ldots, O_{No}\}$ in the scene

with assigned position $\boldsymbol{p}_{o,i} \in \mathbb{R}^3$ $(i = 1, 2, \cdots, N_o)$, the Reinforcement Learning agent should relocate the obstacles that make T unreachable, in order to free up a path to reach it. An object is considered unreachable if it is not possible to find a trajectory that allows the robot to grasp and relocate it without hitting any obstacle. In the initial state s_H, the robot starts in a predefined joints configuration, all the obstacles O_i are in initial positions $\boldsymbol{p}_{o,i}$ and the target T is in the initial position $\boldsymbol{p}_{t,0}$.

The sequence $\mathcal{S}_u$ represents any sequence of objects with cardinality $|\mathcal{S}_u| = u$, and $\mathcal{S}_u^T$ is the sequence obtained from $\mathcal{S}_u$ adding as last element the target T, i.e. $\mathcal{S}_u^T = \{\mathcal{S}_u, T\}$. An object $O_i \in \mathcal{S}_u$ is considered relocatable if all the previous objects in the sequence $\mathcal{S}_u$ can be relocated. If all the objects into the sequence $\mathcal{S}_u$ are relocatable, the sequence is defined feasible.

The following assumptions are made.

Assumption 1 *The scene configuration in terms of target location and obstacles positions is known beforehand*

Assumption 2 *Objects are relocated without affecting the remaining ones*

Assumption 3 *For both obstacles and target, only side grasp is allowed*

3.1.1 System Architecture

A two-layered architecture is designed as shown in Fig. 3.1. In detail, the high-level is represented by the *RL-Task Planner*, which is in charge of learning the optimal sequence of actions, i.e. obstacles to relocate, in order to free up a path toward the target; the low-level layer is the *Motion Planner*, which provides a feedback about the feasibility, g_k, of a specific action, a_k, aimed at relocating an object in terms of fulfillment of robot kinematic constraints.

An action a_k represents a given object to relocate, which is considered unfeasible if the Motion Planner cannot find in a predefined amount of time an obstacle-free path that connects the end-effector initial configuration and the object. In case of a feasible action, the Motion Planner outputs the joint velocities $\dot{\boldsymbol{q}}(t)$ that make the robot actually perform the object relocation, obtaining

a new state s_k. In both cases, a reward r_k, depending on the feasibility g_k of the action a_k is generated.

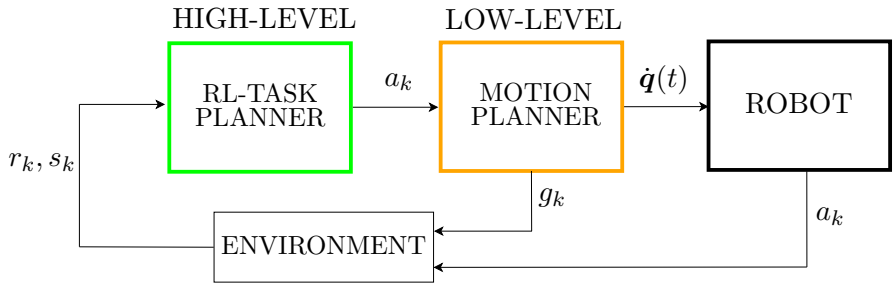

Figure 3.1: Representation of the proposed architecture: the RL-Task Planner chooses the action a_k with an appropriate policy, while the Motion Planner provides information about the feasibility g_k of the chosen action a_k in terms of fulfillment of kinematic constraints. In case of a feasible action, the joint velocities vector $\dot{\boldsymbol{q}}(t)$ is sent to the Robot that actually performs it, relocating the object. The environment elaborates the information related to the feasibility g_k of the action and generates a reward signal r_k and the new state s_k, which are used to update the RL agent.

In the following subsections, algorithmic details about the two layers of the architecture are shown. It is worth noticing that the word *task* is used in both layers, but with different meanings.

Within the low-level layer, it represents the generic elementary control objectives used in the inverse kinematics framework, whereas for the high-level one, it represents a discrete planning for the RL agent.

Low-level: Motion Planner

As previously mentioned, the low-level Motion Planner provides information about the feasibility of the actions a_k requested by the high-level RL-Task Planner and computes the joints velocity vector $\dot{\boldsymbol{q}}(t)$ that allow the robot to reach and relocate the selected object. More in detail, it is composed by three processes (see Fig. 3.2): the actions-objects mapping, the sampling-based algorithm and the Set-based Task-Priority Inverse Kinematics (STPIK)-check [38].

First of all, the action a_k is translated in a desired end-effector pose $\boldsymbol{\eta}_{\text{ee,d}}$, in which the desired position is set as the constant position of the object to

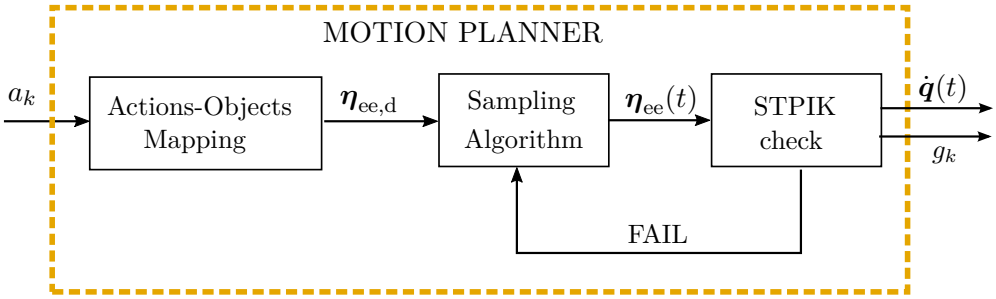

Figure 3.2: Motion planner architecture, designed as three blocks: the Action-Objects Mapping translates the actions in constant desired end-effector poses; the Sampling Algorithm computes obstacle-free trajectories for the end-effector; the STPIK (Set-based Task-Priority Inverse Kinematics) checks the feasibility of the trajectories in terms of joint-level kinematic constraints (joint limits and self-hits).

relocate and the desired orientation is chosen in order to be suitable for the grasping phase. This information feeds the sampling algorithm, which is in charge of finding an obstacle-free, time-varying trajectory $\boldsymbol{\eta}_{\text{ee,d}}(t)$ between the end-effector initial position $\boldsymbol{\eta}_{\text{ee},0}$ and $\boldsymbol{\eta}_{\text{ee,d}}$.

The considered sampling algorithm is the Rapidly-exploring Random Tree (RRT) Connect [99] which is a bidirectional method based on growing two graphs rooted in $\boldsymbol{\eta}_{\text{ee},0}$ and $\boldsymbol{\eta}_{\text{ee,d}}$, respectively. Therefore, the algorithm is able to find a global path connecting the start and final points more efficiently than single tree search approaches. In this step the sampling is performed in Cartesian-space, and the constraints taken into account are defined only in Cartesian-space as well, i.e. the output of the sampling-based algorithm is an obstacle-free path, obtained without considering all the joint-space safety tasks (e.g., joint limits and self-hits). They are handled by the STPIK-check block, which simulates the candidate trajectory $\boldsymbol{\eta}_{\text{ee,d}}(t)$ and checks their fulfillment.

Considering a robot manipulator with n DoFs and being $\boldsymbol{q} = [q_1 \dots q_n]^T$ its joint position vector, the STPIK algorithm allows to perform several tasks simultaneously. In detail, for a generic task $\sigma \in \mathbb{R}^v$, where $v \in \mathbb{N}^+$ is the task dimension, the Closed Loop Inverse Kinematics (CLIK) [77] algorithm can be applied in order to compute the needed joint velocities for achieving a specific

desired value $\boldsymbol{\sigma}_d(t)$

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{\dagger}(\dot{\boldsymbol{\sigma}}_d + \boldsymbol{K}\tilde{\boldsymbol{\sigma}}) \,, \tag{3.1}$$

where $\boldsymbol{J}^{\dagger}$ is the Moore-Penrose pseudoinverse of the task Jacobian matrix $\boldsymbol{J}(\boldsymbol{q}) \in \mathbb{R}^{6\times n}$, $\dot{\boldsymbol{\sigma}}_d \in \mathbb{R}^{v}$ is the time derivative of $\boldsymbol{\sigma}_d$, $\boldsymbol{K} \in \mathbb{R}^{v\times v}$ is a positive-definite gain matrix and $\tilde{\boldsymbol{\sigma}} = \boldsymbol{\sigma}_d(t) - \boldsymbol{\sigma}(t)$ is the task error.

For a redundant robot the number of joints n is greater than the task dimension v, and such redundancy can be exploited to perform multiple tasks simultaneously. In particular, the elementary tasks can be arranged in a hierarchy $\mathcal{H}$ and the solution can be computed by projecting the velocity components of the lower priority tasks onto the null space of the higher priority ones, in order to filter out the components that will affect them. In this way, the accomplishment of the primary task is always guaranteed, while the lower-priority ones are executed *at best*. Therefore, considering a hierarchy composed by K tasks, the joint velocities $\dot{\boldsymbol{q}}$ can be computed recursively as [100]

$$\dot{\boldsymbol{q}}_K = \sum_{i=1}^{K} (\boldsymbol{J}_i \boldsymbol{N}_{i-1}^{A})^{\dagger} (\dot{\boldsymbol{\sigma}}_{i,d} + \boldsymbol{K}_i \tilde{\boldsymbol{\sigma}}_i - \boldsymbol{J}_i \dot{\boldsymbol{q}}_{i-1}) \,, \tag{3.2}$$

where $\boldsymbol{N}_i^A$ is the null space of the augmented Jacobian matrix $\boldsymbol{J}_i^A$, obtained by stacking the task Jacobian matrices from task 1 to i. This task priority-framework has been extended to handle also tasks in which the control objective is to keep the task value $\boldsymbol{\sigma}$ within a certain set, i.e. above a lower threshold and below an upper one. This is the case of tasks such as joint limits, where the control objective is to keep the joints position within their physical limits, or obstacle avoidance, where the control objective is to keep the end-effector of the manipulator at a minimum distance from potential obstacles. This kind of tasks has been defined as *set-based*, and their handling within classical task-priority framework is managed through a proper insertion/removal of tasks in the hierarchy. For more details about the specific employed algorithm, the reader is referred to [101, 102].

In case of a constraint violation, a *fail* signal is sent to the sampling algorithm block that generates a new $\boldsymbol{\eta}_{\mathrm{ee,d}}(t)$, starting a new iteration. The process

terminates when the sampling algorithm finds a path that overcomes the IK-check, or when it cannot find any feasible path in a predefined amount of time.

After the execution of the algorithm, the label g_k is set according to the result of the planning as following

$$g_k = \begin{cases} g_U & \text{if } a_k \text{ is unfeasible} \\ g_O & \text{if } a_k \text{ is feasible and it is related to the relocation of obstacle } O_i \\ g_T & \text{if } a_k \text{ is feasible and it is related to the relocation of the target } T \end{cases} \tag{3.3}$$

High-level: RL-Task Planner

The high-level RL-Task Planner is in charge of learning the optimal sequence of obstacles O_i to relocate in order to reach the target T.

Defining as $\mathcal{S}$ the manifold of all the possible sequences composed by $j \leq N_o$ obstacles and the target T, its cardinality is

$$\xi_t = \sum_{j=1}^{N_o} j! \binom{N_o}{j} . \tag{3.4}$$

Let us define as $a_k \in \mathcal{A}_k = \{a_1, \cdots, a_{N_o}, a_T\}$ the action chosen at a specific timestep k. Within the set $\mathcal{A}_k$, the terms a_i $(i = 1, \cdots, N_o)$ are related to the relocation of the objects O_i, whereas a_T represents the action to relocate the target T.

When the RL-Task Planner selects the action a_k in a given state s_k, it receives a reward r_k that depends on its feasibility g_k. A new approach is presented and it is called Q-Tree. More in detail, it is a technique that replaces the well-known static Q-Matrix of the Q-learning algorithm with a dynamic rooted tree structure that is built over $E_{\max}$ episodes of the algorithm. The advantage of this approach regards the computational burden. Indeed, for this state representation s_k (e.g. sequences of relocated objects until timestep k), considering the classical Q-Learning, all the possible combination must be allocated a-priori. Due the combinatorial nature of the problem, it is very hard, while

considering the Q-Tree, the pre-allocation is not necessary because the structure is built dynamically. Each node of the tree represents a state s_k, with an associated sequence defined as $\mathcal{S}_{s_k} = \{O_{i_1}, O_{i_2}, \cdots, O_{i_U}\}$, that contains information about the objects O_{i_x} relocated until timestep k.

Here, the value associated to edge χ_k between s_k and s_{k+1} is set as the Q-value Q_{χ_k}, and updated at each time-step as in Eq. (2.67).

At the beginning of the first episode, the tree is initialized with a root node s_H, that represents the initial state in which the robot is in a predefined joints configuration and all the objects are in their initial positions. Within each episode $E_h (h = 1, \ldots, E_{\max})$, the tree gets updated at each time-step k after the choice of an action a_k as follows:

- if $O_{i_x} \notin \mathcal{S}_{s_k}$, meaning that the action a_k related to the object O_i has not been chosen in any previous episode, a new node s_{k+1} with associated sequence $\mathcal{S}_{s_{k+1}} = \{\mathcal{S}_{s_k}, a_k\}$ is allocated, updating the edge value between s_k and s_{k+1};
- if $O_{i_x} \in \mathcal{S}_{s_k}$, meaning that the action a_k related to the object O_i has already been chosen in a previous episode, the edge that connects the nodes s_k and s_{k+1} is updated without allocating a new node.

Then, at the beginning of each one of the following episodes, the algorithm starts again from the node s_H keeping the tree structure built until that episode. It is worth noticing that given a specific s_k, a_k is selected from a set that does not contain actions already considered unfeasible in previous episodes in order to speed up the process.

An example of the Q-Tree algorithm is reported in Fig. 3.3.

3.1.2 Exploration Policies

Given a node s_k with an associated $\mathcal{S}_{s_k}$, the RL-Task Planner chooses a_k with the following possible policies:

- **Learning Random Exploration** (**LRND**): it chooses $a_k \in \mathcal{A}_k$ in a completely random manner.

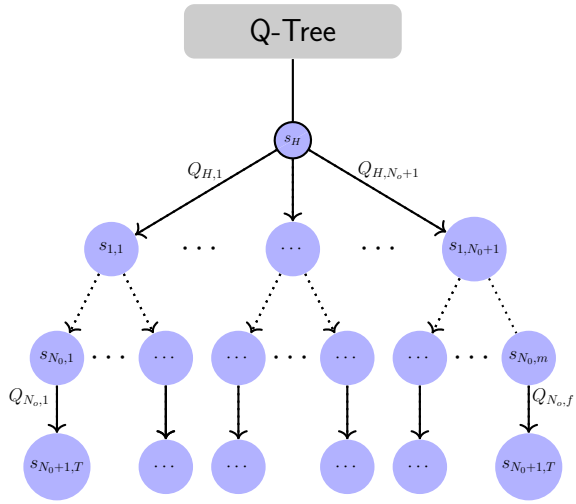

Figure 3.3: Example of complete Q-Tree: The node root s_H represents the initial configuration where all objects are in the initial location. The last nodes contain the target T.

- **Random Exploration with Heuristics** (**H-LRND**): it first tries to reach the target (choosing a_{k_T}). If it is unfeasible, a_k is chosen randomly among the other actions in $\mathcal{A}_k$.
- **ε-Greedy Exploration with Heuristics** (H-εG): it chooses a_k exploiting the ε-Greedy technique. In detail, it chooses a random action a_k with probability ε and the action associated with the maximum edge value with probability $1-\varepsilon$ (this corresponds to the $\max_{a'} Q_*(s', a')$ in Eq. (2.63)).

Regarding the ε-Greedy Exploration with Heuristics policy, in order to ease the exploitation, ε is updated at the beginning of each episode E_h as

$$\varepsilon(h) = \varepsilon_0 \left(\frac{\varepsilon_{\min}}{\varepsilon_0} \right)^{\frac{h-1}{E_{\max}-1}}, \tag{3.5}$$

where ε_0 and $\varepsilon_{\min}$ are the initial and the minimum ε value respectively.

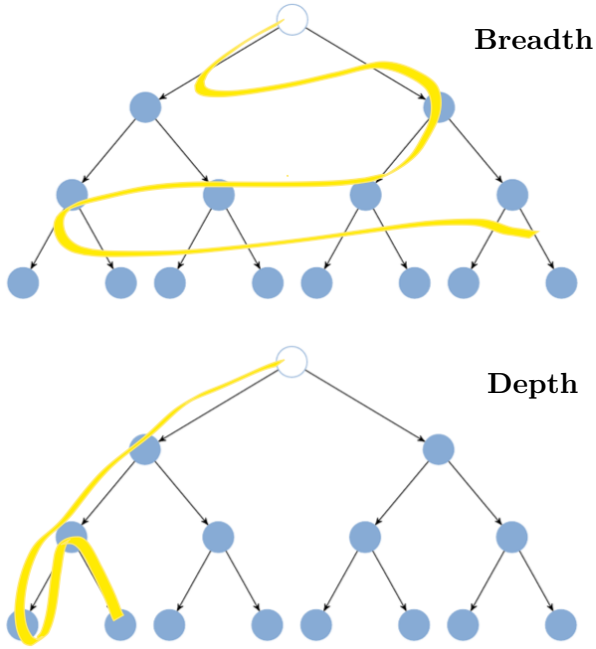

Figure 3.4: Different exploration method: Breadth prefers to explore at the same levels, whereas the Depth one prefers to move forward.

Tree-search Methods

For each one of Q-Tree exploration policies, a comparison between two different search methods for investigating their effect on the learning dynamics is considered. The search methods are shown in Fig. 3.4:

- *Breadth*: the RL-Task Planner explores an entire tree level before moving on the following levels;
- *Depth:* the RL-Task Planner prefers moving towards nodes at following levels rather than the ones at the same level.

In other words, the employed search method affects the behavior of the RL-Task Planner in case of an unfeasible action is chosen. Considering the breadth method the current episode is terminated, whereas considering the depth method it continues choosing another action among the possible ones.

Q-Tree Learning Algorithm

This subsection provides details on Q-Tree learning algorithm. The inputs of the algorithm are:

- Number of obstacles;
- Robot end-effector pose;
- Search method;
- Exploration policy.

At the beginning of the first episode, the Q-Tree is initialized allocating the root node s_H. At each timestep k, the RL-Task Planner chooses an action $\overline{a}_k$ following the selected policy and search method and it queries the low-level Motion Planner, which tries to plan a trajectory for reaching the object associated to a_k. It returns the information about the action feasibility g_k that is then used to compute the reward r_k. At this point the tree is updated adding a new node and updating the corresponding edge value. The procedure gets iterated until one of the termination conditions of the episode is met.

The training phase terminates when the maximum number of episodes $E_{\max}$ is reached, or when the Q-Tree is not significantly updated anymore over consecutive episodes, both in terms of new nodes allocation and edge values. In detail, defining as

$$\overline{Q}_h = \sum_{j=1}^{l-1} |Q_{\chi_j}| \,, \tag{3.6}$$

the sum of all the edge values at the end of the episode h, the termination condition is

$$|\overline{Q}_h - \overline{Q}_{h-1}| \leq \beta \,, \tag{3.7}$$

where $\beta > 0$ is a predefined threshold.

Optimality Analysis

After the training phase, a decision tree is available from which the (sub)-optimal sequence can be extracted. In detail, starting from the root node s_H, the agent selects the action a in the set $\mathcal{A}$ corresponding to the tree branch

Data:
ObstaclesNumber
RobotPose // End-Effector position
SearchMethods // Breadth, Depth
ExplorationPolicy // LRND,H-LRND,H-εG
Result: ActionsSequence
// Allocation of the Tree root node
CreateRootNode();
$\overline{Q}_0 = 0$ // Starting training for the RL-Task Planner
while $h \leftarrow 1 < E_{max}$ *and* $|\overline{Q}_h - \overline{Q}_{h-1}| > \beta$ **do**
  // Reset scene to initial condition
  CurrentState = s_H ;
  for $j \leftarrow 1$ **to** I_{max} **do**
    // Selection next action
    NextAction = TaskPlanner(Policy);
    // Feedback Motion Planner
    MotionPlanner(CurrentState,NextAction);
    r = getReward();
    // Update
    if *IsInTree(CurrentState) == false* **then**
      AddNode(CurrentState);
    end
    UpdateTreeEdgesValue();
  end
end
ActionsSequence = SelectSequenceWithMaximumEdgeValues();

Algorithm 3.1: Q-Tree

with the highest weight, i.e., the maximum $\mathcal{Q}$ value. Indeed, the latter represents the optimal choice leading to the target through the shortest tree nodes sequence. Within the aim to prove its effectiveness, let us take into consideration Eq. (2.67). It can be rewritten as

$$\begin{aligned} Q_{k+1}(s_k, a_k) =& (1-\alpha) Q_k(s_k, a_k) + \\ & \alpha r_{k+1} + \alpha\gamma \max_{a \in \mathcal{A}} Q_k(s_{k+1}, a_{k+1}) \ . \end{aligned} \tag{3.8}$$

From the above equation, it can be possible to notice that it owns the following structure

$$y(k+1) = (1-\alpha)y(k) + \alpha\gamma u(k) \ , \tag{3.9}$$

where $(1-\alpha)$ is the eigenvalue responsible of the convergence time and $y(k)$, $u(k)$ are the output, input of discrete system, respectively. The corresponding system *transfer function* $G(z)$ is defined in the $\mathcal{Z}$ domain as

$$G(z) = \frac{Y(z)}{U(z)} = \frac{\alpha\gamma}{z-(1-\alpha)} \ , \tag{3.10}$$

from which it is possible to obtain the static gain g

$$g = \lim_{z\to 1} G(z) = \gamma \ . \tag{3.11}$$

When the tree reaches the steady state, the edge local value is given by

$$y(k) = \gamma u(k) \ , \tag{3.12}$$

with $y(k)$, $u(k)$ the output and input of the discrete system represented by the single tree edge, respectively. Equation (3.12) is repeated for each edge inside any feasible tree nodes sequence, i.e., linking the root node s_H to the target node T. Then, this affects the reward propagation from T to s_H meaning that a longer sequence implies a larger damping of the tree weights, i.e., in proximity of the root node the edges belonging to longest sequences present smaller weight values. Considering an optimal solution, e.g., at tree level m, it is possible to compute (at steady state) the value of edge weights starting from node s_H, $\overline{Q}_k(s_H, a_k)$ as

$$\overline{Q}_k(s_H, a_k) = \gamma^m r_T \ , \tag{3.13}$$

where r_T is the reward on the target T object. Therefore, in presence of an optimal solution at level m and a sub-optimal solution at level b, with $m < b$, it is straightforward that

$$\overline{Q}_m(s_H, a_m) > \overline{Q}_b(s_H, a_b) \ , \tag{3.14}$$

with actions a_m, a_b corresponding to the optimal and sub-optimal branches, respectively. In virtue of the above consideration, at the end of the training, the optimal sequence of actions can be iteratively retrieved from the tree starting from the root node, s_H, as

$$a_k^\star = \max_{a \in \mathcal{A}} \overline{Q}_k(s_k, a) \ , \tag{3.15}$$

and with $s_{k+1} = Q_k(s_k, a_k^\star)$.

3.2 Simulation and Experiments

Two different case study are considered, where the objects are identical or with different weights, e.g. fragility or physical weights.

Case 1: Identical Objects

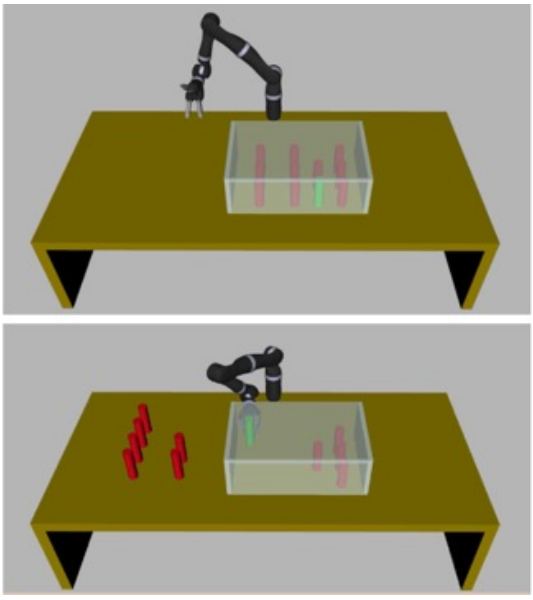

Figure 3.5: An example of cluttered environment. The target is represented by the cylinder in green, while obstacles are in red. Starting from the scenario on the top, the manipulator is required to relocate some obstacles in order to grasp the target (bottom scenario).

In this case, all the objects have the same properties and the reward r_{k_1} function is defined as

$$r_{k_1}(s_k, a_k) = \begin{cases} 0, & \text{if } g_k = g_O \\ -1, & \text{if } g_k = g_U \\ 100, & \text{if } g_k = g_T \end{cases}, \tag{3.16}$$

meaning that, given a state $s \in \mathcal{S}$ and an action $a \in \mathcal{A}$, the reward is:

- 0 if the obstacle i can be relocated according to the feedback provided by the motion planner;
- -1 if either an obstacle or the target cannot be relocated because of occlusions and robot constraints;
- 100 if $a = a_T$ and the target can be grasped and relocated.

According to how mentioned above, for this case only the BFS method exploration is considered, and for all the learning algorithms is set $\alpha = 0.5$ and $\gamma = 0.9$ and the training is repeated 50 times from scratch. In the case of H-εG algorithm, it is $\varepsilon_0 = 1$ and $\varepsilon_{\min} = 10^{-4}$ in Eq. (3.5). The training ends whenever condition Eq. (3.7) is satisfied or the maximum number of episodes $E_{\max} = 5000$ is reached. As reported in Fig. 3.6, H-εG is the best algorithm when it comes to reach for the first time the optimal solution and the interrogation at motion planner system. In addition, in Fig. 3.7 is possible to see that the algorithm H-εG$_b$ reach the steady state condition faster than other algorithms in every scenario for the ε value chosen. This latter depends on whether that the ε introduces variation on the learning dynamic.

Algorithm	E_{ss}	MP_q	$E_{1_{st}}$
LRND$_b$	144	42	20
H-LRND$_b$	119	37	18
H-εG$_b$	115	34	16

Table 3.1: Analysis considering $\alpha = 0.5$, and $\gamma = 0.9$ and as evaluation criteria: the number of episodes E_{ss} necessary to converge, the number of motion planning queries MP_q and the the episode number $E_{1_{st}}$ in which the optimal solution is reached for the first time. This case is for the Scenario 1 with obstacles number $N_o = 5$.

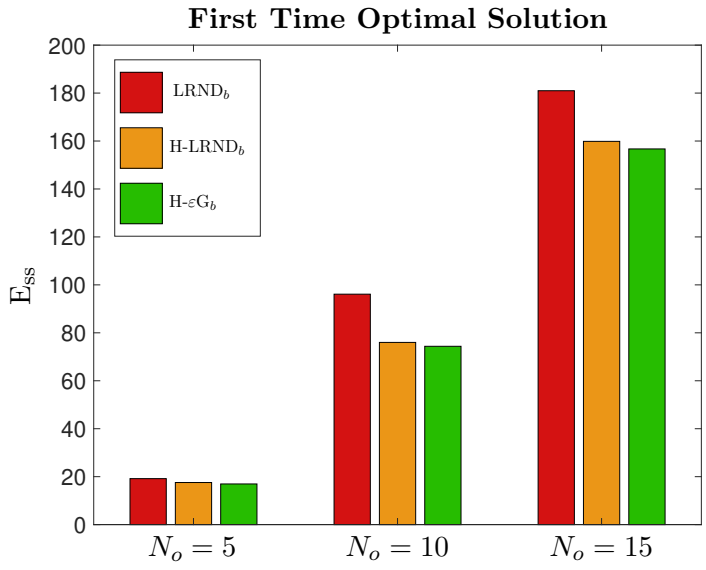

Figure 3.6: Comparison of three learning policies defined for each scenario. The average episodes is calculated on 50 training and represent the first time that the target T is reached through the optimal sequence.

An additional comparison with the no learning technique (RND$_b$) is here considered. The implemented algorithm *simply* explores the graph resorting to a BFS policy by randomly choosing actions until the target is reached. This is clearly a fast method for a very small number of objects, but that becomes quickly intractable. In fact, scenarios $2-3$ described above could not be solved in a *reasonable* amount of time. Concerning the scenario 1 introduced above, 50 trials were executed and an optimal solution is found in $\approx$ 60% of cases (100% in the case of learning techniques). In the remaining 40% of trials, all obstacles are relocated leading to a sub-optimal solutions.

These simulations are related to the Breadth exploration described above and the considered scenarios is reported in Fig. 3.5.

Case 2: Different Objects

Given that, in general, it is possible to find multiple feasible sequences, an optimization procedure can be designed in order to minimize a metric related to the energy spent during the relocation procedure. At this time, without

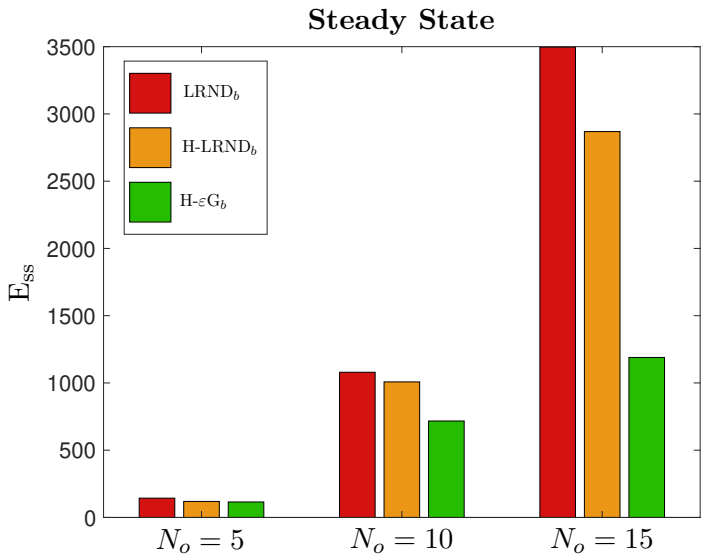

Figure 3.7: Comparison of convergence between learning algorithms defined above for each scenario. The average episodes number is calculated on 50 training.

loss of generality, obstacles with different weights ϕ_i in the range $(0, 1]$ Kg, that is the standard object weight normalized with respect to the KINOVA Jaco^2 7-DoFs maximum payload. It is worth noticing that the weight can be associated to any other physical characteristics of the objects, e.g. fragility or volume. In this perspective, the aim of the optimization problem is to find a feasible sequence $\mathcal{S}_u^T$ that minimizes the cost function:

$$\Phi_{\mathcal{S}_u} = \sum_{i \in \mathcal{S}_u} \phi_i. \tag{3.17}$$

Differently from previous, in this case, all the objects have different properties, with an associated weight and the reward r_{k_s} function is defined as

$$r_{k_2}(s_k, a_k) = \begin{cases} -\phi_i, & \text{if } g_k = g_O \\ -10, & \text{if } g_k = g_U \\ 100, & \text{if } g_k = g_T \end{cases}, \tag{3.18}$$

where ϕ_i is the weight of the object related to the action a_k.

Algorithm	E_{ss}	MP_q	$E_{1_{st}}$
$LRND_b$	1079	236	96
H-$LRND_b$	1007	218	77
H-εG_b	717	188	74

Table 3.2: Analysis considering $\alpha = 0.5$, and $\gamma = 0.9$ and as evaluation criteria: the number of episodes E_{ss} necessary to converge, the number of motion planning queries MP_q and the the episode number $E_{1_{st}}$ in which the optimal solution is reached for the first time. This case is for the Scenario 2 with obstacles number $N_o = 10$.

Algorithm	E_{ss}	MP_q	$E_{1_{st}}$
$LRND_b$	3496	665	181
H-$LRND_b$	2868	588	160
H-εG_b	1189	331	156

Table 3.3: Analysis considering $\alpha = 0.5$, and $\gamma = 0.9$ and as evaluation criteria: the number of episodes E_{ss} necessary to converge, the number of motion planning queries MP_q and the the episode number $E_{1_{st}}$ in which the optimal solution is reached for the first time. This case is for the Scenario 3 with obstacles number $N_o = 15$.

In order to assess the performance of the proposed relocation strategy, a parametric analysis has been conducted considering the following parameters:

- Learning rate α, variable in the interval $[0.3 - 1]$;
- Discount factor γ, variable in the interval $[0.1 - 0.9]$.

The indicators are the same used in the Case 1, but the maximum number of episodes E_{max} for each training in Algorithm 3.1 is set to $E_{max} = 10000$ for the LRND and H-LRND techniques (both breadth and depth search methods) and, for the ε-greedy techniques, E_{max} is set to the maximum number of episodes required by the LRND to converge.

Figure 3.9 reports, as an example, the evolution of $\overline{Q}_h$ in Eq. (3.6) normalized with respect to its steady state value, $\overline{Q}_\infty$, in the case of H-$LRND_b$ and Scenario 1 for different values of α and γ. The figure shows how the tree reaches a steady value after a certain number of episodes and how α and γ affect the converge in this specific training instance.

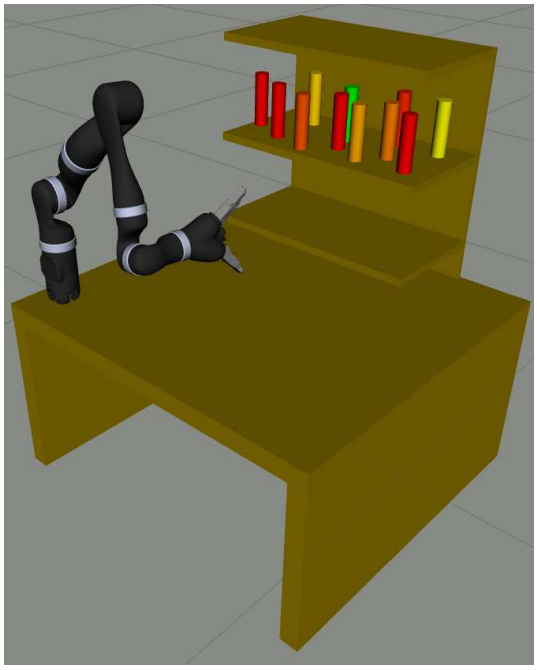

Figure 3.8: The target is represented by a green cylinder, while the obstacles are indicated with a different color, based on weights, which models concepts as fragility or energetic-related metrics. The robot, a KINOVA Jaco^2 in the case study, needs to eventually relocate objects in order to reach the target.

The simulation results are reported both numerically in Tables 3.4-3.6 and graphically in Figs. 3.10-3.12; they are computed by running each case study 50 times and averaging the obtained results.

In the following, the subscripts b and d represent the Breadth and Depth exploration, respectively. As a general consideration, the H-εG$_d$ algorithm exhibits optimal or slightly sub-optimal performance for each of the three considered metrics and scenarios. More in detail, H-εG$_d$ significantly outperforms the other methods concerning E$_{\text{ss}}$ especially in case of complex scenarios (see Table 3.6). With regards to motion planning queries MP$_q$, which equals the number of explored tree branches, H-εG$_b$ outperforms the other ones and slightly H-εG$_d$. Furthermore, regarding the number of episodes E$_{1_{st}}$ necessary for the learning agent to find the first optimal path, it is worth noticing that depth approaches exhibits significantly better performance with respect to breadth

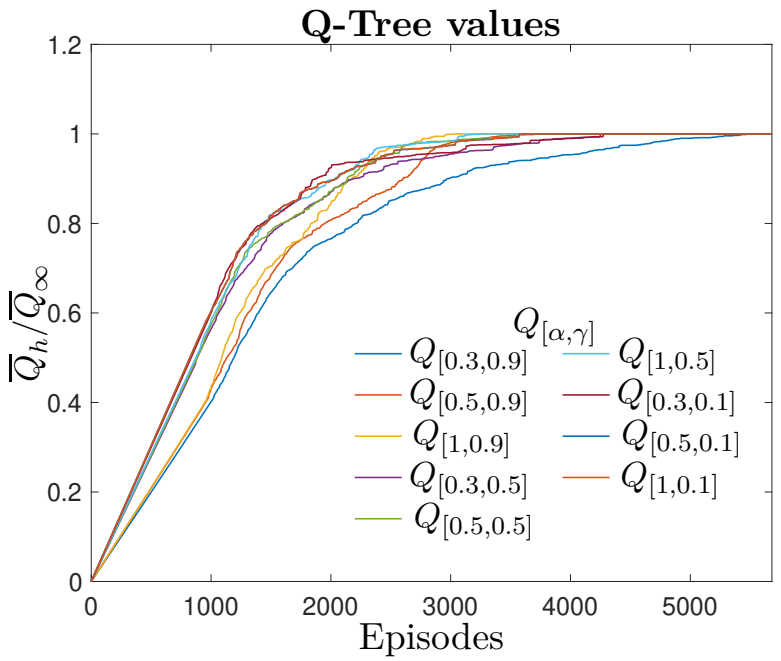

Figure 3.9: Plots represent the sum of all tree edge values $\overline{Q}_h$ normalized with respect to the steady state value $\overline{Q}_\infty$ for different values of α and γ in the case of H-LRND$_b$ approach and Scenario 3.

ones, with very similar results for the three policies LNRD, H-LNRD, H-εG.

In support of Fig. 3.13, the three exploration policies are compared with respect to the performance indexes described above. As it is possible to notice, the magnitude of each indicator is reduced significantly with respect to the Breadth approach by using the Depth approach.

In Figs. 3.14- 3.15, the minimum and maximum average values related to the first time in which the agent reaches the target (through the optimal sequence) for each scenario are reported. In terms of reaching for the first time the optimal solution, the Depth drastically outperforms the Breadth approach.

In the experimental case study, $N_o = 10$ obstacles (bottles) are considered. Furthermore, the perception module is enabled to detect and recognize obstacles and target, providing their positions and orientations accurately.

The scenario is shown in Fig. 3.16 and two shelves are present: the top shelf is

$[\alpha,\gamma]$	Algorithm	E_{ss}	MP_q	$E_{1_{st}}$	$[\alpha,\gamma]$	Algorithm	E_{ss}	MP_q	$E_{1_{st}}$	$[\alpha,\gamma]$	Algorithm	E_{ss}	MP_q	$E_{1_{st}}$
[0.3,0.9]	$LRND_b$	511	84	50	[0.3,0.5]	$LRND_b$	441	84	51	[0.3,0.1]	$LRND_b$	283	84	43
	H-$LRND_b$	487	74	32		H-$LRND_b$	422	74	32		H-$LRND_b$	267	74	28
	H-εG_b	365	69	32		H-εG_b	289	67	39		H-εG_b	235	73	28
	$LRND_d$	449	84	30		$LRND_d$	388	84	26		$LRND_d$	233	84	28
	H-$LRND_d$	414	74	14		H-$LRND_d$	372	74	7		H-$LRND_d$	203	74	8
	H-εG_d	364	72	12		H-εG_d	315	71	15		H-εG_d	165	73	12
[0.5,0.9]	$LRND_b$	355	84	52	[0.5,0.5]	$LRND_b$	296	84	48	[0.5,0.1]	$LRND_b$	246	84	48
	H-$LRND_b$	344	74	31		H-$LRND_b$	282	74	32		H-$LRND_b$	212	74	29
	H-εG_b	229	64	34		H-εG_b	245	67	34		H-εG_b	176	71	32
	$LRND_d$	291	84	22		$LRND_d$	245	84	18		$LRND_d$	176	84	18
	H-$LRND_d$	272	70	12		H-$LRND_d$	226	74	11		H-$LRND_d$	149	74	12
	H-εG_d	271	70	14		H-εG_d	214	70	9		H-εG_d	141	73	12
[1,0.9]	$LRND_b$	170	84	50	[1,0.5]	$LRND_b$	173	84	51	[1,0.1]	$LRND_b$	177	84	45
	H-$LRND_b$	151	74	29		H-$LRND_b$	152	74	34		H-$LRND_b$	142	74	33
	H-εG_b	146	65	39		H-εG_b	148	64	29		H-εG_b	131	69	30
	$LRND_d$	122	84	26		$LRND_d$	104	84	23		$LRND_d$	114	84	20
	H-$LRND_d$	94	74	13		H-$LRND_d$	95	74	11		H-$LRND_d$	87	74	10
	H-εG_d	82	70	12		H-εG_d	86	70	13		H-εG_d	79	72	13

Table 3.4: Parametric analysis considering (α, γ) parameters and as evaluation criteria: the number of episodes E_{ss} necessary to converge, the number of motion planning queries MP_q and the the episode number $E_{1_{st}}$ in which the optimal solution is reached for the first time. This case is for the Scenario 1 with obstacles number $N_o = 5$.

the starting location for both objects and target, which needs to be relocated to the bottom shelf. Objects are initially positioned on the top shelf as in Fig. 3.16. Additionally, the objects have different colors, indicating that they have different weights ϕ_i. The green bottle is the target, while the weights of the other objects are as follows: $\phi_1 = 0.15, \phi_2 = 0.13, \phi_3 = 0.23, \phi_4 = 0.44$, $\phi 5 = 0.65$, $\phi_6 = 0.78$, $\phi_7 = 0.13$, $\phi_8 = 0.34$, $\phi_9 = 0.56$, $\phi_{10} = 0.89$ Kg.

Objects are relocated by the KINOVA Jaco^2 robot equipped by a three finger gripper, manufactured by KINOVA company and available in the LAI-Robotics laboratory of the University of Cassino and Southern Lazio, Italy.

In order to acquire the objects position, an INTEL RealSense D455 RGB-D camera is adopted together with an ArUco marker [103] which provides a common reference frame and *You Only Look Once* (YOLO) software [104] for the objects segmentation. A desktop PC with CPU Intel(R) Core(TM) i9-9900KF 3.60GHz and GPU GeForce RTX 2070 Super equipped with Ubuntu

$[\alpha,\gamma]$	Algorithm	E_{ss}	MP_q	$E_{1_{st}}$	$[\alpha,\gamma]$	Algorithm	E_{ss}	MP_q	$E_{1_{st}}$	$[\alpha,\gamma]$	Algorithm	E_{ss}	MP_q	$E_{1_{st}}$
[0.3,0.9]	$LRND_b$	1606	623	449	[0.3,0.5]	$LRND_b$	1383	622	437	[0.3,0.1]	$LRND_b$	1043	622	470
	H-$LRND_b$	1596	598	410		H-$LRND_b$	1279	598	426		H-$LRND_b$	952	598	414
	H-εG_b	1026	518	427		H-εG_b	868	572	407		H-εG_b	930	593	421
	$LRND_d$	1035	622	19		$LRND_d$	813	622	16		$LRND_d$	485	622	16
	H-$LRND_d$	974	598	16		H-$LRND_d$	688	598	15		H-$LRND_d$	382	598	16
	H-εG_d	673	594	15		H-εG_d	553	595	13		H-εG_d	299	596	15
[0.5,0.9]	$LRND_b$	1275	622	451	[0.5,0.5]	$LRND_b$	1118	622	451	[0.5,0.1]	$LRND_b$	939	622	444
	H-$LRND_b$	1211	598	414		H-$LRND_b$	1061	598	420		H-$LRND_b$	887	598	430
	H-εG_b	853	491	411		H-εG_b	718	548	398		H-εG_b	861	593	412
	$LRND_d$	647	622	15		$LRND_d$	564	622	14		$LRND_d$	396	622	13
	H-$LRND_d$	594	598	19		H-$LRND_d$	458	598	13		H-$LRND_d$	293	598	13
	H-εG_d	468	585	15		H-εG_d	388	592	13		H-εG_d	285	597	16
[1,0.9]	$LRND_b$	910	622	432	[1,0.5]	$LRND_b$	894	622	433	[1,0.1]	$LRND_b$	823	621	451
	H-$LRND_b$	855	597	419		H-$LRND_b$	867	598	400		H-$LRND_b$	800	597	423
	H-εG_b	690	456	375		H-εG_b	630	519	414		H-εG_b	771	590	416
	$LRND_d$	268	622	16		$LRND_d$	255	622	21		$LRND_d$	217	621	10
	H-$LRND_d$	227	597	15		H-$LRND_d$	226	598	12		H-$LRND_d$	190	598	19
	H-εG_d	219	572	21		H-εG_d	222	581	13		H-εG_d	181	592	14

Table 3.5: Parametric analysis considering (α, γ) parameters and as evaluation criteria: the number of episodes E_{ss} necessary to converge, the number of motion planning queries MP_q and the the episode number $E_{1_{st}}$ in which the optimal solution is reached for the first time. This case is for the Scenario 2 with obstacles number $N_o = 10$.

18.04 and MATLAB 2020b is adopted in order to find the optimal sequence by running the RL-Task Planner in Fig. 3.1. The same PC is equipped with the *Robot Operating System* (ROS) framework which is in charge to execute the Motion Planner and control the real robot. The software-hardware architecture is reported in Fig. 3.17.

The perception module receives the real scene from the camera and provides information on the objects pose to the RL-Task Planner implemented in MATLAB. This latter selects an action according with its policy and sends it to the Motion Planner implemented in C++. Finally, if there is a free-obstacle path that satisfies all the joint constraints, the robot receives the computed joint velocities to perform the task.

Each one of the actions selected by the RL-Task Planner is related to the relocation of one of the objects in the scene. Regarding the motion planner, the following task hierarchy is considered:

$[\alpha,\gamma]$	Algorithm	E_{ss}	MP_q	$E_{1_{st}}$	$[\alpha,\gamma]$	Algorithm	E_{ss}	MP_q	$E_{1_{st}}$	$[\alpha,\gamma]$	Algorithm	E_{ss}	MP_q	$E_{1_{st}}$
[0.3,0.9]	$LRND_b$	5035	1674	1049	[0.3,0.5]	$LRND_b$	4519	1672	1021	[0.3,0.1]	$LRND_b$	3542	1658	1045
	H-$LRND_b$	4802	1605	958		H-$LRND_b$	3965	1600	1018		H-$LRND_b$	3213	1585	968
	H-εG_b	2829	1353	987		H-εG_b	2809	1520	993		H-εG_b	3051	1572	978
	$LRND_d$	2511	1675	11		$LRND_d$	1712	1675	13		$LRND_d$	708	1673	11
	H-$LRND_d$	2454	1606	12		H-$LRND_d$	1694	1606	11		H-$LRND_d$	684	1605	13
	H-εG_d	1342	1587	10		H-εG_d	777	1597	11		H-εG_d	658	1600	10
[0.5,0.9]	$LRND_b$	4050	1670	1040	[0.5,0.5]	$LRND_b$	3730	1664	1069	[0.5,0.1]	$LRND_b$	3293	1653	1054
	H-$LRND_b$	3985	1599	963		H-$LRND_b$	3579	1593	971		H-$LRND_b$	3047	1581	995
	H-εG_b	2397	1299	984		H-εG_b	2593	1506	992		H-εG_b	2981	1571	955
	$LRND_d$	1590	1675	14		$LRND_d$	1712	1674	13		$LRND_d$	565	1672	13
	H-$LRND_d$	1562	1606	10		H-$LRND_d$	1694	1606	11		H-$LRND_d$	530	1603	8
	H-εG_d	906	1572	13		H-εG_d	553	1588	11		H-εG_d	499	1596	9
[1,0.9]	$LRND_b$	3278	1655	1027	[1,0.5]	$LRND_b$	3245	1653	1025	[1,0.1]	$LRND_b$	3154	1649	1067
	H-$LRND_b$	3027	1580	987		H-$LRND_b$	3038	1581	980		H-$LRND_b$	2931	1578	956
	H-εG_b	2380	1268	1022		H-εG_b	2452	1500	995		H-εG_b	2832	1566	982
	$LRND_d$	618	1672	12		$LRND_d$	564	1670	10		$LRND_d$	451	1669	13
	H-$LRND_d$	594	1604	15		H-$LRND_d$	634	1605	11		H-$LRND_d$	430	1602	12
	H-εG_d	414	1530	12		H-εG_d	382	1556	11		H-εG_d	383	1591	11

Table 3.6: Parametric analysis considering (α, γ) parameters and as evaluation criteria: the number of episodes E_{ss} necessary to converge, the number of motion planning queries MP_q and the the episode number $E_{1_{st}}$ in which the optimal solution is reached for the first time. This case is for the Scenario 3 with obstacles number $N_o = 15$.

- Mechanical joint limits avoidance;
- Self-collision avoidance;
- Obstacle avoidance between the end-effector and the other objects;
- Position and orientation of the arm's end-effector.

Once the agent is trained, the robot starts relocating objects according to the found sequence, namely $\mathcal{S}_4^T = \{O_9, O_3, O_1, O_6, T\}$; a sequence of snapshots relative to object relocation is reported in Fig. 3.18, while a video of the experiment can be found here[1].

[1] http://www.youtube.com/watch?v=2aTqmWzmiJ8

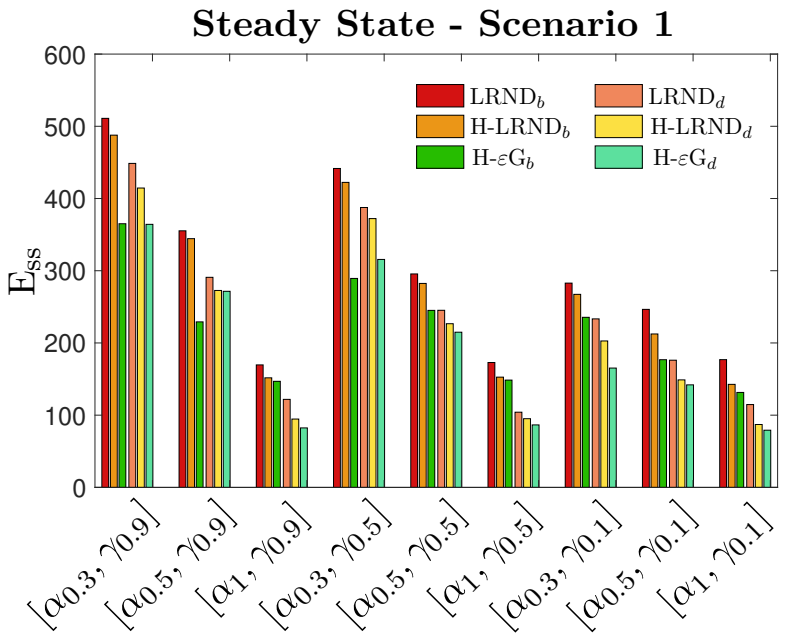

Figure 3.10: Steady State Parametric Analysis: This bar graph is relative to the reaching of optimal solution varying α and γ for the Scenario 1.

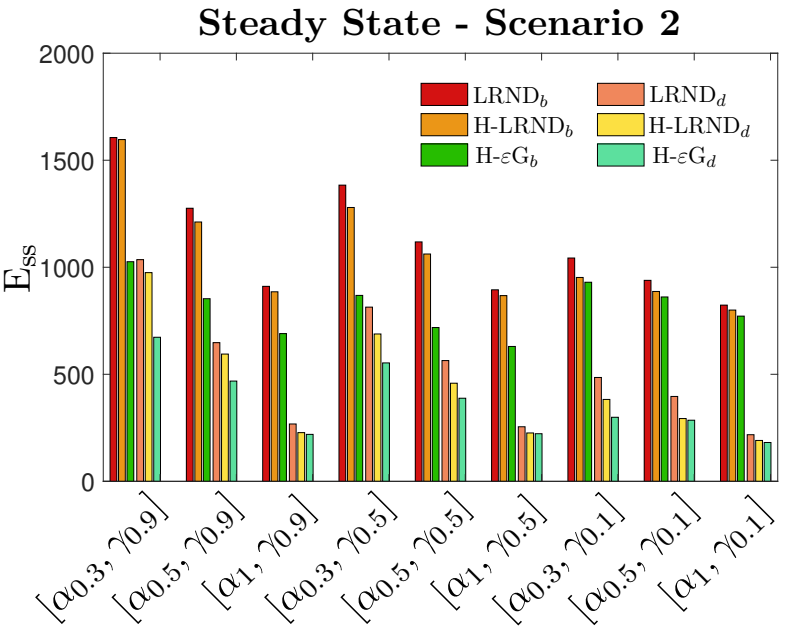

Figure 3.11: Steady State Parametric Analysis: This bar graph is relative to the reaching of optimal solution varying α and γ for the Scenario 2.

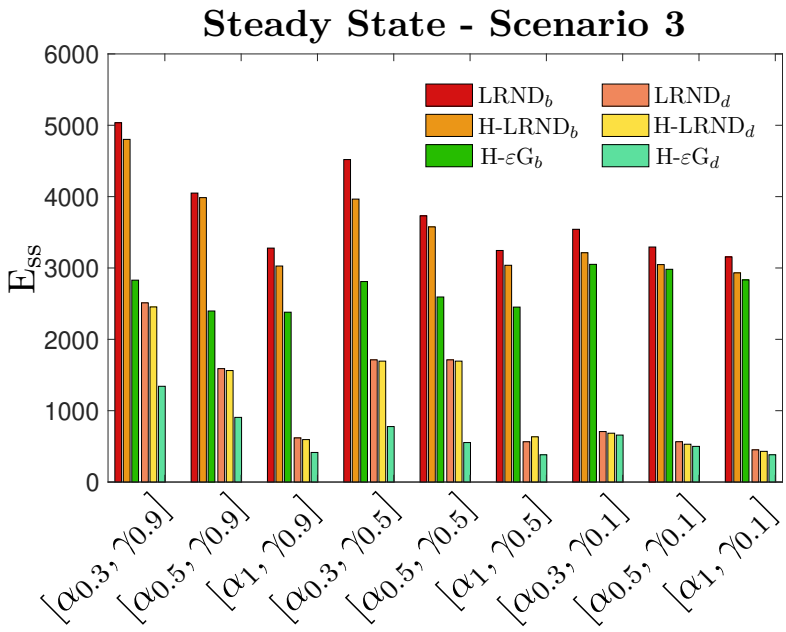

Figure 3.12: Steady State Parametric Analysis: This bar graph is relative to the reaching of optimal solution varying α and γ for the Scenario 3.

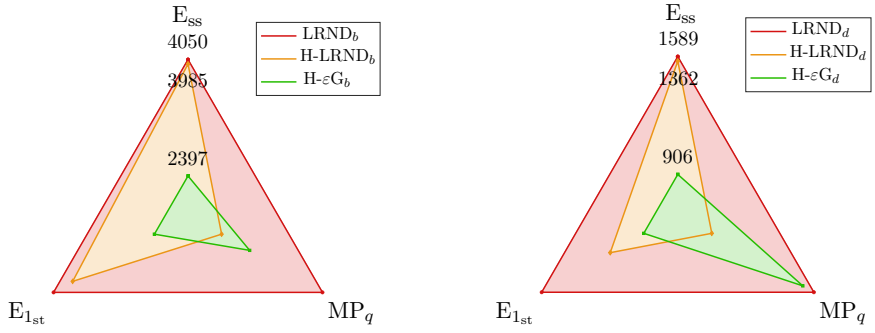

Figure 3.13: Comparison between Breadth (left) and Depth (right) approaches, considering the proposed tree exploration strategies for the Scenario 3, with learning rate $\alpha = 0.5$ and discount factor $\gamma = 0.9$.

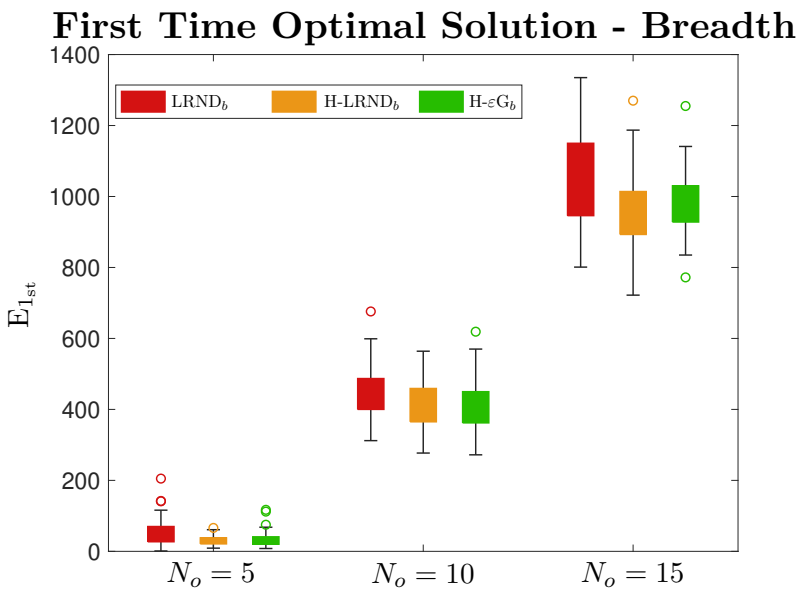

Figure 3.14: Three learning policies comparison: Average, minimum and maximum episode relative at the first time that the agent reaches the target T through the optimal sequence considering Breadth search approach. The statistics are based on 50 training with $\alpha = 0.5$ and $\gamma = 0.9$.

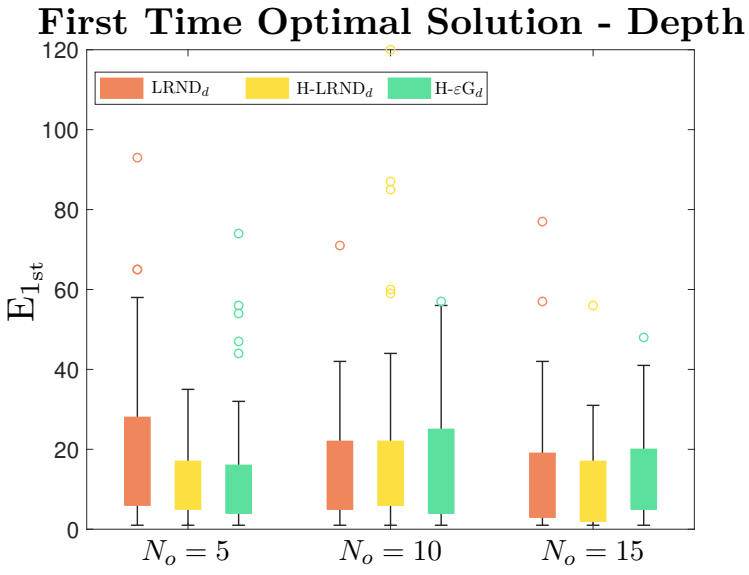

Figure 3.15: Three learning policies comparison: Average, minimum and maximum episode relative at the first time that the agent reaches the target T through the optimal sequence considering Depth search approach. The statistics are based on 50 training with $\alpha = 0.5$ and $\gamma = 0.9$.

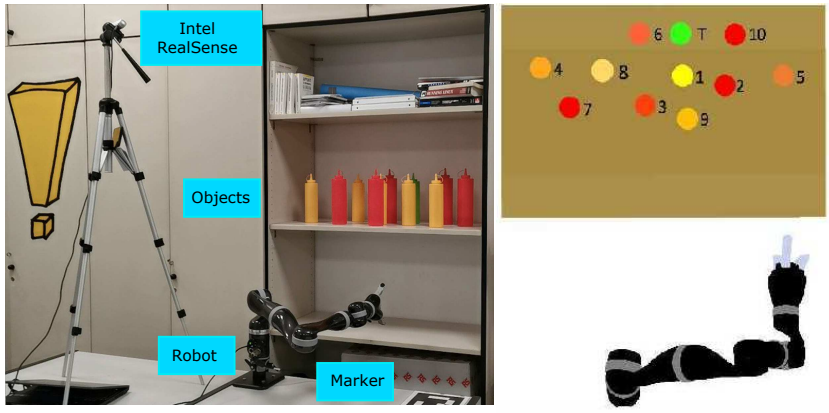

Figure 3.16: Left. The robotic setup adopted to demonstrate the devised approach. Right. A top view representation of target (in green) and obstacles in their initial configuration.

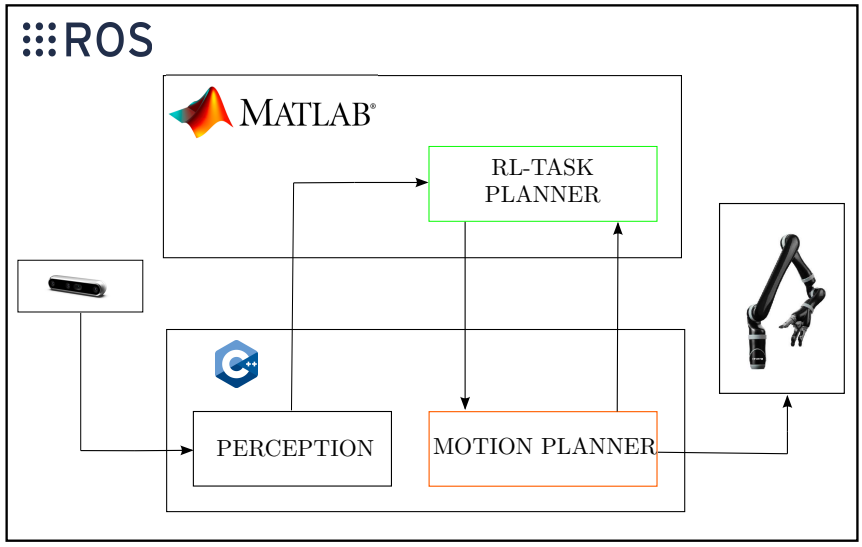

Figure 3.17: Software-Hardware Architecture: The perception module receives the scene from the camera and provides information on the objects pose to the RL-Task Planner (MATLAB). This latter selects an action according to its policy and sends the action to the Motion Planner (C++). Finally, if there is a free-obstacle path that satisfies all the joint constraints, the robot receives the computed joint velocities to perform the task.

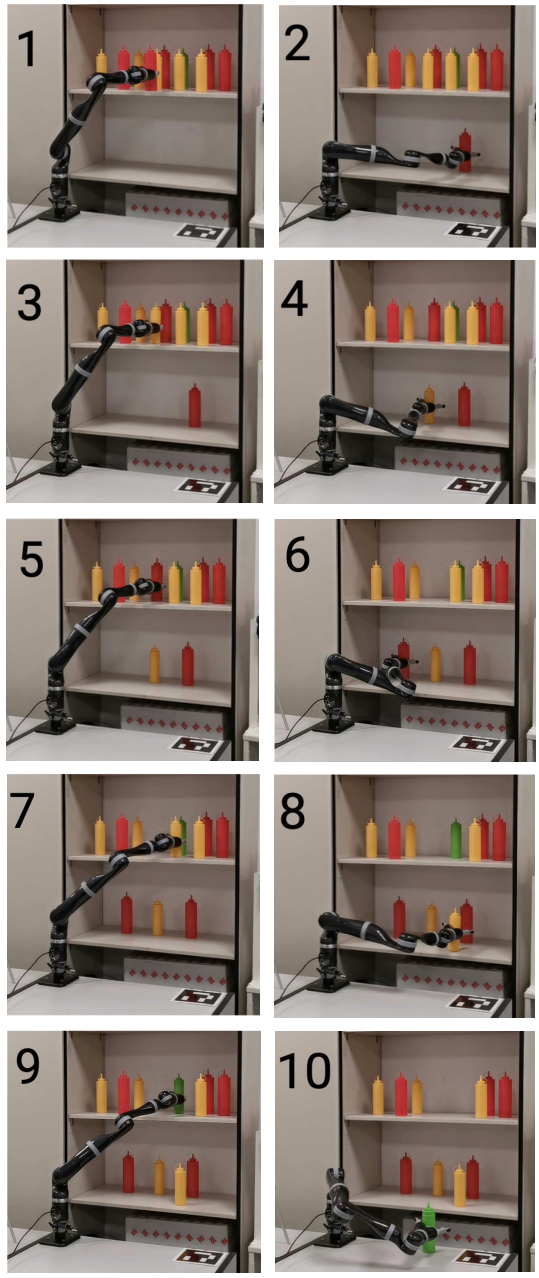

Figure 3.18: Robot manipulator during the validation phase: the KINOVA Jaco^2 is moving the objects in the real world scenario.

Chapter 4

Learning-based Robot Collision Detection

This chapter focuses on the problem of collision detection in an industrial dynamic context.

The time necessary to detect collisions between robot end-effector and environment could be quite long, especially in complex operational scenarios, such as a cluttered environment during a pick-and-place task. For instance, mesh-based approaches may not perform well in certain situations due to the need to reconstruct a mesh from a high volume of information provided by the camera.

A fast tool for collision detection based on deep learning is proposed. The aim is to exploit the knowledge of the robot end-effector mesh, generating depth images and point clouds from it and combine these data with the real ones provided by the camera. Then, these data are sent to different neural networks architectures for detecting the collisions.

Finally, it is validated in a real scenario on KUKA Agilus, an industrial robot manipulator present in the Technology & Innovation Center (TIC) of KUKA Deutschland GmbH, Germany.

4.1 Data Representation and Camera

The method to be adopted required that starting from a mesh, the corresponding depth data is generated. It is an image that contains information on the

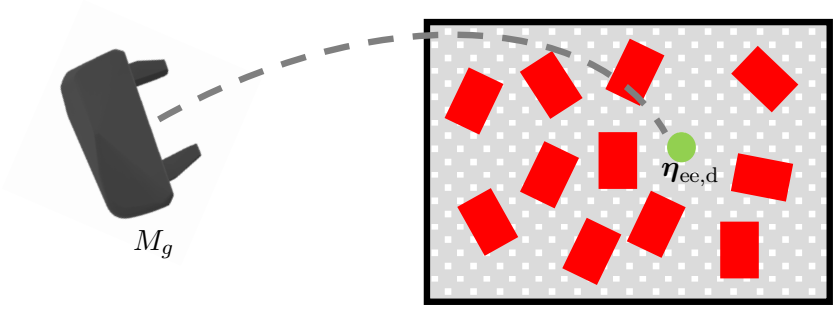

Figure 4.1: The mesh of the robot end-effector M_g is projected into the a desired configuration $\boldsymbol{\eta}_{\text{ee,d}}$ for detecting the collision.

distance of pixel in the image from a camera. Its generation from a mesh is possible using a virtual camera. One of simplest camera model that can be used is called *Pinhole*. Specially, it describes the relationship between 3D projection of the points into an image plane, where the light rays, passing through an aperture, project an inverted image on the opposite side of the camera [105].

The latter is represented by two kinds of parameters: *Extrinsic* and *Intrinsic*. The first ones represent camera rotation and translation in space, while the second ones represent the physical properties of the camera, such as the focal lengths (f_x, f_y), the optical center in pixel (c_x, c_y) and a skew coefficient.

Knowing the depth map it is also possible to obtain the corresponding point cloud, where each pixel represents one point in a 3D scene through the following relationship

$$\begin{cases} z = d/d_s \\ x = (u - c_x)z/f_x \\ y = (v - c_y)z/f_y \end{cases}, \tag{4.1}$$

where d is the depth value, d_s is the depth scale of the camera, and (u, v) are the image coordinates.

4.2 Problem Description

The time necessary for detecting a collisions between robot end-effector and the environment could be really long, according to the complexity of the operational scenario, e.g. cluttered environment during a pick and place task. For example, in several situations, mesh-based approaches may not perform well the collision checking, due to the reconstruction of mesh from an high number of information provided by the camera sensor.

In this work a fast tool for collision detection based on deep learning is proposed. The aim is to exploit the knowledge of the robot end-effector mesh, generating depth images and point clouds from it and combine them with the real ones provided by the camera. Then, this data are sent to different neural networks architectures for detecting the collisions. Finally, it is validated in a real scenario on robot manipulator in a bin picking scenario.

4.3 Data Generation

Two different datasets are built: $\mathcal{D}_{\text{depth}}$ for the depth images and $\mathcal{D}_{\text{pointcloud}}$ for the point clouds. It is possible to load the mesh of the gripper M_g and assign it a desired pose $\boldsymbol{\eta}_{\text{ee,d}}$. Further details on the generation are explained.

4.3.1 Depth Images

An example of real scene is reported in Fig. 4.3.

The procedure for generating $\mathcal{D}_{\text{depth}}$ is reported in Fig. 4.4 and it works as following: the real camera acquires the scene with bin and provides as output the depth image $\boldsymbol{I}_s$; in parallel, a virtual camera (created using parameters and pose of the real camera) generates the depth images $\boldsymbol{I}_g$ for the gripper, to which a desired pose $\boldsymbol{\eta}_{\text{ee,d}}$ is assigned. Once completed, the depth images $\boldsymbol{I}_s$ and $\boldsymbol{I}_g$ are combined, projecting the gripper into the scene through the σ block, generating a complete depth image $\boldsymbol{I}_c$ as following

$$\boldsymbol{I}_c = \Pi(\boldsymbol{I}_s, \boldsymbol{I}_g) \, . \tag{4.2}$$

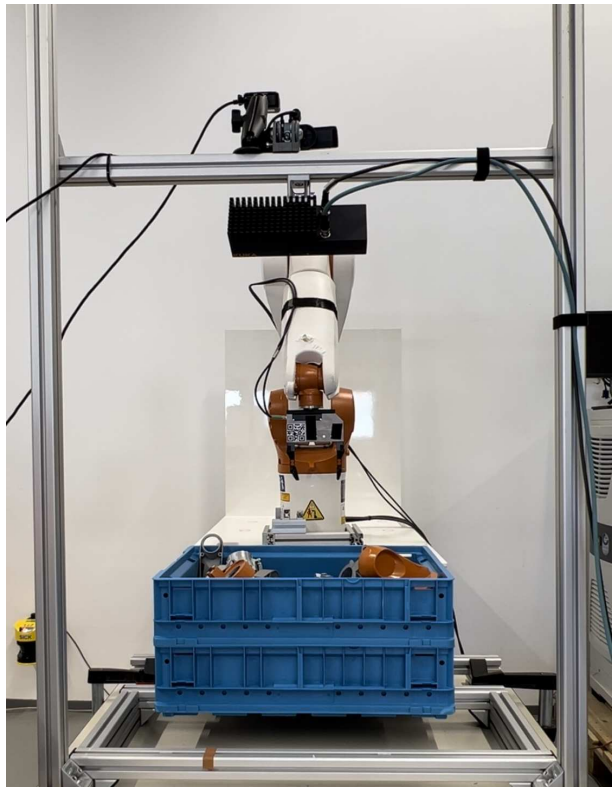

Figure 4.2: Scenario with robot manipulator and a full bin with objects; the camera is top-mounted in order to have a complete view of the bin.

This function Π considers the minimum depth value for each pair of pixels between the scene $\boldsymbol{I}_s$ and the gripper $\boldsymbol{I}_g$. Thus, the objects pixels more close to the camera have a minor depth value, which result predominant compared to the depth value for the pixels belonging to the far away objects. Finally, the block Δ_c generates cropped images $\boldsymbol{I}_s^*$, $\boldsymbol{I}_g^*$ and $\boldsymbol{I}_c^*$ of a new desired resolution (W, H), where in this case are 156×156. The importance of the crop filter lies in reducing the computational load by neglecting parts of the images, while still ensuring that the entire gripper is contained without loss of information (e.g. cut some parts of it). The algorithms only consider the local collision between the gripper and rest of the scene (in this first work the rest of the robot is not considered). As an example, the effect of the crop filter is reported in Fig. 4.5. Thus, the dataset $\mathcal{D}_{\text{depth}}$ is

Figure 4.3: Example of real bin from the camera point of view. The bin contains some different industrial objects.

$$\mathcal{D}_{\text{depth}} = \{[\boldsymbol{I}_c^* \; \boldsymbol{I}_s^* \; \boldsymbol{I}_g^*]_1, [\boldsymbol{I}_c^* \; \boldsymbol{I}_s^* \; \boldsymbol{I}_g^*]_2, \ldots, [\boldsymbol{I}_c^* \; \boldsymbol{I}_s^* \; \boldsymbol{I}_g^*]_{N_s}\} \,, \tag{4.3}$$

where N_s is the number of samples.

4.3.2 Point Clouds

Differently, the dataset $\mathcal{D}_{\text{pointcloud}}$ is generated in a different way: the real camera acquires the scene of the cluttered environment (bin with objects) as before, providing the depth image $\boldsymbol{I}_s$. Then, the point cloud generator block G_{PC} extracts two point clouds $N \times 3$ with N number of points, for the scene $\boldsymbol{PC}_s$ and gripper $\boldsymbol{PC}_g$, respectively using the Eq. (4.1); the block U_{PC} combines them, generating the complete point cloud $\boldsymbol{PC}_c$. Due to the varying nature of the point clouds (the first one comes from the camera, whereas the second one from the mesh model), a downsampling and padding are applied. The complete point cloud is $N \times 4$ because it contains an additional fourth column, which can assume 0 or 1 for the scene or gripper, respectively.

The architecture for generating this dataset is reported in Fig. 4.6 and examples of a complete depth image with the relative point cloud are represented in Fig. 4.7.

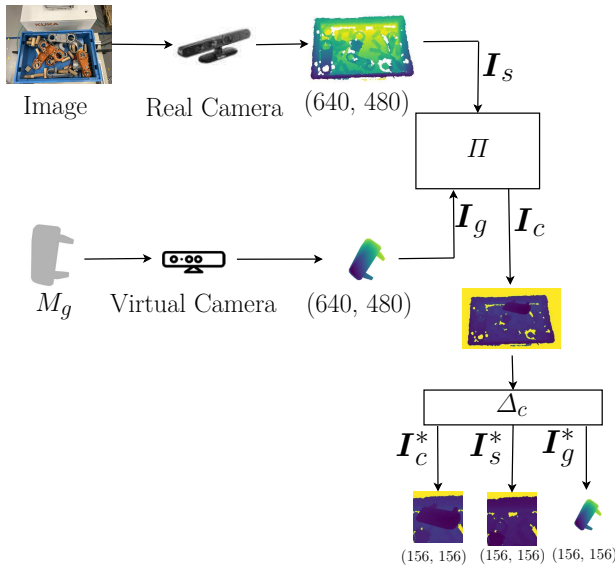

Figure 4.4: Generation dataset $\mathcal{D}_{\text{depth}}$: a camera acquires the scene and the gripper is projected into the scene. The output consists of three depth images: gripper, scene (without gripper) and complete scene (with gripper).

Thus, the dataset $\mathcal{D}_{\text{pointcloud}}$ is

$$\mathcal{D}_{\text{pointcloud}} = \{\boldsymbol{PC}_{c_1}, \boldsymbol{PC}_{c_2}, \ldots, \boldsymbol{PC}_{c_{N_s}}\} . \tag{4.4}$$

The labels are generated using Open3D [106], an open-source library that supports the rapid development of software for handling 3D data, providing a wide selection of data structures and algorithms in C++ and Python.

4.4 Collision Checkers

4.4.1 Geometric-based

As baseline on the computation is taken into consideration an algorithm based on mesh through the libraries Open3D [1] and Trimesh [2]. The latter is a mesh-based library enabled to handle triangular meshes for collision detection based on the very known general purpose FCL [43].

[1] http://www.open3d.org/
[2] http://trimsh.org/

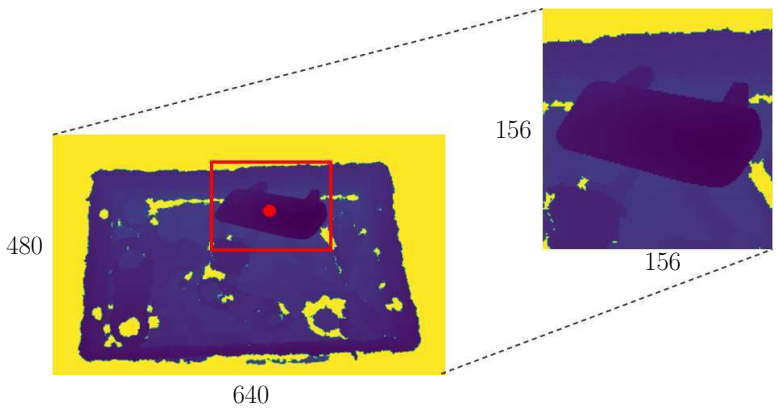

Figure 4.5: Application of the crop filter Δ_c on the complete depth image $\boldsymbol{I}_c$. It reduces the size of the images from (640, 480) to (156, 156).

More in detail, in this work the procedure consists of building an accurate mesh M_s from the point cloud of the scene $\boldsymbol{PC}_s$ through the library Open3D using the *ball pivoting* algorithm proposed in [107]. Then, the mesh of the gripper M_g is projected into the scene with a desired pose $\boldsymbol{\eta}_{\text{ee,d}}$ and finally the collision checker available from the library Trimesh is called on the objects to compute the collision γ_c and the minimum distance d_{min} between the nearest two points of meshes. The flow of the algorithm is reported in Alg. 4.1. Even if the collision checker is faster, there could be a problem to use it into the real world for the time related to the mesh reconstruction.

4.4.2 CNN-based

The first type of architecture being considered is the well-known fully convolutional neural network (which is made only of convolutional layers). The main differences w.r.t. CNNs include independence from the size of the images (in the last layer of the CNN, the fully connected layer depends from the input size), no information loss as opposed to the fully connected layer (generally, it causes loss of spatial information) and advantages in terms of computational cost. In this work considers three different architectures based on CNNs: two fully convolutional neural networks (FCNNs) with one and two depth images, respectively, and the well-known ResNet18 network customized

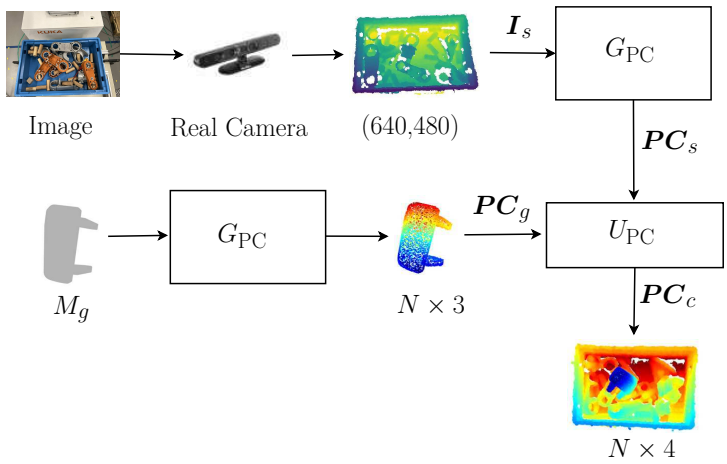

Figure 4.6: Generation dataset $\mathcal{D}_{\text{pointcloud}}$: a real camera acquires the scene providing the depth image of the scene. The PointCloud Generator G_{PC} block creates the pointclouds of the scene and the gripper. The concatenation block U_{PC} combines them generating the complete point cloud.

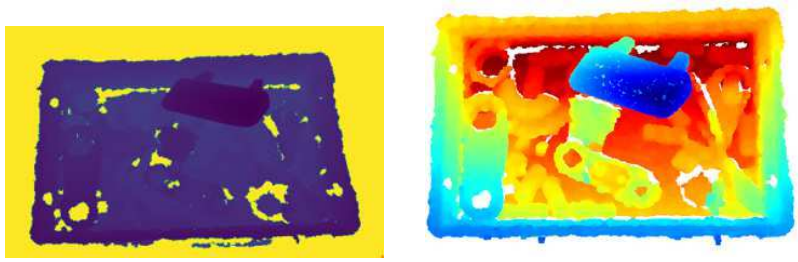

Figure 4.7: In the left part the complete (scene and gripper) depth image is reported, whereas in the right part the complete pointcloud is reported.

for this application. Both the FCNN models consist of four block, each made of three convolutional layers, followed by max pooling and dropout layers. The last layer is a convolutional layer with a kernel of size 1, for compressing the output as a binary classifier.

The architectures are reported in Fig. 4.8 and Fig. 4.9.

FCNN - 1 Depth Image

The input is $\boldsymbol{I}_c^*$, an image 156×156 with 1 channel that contains the gripper projected into the scene.

Data:
$\boldsymbol{I}_s$: // Depth scene
M_g: // Mesh gripper
$\boldsymbol{\eta}_{\text{ee,d}}$: // Desired pose
Result:
γ_c: // Collision 0/1
$d_{\min}$: // Minimum objects distance
// Assigning a desired pose
ApplyTransformation(M_g, $\boldsymbol{\eta}_{\text{ee,d}}$);
// Building point cloud from depth image
$\boldsymbol{PC}_s \leftarrow$ PointCloudGenerator($\boldsymbol{I}_s$);
// Reconstructing the mesh of the scene
$M_s \leftarrow$ CreateMesh($\boldsymbol{PC}_s$);
γ_c, points $\leftarrow$ CollisionChecker(M_s, M_g);
// Computation minimum distance between meshes
$d_{\min} \leftarrow$ ComputeDistance(points);

Algorithm 4.1: Mesh-based Collision Checker

FCNN - 2 Depth Images

Differently from the previous, the inputs are two depth images $\boldsymbol{I}_s^*$ and $\boldsymbol{I}_g^*$, that are scene and gripper with input size 156×156 and 2 channels.

ResNet18

It is a convolutional neural network made of 18 layers deep, capable to classify more than 1000 different objects. In the transfer learning context, there is a possibility to use a pretrained version of the network, which is trained on more than 1 million of images present in the ImageNet database [108]. The original version of the network considers images 224×224, but in this work an edited version is implemented. In the details, only the deep architecture is used (not pretrained), adapting it to work with the images of the dataset $\mathcal{D}_{\text{depth}}$. The architecture is shown in Fig. 4.10. A well-known problem with very deep neural networks is the vanishing gradient when it is back-propagated to earlier layers, because the repeated multiplication may make the gradient very small. On the basis of this, ResNet uses the concept of *residual blocks*, which consider shortcut skip connections in order to jump over some layers.

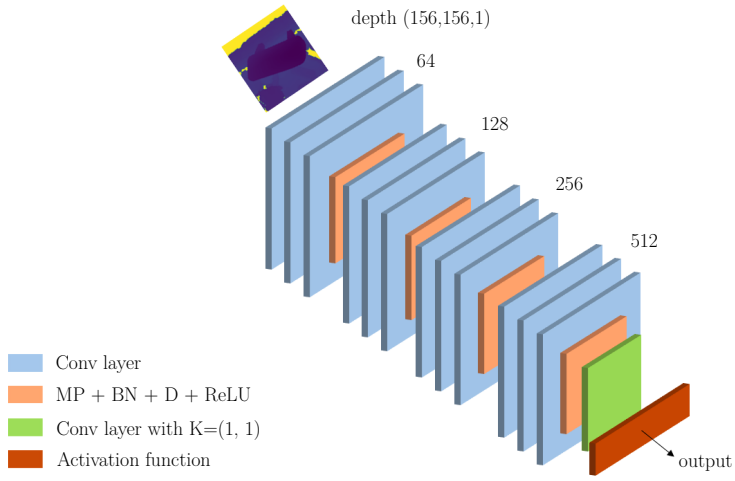

Figure 4.8: Architecture of Fully Convolutional Neural Network with one depth image $(156, 156, 1)$ as input.

4.4.3 PointNet-based

Standard PointNet

When it is necessary handle point clouds, due to the irregularity of structure, it is common to pass through a 3D voxel grids trasformation [50]. Unfortunately, this procedure is really hard and expensive from the computational point of view. The authors in [48] present a Deep Learning Framework, called PointNet that receives in input unordered points. As is well-known, a point cloud is represented by 3D points $\{P_i | i = 1, \ldots, N\}$, where each one P_i is a vector of coordinates $[x\ y\ z]$ and it is possible to add some extra feature channels (e.g. color and other details). In the original paper, the proposed architecture is capable to classify and segment 40 different classes of objects, through reorganizations of the points received as input. In this work, the segmentation from the network is not considered, because the point cloud $\boldsymbol{PC}_c$ is built considering the information about the belonging of the points. On these points a transformation through the T-Net block is applied (e.g. rotation about z-axis), which not affect them in terms of shape or changing in the features. The result is the input of a multilayer perceptron MLP, that transform the extracted

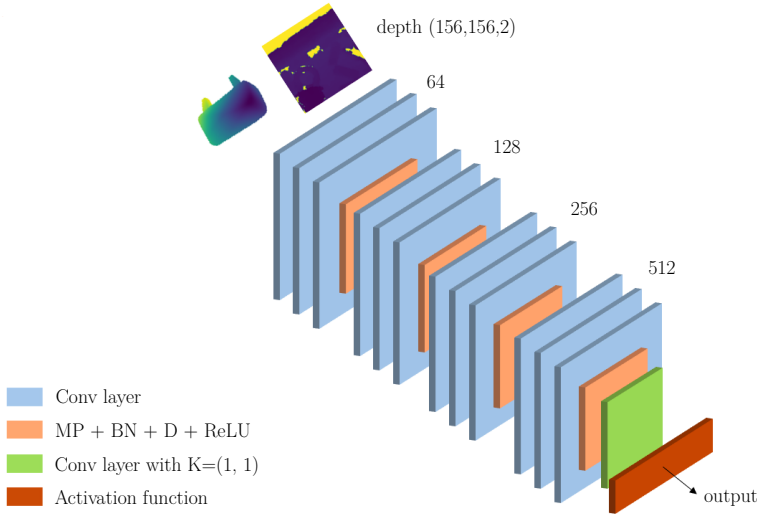

Figure 4.9: Architecture of Fully Convolutional Neural Networks for two depth images $(156, 156, 2)$ as input.

features in another features space with dimensionality 64. Then, there is a feature transform, which aligns each of the points in a canonical space. Another transformation in a different space with dimensionality 1024 is applied in the second MLP. The crucial part of this network is here, in which there is a max pool function. More in detail, each point has 1024 features and the network can learn the shape of the elements present in the data. It means that, even if the order of the input changes, the result is the same thanks to the max pool function. Finally, there is fully connected layer to map the features in one of the k categories to classify. In this particular case, the classes are only two: collision and no-collision, respectively. A representation is reported in Fig. 4.11

Fast-PointNet

In this work, also a short version of PointNet named Fast-PointNet is proposed. The camera used to acquire the data is fixed in the space, thus the T-Net block can be neglected. From the computational point of view, if the point cloud is composed of a large number of points, the multiplication with these matrices could require different times. Due to this consideration, this version is faster than the previous architecture.

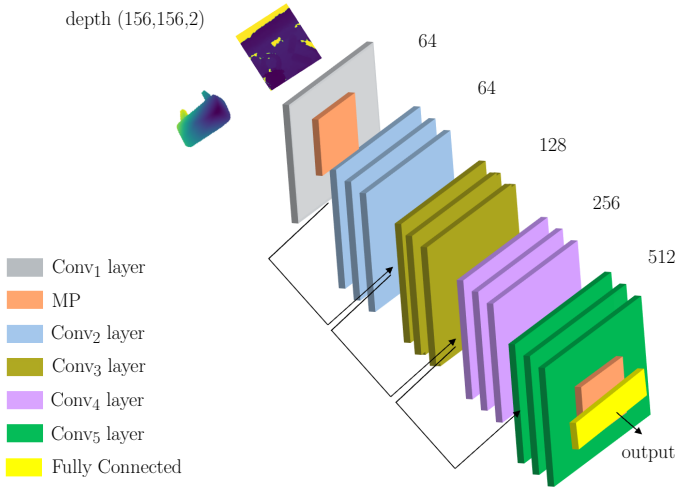

Figure 4.10: Architecture of ResNet18, which receives two depth images $(156, 156, 2)$ as input.

4.4.4 Hybrid-based

MixNet

Finally, a combined architecture of FCNNs and Fast-PointNet entitled MixNet is proposed. It takes into consideration the depth image $\boldsymbol{I}_s^*$ of the entire scene and the point cloud of the gripper $\boldsymbol{PC}_g$. These two architectures are combined, mixing the features extracted from the depth image of the scene and the point cloud of the gripper. The architecture is reported in Fig. 4.12.

The flow for the learning-based collision checkers is reported in Alg. 4.2.

4.5 Simulation & Results

The framework is developed in Python using a combination of PyTorch and ROS (Robot Operating System). The considered robot is the industrial KUKA Agilus located in the Technology & Innovation Center Laboratory of KUKA Deutschland GmbH in Augsburg, Germany (see Fig. 4.2). The Deep Learning

Data:
$\boldsymbol{I}_s$: // Depth scene
M_g: // Mesh gripper
$\boldsymbol{\eta}_{\text{ee,d}}$: // Desired pose
Type: // Architecture: FCNN1D, FCNN2D, ResNet18, Fast-PointNet, PointNet, MixNet
Result:
γ_c: // Collision 0/1
// Assigning desired pose to the gripper
ApplyTransformation(M_g, $\boldsymbol{\eta}_{\text{ee,d}}$);
// Building gripper depth image and point cloud from mesh
$\boldsymbol{I}_g \leftarrow$ DepthGenerator(M_g);
$\boldsymbol{PC}_g \leftarrow$ PointCloudGenerator(M_g);
// Building point cloud of scene
$\boldsymbol{PC}_s \leftarrow$ PointCloudGenerator($\boldsymbol{I}_s$);
// Combining the depth images
$\boldsymbol{I}_c \leftarrow$ Minimum($\boldsymbol{I}_s$, $\boldsymbol{I}_g$);
// Applying a crop filter on complete depth image
$\boldsymbol{I}_c^*, \boldsymbol{I}_s^*, \boldsymbol{I}_g^* \leftarrow$ Filter($\boldsymbol{I}_c$);
// Combining the point clouds
$\boldsymbol{PC}_c \leftarrow$ Union($\boldsymbol{PC}_s$,$\boldsymbol{PC}_g$);
// Selecting learning architectures and data
// Query for the collision check
if *Type == FCNN1D* **then**
  $\gamma \leftarrow$ FCNN1D($\boldsymbol{I}_c^*$)
end
else if *Type == FCNN2D* **then**
  $\gamma \leftarrow$ FCNN2D($\boldsymbol{I}_s^*$, $\boldsymbol{I}_g^*$)
end
else if *Type == ResNet18* **then**
  $\gamma \leftarrow$ ResNet18($\boldsymbol{I}_c^*$)
end
else if *Type == Fast-PointNet* **then**
  $\gamma \leftarrow$ Fast-PointNet($\boldsymbol{PC}_c$)
end
else if *Type == PointNet* **then**
  $\gamma \leftarrow$ PointNet($\boldsymbol{PC}_c$)
end
else
  $\gamma \leftarrow$ MixNet($\boldsymbol{I}_s$, $\boldsymbol{PC}_g$)
end

Algorithm 4.2: Learning-based Collision Checkergi

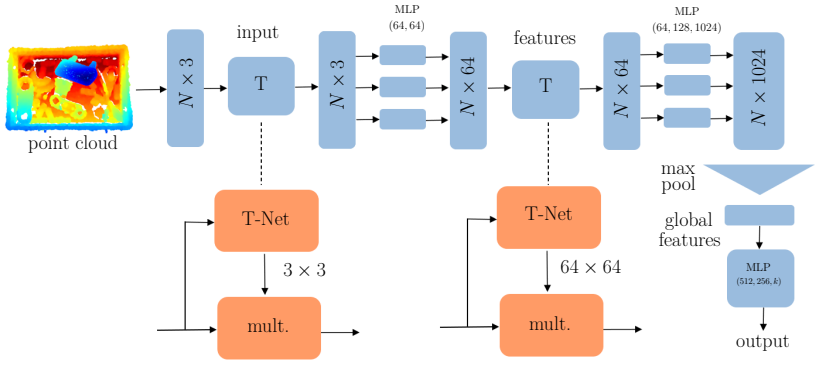

Figure 4.11: Architecture of PointNet receives a complete point cloud, which contains scene and gripper as input.

Model	Train[%]	Val[%]	Test[%]	Prec.	Rec.	F1-Sc.
Fast-PointNet	98.0502	93.1321	93.1989	0.9487	0.9133	0.9307
PointNet	98.1214	92.0320	91.3652	0.9212	0.9046	0.9129
$FCNN_{1D}$	92.1215	91.3485	90.9818	0.9276	0.8889	0.9078
$FCNN_{2D}$	90.9500	92.4487	90.9484	0.9915	0.8259	0.9012
ResNet18	91.0578	90.2483	90.6984	0.9485	0.8606	0.9024
MixNet	98.5964	93.1988	92.6321	0.9607	0.8890	0.9234

Table 4.1: Analysis of learning-based methods for collision detection on training, validation and test set.

models are trained on a desktop DELL Alienware equipped with CPU Intel I9.9980XE 3.0 Ghz x 36 and GPU NVIDIA RTX 2080 Ti 12 Gb.

A scheme is reported in Fig. 4.13

They are evaluated in the real world considering scenes with objects never seen before, showing the capability to generalize in different conditions.

Regarding the mesh-based checker it spends an average time approximately 1.90 [s] for reconstructing the mesh from the related point cloud, even if the collision checker is faster. According with the Tab. 4.1, the methods based on the point clouds outperform the others in terms of accuracies. Indeed, from Table 4.2 it is possible to notice a sensible advantage in terms of times to detect collision in favour of Fast-PointNet with respect to the classic PointNet version, which leads it comparable with the others based on depth images.

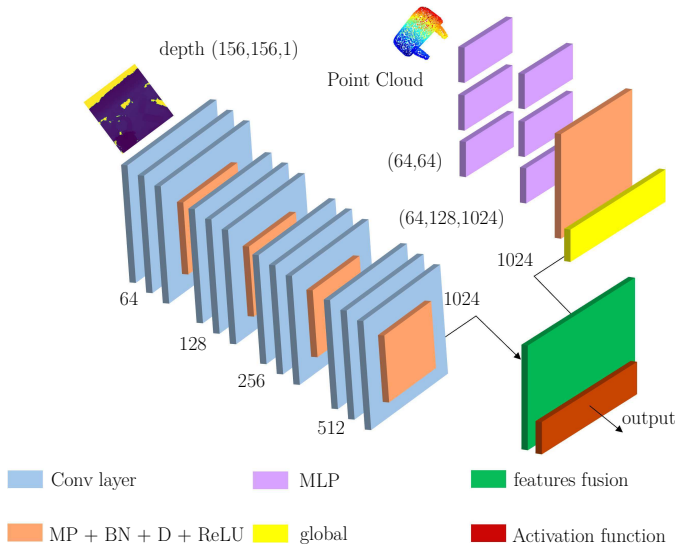

Figure 4.12: Architecture of MixNet, which receives a depth image $(156, 156, 1)$ and a point cloud as input.

The experiments consist of 10 different cases as reported in Fig. 4.14. Case 1 is a simple scenario made of two sub-cases: no-collision and collision. In the first one an empty bin is considered and for the algorithms is easy to detect the no-collision for the desired assigned configuration; then, a stack of objects is added in the same desired position to generate a collision with the gripper; case 2, where the bin is filled with typical industrial objects, providing a detection more complex than previous because of the large amount of information into data; cases 3-4 are based on different bin and objects in the scene, i.e. plastic fishes. The first four cases are performed for easy of using the same pose. Cases 5-6 put emphasis on the no-collision, moving the gripper very close to the objects, whereas cases 7-10 consider different poses where there are collision with small parts of the gripper (e.g. fingers). As explained above, this method is based only on the collision detection with the gripper and no with the rest of the robot. The video is available here[3].

[3] https://webuser.unicas.it/lai/robotica/video/collision_detection.mp4

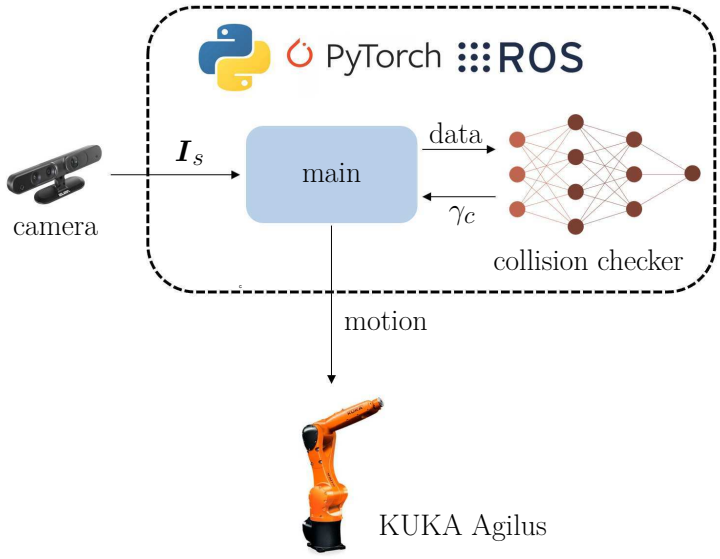

Figure 4.13: Software Architecture: the main block receives the depth image by the camera and, building the data (depth image or point cloud) and sends them to a neural network allowed to detect collisions with the environment; the collision checker returns a feedback that compute the control signal for the KUKA Agilus, moving it in the desired configuration.

Model	**CPU** [s]	**GPU** [s]
Fast-PointNet	0.0240	0.0009
PointNet	0.0543	0.0025
FCNN_{1D}	0.0699	0.0006
FCNN_{2D}	0.0268	0.0008
ResNet18	0.0231	0.0014
MixNet	0.0088	0.0010

Table 4.2: Times for feeding a sample into the models using CPU-GPU.

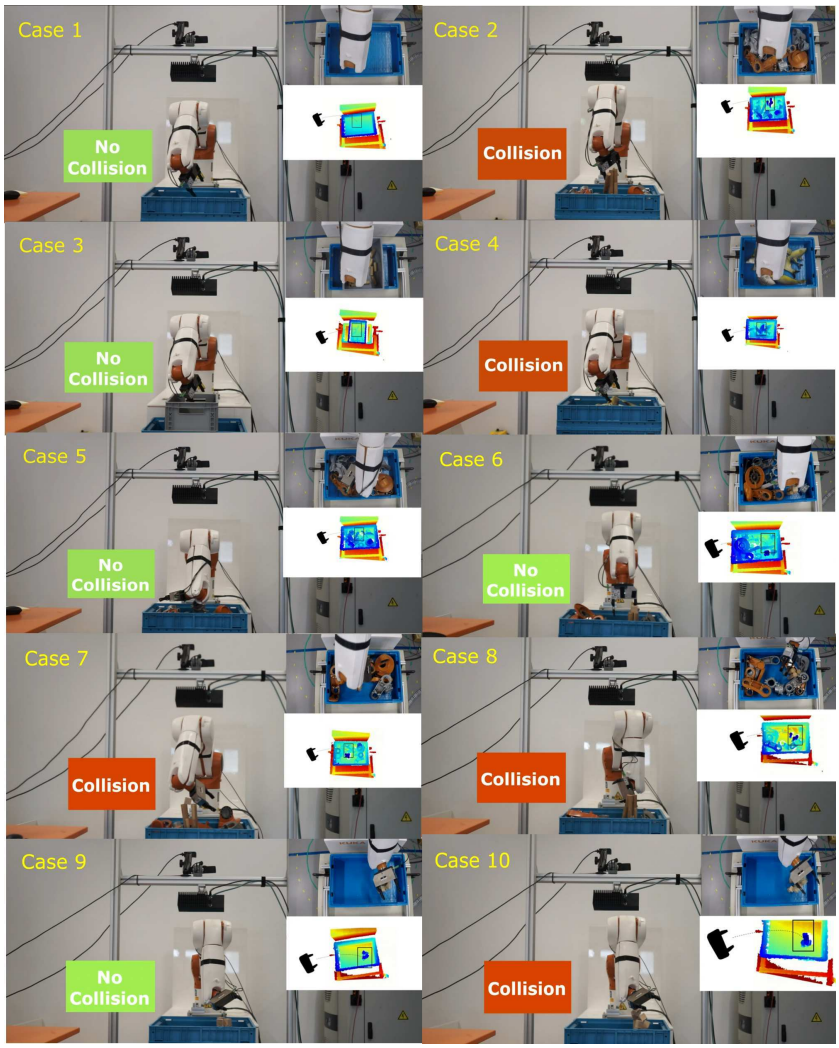

Figure 4.14: Experimental validation of KUKA Agilus.

Chapter 5

Deep Learning for Task Priority Inverse Kinematics

This chapter is dedicated to a possible solution of robot motion generation without violation of constraints in case of redundant robots.

Typical metrics to maximize in Eq. (2.23) are manipulability in Eq. (2.24) and the distance from the joint limits of the arm in Eq. (2.25). 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 [109]. It is worth noticing that optimizing these metrics employing Eq. (2.23) 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.

The final goal of adding a secondary task is to increase the 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 such a case, the best choice might be to not optimize any function at all, employing Eq. (3.1) instead of Eq. (2.27).

For these reasons, the problem that is addressed here 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, among three different possibilities: i) do not optimize anything, ii) maximize the manipulability or iii) maximize the distance from the joint limits. The aim is to show that using the proposed method, an enlargement of the workspace of the robot can be appreciated. The proposed solution is validated on a three-link planar robot in simulation.

5.1 Multi-class Problem

Considering p possible control algorithms to employ $\mathcal{F}_1, \mathcal{F}_2 \ldots, \mathcal{F}_p \in \mathcal{F}$, 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, label $\boldsymbol{\zeta}$ is defined as

$$\boldsymbol{\zeta} = \begin{bmatrix} \zeta_1 & \zeta_2 & \ldots & \zeta_p \end{bmatrix}^T \quad i = 1, 2, \ldots, p \,, \tag{5.1}$$

where $\zeta_1, \zeta_2 \ldots \zeta_p \in \{0, 1\}$ are values that represent the violation of at least one of the constraints in $\mathcal{C}$ employing the algorithms in $\mathcal{F}$: 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, three possible algorithms are considered: $\mathcal{F}_1$ is represented by Eq. (3.1), without optimizing any function; $\mathcal{F}_2$ is Eq. (2.27) optimizing w_{m} as in Eq. (2.24); $\mathcal{F}_3$ is Eq. (2.27) optimizing w_{jl} as in Eq. (2.25). Regarding the constraints, there is a minimum value for the manipulability to consider, as well as minimum and maximum values for the joint positions. Taking into account the label $\boldsymbol{\zeta}$, if $\zeta_1 = 1$, it means that employing $\mathcal{F}_1$ none of the constraints would be violated; if $\zeta_1 = 0$, 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 ζ_2, associated with the employment of $\mathcal{F}_2$, and ζ_3, associated with $\mathcal{F}_3$. The labels associated with the classes to identify are listed in Table 5.1.

Class: c	No Opt.: ζ_1	Manip: ζ_2	J. Limits: ζ_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	

Table 5.1: Mapping for labelling: 0 is related to the violation of constraints, 1 if there is no violation.

5.1.1 Learning Model

The designed neural network for classifying what function to optimize in the null space is a fully connected network, as shown in Fig. 5.1. The input is represented by the vector

$$\boldsymbol{u} = \begin{bmatrix} \boldsymbol{q}_0 \\ \boldsymbol{\eta}_{\mathrm{ee,d}} \end{bmatrix}, \tag{5.2}$$

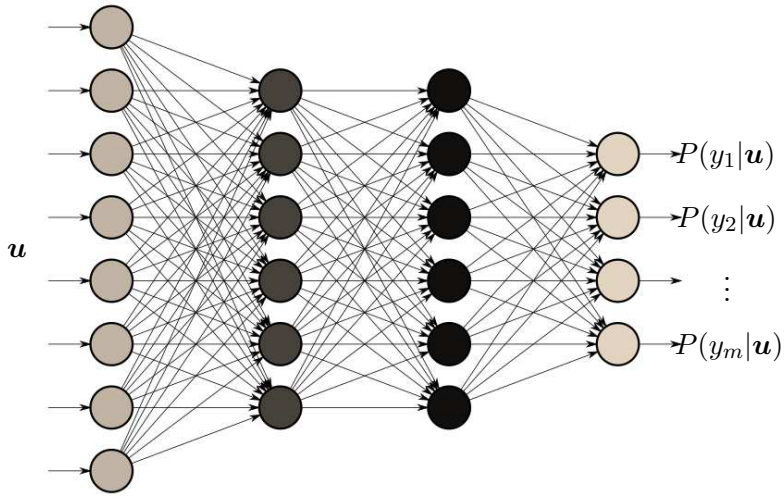

Figure 5.1: Architecture of fully connected neural network: $\boldsymbol{u}$ is the input and $P(y|\boldsymbol{u})$ are the probabilities of the output.

where $\boldsymbol{q}_0 \in \mathbb{R}^n$ is the vector of initial joint configuration and $\boldsymbol{\eta}_{\text{ee,d}} \in \mathbb{R}^3$ is the desired position of its 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. This latter is an important choice in the design phase of the models to use. 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 a neural network consists of the minimization of an error function through an optimization algorithm called 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

$$W^j \sim \mathcal{N}(\mu, \sigma^2 = \frac{1}{n^{j-1}}) \,, \tag{5.3}$$

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 $P(y = c_i|\boldsymbol{u})$ of the neural network that a particular sample belongs to a certain class. From a mathematical perspective, it is defined as follows

$$f_{\text{softmax}}(y_i) = \frac{e^{y_i}}{\sum_{k=1} e^{y_k}} \ . \tag{5.4}$$

In combination with the f_{softmax} activation function, the Negative-Log Likelihood (NLL) is used. 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. Finally, the loss function is expressed as

$$\mathcal{L}_{\text{NLL}} = -\log(y_k) \ . \tag{5.5}$$

5.2 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 $\boldsymbol{u}$ of the DNN in Eq. (5.2) 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 are represented in Fig. 5.2. 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.

5.2.1 Dataset Generation

The dataset consists of a set of instances for each class obtained sampling on the entire workspace of the robot, starting from random initial configurations $\boldsymbol{q}_0$ selected respecting the imposed joint limits and desired end-effector positions $\boldsymbol{\eta}_{\text{ee,d}}$ in a range $[-3.0, 3.0]$ m of the cartesian space. Regarding the less frequent

classes, e.g. c_5, a denser local sampling around the region of interest is applied in order to increase the probability to found other instances of the same class.

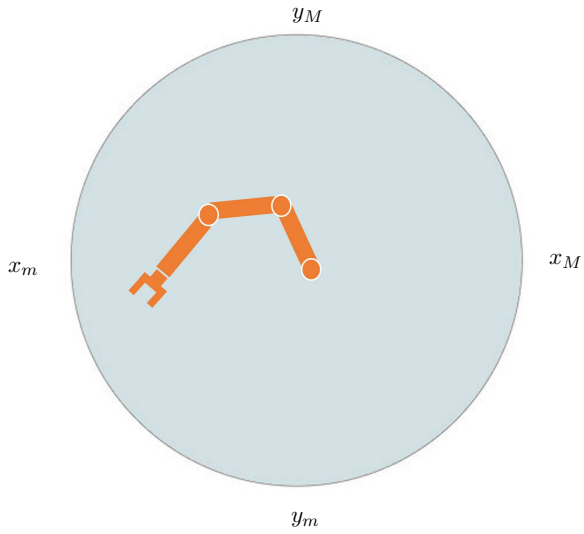

Figure 5.2: 3-link planar Robot and its workspace without joint limits.

Furthermore, the dataset is split into three parts: *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.

5.2.2 Results

Looking at the results, it is possible to notice that the model is capable to detect and correctly classify which tasks to optimize in the Null-space of the Jacobian, reaching the 94% on the test set. Figure 5.3 shows the entire robot workspace divided into regions with different colors, 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 (see Table 5.1) 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. Explaining 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 $x \approx [-0.05, 0.2]^T$ and $y \approx [0.05, 0.3]^T$ or $x \approx [-0.9, -0.3]^T$

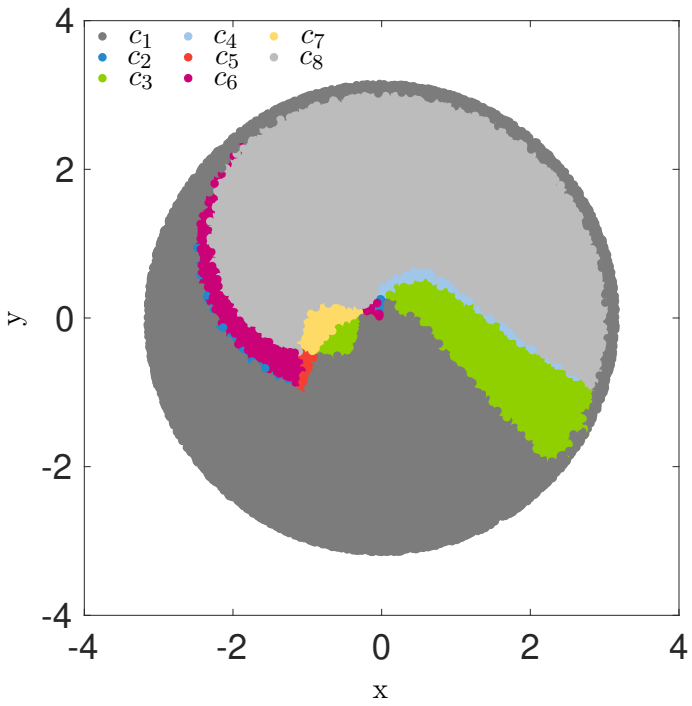

Figure 5.3: Sampled workspace of the 3-link robot arm for the initial configuration $[\pi/4 \quad \pi/3 \quad \pi/3]^T$ with a robot base position in $(0, 0)$.

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. 5.4.

As an illustrative example of c_4, Fig. 5.5 shows the evolution of 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 $\mathcal{F}_1$, whereas $\mathcal{F}_2$ and $\mathcal{F}_3$ do not cause any constraint violation.

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

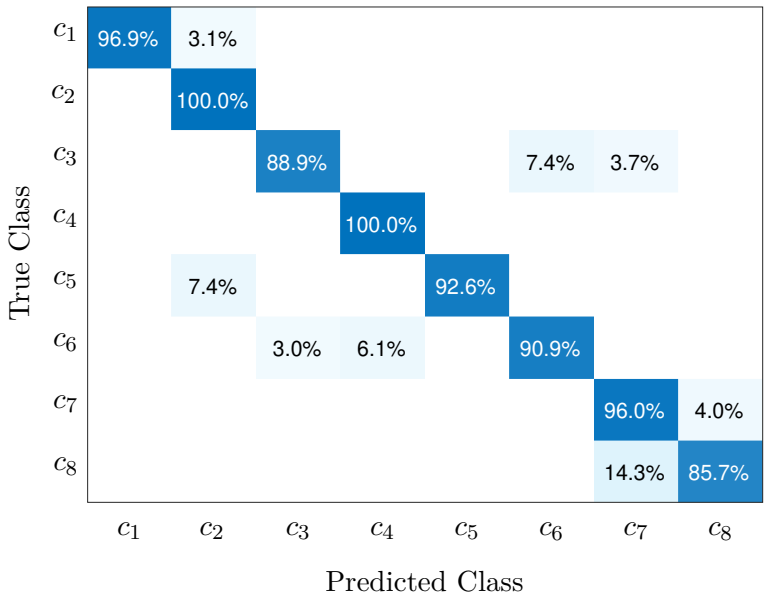

Figure 5.4: Confusion Matrix normalized for row. The diagonal represents the classes correctly classified, whereas the others value represent the classes wrongly classified.

Table 5.2, which reports the areas of the workspaces normalized with respect to the union. Analyzing the results, it is possible to notice that 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%.

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	■ + ■ + ■	$c_1, c_2 \dots c_8$	1.00

Table 5.2: Area of the workspaces obtained with the three algorithms and their union.

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

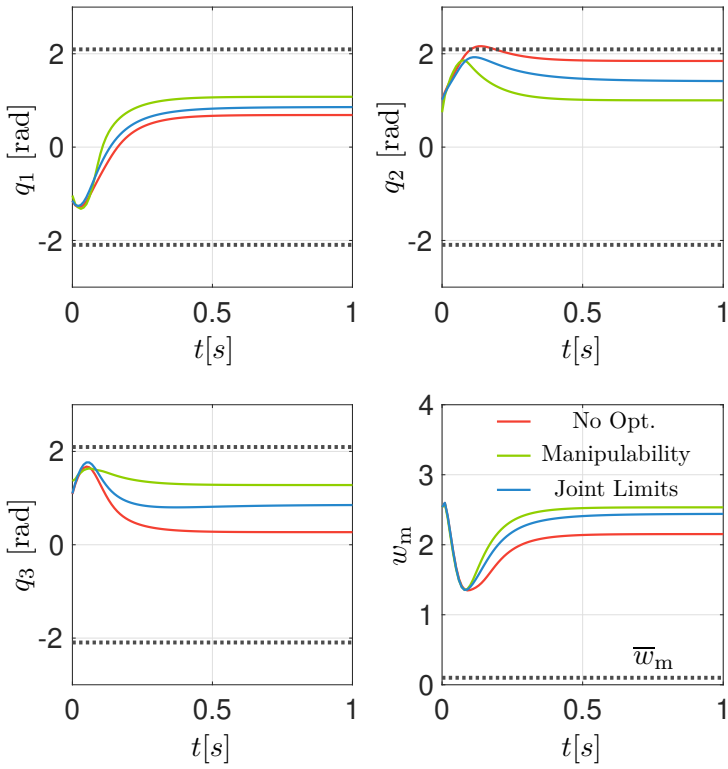

Figure 5.5: Top-left: evolution of the first joint position using and their joint limits (dotted line). Top-right: evolution of the second joint position and their joint limits (dotted line). Bottom-right evolution of the third joint position and their joint limits (dotted line). Bottom-right: evolution of the manipulability functionals and their thresholds (dotted line). The color of the plots are related to the algorithms $\mathcal{F}_1, \mathcal{F}_2$ and $\mathcal{F}_3$.

$[\pi/4 \quad \pi/3 \quad -\pi/3]^T$ 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. 5.6.

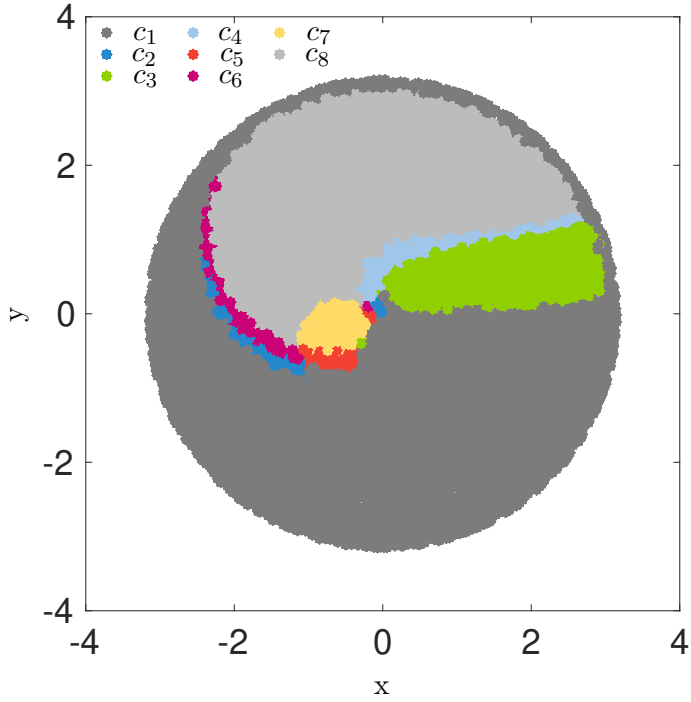

Figure 5.6: Sampled workspace of the 3-link robot arm for the initial configuration $[\pi/4 \quad \pi/3 \quad -\pi/3]^T$ with a robot base position in $(0, 0)$.

Chapter 6

Conclusions and Future Works

This thesis work is focused on the design of architectures based on Machine Learning and model-based control for robot manipulators to solve different tasks. In particular, the following topic are addressed:

- Task Motion Planning (TAMP): an architecture was proposed to solve the problem of retrieving an object from clutter based on application of reinforcement learning on dynamic tree structure;
- Learning-based Robot Collision Detection: tool for detecting collisions between robot end-effector and environment, through neural networks capable to handle depth images and point clouds;
- Deep Learning application in Task-Priority Inverse Kinematics: application of neural network for deciding if and what task optimize in the null-space for redundant robot.

Regarding the TAMP architecture, the main advantage is the reduction of computational load during the training of the agent, due to the not-pre-allocation of the entire space necessary to the convergence. This architecture combines a low level model-based system made of motion planner, with an high level system made of reinforcement learning task planner for solving the problem of retrieving an object target from clutter. The objective is to find the optimal sequence of obstacles to relocate taking into account kinematic constraints and

minimizing the energy spent by the robot. Different techniques guided and not-guided by heuristics during the exploration phase based on the well-known Breadth or Depth strategies were proposed and compared. The method that consider heuristics combined with the Depth exploration outperforms the other ones. Recalling that this work is related to the use of Markov Decision Process (MDP) with the assumption that the scene is visible a-priori, a possible future work can refer to an extension towards the Partially Observable Markov Decision Process (POMDP) to consider the case of partially visible objects. Then, another possible extension is related to the comparison with a DQN in order to evaluate the performances w.r.t. the proposed Q-Tree.

Concerning the collision detection research topic, an architecture based on deep learning to detect collisions is proposed. Even if well-known collision checkers based on mesh can detect collisions with high precision and faster speed, in some cases they require a significant amount of time. This leads to be unaddressable in the several real world application, becoming the bottleneck for other systems, e.g. motion planner. This is the reason why, machine learning approaches could be very useful. The proposed approaches are based on depth images and point clouds. In particular, starting from the CAD model of the robot end-effector, a depth images and point cloud are generated. These latter are projected with the desired pose into the scene acquired by the camera to detect collisions. From the results, a proposed method named Fast-PointNet, which uses point clouds, outperforms the others on depth images, in terms of accuracy, reaching an accuracy of approximately 93% using only 60k samples. The required efficiency in this case is very high, but as it is known, the threshold used for the binary classification is adaptable to the safe of problem increasing the recall value for more conservative applications. A possible future work could be to consider the rest of the robot and to consider other different data representation, e.g. 3D voxels.

Finally, concerning the application of machine learning on task priority inverse kinematics, a supervised learning process to properly handle optimization tasks has been investigated in this thesis work. A simple case study consisting in a 3-link planar robot manipulator already exhibits interesting properties and improvements under the proposed approach. Since the simulation is performed

only from a kinematics perspective, the behavior and result on a real system would be the same.

Future research directions include: investigation of the role of the optimization gains in the regions formation; introduction of a metric instead of the binary classification 0/1 indicating (un)reachability and extension to full dimensional robotic structures; a comparison using RGB image and 3D voxel data representation for the robot workspace and test on a more complex robot.

Chapter 7

Appendix

One of the contributions of this work is that our research can be reproduced by a third party. The community is becoming more and more aware of the importance of making obtained results reproducible and, to the aim, tools like Code Ocean are becoming popular among researchers. Concerning the results presented in this thesis, the reproduction of experimental tests can be done by adopting the MATLAB software together with the CVX toolbox [110] where the code has been created by adopting Matlab R2018b on Ubuntu 16.10, even though it has been tested also on Windows and OS X operative systems and Matlab R2016b and later version. A Matlab basic installation is enough and no particular package other than CVX is required.
In the details, CVX toolbox is a modeling system for constructing and solving convex optimization problems by also considering LMI constraints [88] and was used for CLS-1, CLS-2 and CLS-3 algorithms. The code is shared with the community as supplementary material and it is available on

- *Code Ocean* at [111]. However, this tool lacks of advanced functionalities; for instance, it only allows headless execution without plots and other graphics which make less intuitive the adoption of the developed tool. On the other side, no software installation is required on the local computer;
- *IEEE DataPort* at [112] as open-access. This version presents the overall code with a Graphical User Interface (GUI) and interactive plots. In this case, it is required to locally download the code and execute it by using its own Matlab installation.

The Graphical User Interface was designed to ease the use of software and is shown in Fig. 7.1.

Instructions to use both version of the code can be found at Sect .7.2.

In the details, the main steps to be followed are

- `load ID` to load one of the 5 predefined trajectories for the purpose of identification;
- `load VAL` to load one of the 5 predefined trajectories for the purpose of validation;
- `Identification` to compute the dynamic parameters according to algorithms explained in Section 2.1.6;
- `Validation` to compute the identification error and to show plots comparing over time the acquired torques and the reconstructed ones by exploiting the chosen validation trajectory and the method selected by the drop-down menu (for instance, CLS-3 in Fig. 7.1).

Moreover, the main functionalities of the software are implemented in

- `identification_ULS` for ULS method [113];
- `identification_CLS1` for CLS-1 method (inspired by) [84];
- `identification_CLS2` for CLS-2 method [86];
- `identification_CLS3` for CLS-3 method [88, 90];
- `validation_CAD` to validate on the selected trajectories the CAD method;
- `validation_CLS1` to validate on the selected trajectories the CLS-1 method;
- `validation_CLS2` to validate on the selected trajectories the CLS-2 method;
- `validation_CLS3` to validate on the selected trajectories the CLS-3 method;
- `validation_comparison` to compare among all the identification methods;

- `TestPropertiesB` to verify the inertia matrix properties (symmetry, positiveness and conditioning number) on 10.000 randomly generated configurations as required by ULS and CLS-1 method that do not guarantee these properties by construction.

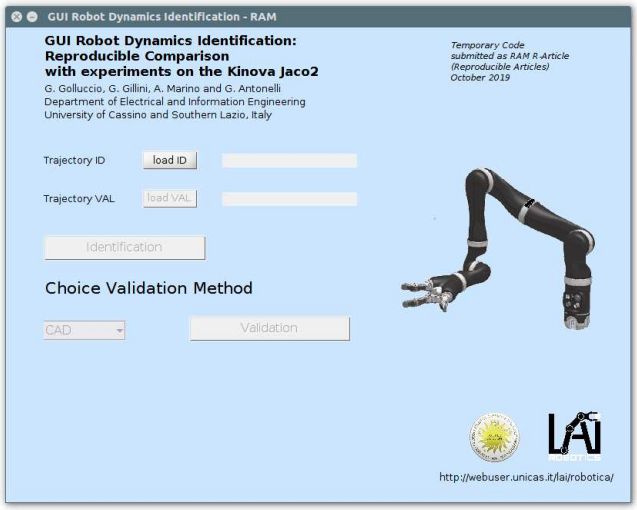

Figure 7.1: Interface of GUI Matlab necessary to reproduce experiments.

7.1 Notes on the positive definiteness of dynamic matrix $\boldsymbol{M}(\boldsymbol{q})$

In this section, it is shown how the constraint on the positive definiteness property of matrix $\boldsymbol{L}_i$ $(i = 1, 2, \ldots, n)$ affects the positive definiteness of the inertia matrix $\boldsymbol{M}(\boldsymbol{q})$ in Eq. (2.42). To this aim, the kinetic energy of a open-chain manipulator is

$$\mathcal{T} = \frac{1}{2}\dot{\boldsymbol{q}}^{\mathrm{T}}\boldsymbol{M}(\boldsymbol{q})\dot{\boldsymbol{q}} = \sum_{i=1}^{N}\mathcal{T}_i \tag{7.1}$$

being $\mathcal{T}_i$ the kinetic energy associated to link i (without loss of generality, only the link without motor is considered). Therefore, the positiveness of $\mathcal{T}_i$ $(\forall\, i)$ is a sufficient and necessary condition for the positive definiteness of $\boldsymbol{M}(\boldsymbol{q})$.

Concerning the kinetic energy $\mathcal{T}_i$ of the single rigid body, it holds

$$\mathcal{T}_i = \frac{1}{2} m_i \boldsymbol{v}_{c_i}^{i\mathrm{T}} \boldsymbol{v}_{c_i}^i + \frac{1}{2} \boldsymbol{\omega}_i^{i\mathrm{T}} \boldsymbol{C}_i \boldsymbol{\omega}_i^i \tag{7.2}$$

where $\boldsymbol{v}_{c_i}$ is the velocity of the center of mass of the link i and $\boldsymbol{C}_i$ is the inertia tensor about the center of mass. Equation (7.2) can be conveniently rewritten as

$$\mathcal{T}_i = \frac{1}{2} \boldsymbol{\nu}_{C_i}^{i\,\mathrm{T}} \begin{bmatrix} m_i \boldsymbol{I}_3 & \boldsymbol{O}_3 \\ \boldsymbol{O}_3 & \boldsymbol{C}_i \end{bmatrix} \boldsymbol{\nu}_{c_i}^i \tag{7.3}$$

where $\boldsymbol{O}_n \in \mathbb{R}^{n \times n}$ is the null square matrix, $\boldsymbol{\nu}_{c_i}^i = \begin{bmatrix} \dot{\boldsymbol{p}}_{c_i}^{i\,\mathrm{T}} & \boldsymbol{\omega}_i^{i\mathrm{T}} \end{bmatrix}^{\mathrm{T}}$ and where the core matrix of the quadratic form is positive definite if and only if $m_i > 0$ and $\boldsymbol{C}_i \succ 0$.

The same kinetic energy can be expressed with respect to the velocity of the origin of the link frame by considering that

$$\begin{cases} \boldsymbol{p}_{c_i}^i = \boldsymbol{p}_i^i + \boldsymbol{r}_{i,c_i}^i \\ \dot{\boldsymbol{p}}_{c_i}^i = \dot{\boldsymbol{p}}_i^i - \boldsymbol{S}(\boldsymbol{r}_{i,c_i}^i) \boldsymbol{\omega}_i^i \\ \boldsymbol{C}_i = \boldsymbol{L}_i - m_i \boldsymbol{S}(\boldsymbol{r}_{i,c_i}^i)^{\mathrm{T}} \boldsymbol{S}(\boldsymbol{r}_{i,c_i}^i) \end{cases} \tag{7.4}$$

In virtue of the equations above and by defining the vector $\boldsymbol{\nu}_i^i$ as $\boldsymbol{\nu}_i^i = \begin{bmatrix} \boldsymbol{v}_i^{i\mathrm{T}} & \boldsymbol{\omega}_i^{i\mathrm{T}} \end{bmatrix}^{\mathrm{T}}$, Eq. (7.3) is rewritten as

$$\begin{aligned} \mathcal{T}_i &= \frac{1}{2} \boldsymbol{\nu}_i^{i\mathrm{T}} \begin{bmatrix} m_i \boldsymbol{I}_3 & -m_i \boldsymbol{S}(\boldsymbol{r}_i^i) \\ -m_i \boldsymbol{S}(\boldsymbol{r}_i^i)^{\mathrm{T}} & \boldsymbol{C}_i + m_i \boldsymbol{S}(\boldsymbol{r}_i^i)^{\mathrm{T}} \boldsymbol{S}(\boldsymbol{r}_i^i) \end{bmatrix} \boldsymbol{\nu}_i^i \\ &= \frac{1}{2} \boldsymbol{\nu}_i^{i\mathrm{T}} \begin{bmatrix} m_i \boldsymbol{I}_3 & -\boldsymbol{S}(m_i \boldsymbol{r}_{i,C_i}^i) \\ -\boldsymbol{S}(m_i \boldsymbol{r}_{i,C_i}^i)^{\mathrm{T}} & \boldsymbol{L}_i \end{bmatrix} \boldsymbol{\nu}_i^i \\ &= \frac{1}{2} \boldsymbol{\nu}_i^{i\mathrm{T}} \boldsymbol{H}_i \boldsymbol{\nu}_i^i \end{aligned} \tag{7.5}$$

where the expression of $\boldsymbol{H}_i$ is implicitly defined. It can be easily noticed that the positive definiteness of $m_i \boldsymbol{I}_3$ and $\boldsymbol{L}_i$ (whose elements appears in $\boldsymbol{\pi}_{\mathrm{full}}$ and are the result of the identification process) does not guarantee the positive definiteness of the quadratic form because of the off-diagonal terms $m_i \boldsymbol{S}(\boldsymbol{r}_{i,c_i}^i) = \boldsymbol{S}(m_i \boldsymbol{r}_{i,c_i}^i) = \boldsymbol{S}(\boldsymbol{m}_{c_i})$ which is related to the first moment of

mass.
Therefore, the identification process might lead to a not always positive kinetic energy that, in turn, leads to $\boldsymbol{M}(\boldsymbol{q})$ not being positive definite. Finally, concerning the positive definiteness of matrix $\boldsymbol{H}_i$ in Eq. (7.5) which is of type

$$\boldsymbol{H}_i = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{B}^{\mathrm{T}} & \boldsymbol{C} \end{bmatrix} \tag{7.6}$$

the concept of Schur complement of matrix $\boldsymbol{H}_i$ with respect to $\boldsymbol{A}$ can be resorted. In details, it holds that matrix $\boldsymbol{H}_i \succ 0$ if and only if

$$\begin{cases} \boldsymbol{A} \succ 0 \\ \boldsymbol{C} - \boldsymbol{B}^{\mathrm{T}}\boldsymbol{A}^{-1}\boldsymbol{B} \succ 0 \end{cases} \tag{7.7}$$

which applied to our case leads to

$$\begin{cases} m_i > 0 \\ \boldsymbol{L}_i - m_i \boldsymbol{S}(\boldsymbol{r}^i_{i,c_i})^{\mathrm{T}} \boldsymbol{S}(\boldsymbol{r}^i_{i,c_i}) \succ 0 \end{cases} \tag{7.8}$$

that automatically holds if Steiner theorem is correctly applied.

As far as it concerns the identification process which requires to separately identify all or some of the components of $\boldsymbol{L}_i$ and of $m_i \boldsymbol{r}^i_{i,c_i}$, the constraints in Eq. (7.8) must be ensured on the components which effectively appear in the dynamics of the manipulator.

7.2 User Guide

Here, the user guide to replicate the experiments presented in the paper is reported. This section does not intend to explain the code architecture, but only how to reproduce the experiment results shown in the paper and how to use the `GUI` to run new experiments. A general idea of the software structure is illustrated in Fig. 7.2.

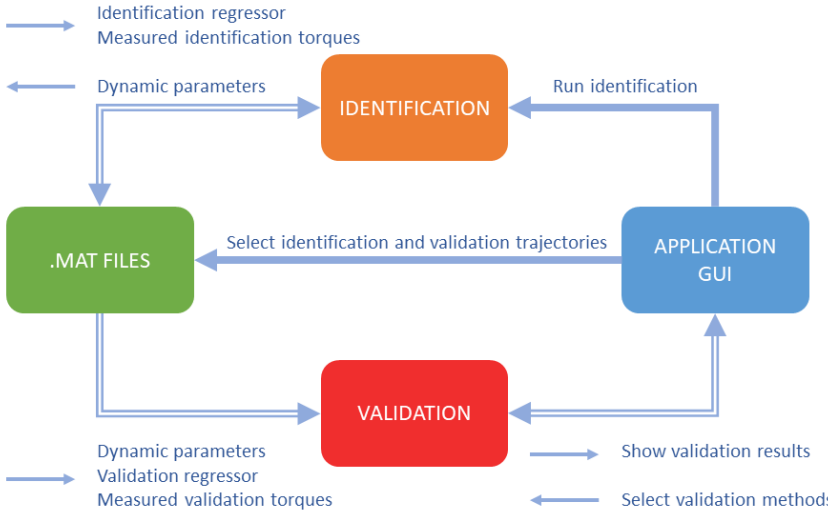

Figure 7.2: Software architecture.

Code Ocean version

As suggested in *RAM Information for Authors*[1], Code Ocean platform provides the possibility of reproducing experimental results by cloud computing. In order to run the provided code, a Code Ocean account is required by adopting the sign-in functionality. By accessing the link at [111], the code interface as in Fig. 7.3 appears and it is possible to run the code by using the *Reproducible Run* button. This button is made actually active after the duplication of the code in its own account which can be made by *Capsule→Duplicate* functionality.

By editing the file *main.m*, by modifying the variables:

- *Identification_Trajectory* (values: $1, 2, 3, 4, 5$) at line 63, it is possible to select one of the 5 trajectory to be used for the identification;
- *Validation_Trajectory* (values: $1, 2, 3, 4, 5$) at line 66, it is possible to select one of the 5 trajectory to be used for the validation;

[1] https://www.ieee-ras.org/publications/ram/information-for-authors-ram

- *algorithm* (values: 'ALL', 'CAD', 'ULS' 'CLS-1','CLS-2','CLS-3') at line 69, it is possible to select one of the 5 methods to be used for the identification of the KINOVA Jaco^2 robot. By setting this variable to 'ALL', all methods are run.

It is possible to observe that on the right frame are present two folders: *mat* and *Plot*. The former contains the dynamic parameters identified by the selected identification methods, while the latter contains the relative plot results saved as *.pdf* files. Furthermore, the console in the bottom frame shows the reconstruction errors. Unfortunately, among the Code Ocean limitations, files cannot be larger than 100 MB in size; then, the duration of the 5-th trajectory has been reduced in time in order to have the corresponding regressor meet the size limitation. For this reason, results relative to this trajectory are slightly different from the values reported in the paper and the ones produced by locally running the code as in Sect. 7.2.

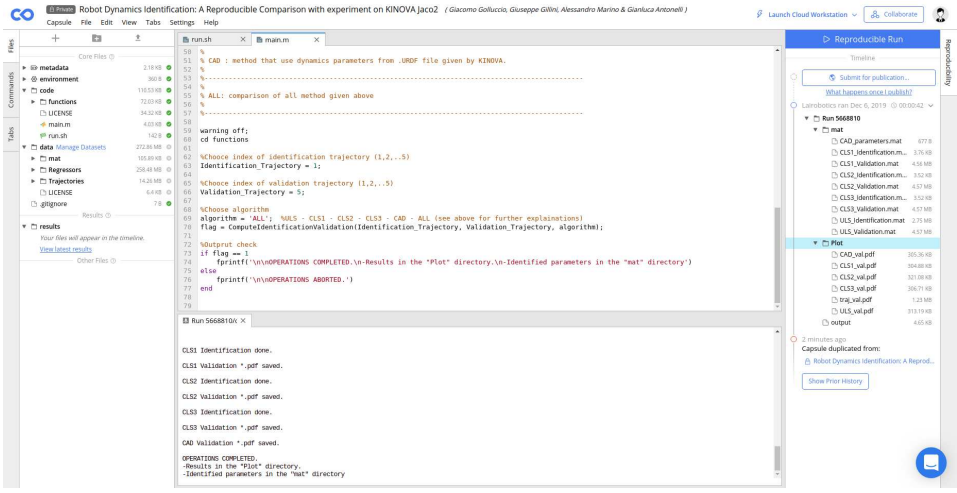

Figure 7.3: Code Ocean platform.

IEEE DataPort

Through the available `GUI` described in details in the following, it can be selected which trajectories to use to compute the identification and validation

regressors stored as *.mat* files. These files are needed to run the identification process which generates an estimate of the dynamic parameters of the KINOVA Jaco^2 robot which are stored in a *.mat* file as well. Finally, for each identification method the related validation results are computed and shown by using the previously stored identification *.mat* files.

7.2.1 Running the code

The software was created under MATLAB 2018b and Ubuntu 16.10 which represents the recommended configuration. However, the software has been tested on different platforms (Windows 10 and OS X High Sierra) with MATLAB 2016b and newer versions. The only issue experienced on some Windows systems is represented by misalignment of characters in the `GUI` which does not undermine the core functionalities of the `GUI` itself.

Required packages are:

- basic installation of Matlab 2016b or newer version;
- CVX MATLAB toolbox available at [110].

As mentioned before, for the software to run it is necessary to download the CVX tool for the specific operating system at [110]. The version including *Gurobi and/or MOSEK* as in Fig. 7.4 has to be selected.

OS	mexext	Download links		SDPT3	SeDuMi	Gurobi	MOSEK
Standard bundles, including Gurobi and/or MOSEK							
Linux	mexa64	cvx-a64.zip	cvx-a64.tar.gz	✓	✓	✓	✓
Mac	mexmaci64	cvx-maci64.zip	cvx-maci64.tar.gz	✓	✓	✓	✓
Windows	mexw64	cvx-w64.zip	cvx-w64.tar.gz	✓	✓	✓	✓

Figure 7.4: CVX download page.

The CVX folder contained in the downloaded archive needs to be unzipped in the main folder of the identification too provided as in Fig. 7.5.

Once the Matlab program has been launched, it is required to change the current working directory as shown in Fig. 7.6 (it is possible to see the script *main.m* in the current directory).

Figure 7.5: CVX folder extracted in the identification toolbox folder.

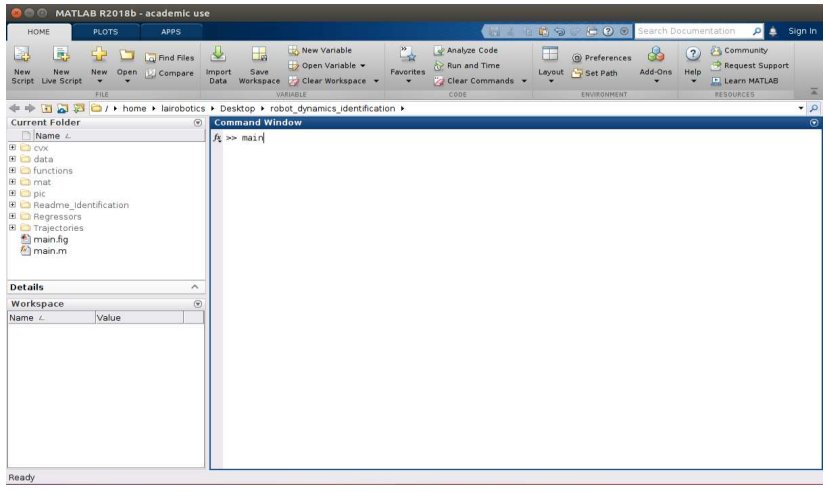

Figure 7.6: MATLAB Environment with initialized working directory.

First, it is necessary to run the script *add_cvx_path* in the Matlab environment in order to setup the CVX tool. In order to launch the application, it is needed to run the script `main.m` and the application `GUI` as seen in Fig. 7.7 will appear. At this point, it is possible to choose the identification and validation trajectories by using the load button as seen in Fig. 7.8.

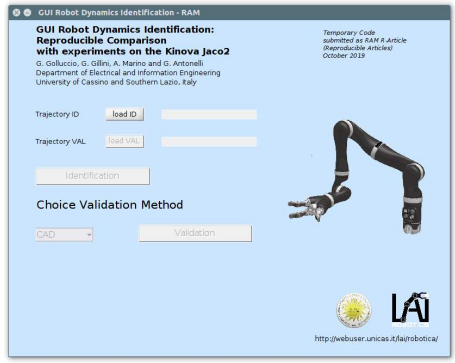

Figure 7.7: GUI how it appears after the file *main.m* has been launched.

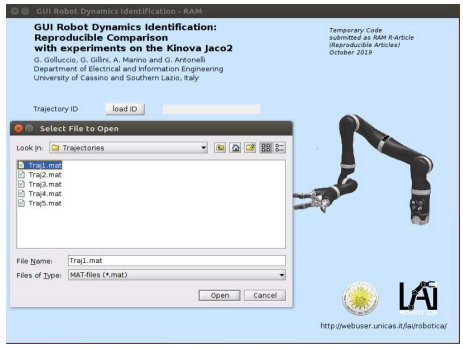

Figure 7.8: Trajectory selection window after either *load ID* or *load VAL* buttons are push.

Regressor matrices associated to trajectories are stored in the *Regressors* directory and their computation is skipped if the same trajectory is selected in successive identification and validation runs. The regressors associated to trajectories provided with the code have been pre-computed and made available in the directory mentioned above. In the opposite case, Fig. 7.9 shows the progress bar relative to the regressor computation.

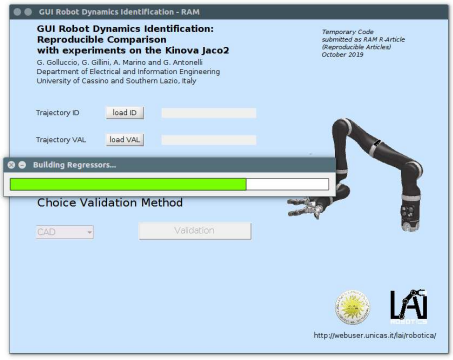

Figure 7.9: Building regressor phase with a progress bar showing the elapsed and remaining time.

After the loading of the identification and validation trajectories, it is possible to click on the *Identification* button as in Fig. 7.10.

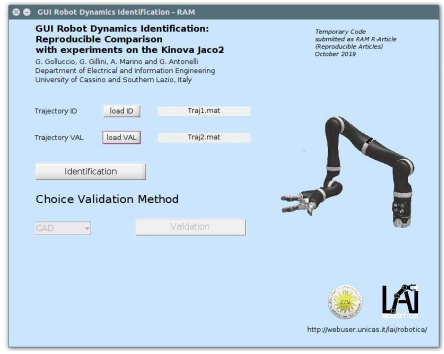

Figure 7.10: *Identification* button enabled after the identification and validation trajectories have been selected and the relative regressors computed.

A progress bar will appear in order to quantify the remaining time relative to the the selected identification method as shown in Fig. 7.11.

Once the identification is completed, it is possible to validate the dynamic parameters by choosing the validation method from the menu as seen in Fig. 7.12 (CLS3 in the image). If *All* is selected, all the identification methods are validated. After the computation is done, plots over time of significant variables are drawn as in Fig. 7.13, which also shows the average reconstruction *Error*, ς, and the *Relative Error*, ς_r, on the GUI.

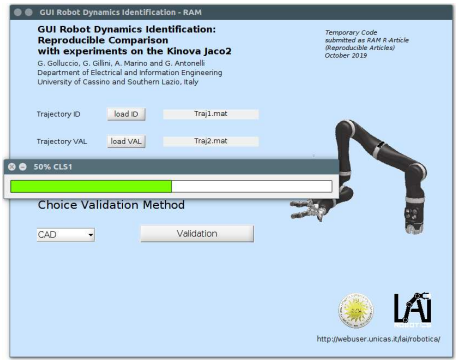

Figure 7.11: Progress bar showing the elapsed and remaining time once the *Identification* button is pushed.

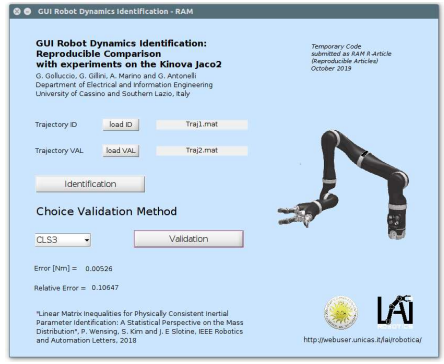

Figure 7.12: Validation button enabled with *CLS3* method selected.

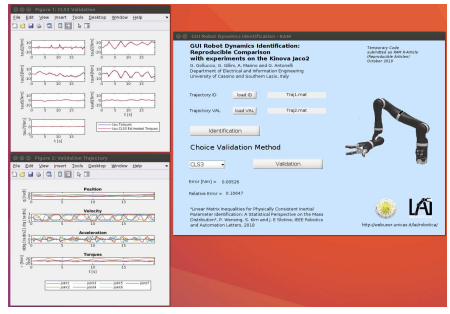

Figure 7.13: Numerical results displayed at the end of the validation process.

Bibliography

[1] G. Gillini, P. Di Lillo, and F. Arrichiello, "An assistive shared control architecture for a robotic arm using eeg-based bci with motor imagery," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4132–4137, IEEE, 2021.

[2] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez, "Integrated task and motion planning," *Annual review of control, robotics, and autonomous systems*, vol. 4, pp. 265–293, 2021.

[3] T. Lozano-Pérez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," *Communications of the ACM*, vol. 22, no. 10, pp. 560–570, 1979.

[4] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, "Motion planning with sequential convex optimization and convex collision checking," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1251–1270, 2014.

[5] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[6] F. Lagriffoul, D. Dimitrov, J. Bidot, A. Saffiotti, and L. Karlsson, "Efficiently combining task and motion planning using geometric constraints," *The International Journal of Robotics Research*, vol. 33, no. 14, pp. 1726–1747, 2014.

[7] F. Bertoncelli, M. Selvaggio, F. Ruggiero, and L. Sabattini, "Task-oriented contact optimization for pushing manipulation with mobile robots," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1639–1646, IEEE, 2022.

[8] N. Correll, K. E. Bekris, D. Berenson, O. Brock, A. Causo, K. Hauser, K. Okada, A. Rodriguez, J. M. Romano, and P. R. Wurman, "Analysis and observations from the first amazon picking challenge," *IEEE Transactions on Automation Science and Engineering*, vol. 15, no. 1, pp. 172–188, 2016.

[9] F. Ceola, E. Tosello, L. Tagliapietra, G. Nicola, and S. Ghidoni, "Robot task planning via deep reinforcement learning: a tabletop object sorting application," in *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pp. 486–492, IEEE, 2019.

[10] C. Nam, J. Lee, S. H. Cheong, B. Y. Cho, and C. Kim, "Fast and resilient manipulation planning for target retrieval in clutter," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3777–3783, IEEE, 2020.

[11] J. Lee, Y. Cho, C. Nam, J. Park, and C. Kim, "Efficient obstacle rearrangement for object manipulation tasks in cluttered environments," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 183–189, IEEE, 2019.

[12] M. Stilman and J. Kuffner, "Planning among movable obstacles with artificial constraints," *The International Journal of Robotics Research*, vol. 27, no. 11-12, pp. 1295–1307, 2008.

[13] K. Hang, J. A. Stork, F. T. Pokorny, and D. Kragic, "Combinatorial optimization for hierarchical contact-level grasping," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 381–388, IEEE, 2014.

[14] M. Stilman, J.-U. Schamburek, J. Kuffner, and T. Asfour, "Manipulation planning among movable obstacles," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pp. 3327–3332, IEEE,

2007.

[15] W. Yuan, K. Hang, D. Kragic, M. Y. Wang, and J. A. Stork, "End-to-end nonprehensile rearrangement with deep reinforcement learning and simulation-to-reality transfer," *Robotics and Autonomous Systems*, vol. 119, 2019.

[16] M. Eppe, P. D. Nguyen, and S. Wermter, "From semantics to execution: Integrating action planning with reinforcement learning for robotic causal problem-solving," *Frontiers in Robotics and AI*, vol. 6, p. 123, 2019.

[17] K. B. Shimoga, "Robot grasp synthesis algorithms: A survey," *The International Journal of Robotics Research*, vol. 15, no. 3, pp. 230–266, 1996.

[18] A. Bicchi and V. Kumar, "Robotic grasping and contact: A review," in *Proceedings 2000 ICRA. Millennium conference. IEEE international conference on robotics and automation. Symposia proceedings (Cat. No. 00CH37065)*, vol. 1, pp. 348–353, IEEE, 2000.

[19] J. Bohg, A. Morales, T. Asfour, and D. Kragic, "Data-driven grasp synthesis—a survey," *IEEE Transaction on robotics*, vol. 30, no. 2, pp. 289–309, 2013.

[20] C. Eppner, S. Höfer, R. Jonschkowski, R. Martín-Martín, A. Sieverling, V. Wall, and O. Brock, "Lessons from the amazon picking challenge: Four aspects of building robotic systems.," in *Robotics: Science and Systems*, 2016.

[21] P. R. Wurman and J. M. Romano, "Amazon picking challenge 2015," *AI Magazine*, vol. 37, no. 2, pp. 97–99, 2016.

[22] A. ten Pas, M. Gualtieri, K. Saenko, and R. Platt, "Grasp pose detection in point clouds," *The International Journal of Robotics Research*, vol. 36, no. 13-14, pp. 1455–1473, 2017.

[23] J. A. Haustein, J. King, S. S. Srinivasa, and T. Asfour, "Kinodynamic randomized rearrangement planning via dynamic transitions between statically stable states," in *2015 IEEE International Conference*

on Robotics and Automation (ICRA), pp. 3075–3082, IEEE, 2015.

[24] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, "Incremental task and motion planning: A constraint-based approach.," in *Robotics: Science and systems*, vol. 12, p. 00052, Ann Arbor, MI, USA, 2016.

[25] G. Havur, G. Ozbilgin, E. Erdem, and V. Patoglu, "Geometric rearrangement of multiple movable objects on cluttered surfaces: A hybrid reasoning approach," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 445–452, IEEE, 2014.

[26] H. Karami, A. Thomas, and F. Mastrogiovanni, "A task-motion planning framework using iteratively deepened and/or graph networks," *arXiv preprint arXiv:2104.01549*, 2021.

[27] B. Bonet and H. Geffner, "Planning as heuristic search," *Artificial Intelligence*, vol. 129, no. 1-2, pp. 5–33, 2001.

[28] A. H. Qureshi, A. Mousavian, C. Paxton, M. C. Yip, and D. Fox, "Nerp: Neural rearrangement planning for unknown objects," *arXiv preprint arXiv:2106.01352*, 2021.

[29] M. Q. Mohammed, K. L. Chung, and C. S. Chyi, "Review of deep reinforcement learning-based object grasping: Techniques, open challenges and recommendations," *IEEE Access*, 2020.

[30] P. Kormushev, S. Calinon, and D. G. Caldwell, "Reinforcement learning in robotics: Applications and real-world challenges," *Robotics*, vol. 2, no. 3, pp. 122–148, 2013.

[31] W. Bejjani, W. C. Agboh, M. R. Dogar, and M. Leonetti, "Occlusion-aware search for object retrieval in clutter," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4678–4685, IEEE, 2021.

[32] Y. Deng, X. Guo, Y. Wei, K. Lu, B. Fang, D. Guo, H. Liu, and F. Sun, "Deep reinforcement learning for robotic pushing and picking in cluttered

environment," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 619–626, IEEE.

[33] B. Wu, I. Akinola, and P. K. Allen, "Pixel-attentive policy gradient for multi-fingered grasping in cluttered scenes," in *2019 IEEE/RSJ Intelligent Conference on Intelligent Robots and Systems (IROS)*, pp. 1789–1796, IEEE, 2019.

[34] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[35] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[36] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*, pp. 1928–1937, PMLR, 2016.

[37] J. Pan and D. Manocha, "Fast probabilistic collision checking for sampling-based motion planning using locality-sensitive hashing," *The International Journal of Robotics Research*, vol. 35, no. 12, pp. 1477–1496, 2016.

[38] D. Di Vito, M. Bergeron, D. Meger, G. Dudek, and G. Antonelli, "Dynamic planning of redundant robots within a set-based task-priority inverse kinematics framework," in *2020 IEEE Conference on Control Technology and Applications (CCTA)*, pp. 549–554, IEEE, 2020.

[39] 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 International Conference on Advanced Robotics (ICAR)*, pp. 676–681, IEEE, 2021.

[40] 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 *ICINCO*, pp. 130–137, 2021.

[41] 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.

[42] B. Akgun and M. Stilman, "Sampling heuristics for optimal motion planning in high dimensions," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2640–2645, IEEE, 2011.

[43] J. Pan, S. Chitta, and D. Manocha, "Fcl: A general purpose library for collision and proximity queries," in *2012 IEEE International Conference on Robotics and Automation*, pp. 3859–3866, IEEE, 2012.

[44] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space," *IEEE Journal on Robotics and Automation*, vol. 4, no. 2, pp. 193–203, 1988.

[45] M. Kleinbort, O. Salzman, and D. Halperin, "Collision detection or nearest-neighbor search? on the computational bottleneck in sampling-based motion planning," *arXiv preprint arXiv:1607.04800*, 2016.

[46] P. Jiménez, F. Thomas, and C. Torras, "Collision detection algorithms for motion planning," *Robot Motion Planning and Control*, pp. 305–343, 1998.

[47] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[48] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 652–660, 2017.

[49] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," in *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 1912–1920, 2015.

[50] Y. Zhou and O. Tuzel, "Voxelnet: End-to-end learning for point cloud based 3d object detection," in *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 4490–4499, 2018.

[51] D. Maturana and S. Scherer, "Voxnet: A 3d convolutional neural network for real-time object recognition," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 922–928, IEEE, 2015.

[52] P. Schmidt, N. Vahrenkamp, M. Wächter, and T. Asfour, "Grasping of unknown objects using deep convolutional neural networks based on depth images," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6831–6838, IEEE, 2018.

[53] K. Huebner, K. Welke, M. Przybylski, N. Vahrenkamp, T. Asfour, D. Kragic, and R. Dillmann, "Grasping known objects with humanoid robots: A box-based approach," in *2009 International Conference on Advanced Robotics*, pp. 1–6, IEEE, 2009.

[54] I. Lenz, H. Lee, and A. Saxena, "Deep learning for detecting robotic grasps," *The International Journal of Robotics Research*, vol. 34, no. 4-5, pp. 705–724, 2015.

[55] L. Berscheid, T. Rühr, and T. Kröger, "Improving data efficiency of self-supervised learning for robotic grasping," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 2125–2131, IEEE, 2019.

[56] M. Sundermeyer, A. Mousavian, R. Triebel, and D. Fox, "Contact-graspnet: Efficient 6-dof grasp generation in cluttered scenes," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 13438–13444, IEEE, 2021.

[57] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, "Deepsdf: Learning continuous signed distance functions for shape representation," in *Proceedings of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition*, pp. 165–174, 2019.

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

[59] M. Danielczuk, A. Mousavian, C. Eppner, and D. Fox, "Object rearrangement using learned implicit collision functions," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6010–6017, IEEE, 2021.

[60] 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.

[61] 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)*, pp. 6248–6253, IEEE, 2015.

[62] 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.

[63] 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.

[64] 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.

[65] 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.

[66] 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*, pp. 1283–1290, IEEE, 2011.

[67] 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.

[68] 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.

[69] 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)*, pp. 3944–3949, IEEE, 2015.

[70] 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)*, pp. 6416–6421, IEEE, 2015.

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

[72] 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.

[73] 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.

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

[75] 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.

[76] 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 International Conference on Informatics in Control, Automation and Robotics - ICINCO*, pp. 130–137, INSTICC, SciTePress, 2021.

[77] 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.

[78] E. Magrini, F. Flacco, and A. De Luca, "Estimation of contact forces using a virtual force sensor," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2126–2133, IEEE, 2014.

[79] K. Yoshida and W. Khalil, "Verification of the positive definiteness of the inertial matrix of manipulators using base inertial parameters," *The International Journal of Robotics Research*, vol. 19, no. 5, pp. 498–510, 2000.

[80] G. Antonelli, F. Caccavale, and P. Chiacchio, "A systematic procedure for the identification of dynamic parameters of robot manipulators," *Robotica*, vol. 17, pp. 427–435, 1999.

[81] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: modelling, planning and control*. Springer Verlag, 2009.

[82] J. Hollerbach, W. Khalil, and M. Gautier, *Model identification*, pp. 113–137. Cham: Springer International Publishing, 2016.

[83] P. Nadeau, M. Giamou, and J. Kelly, "Fast object inertial parameter identification for collaborative robots," in *2022 International Conference on Robotics and Automation (ICRA)*, pp. 3560–3566, IEEE, 2022.

[84] C. Gaz, F. Flacco, and A. De Luca, "Extracting feasible robot parameters from dynamic coefficients using nonlinear optimization methods," in *2016 IEEE international conference on robotics and automation (ICRA)*, pp. 2075–2081, IEEE, 2016.

[85] A. Jubien, M. Gautier, and A. Janot, "Dynamic identification of the Kuka LWR robot using motor torques and joint torque sensors data," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 8391–8396, 2014.

[86] C. D. Sousa and R. Cortesão, "Physical feasibility of robot base inertial parameter identification: A linear matrix inequality approach," *The International Journal of Robotics Research*, vol. 33, no. 6, pp. 931–944, 2014.

[87] M. Gautier, "Numerical calculation of the base inertial parameters of robots," *Journal of robotic systems*, vol. 8, no. 4, pp. 485–506, 1991.

[88] P. M. Wensing, S. Kim, and J.-J. E. Slotine, "Linear matrix inequalities for physically consistent inertial parameter identification: A statistical perspective on the mass distribution," *IEEE Robotics and Automation Letters*, vol. 3, no. 1, pp. 60–67, 2018.

[89] S. Traversaro, S. Brossette, A. Escande, and F. Nori, "Identification of fully physical consistent inertial parameters using optimization on manifolds," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5446–5451, IEEE, 2016.

[90] C. D. Sousa and R. Cortesão, "Inertia tensor properties in robot dynamics identification: A linear matrix inequality approach," *IEEE/ASME Transactions on Mechatronics*, vol. 24, pp. 406–411, Feb 2019.

[91] J. Jovic, A. Escande, K. Ayusawa, E. Yoshida, A. Kheddar, and G. Venture, "Humanoid and human inertia parameter identification using hierarchical optimization," *IEEE Transactions on Robotics*, vol. 32, pp. 726–735, June 2016.

[92] M. Gautier and G. Venture, "Identification of standard dynamic parameters of robots with positive definite inertia matrix," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5815–5820, IEEE, 2013.

[93] B. Armstrong, "On finding exciting trajectories for identification experiments involving systems with nonlinear dynamics," *The International journal of robotics research*, vol. 8, no. 6, pp. 28–48, 1989.

[94] C. Presse and M. Gautier, "New criteria of exciting trajectories for robot identification," in *Proceedings 1993 IEEE International Conference on Robotics and Automation*, (Atlanta, GA), pp. 907–912, 1993.

[95] KINOVA, "Official ros packages for kinova robotic arms." https://github.com/Kinovarobotics/kinova-ros, 2019.

[96] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3, pp. 279–292, 1992.

[97] W. S. McCulloch and W. Pitts, "Neurocomputing: Foundations of research," *ch. A Logical Calculus of the Ideas Immanent in Nervous Activity*, pp. 15–27, 1988.

[98] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[99] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, vol. 2, pp. 995–1001, IEEE, 2000.

[100] B. Siciliano and J.-J. E. Slotine, "A general framework for managing multiple tasks in highly redundant robotic systems," in *Proc. Fifth International Conference on Advanced Robotics (ICAR)*, (Pisa, Italy), pp. 1211–1216, IEEE, 1991.

[101] P. Di Lillo, F. Arrichiello, D. Di Vito, and G. Antonelli, "BCI-controlled assistive manipulator: developed architecture and experimental results," *IEEE Transaction on Cognitive and Developmental Systems*, pp. 1–1, 2020.

[102] P. Di Lillo, E. Simetti, F. Wanderlingh, G. Casalino, and G. Antonelli, "Underwater intervention with remote supervision via satellite communication: Developed control architecture and experimental results within the dexrov project," *IEEE Transaction on Control Systems Technology*, vol. 29, no. 1, pp. 108–123, 2021.

[103] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognition*, vol. 47, no. 6, pp. 2280–2292, 2014.

[104] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[105] A. M. Andrew, "Multiple view geometry in computer vision," *Kybernetes*, 2001.

[106] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D: A modern library for 3D data processing," *arXiv:1801.09847*, 2018.

[107] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin, "The ball-pivoting algorithm for surface reconstruction," *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, no. 4, pp. 349–359, 1999.

[108] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.

[109] 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.

[110] http://cvxr.com/cvx/download/.

[111] https://codeocean.com/capsule/31dcd5d6-7a11-4002-87f5-4d904d13b98a.

[112] https://ieee-dataport.org/documents/robot-dynamics-identification.

[113] J. Swevers, C. Ganseman, D. B. Tukel, J. de Schutter, and H. Van Brussel, "Optimal robot excitation and identification," *IEEE Transactions on Robotics and Automation*, vol. 13, pp. 730–740, Oct 1997.