Università degli Studi di Padova

Doctoral Thesis

Cognitive Task Planning for Smart Industrial Robots

Author:
Elisa Tosello

School Director:
Prof. Matteo Bertocco

Coordinator:
Prof. Carlo Ferrari

Supervisor:
Prof. Enrico Pagello

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy

in the

Intelligent Autonomous Systems Laboratory (IAS-Lab)
Department of Information Engineering (DEI)

Area of interest:

Information and Communication Science and Technologies (ICT)

July 31, 2016

To my parents for their unconditional love,
support and encouragement in following my dreams.

Abstract

Cognitive Task Planning for Smart Industrial Robots

by Elisa Tosello

Keywords: Task Planning, Navigation Among Movable Ostacles, Manipulation, Cloud Robotics, Cognitive Robotics, Ontology, Robot Operating System.

This research work presents a novel Cognitive Task Planning framework for Smart Industrial Robots. The framework makes an industrial mobile manipulator robot Cognitive by applying Semantic Web Technologies. It also introduces a novel Navigation Among Movable Obstacles algorithm for robots navigating and manipulating inside a firm.

The objective of Industrie 4.0 is the creation of Smart Factories: modular firms provided with cyber-physical systems able to strong customize products under the condition of highly flexible mass-production. Such systems should real-time communicate and cooperate with each other and with humans via the Internet of Things. They should intelligently adapt to the changing surroundings and autonomously navigate inside a firm while moving obstacles that occlude free paths, even if seen for the first time. At the end, in order to accomplish all these tasks while being efficient, they should learn from their actions and from that of other agents.

Most of existing industrial mobile robots navigate along pre-generated trajectories. They follow electrified wires embedded in the ground or lines painted on the floor. When there is no expectation of environment changes and cycle times are critical, this planning is functional. When workspaces and tasks change frequently, it is better to plan dynamically: robots should autonomously navigate without relying on modifications of their environments. Consider the human behavior: humans reason about the environment and consider the possibility of moving obstacles if a certain goal cannot be reached or if moving objects may significantly shorten the path to it. This problem is named Navigation Among Movable Obstacles and is mostly known in rescue robotics. This work transposes the problem on an industrial scenario and tries to deal with its two challenges: the high dimensionality of the state space and the treatment of uncertainty.

The proposed NAMO algorithm aims to focus exploration on less explored areas. For this reason it extends the Kinodynamic Motion Planning by Interior-Exterior Cell Exploration algorithm. The extension does not impose obstacles avoidance: it assigns an importance to each cell by combining the efforts necessary to reach it and that needed to free it from obstacles. The obtained algorithm is scalable because of its independence from the size of the map and from the number, shape, and pose of obstacles. It does not impose restrictions on actions to be performed: the robot can both push and grasp every object. Currently, the algorithm assumes full world knowledge but the environment is reconfigurable and the algorithm can be easily extended in order to solve NAMO problems in unknown environments. The algorithm handles sensor feedbacks and corrects uncertainties.

Usually Robotics separates Motion Planning and Manipulation problems. NAMO forces their combined processing by introducing the need of manipulating multiple objects, often unknown, while navigating. Adopting standard precomputed grasps is not sufficient to deal with the big amount of existing different objects. A Semantic Knowledge Framework is proposed in support of the proposed algorithm by giving robots the ability to learn to manipulate objects and disseminate the information gained during the fulfillment of tasks. The Framework is composed by an Ontology

and an Engine. The Ontology extends the IEEE Standard Ontologies for Robotics and Automation and contains descriptions of learned manipulation tasks and detected objects. It is accessible from any robot connected to the Cloud. It can be considered a data store for the efficient and reliable execution of repetitive tasks; and a Web-based repository for the exchange of information between robots and for the speed up of the learning phase. No other manipulation ontology exists respecting the IEEE Standard and, regardless the standard, the proposed ontology differs from the existing ones because of the type of features saved and the efficient way in which they can be accessed: through a super fast Cascade Hashing algorithm. The Engine lets compute and store the manipulation actions when not present in the Ontology. It is based on Reinforcement Learning techniques that avoid massive trainings on large-scale databases and favors human-robot interactions.

The overall system is flexible and easily adaptable to different robots operating in different industrial environments. It is characterized by a modular structure where each software block is completely reusable. Every block is based on the open-source Robot Operating System. Not all industrial robot controllers are designed to be ROS-compliant. This thesis presents the method adopted during this research in order to Open Industrial Robot Controllers and create a ROS-Industrial interface for them.

Sommario

Cognitive Task Planning for Smart Industrial Robots

di Elisa Tosello

Parole chiave: Task Planning, Navigation Among Movable Ostacles, Manipulation, Cloud Robotics, Cognitive Robotics, Ontology, Robot Operating System.

Questa ricerca presenta una nuova struttura di Pianificazione Cognitiva delle Attività ideata per Robot Industriali Intelligenti. La struttura rende Cognitivo un manipolatore industriale mobile applicando le tecnologie offerte dal Web Semantico. Viene inoltre introdotto un nuovo algoritmo di Navigazione tra Oggetti Removibili per robot che navigano e manipolano all'interno di una fabbrica.

L'obiettivo di Industria 4.0 è quello di creare Fabbriche Intelligenti: fabbriche modulari dotate di sistemi cyber-fisici in grado di customizzare i prodotti pur mantenendo una produzione di massa altamente flessibile. Tali sistemi devono essere in grado di comunicare e cooperare tra loro e con gli agenti umani in tempo reale, attraverso l'Internet delle Cose. Devono sapersi autonomamente ed intelligentemente adattare ai costanti cambiamenti dell'ambiente che li circonda. Devono saper navigare autonomamente all'interno della fabbrica, anche spostando ostacoli che occludono percorsi liberi, ed essere in grado di manipolare questi oggetti anche se visti per la prima volta. Devono essere in grado di imparare dalle loro azioni e da quelle eseguite da altri agenti.

La maggior parte dei robot industriali mobili naviga secondo traiettorie generate a priori. Seguono fili elettrificati incorporati nel terreno o linee dipinte sul pavimento. Pianificare a priori è funzionale se l'ambiente è immutevole e i cicli produttivi sono caratterizzati da criticità temporali. E' preferibile adottare una pianificazione dinamica se, invece, l'area di lavoro ed i compiti assegnati cambiano frequentemente: i robot devono saper navigare autonomamente senza tener conto dei cambiamenti circostanti. Si consideri il comportamento umano: l'uomo ragiona sulla possibilità di spostare ostacoli se una posizione obiettivo non è raggiungibile o se tale spostamento può accorciare la traiettoria da percorrere. Questo problema viene detto Navigazione tra Oggetti Removibili ed è noto alla robotica di soccorso. Questo lavoro traspone il problema in uno scenario industriale e prova ad affrontare i suoi due obiettivi principali: l'elevata dimensione dello spazio di ricerca ed il trattamento dell'incertezza.

L'algoritmo proposto vuole dare priorità di esplorazione alle aree meno esplorate, per questo estende l'algoritmo noto come Kinodynamic Motion Planning by Interior-Exterior Cell Exploration. L'estensione non impone l'elusione degli ostacoli. Assegna ad ogni cella un'importanza che combina lo sforzo necessario per raggiungerla con quello necessario per liberarla da eventuali ostacoli. L'algoritmo risultante è scalabile grazie alla sua indipendenza dalla dimensione della mappa e dal numero, forma e posizione degli ostacoli. Non impone restrizioni sulle azioni da eseguire: ogni oggetto può venir spinto o afferrato. Allo stato attuale, l'algoritmo assume una completa conoscenza del mondo circonstante. L'ambiente è però riconfigurabile di modo che l'algoritmo possa venir facilmente esteso alla risoluzione di problemi di Navigazione tra Oggetti Removibili in ambienti ignoti. L'algoritmo gestisce i feedback dati dai sensori per correggere le incertezze.

Solitamente la Robotica separa la risoluzione dei problemi di pianificazione del movimento da quelli di manipolazione. La Navigazione tra Ostacoli Removibili forza il loro trattamento combinato introducendo la necessità di manipolare oggetti diversi, spesso ignoti, durante la navigazione.

Adottare prese pre calcolate non fa fronte alla grande quantità e diversità di oggetti esistenti. Questa tesi propone un Framework di Conoscenza Semantica a supporto dell'algoritmo sopra esposto. Esso dà ai robot la capacità di imparare a manipolare oggetti e disseminare le informazioni acquisite durante il compimento dei compiti assegnati. Il Framework si compone di un'Ontologia e di un Engine. L'Ontologia estende lo Standard IEEE formulato per Ontologie per la Robotica e l'Automazione andando a definire le manipolazioni apprese e gli oggetti rilevati. È accessibile a qualsiasi robot connesso al Cloud. Può venir considerato I) una raccolta di dati per l'esecuzione efficiente ed affidabile di azioni ripetute; II) un archivio Web per lo scambio di informazioni tra robot e la velocizzazione della fase di apprendimento. Ad ora, non esistono altre ontologie sulla manipolazione che rispettino lo Standard IEEE. Indipendentemente dallo standard, l'Ontologia proposta differisce da quelle esistenti per il tipo di informazioni salvate e per il modo efficiente in cui un agente può accedere a queste informazioni: attraverso un algoritmo di Cascade Hashing molto veloce. L'Engine consente il calcolo e il salvataggio delle manipolazioni non ancora in Ontologia. Si basa su tecniche di Reinforcement Learning che evitano il training massivo su basi di dati a larga scala, favorendo l'interazione uomo-robot. Infatti, viene data ai robot la possibilità di imparare dagli umani attraverso un framework di Apprendimento Robotico da Dimostrazioni.

Il sistema finale è flessibile ed adattabile a robot diversi operanti in diversi ambienti industriali. È caratterizzato da una struttura modulare in cui ogni blocco è completamente riutilizzabile. Ogni blocco si basa sul sistema open-source denominato Robot Operating System. Non tutti i controllori industriali sono disegnati per essere compatibili con questa piattaforma. Viene quindi presentato il metodo che è stato adottato per aprire i controllori dei robot industriali e crearne un'interfaccia ROS.

List of Publications

Papers closely related to the scientific achievements of this Thesis:

I **E. Tosello**, Z. Fan, A. C. Gatto, and E. Pagello
Cloud-based Task Planning for Smart Robots
The 14th International Conference on Intelligent Autonomous Systems (IAS-14); Proceedings of. July 3-7, **2016**. Shanghai (China).

II N. Castaman, **E. Tosello**, and E. Pagello
A sampling-based Tree Planner for Navigation Among Movable Obstacles
ISR 2016; 47th International Symposium on Robotics; Proceedings of. June 21-22, **2016**. Munich (Germany). pp. 292-299.

III **E. Tosello**, Z. Fan, and E. Pagello
A Semantic Knwoledge Base for Cognitive Robotics Manipulators
Toward Intelligent Social Robots - Current Advances in Cognitive Robotics, Workshop on. Nov 3rd, **2015**. Seul (Korea).

IV S. Michieletto, **E. Tosello**, F. Romanelli, V. Ferrara, and E. Menegatti
ROS-I Interface for COMAU Robots
Simulation, Modeling, and Programming for Autonomous Robots. 4th International Conference, SIMPAR 2014; Proceedings of. October 20-23, **2014**. Bergamo (Italy). Online ISBN: 978-3-319-11900-7. pp. 243-254.

V **E. Tosello**, S. Michieletto, A. Bisson, E. Pagello, and E. Menegatti
A Learning from Demonstration Framework for Manipulation Tasks
ISR/Robotik 2014; 45th International Symposium on Robotics; Proceedings of. June 2-3, **2014**. Munich (Germany). ISBN: 978-3-8007-3601-0. pp. 1-7.

VI Z. Fan, **E. Tosello**, M. Palmia, and E. Pagello
Applying Semantic Web Technologies to Multi-Robot Coordination
NRF-IAS-2014; New Research Frontiers for Intelligent Autonomous Systems; Workshop. July 19, **2014**. Venice (Italy).

Contribution report

The author of this Thesis has given a main contribution to Papers I, II, III, IV and VI, while she has only partially contributed to Paper V through experiments implementation, data analysis and discussion of results.

The following papers are not strictly correlated to the main topics developed in this Thesis.

- **E. Tosello**, S. Michieletto, E. Pagello
  Training master students to program both virtual and real autonomous robots in a teaching laboratory
  EDUCON 2016; IEEE Global Engineering Education Conference; Proceedings of. April 10-13, **2016**. Abu Dhabi (AUE). pp. 621 - 630.

- S. Michieletto, **E. Tosello**, E. Pagello, E. Menegatti
  Teaching Humanoid Robotics by means of Human Teleoperation through RGB-D Sensors
  Robotics and Autonomous Systems, **Journal of**. Available online 23 October **2015**.

- N. Boscolo, **E. Tosello**, S. Tonello, M. Finotto, R. Bortoletto, and E. Menegatti
  A Constraint Based Motion Optimization System for Quality Inspection Process Improvement
  Simulation, Modeling, and Programming for Autonomous Robots. 4th International Conference, SIMPAR 2014; Proceedings of. October 20-23, **2014**. Bergamo (Italy). Online ISBN: 978-3-319-11900-7. pp. 545-553.

- **E. Tosello**, R. Bortoletto, S. Michieletto, E. Pagello, and E. Menegatti
  An Integrated System to approach the Programming of Humanoid Robotics
  4th International Workshop Teaching Robotics, Teaching with Robotics & 5th International Conference Robotics in Education; Proceedings of. July 18, **2014**. Padova (Italy). ISBN 978-88-95872-06-3. pp. 93-100.

- D. Kurabayashi, Y.Takahashi, R. Minegishi, **E. Tosello**, E. Pagello, R. Kanzaki
  Property Investigation of Chemical Plume Tracing Algorithm in an Insect Using Biomachine Hybrid System
  Living Machines. 29 July - 2 August, **2013**. London (United Kindom). pp. 131-142.

Paper published prior PhD:

- S. Tonello, G. P. Zanetti, M. Finotto, R. Bortoletto, **E. Tosello**, E. Menegatti
  WorkCellSimulator: A 3D Simulator for Intelligent Manufacturing
  Simulation, Modeling, and Programming for Autonomous Robots. Springer Verlag Berlin. November 5-8, **2012**. Tsukuba (Japan). pp. 311-322.

Acknowledgments

I am grateful to Prof. Pagello, my supervisor and my mentor, for the opportunity he gave me by working in the world of Research. He gave me his passion, almost devotion, for Robotics. A man always ready to put into practice new ideas and lead innovation in this field. I want to thank him for giving me the opportunity to make my little Robotics contribution. Thanks also to Prof. Menegatti for welcoming me in the Intelligent Autonomous System Lab. They both gave me the opportunity to professionally and personally grow, improve myself... and travel!

Huge thanks to my colleagues for sharing ideas and hard work. Special thanks to Roberto, my right-hand man and my referee in the lab. "Cicca?".

Thanks to Prof. Kavraki. She let me conclude this chapter of my life becoming a Citizen of the World. I will do my best!

Thanks to my old friends, the ones who never get tired to ask me out and feel say no, the ones who were awake at night to keep me company while I was preparing the presentation for my first conference. Thanks to those I've known all my life. Vale, sometimes we haven't heard in ages but every time it is like it was yesterday. I know you will be close to me even when I will be away! Thanks to passing friends because they lightened my journey. Thanks to friends of all nationalities, the ones I met at summer schools and conferences.. Spanish, Turkish, German, French, Chinese (nationalities are too many to list all) people that made me realize that the world is full of people who want to improve it. Pedro, you are one of them!

Particular thanks to my parents, Manuela and Corrado, because if I came up here I owe all it to them, their unconditional love, and sacrifices. Thanks for accepting my choices (even those you do not understand :P).

Thanks to my cousin Betty. She is like a sister for me!

And finally thanks to the man who read all my papers and is still waiting for his "robot carrying beer", Francesco. Thanks for having supported me all the times I thought of giving up. Thanks for understanding my studies in Robotics. Believe! That robot will come!

Elisa

Padova, Italy, July 2016

Contents

List of Figures

List of Tables

List of Abbreviations

AGILE	Antipodal Grasping Identification and LEarning
AGV	Automatic Guided Vehicles
ANN	Approximate Nearest Neighbor
B-Spline	Basis Spline
BIC	Bayesian Information Criterion
BSD	Berkeley Software Distribution
CoM	Center of Mass
CORA	Core Ontology for Robotics & Automation
CORE	Cloud-base Object Recognition Engine
DBMS	Database Management System
DoF	Degree of Freedom
DoG	Difference of Gaussians
DSP	Digital Signal Processing
EM	Expectation Maximization
EST	Expansive Space Trees
GMM	Gaussian Mixture Model
GMR	Gaussian Mixture Regression
JSON	JavaScript Object Notation
KPIECE	Kinodynamic Motion Planning by Interior-Exterior Cell Exploration
LoG	Laplacian of Gaussian
MAP	Manufacturing Automation Protocol
MPC	Model Predictive Control
NAMO	Navigation Among Movable Obstacles
OpenCV	Open source Computer Vision library
ORA	Ontology for Robotics & Automation
ORL	Open realistic Robot Library
OSD	Object Segmentation Database
OWL	Web Ontology Language
PCA	Principal Component Analysis
PCL	Point Cloud Library
RAS	Robotics and Automation Society
RDF	Resource Description Format
RLfD	Robot Learning from Demonstration
RL	Reinforcement Learning
ROS-I	Robot Operating System-Industrial
ROS	Robot Operating System
RRT	Rapidly-exploring Random Trees
RTAI	Real-Time Application Interface

RTnet	Real-Time networking
SBL	Single-query Bidirectional probabilistic roadmap planner with Lazy collision checking
SHOT	Signature of Histograms of OrienTations
SIFT	Scale Invariant Feature Transform
SMP+	System Master Processor
SOM	Semantic Object Map
SOM	Semantic Object Maps
SPARQL	Simple Protocol and RDF Query Language
SVM	Storage Virtual Machine
URDF	Universal Robot Description Format
W3C	World Wide Web Consortium
WG	Working Group

1

Background and Motivation

The first Chapter depicts the Motivation of the conducted Research. Challenges and Contributions are highlighted.

Contents

The goal of this thesis is proposing a novel approach for Cognitive Task Planning for Smart Industrial Robots.

Focusing on tasks such as Navigation and Manipulation, the aim is that of giving industrial mobile manipulator robots the ability of autonomously navigate in a factory while deciding if moving obstacles could shorter the path or could let the achievement of goals otherwise unreachable. The potential of solving Task Planning problems while combining Navigation and Manipulation capabilities is then given to robots, together with the Cognitive capabilities required to improve robots autonomy. Every robot should be able to learn how to accomplish a task from other agents and should solve tasks by applying what learnt. This is especially useful in robotics manipulation because of the large variety of existing manipulable objects and of the large variety of possible gripper manipulation poses. For this purpose, an Open Semantic Framework for Knowledge acquisition is proposed in support of the Navigation planner and can be treated as an information resource for robots performing Manipulation tasks.

1.1 Motivations

In the cyber-physical production world of Industrie 4.0, the idea of populating factories with robots, designed merely as universal machines to boost productivity, and transporting goods through static production lines is becoming outdated. The aim is that of increasingly evolving robots into factory servants and assistants able to support human workers and accomplish multiple tasks. They should act as AGVs able to substitute conveyor belts by moving in swarms and autonomously finding their destinations without floor markings, induction loops or magnets. They should be able to move products, workpieces or entire productions units, and reposition them in different locations. Moreover, the possibility of adopting a safe and productive human-robot interaction in presence of ambiguous situations and tight workspaces should be the everyday. Robots should collaborate with humans or learn from humans, e.g., through a RLfD framework [V], [1]. Multi-robots cooperative systems should be adopted to speed up and improve the way to operate. A team of robots, in fact, can subdivide tasks according to their abilities [VI].

Formally, this means that Smart Robots should populate Smart Factories: versatile and efficient Artificial Intelligence systems able to support human workers. Versatile because robots should be able to perceive and independently, safely, and timely adapt to external influences and to the resulting constantly changing surroundings. Efficient because, in order to intelligently act, robots should be able to learn from past experiences, errors, actions performed by other agents (either humans or robots), and build on their capabilities based on that knowledge. These capabilities cannot arise from precomputed software routines. This thesis argues that robots should be able to reason and act like humans: they should decide which path to cover while moving in swarms or if moving obstacles out of their way can shorter that path. They should learn how to manipulate objects and they should share what learnt in order to improve the global knowledge growth. This means that robots should be Cognitive: agents able to learn how to solve new tasks and increase their capabilities based on that knowledge. Moreover, they should combine their Navigation and Manipulation capabilities in order to solve those tasks. For these reasons, this thesis focuses on the resolution of Motion Planning problems through a Cognitive combination of robotics Navigation and Manipulation capabilities.

Over the last decades, many Motion Planning algorithms have been developed in order to find a continuous robot motion connecting a robot start configuration q_S and a goal configuration q_G. Usually, these algorithms limit the search of the path within the collision-free space, while

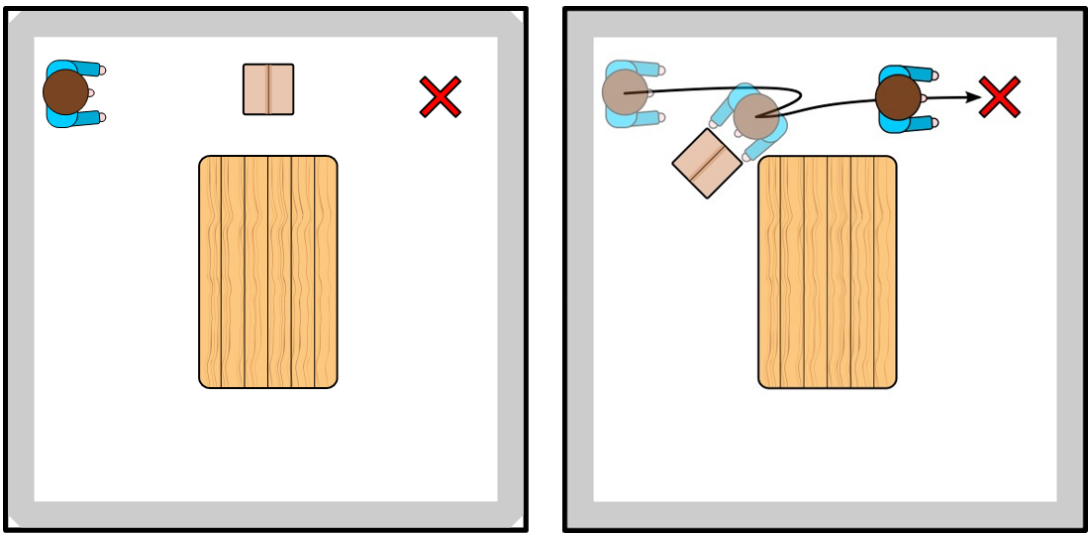


Figure 1.1: A Navigation Among Movable Obstacles scenario.

avoiding contacts with obstacles in the scene. They ignore the fact that mobile manipulators have both navigation and manipulator capabilities and they are in contrast to how humans naturally act: humans do not hesitate to utilize their manipulation capabilities to modify the environment, create gaps among obstacles, and minimize efforts and time necessary to reach a goal. Figure 1.1 depicts a symbolic scenario: instead of circumnavigating the collision space that occludes the goal, humans first try to move the obstacle. If the manipulation is too onerous (e.g., the object is heavy, the time required to manipulate the object exceeds the time to perform the collision-free path, etc.), then they will interrupt the task and walk through the existing free space. Robots should demonstrate similar behavior. This thesis highlights the need of a Motion Planning solver that combines navigation and manipulation capabilities for robots in automation while keeping in mind that other fields argue the same need. Service robots, for example, should be able to autonomously navigate inside a home or an office while opening doors or moving chairs. Rescue robots should be able to act in areas affected by disasters such as mining accidents, floods, and earthquakes; they should be able to decide when moving obstacles out of their way in order to reach and save humans in the shortest possible time. NAMO is an instance of Motion Planning that gives robots the ability to reason about the environment and choose when manipulating obstacles [2]. It plans robot movements taking into consideration the possibility of moving objects if the goal is otherwise unreachable or if moving objects may significantly shorten the path. This thesis proposes a Sampling-based tree planner that solves NAMO problems by giving robots the capability of finding the shorter path to a goal pose eventually manipulating objects that obstruct free navigable spaces. The planner is designed for cognitive robots and tested on mobile industrial ones [II].

Once established the will to combine robotics navigation and manipulation capabilities, a second fact should be take on: a wide variety of manipulable objects exist. Given the large number of different existing objects and robotic grippers, off-line precomputing all manipulation actions is inefficient or even impossible. It requires robots to be equipped with large on board CPU stacks and it does not make them intelligent. Instead, robots should be able to perceive and explore the surrounding workspace in order to detect objects to be manipulated, obstacles to be avoided, or event the agents with which they are collaborating. They have to be able to compute or learn the arm's path that allows them to approach objects without colliding and the gripper data necessary for the manipulation (e.g., the grasp pose, the gripping force, the stability of finger contacts). Given an object seen for the first time, they could combine a trial-and-error Reinforcement Learning

technique with a matching phase that compares the achieved results with the desired ones. For all known objects or similar ones, they should exploit the already adopted strategies [3, 4]. This thesis argues the lack of a sharable, Cloud-based Semantic Knowledge Base that can be adopted as an information resource for intelligent, autonomous, cognitive robots performing manipulation tasks. Its adoption will overcome the limited capacity of robots on-board data memory while offering a common medium to communicate and share knowledge. This will speed up the learning phase of each manipulator robot. Imagine a robot that can learn how to manipulate an object (e.g., by exploring the workspace or observing actions performed by other agents) and can save the acquired information on a common Knowledge Base, together with data describing detected objects and explored environments. In this way, the information can be shared with other robots, facilitating future replicas by the same robot or by different robots with the same assignment. The purpose is not to formulate a *cognitivist* precoded symbolic representation of the workspace; the aim is to create a Knowledge Base able to improve the *co-determination* of *emergent systems*. Traditional cognitive modeling approaches involve symbolic coding scheme as a means for depicting the world. This symbolic representation originates a designer-dependent action domain [5] that is successful if the system acts under the conditions specified by descriptors; otherwise, a semantic gap [6] between perception and possible interpretation follows and must be bridged by implementing ad-hoc frameworks [7]. Co-determination is the solution. It means that the agent constructs its reality (its world) because of its operations in the world: intelligently acting is functionally dependent on the interaction between perception and cognition. With this aim, the proposed Open Semantic Framework is composed of an Ontology and a Cloud-based Engine. The former stores data about objects and actions necessary to manipulate them. The latter detects objects in the scene and retrieves their manipulation actions from the Ontology. If no ontological data exists, the Engine generates and stores it within the Ontology. Adopting a Cloud server allows to offload robots from CPU-heavy tasks and to perform intensive computation while meeting the hard real-time constraints of operations. The Web Ontology, instead, lets the definition of a common vocabulary that ensures a common understanding during the interaction as well as an efficient data transfer and integration.

Finally, in order to guarantee the flexibility and adaptability of the system according to the assigned task (e.g., navigation or manipulation), the robot has to be part of a modular framework structured into reusable blocks. Every block should identify a function that accomplishes part of the task. In this way, the framework becomes independent from the blocks nature and allows users to customize their applications according to their needs: they can use the entire system or maintain only some of its modules. Hence, code reusability makes the software portable and manufacturer-independent. Updating single blocks is easier and faster, as well as finding and fixing bugs. This improves robustness and efficiency. Blocks composition is dynamic; hence, the functionality of the system can change at runtime in accordance to the highlighted flexibility and adaptability requirements. This thesis argues the lack of consistent incentives offered to software developers in order to stimulate them in taking the reuse direction. There exist projects that finance the scientific community in order to apply its research on industrial products. However, most of manufactures do not support the community efforts: these researches represent a long-term cost and fall outside of industrial short-term goals. For this reason, most of existing industrial applications are robot dependent: programmers implement robotics software packages for specific robot platforms performing tasks in specific environments. Applications, similar to the existing ones, are every time implemented from scratch. In order to make the proposed framework reusable, ROS has been adopted [8].

Not all industrial robot controllers are ROS-compliant. The last part of this thesis focuses on the efforts done during this research to combine ROS high-level functionalities with the low-level

reliability and safety of industrial robot controllers in order to create a robust and reliable software. The validity of the system is proved by describing the results obtained after the opening of a Comau robot controller and the implementation of its ROS interface. Opening a controller is essential in order to make it ROS-compliant. According to Technical Committee of Open Systems of IEEE, "An open system provides capabilities that enable properly implemented applications to run on a variety of applications and present a consistent style of interaction with the user" [9]. From user point of view, the "openness" of systems consists in capabilities to integrate, extend, and reuse software modules in control systems [10]. In our case, for example, the robot has to detect the object on the surrounding workspace in order to manipulate it. Hence, vision sensors integration is essential. Even opening industrial robot controllers is not an established habit and, for this reason, it falls among the nowadays robotics challenges. Two features have to be considered when dealing with an industrial robot. The former is its insertion in an industrial environment populated by people: the robot has to be designed observing several requirements, such as industrial standards and safety regulations. The latter regards the maintenance of the trade secret. The combination of these two factors has always led manufactures to keep their controllers closed to the users who want to improve their capabilities and extend their computational power. Closure hinders the robotics performance improvement: software modules are application-specific, they cannot be reused for other applications and cannot be modified, external hardware such as sensors and vision systems cannot be integrated. Hence, designing and testing new algorithms in a short time and at limited costs is not possible. Many robot manufactures have realized that this policy is counterproductive, limiting the possibility of offering flexible, integrable and highly reconfigurable robotics units. Reducing the manufacturer "lock-in" and increasing the innovation changes will speed up the 4th Industrial Revolution.

1.2 Challenges

The goal of this thesis is to develop a practical system that allows a robot to reason about the environment and eventually move obstacles away. The robot should be able to cognitive learn how to solve the planning and flexibly adapt to the changing surroundings. Multiple challenges address the framework.

Real-timing In order to create a reusable and adaptable industrial framework, the industrial robot controller should be opened and made ROS-compliant. The controller should have real-time characteristics in order to let the robot real-time react to external influences and sensor feedbacks. This is a challenge because of the non real-time ROS nature. Moreover, a real-time Web connection between the ROS framework and the Cloud Engine has to be guaranteed: once detected the object to be manipulated, the robot should real-time access the Web Ontology to retrieve manipulation data. Delays could compromise the successful outcome of the action. If no data has been stored on the Ontology, the robot has to try to manipulate (or learn to manipulate) the object itself in real-time.

Dimensionality When solving a NAMO problem, the planner has to face very high dimensional spaces. This challenge is related to the size of the space, the lack of meaningful heuristics for choosing objects, and the complex nonlinearity of object interaction that leads to local minima. In order to understand the state space dimensionality, consider navigating in a room populated by manipulable obstacles. As stated by Levihn in his PhD thesis [11], planning for a single mobile robot with a fixed number of DoFs is not enough. NAMO requires to consider also the displacement of any movable obstacle taking the dimension of the state space exponential in the number of objects it contains. Formally, if $\mathscr{C}_r \in \mathbb{R}^{d_r}$ is the configuration of a robot r (d_r is the

dimension of the robot state space, i.e. the number of its DoFs) and $\mathscr{C}_o \in \mathbb{R}^{d_o}$ is the configuration of an object o (d_o is the dimension of the object state space), then an environment composed of a robot r and N obstacles o_i ($1 \leq i \leq N$) will be represented by the product of subspaces $\mathscr{C}_r \times \mathscr{C}_{o_1} \times ... \times \mathscr{C}_{o_i} \times \mathscr{C}_{o_N} \in \mathbb{R}^{(d_r+Nd_o)}$. If d is the discretization resolution of every dimension of the considered discretized environment, then the number of possible states will be $O(d^{(d_r+Nd_o)})$. Proper heuristics should be formulated for efficiently selecting the objects to be displaced and deciding where to move them. As done by Stilman in [12], without loss of generality, this thesis reduces the navigation domain to a sub-domain, namely L_1, where disconnected components of free-space could be connected independently by moving a single obstacle. If heuristics are correctly formulated, the implemented algorithm will be scalable: an algorithm characterized by a computational time of an order of magnitude such that the system will be as independent as possible from the size of the map and from the number, size and shape of obstacles.

Uncertainty Usually, real robots do not have a complete knowledge of their workspace. While acting, they have to perceive the environment and exploit sensor feedbacks in order to auto-localize, locate and recognize objects, and verify the outcome of their actions. Existing sensors have limited capacities, resulting in state and action uncertainties while physical interacting with the environment. While moving, for example, motor engines could be imprecise, the terrain could be rough or characterized by an unexpected friction. For those reasons, the robot final state could not correspond to the desired one. While grasping or pushing an object, the weight of that object could be different from the estimated one and the action could not reflect what planned. Also in this case, friction and the dynamic features of the object could affect the motion. Improving existing objects recognition and localization methods is beyond the scope of this thesis but existing tools accomplishing these functions should be well employed in order to give robots the capability of manage state and action uncertainties.

Autonomy A robot is autonomous if it does not need human intervention. This objective is especially challenging when sharing and exchanging data. Communication protocols should be programming language and transport agnostic. Information should be represented in a machine-understandable format: the meaning of the content should be represented explicitly, separately in terms of logical axioms that a robot can understand, and these logical axioms should be well-defined. This means that a common vocabulary should be defined in order to ensure a common understanding during the interaction as well as an efficient data transfer and integration.

1.3 Contributions

A very short description of the contributions of this thesis with respect to the exposed challenges follows. Their implementations will be detailed in the following chapters.

Real-timing In order to guarantee the real-time performances of the implemented Task Planning framework, the framework is composed of ROS blocks. The industrial robot controller has been opened and a ROS interface has been implemented in order to let the framework, and more generally ROS users, send commands to this controller. The connection between the framework and the Cloud-Engine is as real-time as possible.

The controller has been made real-time by adding an external PC based on GNU/Linux Operating System. The PC extends the standard kernel with real-time capabilities by using the RTAI module [13]. In addition, a real-time communication layer is provided using the RTnet stack [14], an open source hard real-time network protocol compatible with RTAI. The gap of ROS of not being real-time has been overcome by developing a communication node connecting the real robot with

the ROS module. The node includes both a real-time and a normal priority thread. The former establishes the connection between the controller and the real robot. The latter provides the controller with the motion data: if a ROS compliant platform aims to control the robot by sending it specific motion directives, then the thread will deliver these motion commands to the controller in the form of ROS messages. If a non-ROS compliant processor has the same request, the thread will send the motion primitives to the controller in the form of TCP/IP messages.

On the other hand, the Cloud infrastructure is based on a service architecture (synchronous). The communication between nodes of the ROS Client framework and that of the Cloud Engine is based on Python WebSockects over HTTP and ros_bridge [15], a middleware abstraction layer that provides socket-based programmatic access to robot interfaces and algorithms by ROS. All messages are transported as serialized ROS messages represented as JSON[i] objects. Within the Cloud Engine, the Ontology access is "super-fast" thanks to the implementation of a "super-fast" Similar Search routine: for every object stored on the Ontology, its descriptors are saved also on a server folder. When detecting an object, descriptors of the folder are compared using a super-fast cascade hashing algorithm (ten times faster than other image matching algorithms) [16]. The comparison returns the name of the class representing similar descriptors. The name is used to filter the Ontology. This procedure lets query the Ontology only once. As result, the architecture has real-time performances and is scalable.

Dimensionality A Sampling-based Tree Planner has been implemented in order to solve NAMO L_1 problems. The planner exploits the exploration strategy of KPIECE, which focuses exploration on less covered areas, by assigning an importance function that gives priority to unexplored spaces (see Appendix A). As stated in [17, 18], KPIECE projects the state space to a lower dimensional Euclidean space and discretizes it by using a grid. The discretization is used to estimate the coverage of the state space and to evaluate cells goodness: for every cell, the algorithm saves the number of times it has been explored and the progress achieved by exploring it. Combining collected information, KPIECE is able to deterministically select the regions to explore: the best less explored ones. While KPIECE searches for a collision-free path, the NAMO importance function takes into consideration also those areas populated by obstacles and the effort done by robots to remove them. The algorithm extends KPIECE because, if compared with other Motion Planning algorithm, KPIECE makes a better use of the information collected during the planning process. This information is used to decrease the amount of forward propagation the algorithm needs. As consequence, both runtime and memory requirements decrease making the algorithm suitable to handle high dimensional systems with complex dynamics. The obtained algorithm is locally optimal and its computation time has an order of magnitude that makes it almost independent from the size of the map and from the number, shape, and position of obstacles.

Uncertainty A Microsoft Kinect vision sensor is mounted on the robot. Data filters, feature descriptions, and segmentation techniques of the Point Cloud Library [19, 20] are exploited in order to process the Kinect signals and extract the scene, segment and recognize objects, and extract their poses. This information is used to update the NAMO occupancy map, recalculate the path if necessary, verify the outcome of an action (manipulation or navigation), and eventually plan a corrective action (see Appendix C).

Autonomy An Ontology has been implemented in order to define a common manipulation vocabulary. It encodes information about assigned tasks, objects, and their manipulation data. Defining an Ontology instead of an SQL Database makes robots more autonomous. In fact, the

[i] JSON is a lightweight data-interchange format that is easy for humans to read and write and for machines to parse and generate

logical data structure of a DBMS is limited in scope and biased toward the employed implementation strategy. An Ontology, instead, makes all pertinent knowledge about a domain explicit and is represented in a computer-interpretable format. This format allows a software to reason on that knowledge in order to infer additional information. Following these concepts, the implemented Ontology extends the work proposed by the IEEE RAS ORA WG [21, 22, 23] by covering the manipulation task domain. Data is represented through the union of the Resource Description Format (RDF)[ii] and the Web Ontology Language (OWL)[iii], namely OWL Full. While RDF is used to define data structure, OWL adds semantic to the schema and allows the user to specify relationships among data. OWL Full allows an Ontology to augment the meaning of the RDF vocabulary by guaranteeing the maximum expressiveness of OWL and the syntactic freedom of RDF. Indeed, OWL is adopted by the W3C and is the representation language used by the RAS ORA WG.

1.4 Thesis Outline

The rest of this work is organized as follows:

- **Chapter 2 - Navigation Among Movable Obstacles** presents the proposed algorithm to solve NAMO problems while focusing exploration on less explored areas;
- **Chapter 3 - Manipulation** proposes some methods to Cognitive manipulate objects: using a Robot Learning from Demonstration Framework or a Reinforcement Learning technique;
- **Chapter 4 - Open Semantic Framework** collects efforts done to tap into the Cloud and develop an Open Semantic Framework for Knowledge acquisition of Cognitive robots performing Manipulation tasks;
- **Chapter 5 - Flexible robots: Open Controllers and ROS-I Interfaces** details contributions of this thesis to robots flexibility. It focuses on the Opening of Industrial Robot Controllers and the development of a common ROS-I control interface. It is focused on the Comau case;
- **Chapter 6 - Conclusions and Future Work** provides concluding remarks and outlines future research directions.

[ii] Resource Description Format (RDF): http://www.w3.org/RDF
[iii] Web Ontology Language (OWL): http://www.w3.org/TR/owl-features/

2

Navigation Among Movable Obstacles

This chapter describes the algorithm developed to solve Navigation Among Movable Obstacles problems. It exploits the exploration strategy of the Kinodynamic Motion Planning by Interior-Exterior Cell Exploration algorithm in order to favor exploration on less explored areas.

Contents

2.1 State of Art

As stated in [2, 24, 25, 26], Wilfong [27] first proved that deterministic NAMO with an unconstrained number of obstacles is NP-hard. Demaine [28] showed that even when considering only unit square obstacles the problem remains NP-hard. Stilman first simplified the problem and made it a practical challenge that can be addressed with existing hardware and computation. In [2], he defined and solved a subclass of NAMO problems, namely L_1, where disconnected components of free-space could be connected independently by moving a single obstacle. These planners solved NAMO problems assuming full world knowledge but real robot systems only have limited sensor range and partial environment knowledge.

Improvements solve NAMO problems in unknown environments. An example is [25]: the robot gains the information about the position and size of objects through the use of a laser range finder. Objects movability, instead, is determined by interaction. The algorithm discretizes the environment in a N x M grid and calculates a plan from q_S to q_G through an A*-Search using the Euclidean distance as heuristic. Every unknown cell in the map is assumed to be free and every object is considered movable unless a failed push action is performed. Authors presented a baseline as well as an optimized approach. The baseline approach calculates plans for all possible actions on all known objects for any change in the environment. The method does not scale for larger environments. The optimized method does not automatically recompute plans if new information becomes available. When encountering new obstacles, it recomputes plans only when the current one becomes invalid due to collisions. For every new obstacle considered for the re-plan, it evaluates push actions limiting their number by the cost of just avoiding the obstacle itself. It maintains an ordered list of costs relative to the manipulation of each object and, at every step, it selects the minimum cost one. With respect to the baseline, the optimized approach reduces the number of objects that have to be evaluated during the selection but it does not guarantee local optimality, it supports only push actions, and it constrains obstacle shapes to be rectangular.

In [26], Levihn improved [25] guaranteeing local optimality. This new formulation of the algorithm does not restrict the robot action set to pushes and does not restrict objects to be rectangular: they can have arbitrary shapes. It introduces a dynamic bound that limits the number of obstacles evaluations. This bound is based on the cost of all previously evaluated objects and on that of moving into the collision free space. It maintains two lists of cost underestimates for every plan. Each entry of the first list refers to the minimum distance that the robot has to cover if it decides to move a certain object. The minimum distance assumes free-space in all other areas then that covered by that object, even in known parts of the environment, in order to achieve independence of the entries from future environment changes. Every entry of the second list, instead, refers to the information of the obstacles already evaluated. This means that it stores, according to the index of the movable object, the lowest value that was determined in the most recent evaluation of a NAMO plan displacing that object. The use of these two lists and of the dynamic bound increases the computational efficiency and guarantees local optimality. Levihn proved that the algorithm guided the robot to a desired goal in a time between 18 and 50 seconds in an environment containing between 2 and 70 randomly generated obstacles in randomized configurations.

In [29], Kakiuchi presented a solution for NAMO problems in unknown environments and experimented it on the humanoid robot HRP-2 using only on-board sensors. Movable obstacles are detected using active sensors and a color range sensor. When an obstacle is moved, the perception of the environment is reconstructed.

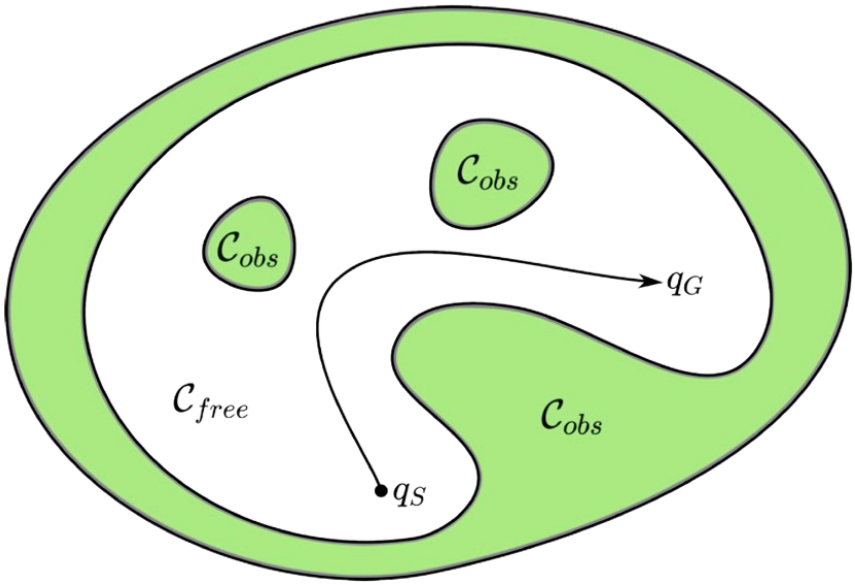


Figure 2.1: The Configuration Space.

2.2 Problem Statement

Consider a robot acting in a workspace W populated by obstacles. The environment can be modeled as a three-dimensional Euclidean Space with

- r - the robot. $r \in \mathbb{R}^N$ for a robot with N DoFs. Every DoF is represented by a polyhedral link;
- $\mathscr{O}$ - the set of obstacles, i.e. the collision space.

In detail, $\mathscr{O} = \mathscr{O}_{fixed} \cup \mathscr{O}_{movable}$, with:

- $\mathscr{O}_{fixed} = \{o_{f_1}, ..., o_{f_i}, ..., o_{f_m}\}$ - the set of polygonal *Fixed Obstacles* that must be avoided;
- $\mathscr{O}_{movable} = \{o_{m_1}, ..., o_{m_j}, ..., o_{m_k}\}$ - the set of polygonal *Movable Obstacles* that can be manipulated by applying forces at allowable contacts.

Definition 1 (LaTombe [30]): A **Motion Planning** problem aims to find a collision free path τ from a robot Start configuration, q_S, to a Goal configuration, q_G:

$$\tau : [0,1] \rightarrow \mathscr{C}_{free} \;\; with \;\; \tau(0) = q_S, \tau(1) = q_G \tag{2.1}$$

where $\mathscr{C}_{free}$ is the free space. Formally, if $\mathscr{C}$ is the configuration space, i.e. the set of all possible robot configurations, that means the set of all transformations that can be applied to a robot given its kinematics, then

$$\mathscr{C}_{free} = \{q \in \mathscr{C} | A(q) \cap \mathscr{O} = \emptyset\} \tag{2.2}$$

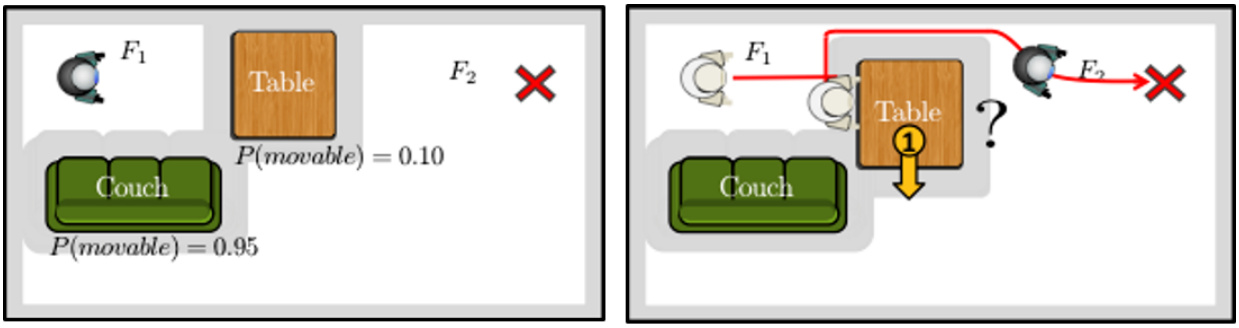
is the free space, with $A(q)$ the robot in configuration $q \in \mathscr{C}$.

$$\mathscr{C}_{obs} = \{q \in \mathscr{C} | A(q) \cap O \neq \emptyset\} = \mathscr{O}_{fixed} \cup \mathscr{O}_{movable} = \mathscr{O} \tag{2.3}$$

is the collision space, and

$$\mathscr{C} = \mathscr{C}_{free} \cup \mathscr{C}_{obs} \tag{2.4}$$

is the Configuration Space (see Figure 2.1).



(a) Configuration (b) Obtained Solution

Figure 2.2: Stilman NAMO planner.

According to this definition, the path τ is composed of a sequence $(\mathscr{W}^S, ..., \mathscr{W}^t, ..., \mathscr{W}^G)$ where

$$\mathscr{W}^t = (t, q_t) \in \mathscr{C}_{free} \tag{2.5}$$

is the non colliding world state at time t. $\mathscr{W}^S$ is the world state at the robot initial configuration, q_S, and the goal is to achieve a robot final configuration, q_G.

Definition 2 (Stilman [12]): A **Navigation Among Movable Obstacles** (NAMO) problem aims to find a path τ from a robot Start configuration, q_S, to a Goal configuration, q_G, by giving the robot the ability to manipulate obstacles if this can shorter the path or let the achievement of an otherwise unreachable q_G (see Figure 2.2):

$$\tau : [0, 1] \rightarrow C \;\; with \;\; \tau(0) = q_S, \tau(1) = q_G \tag{2.6}$$

Then, the path τ is composed by a sequence $(\mathscr{W}^S, ..., \mathscr{W}^t, ..., \mathscr{W}^G)$ where

$$\mathscr{W}^t = (t, q_t, o^t_{m_1}, ..., o^t_{m_j}, ..., o^t_{m_k}) \in \mathscr{C} \tag{2.7}$$

is the world state at time t. $\mathscr{W}^S$ depicts the world state at the robot initial configuration, q_S, and that of each movable obstacle $o^S_{m_j}$. The goal is to achieve a robot final configuration q_G.

In other words, an instance of the NAMO problem can be formally defined by the tuple $S = (\mathscr{C}_{movable}, \mathscr{U}, q_S, q_G, f)$ where:

- $\mathscr{C}_{movable} = \mathscr{C}_{free} \cup \mathscr{O}_{movable}$ is the allowed robot state space;
- $\mathscr{U}$ is the control space;
- q_S is the robot start state;
- q_G is the goal;
- $f : \mathscr{Q} \times \mathscr{U} \rightarrow \mathscr{T}g\mathscr{Q}$ is the forward routine describing the dynamics, where $\mathscr{T}g\mathscr{Q}$ is the tangent bundle of $\mathscr{Q}$.

A solution of the NAMO problem instance consists of a sequence of controls $u_1, ..., u_n \in \mathscr{U}$ and times $t_1, ..., t_n \in \mathbb{R}^{\geq 0}$ such that $q_0 = q_S$, $q_n = q_G$ and $q_k \in \mathscr{C}_{free} \cup \mathscr{O}_{movable}$, $k = 1, ..., n-1$, can be obtained sequentially by integrating f. This means that the motion plan can iterate navigation and objects manipulation routines until the robot reaches q_G.

2.3 Action Space

NAMO lets a robot combine two actions, *Navigation* and *Manipulation*, by iterating a two steps procedure. First, the robot moves to a contact state through a *Navigation* action, then it applies a *Manipulation* action in order to displace the object. The robot also uses *Navigation* to reach a Goal state. Taking into consideration concepts proposed by Stilman in [12], each action is parameterized by a path $\tau(q_i, q_j)$ that defines the motion of the robot between two configurations: $\tau[0,1] \rightarrow \mathscr{C}$ where $\tau[0] = q_i$ and $\tau[1] = q_j$. Details follow.

2.3.1 Navigation

Navigation means the robot's ability to plan a path towards some Goal locations while avoiding contacts with obstacles. While navigating, the robot does not have to displace any obstacle by collision or friction. It simply moves its joints as specified by τ:

$$Navigation : (\mathscr{W}^t, \tau(q_t, q_{t+1}) \rightarrow \mathscr{W}^{t+1} \tag{2.8}$$

2.3.2 Manipulation

Manipulation refers to the capability of the robot to modify its workspace by displacing objects. It includes both grasp and push routines. Depending on the routine, the robot approaches its end-effector to the object and searches for the appropriate end-effector/object contacts. Given an object to be grasped (or pushed) multiple contacts can be generated depending on the orientation of the end-effector with respect to the object and depending on the gripper joints configurations. Moreover, distinct grasps (or pushes) can lead to different object motions given the same end-effector trajectory. Let $\mathbf{G}(\mathscr{O}_i) = \{\mathscr{G}_i\}$ be the set of relative transformations between the robot end-effector and the object $\mathscr{O}_i$ that constitute a grasp contact. Let $\mathbf{P}(\mathscr{O}_i) = \{\mathscr{P}_i\}$ be the set of relative transformations between the robot end-effector and the object $\mathscr{O}_i$ that constitute a push contact. Then,

$$Manipulation : (\mathscr{W}^t, \mathscr{O}_i, \mathscr{P}_i \cup \mathscr{G}_i, \tau(q_t, q_{t+1}) \rightarrow \mathscr{W}^{t+1} \tag{2.9}$$

That means *Manipulation* maps a world state, contact, and path to a new world state where the robot and $\mathscr{O}_i$ have been displaced. The action is valid when neither the robot not object collide or displace other objects.

Every *Manipulation* translates the trajectory of the robot into a motion for the object. The motion depends on which the object is being grasped or pushed:

Grasp A grasped object remains at a fixed transform relative to the robot end-effector. In addition to requiring collision-free paths, a valid grasp constrains the initial state of the robot end effector by imposing some form closures. The observance of these constraints indicate that a desired motion of the robot will not cause it to release the object. It is also possible to constrain grasps with regard to robot force capabilities. Grasping allow to easily predict the possible displacement of an object by looking at robot and object geometry. However, some objects, such as large boxes, are difficult or impossible to be grasped. Moreover, restrictions can be imposed on robot poses and gripper motions. They can all make grasping impossible.

Push Pushes are non-prehensile manipulation actions that do not require grasps with closure. Given any contact between the robot and the object, pushing manipulation restricts the path that

the manipulator can follow in order to maintain a fixed transform between the end-effector and the object. The most studied version of pushing manipulation is based on static friction during pushing [31]. At sufficiently low velocities, static friction prevents the object from slipping with respect to the contact surface. Given the friction coefficient and the friction center of the object we can restrict robot paths to those that apply a force inside the friction cone for the object. Pushes are more general than grasps but they need more detailed modeling because they require the friction management. Incorrect pushes could bring to slips and unplanned behaviors.

2.4 Problem Complexity

Combining the choose of paths for *Navigation* and that of paths and contacts for *Manipulation* means that, when planning motions, robots have to take into consideration both the state of the environment and that of all movable obstacles. This constraint exponentially increases the dimensionality of the state space and, as stated by Wilfong[27], makes complete planning for full domain computationally infeasible. In fact, there is no method for identifying non-trivial manipulations. Any *Manipulation* changes the world state and the set of accessible configurations. Even if the change seems to decrease immediate accessibility, it is possible that *Manipulation* opens spaces for future displacement of other objects.

In order to reduce the problem complexity, the free space $\mathscr{C}_{free}$ can be partitioned into disjoint sets of robot configurations closed under *Navigation*: $\{\mathscr{C}_1, \mathscr{C}_2, ..., \mathscr{C}_n\}$, with n the number of disjoint free spaces among obstacles. Partitioning the free space results in an abstraction for identifying useful manipulation subgoals.

Definition 3 (Stilman[12]): A *keyhole*, $\mathscr{K}(\mathscr{W}^1, \mathscr{C}_i)$, is a subgoal problem in NAMO that specifies a start state, $\mathscr{W}^1$, and a component of free space, $\mathscr{C}_i$. The goal is to find a sequence of operators (*Navigations* and *Manipulations*) that results in $\mathscr{W}^2$ s.t. every free configuration in $\mathscr{C}_i$ is accessible to the robot:

$$\mathscr{C}_i \cap \mathscr{C}_{free}(\mathscr{W}^2) \subset \mathscr{C}_{acc}(\mathscr{W}^2) \quad and \quad \mathscr{C}_i \cap \mathscr{C}_{free}(\mathscr{W}^2) \neq \emptyset \tag{2.10}$$

To restrict the number of obstacles moved in solving a keyhole we define a *keyhole solution* according to the maximum number of permitted *Manipulations*.

Definition 4 (Stilman [12]): A *k-solution* to $\mathscr{K}(\mathscr{W}^1, \mathscr{C}_i)$ is a sequence of valid actions including at most k *Manipulations* that results in $\mathscr{W}^2$ satisfying Eq. 2.10.

In order to solve a NAMO problem, *a robot must resolve a sequence of keyholes until* $q_t \in \mathscr{C}_G$ *at some future time t.* It is therefore possible to define a class of problems where keyholes can be solved independently.

Definition 5 (Stilman [12]): A NAMO problem, $(\mathscr{W}^0, q_G)$, is *linear of degree k*, or L_k, if and only if:

1. There exists an ordered set of $n \geq 1$ distinct configuration space components, $\{\mathscr{C}_1, ..., \mathscr{C}_n\}$ such that $\mathscr{C}_n = \mathscr{C}_G$ and $\mathscr{C}_1 = \mathscr{C}_{acc}(\mathscr{W}_0)$;

2. For any i $(0 < i < n)$, any sequence of *k-solutions* to $\{\mathscr{K}(\mathscr{W}_{j-1}, \mathscr{C}_j) | j < i\}$ results in $\mathscr{W}_{i-1}$ such that there exists a *k-solution* to $\mathscr{K}(\mathscr{W}_{i-1}, \mathscr{C}_i)$;

3. If $n > 1$, any sequence of *k-solutions* to $\{\mathscr{K}(\mathscr{W}_{j-1}, \mathscr{C}_j) | j < n\}$ results in $\mathscr{W}_{n-1}$ such that there exists a *k-solution* to $\mathscr{K}(\mathscr{W}_{n-1}, \mathscr{C}_n)$ that results in $\mathscr{W}_n$ where $q_j \in \mathscr{C}_{free}(\mathscr{W}_n)$.

The definition ensures that once the robot has reached a configuration in $\mathscr{C}_i$ it can always access $\mathscr{C}_{i+1}$. The third condition guarantees the existence of a final keyhole solution such that the goal is in $\mathscr{C}_G \subset \mathscr{C}_{free}(\mathscr{W}^n)$.

2.5 Idea

This thesis focuses on NAMO problems that are *linear of degree 1*, or L_1, where disconnected components of free space can be connected independently by moving a single obstacle. The idea is to focus exploration on less explored areas. For this reason, extending KPIECE [17, 18] seams a good choice [II]. KPIECE is a Sampling-based Motion Planning algorithm, in the sense that it explores the state space of the robotic system by growing a tree of valid motions from the start state of the system towards a goal region, using a model of motion [32]. Its strength, as well as innovation, is focusing the exploration on less explored areas. It projects the state space to a lower dimensional Euclidean space and discretizes it by using a grid. The discretization is used to estimate the coverage of the state space and to evaluate cells goodness: for every cell, the algorithm saves the number of times it has been explored and the progress achieved by exploring it. Combining collected information, KPIECE is able to deterministically select the regions to explore: the best less explored ones (see Appendix A). The algorithm we want to propose should adopt the same exploration strategy but should not impose obstacle avoidance. The importance function assigned to each cell should combine efforts necessary to reach it and that needed to free it from obstacles.

Focusing on the structure of actual robots, Motion Planning should not longer ignore their complexity focusing on the resolution of basic geometric problems, such as the piano movers' problem [33], and hoping on controllers able to follow the paths by keeping velocities slow. It should also account for kinematic and dynamic constraints such as friction, gravity, limits in forces. Being KPIECE a tree planner that only depends on forward propagating the model of motion - numerically evaluating motions only forward in time - it is suitable for this scenario. With respect to other forward propagation algorithms, e.g. RRT [34], EST [35] , or SBL, KPIECE improves the decision phase by making better use of the information collected during the planning process. This information is used to decrease the amount of forward propagation the algorithm needs. As consequence, both runtime and memory requirements decrease making the algorithm suitable to handle high dimensional systems with complex dynamics.

The algorithm will focus on the resolution of L_1 NAMO problems in known environments. However, it will be able to deal uncertainties using sensor feedbacks. This feature will make it suitable for the resolution of NAMO problems in unknown environments.

2.6 Proposed Solution

2.6.1 Navigation

As for KPIECE, every $q \in \mathscr{C}$ is projected into an Euclidean space E through a projection $Proj$. If $p = Proj(q)$, then the coordinates of p in E will be:

$$Coord(p) = Coord((p_1, ..., p_k)) = (\lfloor \frac{p_1 - o_1}{d_1} \rfloor, ..., \lfloor \frac{p_k - o_k}{d_k} \rfloor) \tag{2.11}$$

Algorithm 1 NAMO

Input: $\mathscr{C}$: the state space, q_S: the start state; q_G: the goal state
Output: $Path$: a set of states minimizing the cost for reaching q_G

Project $\mathscr{C}$ into and Euclidean space E: $i = Proj(q_i)$ with $q_i \in \mathscr{C}$
Discretize E through a grid G
Create a tree data-structure T of nodes v_i: v_i refers to $q_i \in Cell(i)$
for each $Cell(i) \in T$ **do**
    $Cell(i).distance \leftarrow A^*(Cell(i), Cell(q_G))$
    $Cell(i).weight \leftarrow weight$
    $Cell(i).selection, Cell(i).visits \leftarrow 0$
end for
Let v_S and v_G be the q_S and q_G nodes respectively
T.pushBack(v_S)
$Cell(q_S).visits{+}{+}$
$current \leftarrow v_S$
while $current \neq v_G$ **do**
    $current \leftarrow$ SelectBestCurrentNode(T)
    $Cell(current).selection{+}{+}$
    $next \leftarrow$ SelectBestNextNode($current$)
    $Cell(next).visits{+}{+}$
    T.pushBack($next$)
end while
for each $v_i \in T$ **do**
    $Path$.pushBack(q_i)
end for
Let the robot perform $Path$

where $(p_1, ..., p_k)$ are the components of p, $(o_1, ..., o_k)$ is the origin of the Euclidean map, and $(d_1, ..., d_k)$ is its resolution.

E is discretized through a grid G of NxM cells of length d. Without loss of generality, d is chosen as the propagation step size of the expansion tree. This means that $d = d_1 = ... = d_k$ will be the resolution of the map.

$$Cell(p) = \{q \in Q | Coord(Proj(q) \in Cell(p)\} \tag{2.12}$$

defines, for every $p \in E$, the corresponding cell of G.

A tree data structure T is defined. Every vertex $v_i \in T$ refers to the cell $Cell(i) \in G$; v_i points to the state of Q projected into $Cell(i)$ and used for the propagation. The algorithm proceeds as described in Algorithm 1.

T is initialized with v_S referring to $q_S \in Cell(q_S)$. At every iteration, the importance of cells referring to the current node and to its neighbors are updated. The node of the tree referring to the most important cell is selected and a state of it is chosen for the expansion. The process iterates until T reaches $Cell(q_G)$.

The importance of $Cell(i)$ is defined as:

$$Importance(i) = \frac{1}{\frac{1 + Distance(i) + Weight(i)_{TOT}}{1 + Selection(i) + Visits(i)}} \tag{2.13}$$

where

- $Distance(i)$ is the effort done to cover the Euclidean distance separating $Cell(i)$ from $Cell(q_G)$. It reflects the A* search;

Algorithm 2 Select The Best Current Node

```
1: function SelectBestCurrentNode(T)
2:     current ← T.begin()
3:     max_importance ← GetImportance(Cell(current))
4:     best_current ← current
5:     while current ≠ T.end() do
6:         current ← T.next()
7:         importance ← GetImportance(Cell(current))
8:         if importance > max_importance then
9:             max_importance ← importance
10:            best_current ← current
11:        end if
12:    end while
13:    return best_current
14: end function
```

Algorithm 3 Select The Best Next Node

```
1: function SelectBestNextNode(n)
2:     best_next ← a random neighbour of n
3:     max_score ← 0
4:     while n has non-selected neighbours do
5:         next ← a random neighbour of n
6:         score ← GetImportance(Cell(next))
7:         if score > next_score then
8:             max_score ← score
9:             best_next ← next
10:        end if
11:    end while
12:    return best_next
13: end function
```

- $Selection(i)$ is the number of times that $Cell(i)$ was selected for expansion;
- $Visits(i)$ refers to the number of times that $Cell(i)$ was considered during the selection phase, namely it is the coverage of $Cell(i)$;
- $Weight(i)_{TOT}$ is the cost of the path to be performed in order to reach $Cell(i)$, i.e., the sum of weights of the cells that NAMO sampled as parents of $Cell(i)$.

$Weight(i)$ assigned to $Cell(i)$ is defined as follows:

$$Weight(i) = \alpha \cdot Reach(i) + \sum_{k} (\beta \cdot Move(k, i) + \gamma \cdot Return(i)) \tag{2.14}$$

where

- $Reach(i)$ is the effort done by the robot in order to reach q_i within $Cell(i)$ from the current state;
- $Move(k, i)$ is the effort required to remove the k-th obstacle from $Cell(i)$ and place it out of the Euclidean distance separating $Cell(q_S)$ and $Cell(q_G)$ ($0 \leq k \leq n$, with n the number of obstacles within $Cell(i)$);
- $Return(i)$ is the effort required to come back.

In order to homologue data, efforts are represented as time variables.

Note: Unknown regions are considered free. Unknown objects have a null weight. These assumptions encourage the exploration of unknown areas.

Importance can be computed in constant time since all the values it depends on can be made readily available. Once visited a cell, its coverage is updated. Once selected a cell from which continuing the expansion, its selection rate is incremented and a robot state within it is sampled. A chain of states $Path = (q_0, ..., q_k, ..., q_n)$ results, with $q_0 = q_S$ and $q_n = q_G$. The robot real-time performs the $Path$.

It is easy to observe that, as for KPIECE, NAMO prefers expanding from cells that are less covered rather than from cells that are well covered. Cells that have been selected for expansion fewer times are preferred over cells that have been selected many times. Moreover, NAMO gives priority to cells closer to the goal, i.e., less explored areas; and it prioritizes cells that carry the robot to make the least effort, combining navigation and manipulation efforts. Studies show that considering these heuristics in the selection of cells work well in practice [17]. The author of this thesis thinks that formulating these heuristics will facilitate the resolution of the NAMO problem in unknown environments.

2.6.2 Manipulation

Once detected the obstacle on the path, the robot has to decide which manipulation action to apply in order to move it. Generally, if the object is small enough (width or length less than the gripper maximum opening) and has a weight that observes the robot payload, the robot tries to grasp it, otherwise it proceeds with a push. Remember that unknown obstacles are considered weightless. For that objects, the robot evaluates only their size. Depending on the action, a different method has been implemented in order to geometrically compute the new object position (the re-estimation of the orientation is unnecessary). Every approach refers to the obstacles positions (current or next) as the positions of their CoM. Actions are generated for a generic mobile manipulator robot.

2.6.2.1 The relocation routine of graspable objects

Figure 2.3 depicts the new position generation routine of a graspable object. Starting from the manipulator $\backslash base\_link$, n positions are generated around the manipulator robot. Every position (x_i, y_i) $(0 \leq i < n)$ has the following coordinates:

$$x_i = r_i \cdot \sin\theta_i \tag{2.15}$$

$$y_i = r_i \cdot \cos\theta_i \tag{2.16}$$

where r_i is the radius:

$$r_i = \begin{cases} min & i = 0 \\ r_{i-1} + \Delta r & otherwise \\ max & i = n-1 \end{cases} \tag{2.17}$$

Δr is the radius resolution, min is the minimum reach of the robotics arm allowing the placement of the object out of the footprint polygon of the mobile base, and max is its maximum reach. θ_i is the direction:

$$\theta_i = \begin{cases} 0 & i = 0 \\ \theta_{i-1} + \Delta\theta & otherwise \\ 2\pi & i = n-1 \end{cases} \tag{2.18}$$

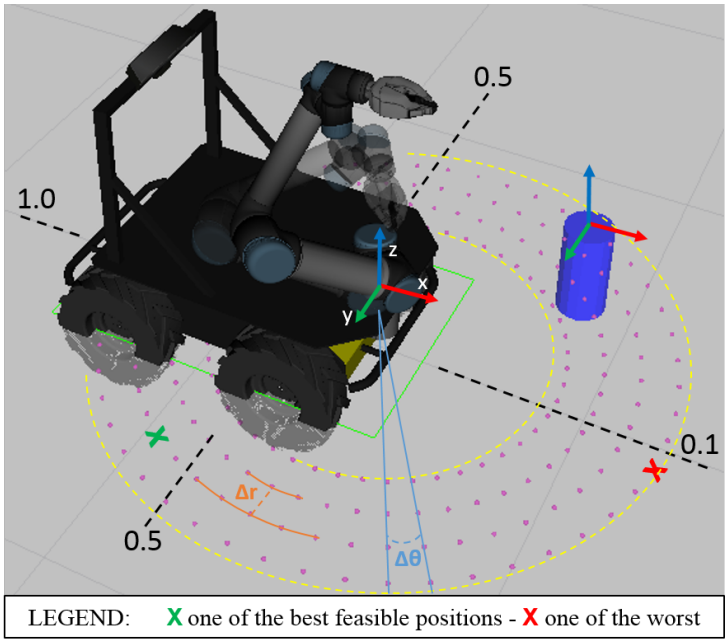


Figure 2.3: The positions generation routine of a graspable obstacle.

$\Delta\theta$ is the angle resolution.

In order to facilitate the new position selection, a weight w_i is assigned to every (x_i, y_i):

$$w_i = \begin{cases} 1 & (x_i, y_i) \textit{ behind the robot} \\ 0.1 & (x_i, y_i) \textit{ in front of the robot} \\ -\cos\theta & \textit{otherwise} \end{cases}$$

Positions are inserted in an ordered list L depending on w_i and d_i, the Euclidean distance between (x_i, y_i) and the manipulator $\backslash base\_link$:

$$L = [(x_1, y_1, w_1, d_1), ..., (x_i, y_i, w_i, d_i), ..., (x_j, y_j, w_j, d_j), ..., (x_n, y_n, w_n, d_n)] \tag{2.19}$$

with $w_1 > ... > w_i > ... > w_j > ... > w_n$. If $w_i = w_j$, than $d_i \leq d_j$.

Typically, human navigation routines prefer a forward motion instead of a backward one. Inspired from this behavior, the implemented positions generation process prefers to relocate encountered obstacles behind the robot. This decision should minimize the probability of reconsidering the object again while solving the NAMO problem.

If, during the displacement, no $(x_i, y_i) \in L$ is kinematically feasible and collision-free, than the constraint $r_i \leq max$ is relaxed allowing the motion of the mobile base. Depending on the relaxation, L is reformulated.

2.6.2.2 The relocation routine of pushable objects

Figure 2.4 depicts the positions generation process of a pushable object. A *Zig-Zag* mode is adopted: new positions are generating to the right or the left of the current one taking into account the space that the robot requires to move. (x_i, y_i) $(0 \leq i < n)$ has coordinates:

$$x_i = x_0 + i\Delta d \tag{2.20}$$

$$y_i = y_0 \pm l \tag{2.21}$$

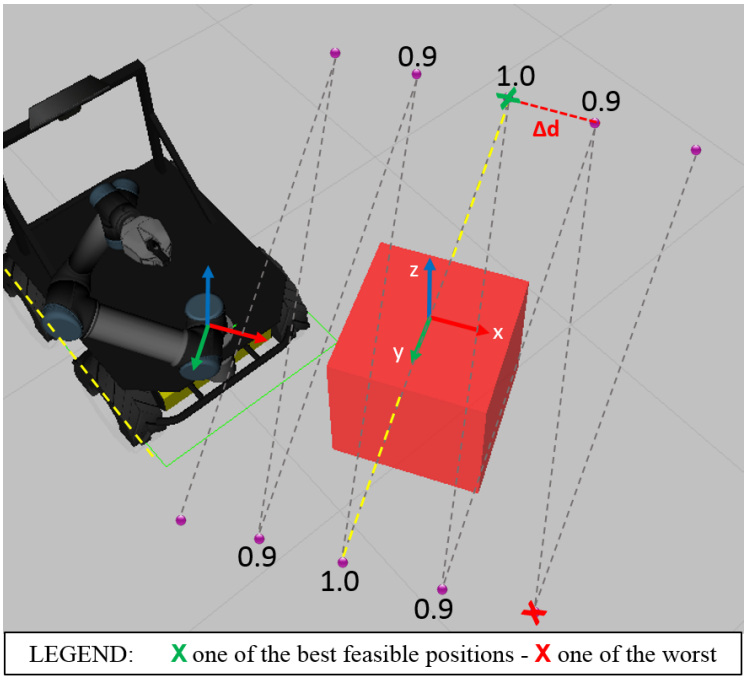


Figure 2.4: The positions generation process of a pushable obstacle.

where (x_0, y_0) is the current position of the object, Δd is the distance resolution, and l is the robot maximum side.

As for the *Grasp* routine, a weight w_i is assigned to every (x_i, y_i). The adopted rule follows:

$$w_i = \begin{cases} 1 & x_i = x_0 \\ \frac{1}{d_i} & otherwise \end{cases}$$

where $d_i = D((x_i, y_i), (x_0, y_0))$ is the Euclidean distance between the new and the current object position with respect to the object reference system. An ordered list

$$L = [(x_1, y_1, w_1), ..., (x_i, y_i, w_i, d_i), ..., (x_j, y_j, w_j, d_j), ..., (x_n, y_n, w_n)] \tag{2.22}$$

is formulated with $w_1 \geq ... \geq w_i \geq ... \geq w_j \geq ... \geq w_n$.

In case of failed displacement, the routine increases l. L is reformulated.

2.7 Implementation Details

2.7.1 Navigation

The implementation extends the ROS Indigo *Navigation* stack[i] and, in detail, its *costmap_2d* package.

costmap_2d discretizes the environment into a grid and builds a 2D map of the robot workspace by distinguishing the environment between free or obstructed. This information is used by the planner in order to compute a collision-free path between q_S and q_G. No distinction is made between obstacles in the scene: every object refers to a collision.

[i] http://wiki.ros.org/navigation

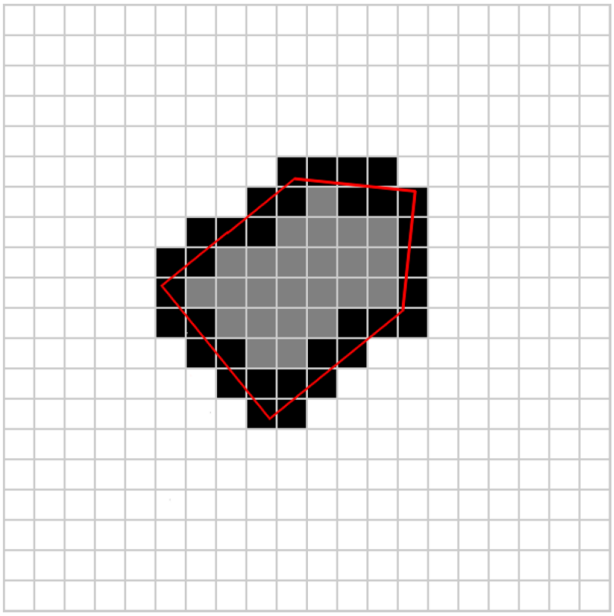


Figure 2.5: From Robot Footprint to Cells.

In our package, every 3D object is characterized by a displacement weight. Like for *costamap_2d*, the sole of every 3D object is mapped into a 2D polygon. Thanks to the ray-tracing algorithm of Bresenham [36], the polygon is projected on the grid: the algorithm first extracts cells corresponding to the footprint vertexes, then it determines the points of the 2D raster that approximates the straight lines between every two vertexes. This approach is the same used to draw line primitives in a bitmap image. Every point of every line is mapped into the corresponding cell. Once boundary cells are extracted, inner cells derived (see Figure 2.5).

If two objects are in the same position of the XL plane but have different Z coordinates, their soles will be projected onto the same cells. Different from *costamap_2d*, our implementation verifies the height from the ground at which every object is situated. If the height is such that the object does not influence the robot motion, then that object will not be taken into consideration during the building of the collision map. Moreover, each cell links the 3D objects that belong to it, this means that every cell has a weight equal to the sum of the weights of obstacles that belong to it. Only if the cell refers to free space, then it will have no weight.

Starting from this information, the planner assigns an importance function to each cell and a tree is built (see Section 2.6). In order to build the tree and understand which obstacles can constitute a collision and should be eventually displaced, the polygonal footprint of the robot is extracted and used as robot reference.

2.7.2 Manipulation

Once L has been computed, the displacement routine starts: it extracts the positions in L starting from the first and try to place there the object. The robot approaches its end-effector to the object and grasps (or pushes) it until it reaches the goal position. In this phase, the geometry of objects to be manipulated and of the gripper are known. For this reason, gripper joints configurations necessary to manipulate objects are pre coded through the *MoveIt! Simple Grasps* tool developed by Dave T. Coleman[ii]. *MoveIt!* is used to plan the motion. It provides a collision checking for

[ii]D. T. Coleman: MoveIt! Simple Grasps, available at https://github.com/davetcoleman/moveit_simple_grasps

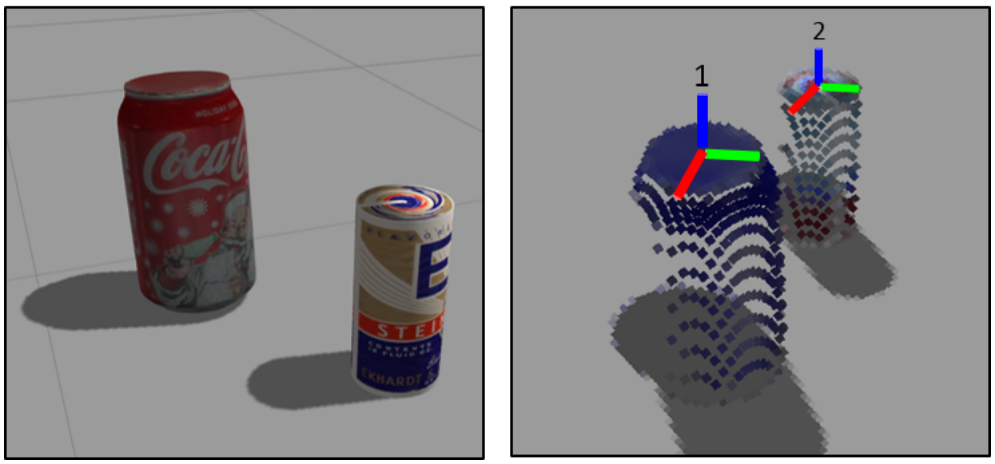


Figure 2.6: Objects in the scene and their segmentations. A marker is visible for every reference system. Objects are ordered depending on the distance from the robot.

the mobile robot, the robot arm, and the gripper during the motion from the current to the goal position.

2.7.3 Uncertainty

While moving in a workspace, robots may have to deal with unexpected events. Moreover, both object positions and robot movements may be subject to uncertainty. For these reasons, sensors should be mounted on every robot and every automaton should be able to correct its actions based on sensor feedbacks. In our case, the Point Cloud Library [19] and the ROS *rail_segmentation* package[iii] have been exploited in order to implement a routine able to read signals of a vision sensor and process them. *rail_segmantation* provides tabletop segmentation functionalities for handle objects given a Point Cloud (see Appendix C for further details): the robot detects the objects on ground, segments them, and extracts their coordinates (See Figure 2.6). This information is used to update the occupancy map and eventually recompute the NAMO path.

If the robot detects a new unknown object, its mass is considered negligible. If it detects new unknown spaces, they are considered free. These choices arise robot curiosity and led it to explore less explored areas.

2.8 Experiments

Experiments aim to prove the effectiveness of the proposed solution when dealing with complex dynamic systems, i.e. large robot state spaces. Tests started from a point-like robot: only if the algorithm is scalable for a simple system, it can be scalable when taking into consideration the dynamic of a complex robot. Then, we considered the projection of a robot footprint on a two-dimensional plane and tested the changes on time performances. At the end, a simulated robot was analyzed. We evaluated its correct behavior when moving among simulated movable obstacles and dealing with uncertainties.

All tests were performed on known environments.

[iii] http://wiki.ros.org/rail_segmentation

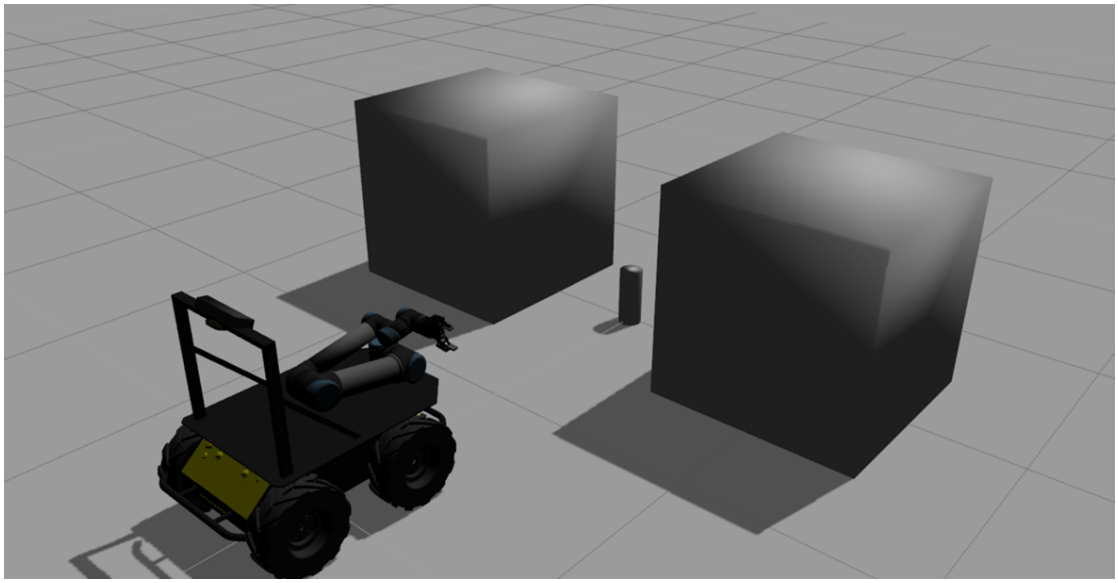


Figure 2.7: The robot in the scene.

Time performances refers to tests performed on a MacBook Pro equipped with an Intel Core i7 2.2 GHz quad-core processor and 16 GB DDR3 RAM memory.

2.8.1 Experiment 1: a Point-like Robot

The robot was reduced to a point reflecting its CoM. Two different C++ versions of the algorithm were compared: the proposed one and its *random* version. While our algorithm selects the leaf from which expanding the tree and its subsequent neighbor by always choosing the most important node, KPIECE selects the leaf as the most important existing one but the neighbor is randomly selected among states in the neighborhood. Comparing these two approaches lets proving the computational time improvement achieved by the proposed algorithm, as well as its good behavior with respect to the growth of the map size and of the number of obstacles.

Two different scenario were set up. In both setups, obstacles were polygons of random size, randomly placed on the map. For simplicity, three different weights [1, 3, 5] were considered. Weights were randomly assigned to obstacles.

Setup 1: Two types of maps were created: one with 70 obstacles and one with 100 obstacles. Every map was discretized into a 25x25 and a 50x50 grid. 100 different maps were generated depending on the size of the grid and on the number of obstacles. For each map, tests were repeated 100 times.

Setup 2: 1) Given a 50x50 grid, the number of obstacles populating it was varied from 10 to 250. 2) Given 70 obstacles, the size of the grid was changed from 20x20 to 100x100.

2.8.2 Experiment 2: a Footprint Robot

Considering the robot footprint instead of its CoM lets verify the accuracy of the algorithm in a simple but possible real scenario. Moreover, it aims to confirm the effectiveness of the proposed solution while considering the robot occupancy.

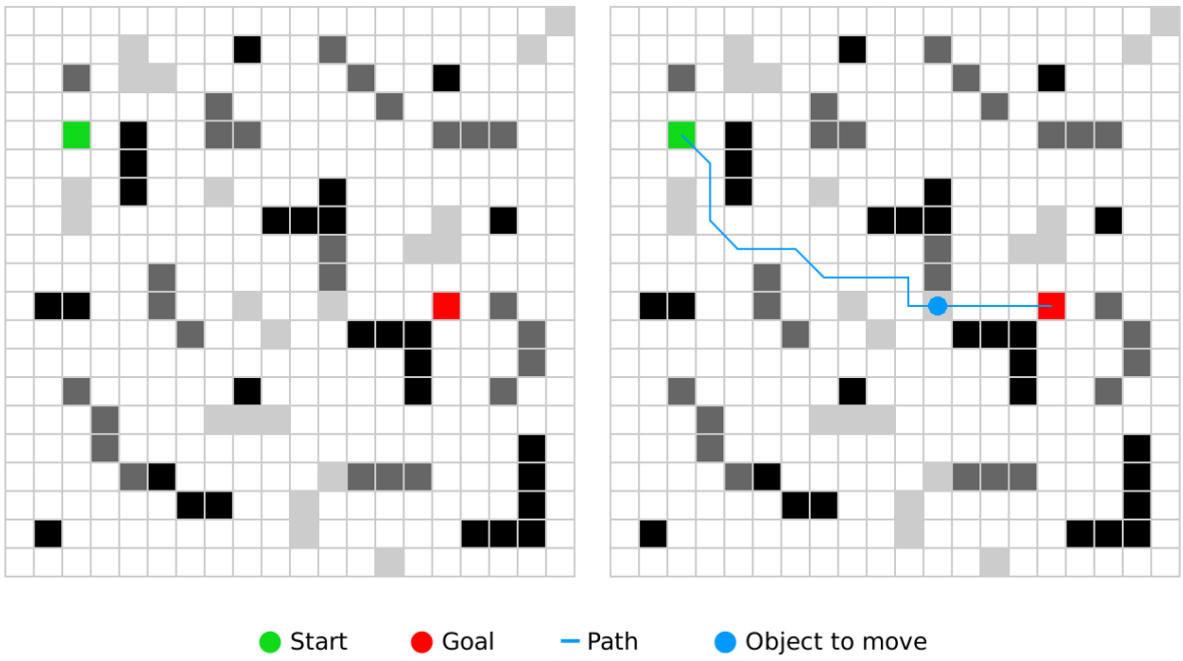


Figure 2.8: Experiment 1: A point-like robot moving on a map. One obstacle is selected for the relocation routine.

Obstacles	25x25		50x50	
	determ.	random	determ.	random
70	0.265	1.021	0.389	3.742
100	0.257	1.350	0.380	3.590

Table 2.1: Mean time of 10000 executions (time expressed in ms).

Obstacles	25x25		50x50	
	determ.	random	determ.	random
70	1.056	1.278	1.000	1.333
100	1.126	1.280	1.000	1.306

Table 2.2: Mean obstacles weight of 10000 executions.

A map populated by three obstacles was generated. Obstacles were placed in front of the robot in order to form a barrier. As depicted in Figure 2.11, the barrier was composed of two heavy obstacles (back rectangles) and an obstacle observing the robot payload (the gray rectangle). The robot had to reach a position beyond the barrier while deciding it navigating in the free space or moving some obstacle. Two footprint were considered: one larger than the other. For every type of footprint, experiments were repeated 10 times.

2.8.3 Experiment 3: a Simulated Robot

Gazebo was used as simulator. An Husky mobile robot equipped with an UR5 robot manipulator, a Robotiq 2-finger gripper, and a Microsoft Kinect vision sensor was in the scene (see Figure 2.7). The robot had to reach a position behind the barrier.

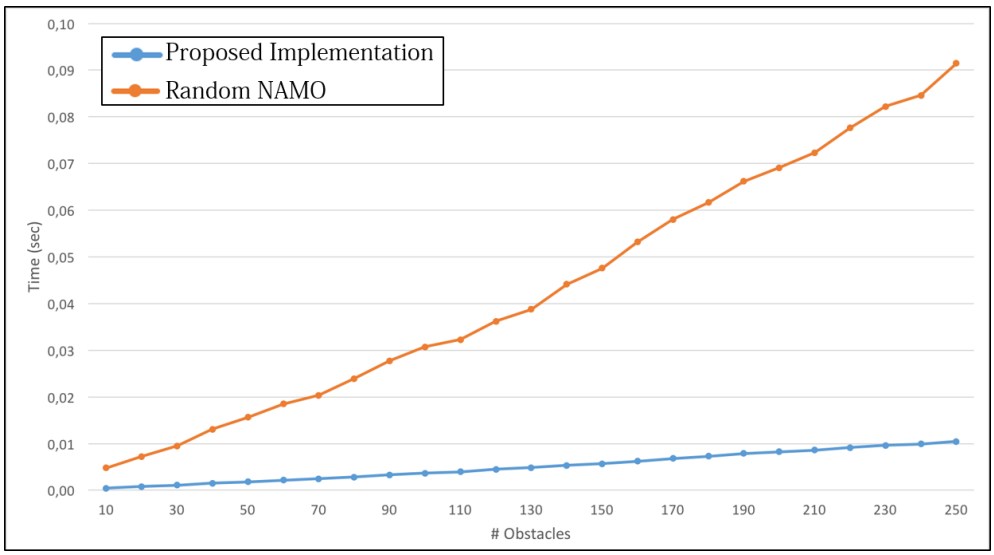


Figure 2.9: Computational time with respect to the number of obstacles populating the grid.

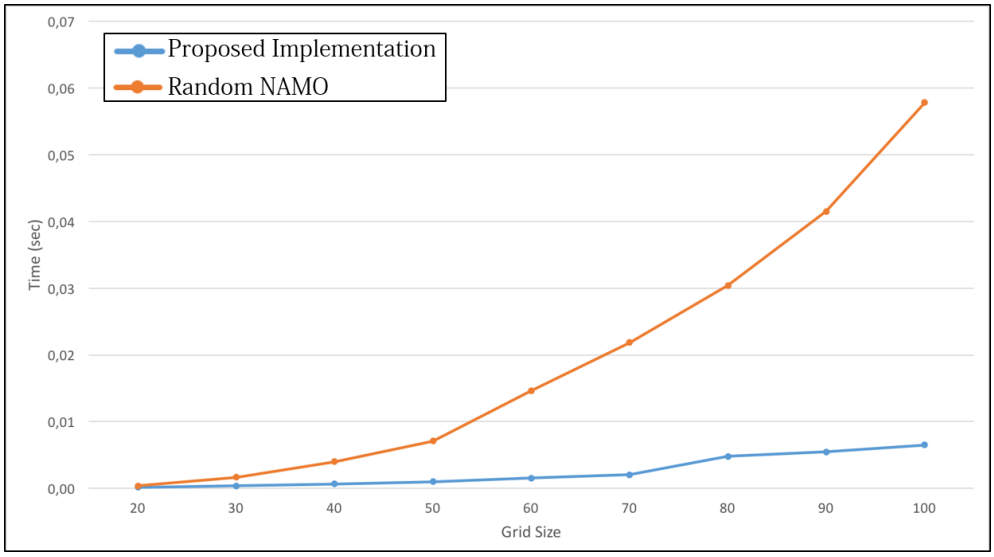


Figure 2.10: Computational time with respect to the size of the map.

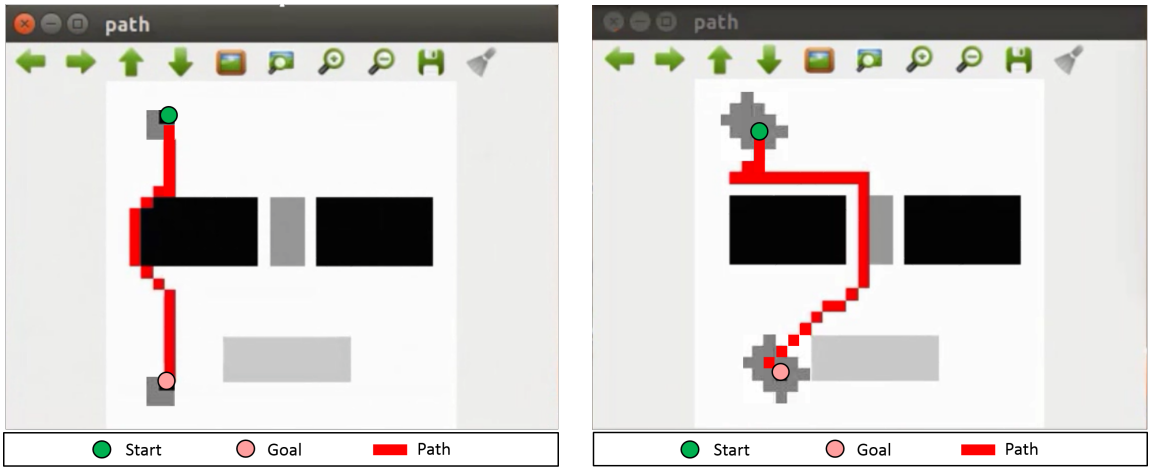
2.9 Results

2.9.1 Experiment 1

Figure 2.8 depicts a point-like robot performing a path from a Start to a Goal point while adopting the proposed methodology. Is is noted that the Goal was not occluded. The robot could freely choose if navigating along the free space or moving obstacles. It had to evaluate if moving obstacles could shorter the path. At the end, one obstacle was selected for the relocation routine.

Table 2.1 proves that the elaboration time is not influenced by the number of obstacles. The implemented version is faster than the *random* one and less affected by the growth of the grid size.

Table 2.2 shows that, on average, only light obstacles are chosen. The implemented version selects less obstacles than the *random* one. The choice shows the ability of the robot to select those objects whose displacement requires less effort.



(a) 2x2 pixels Footprint (b) 3x4 pixels Footprint

Figure 2.11: Experiment 2: Robot Fooprint

Table 2.3: Results of 10 attempts. Behaviors of a 2x2 and a 3x4 pixels robot footprints are compared (time expressed in seconds).

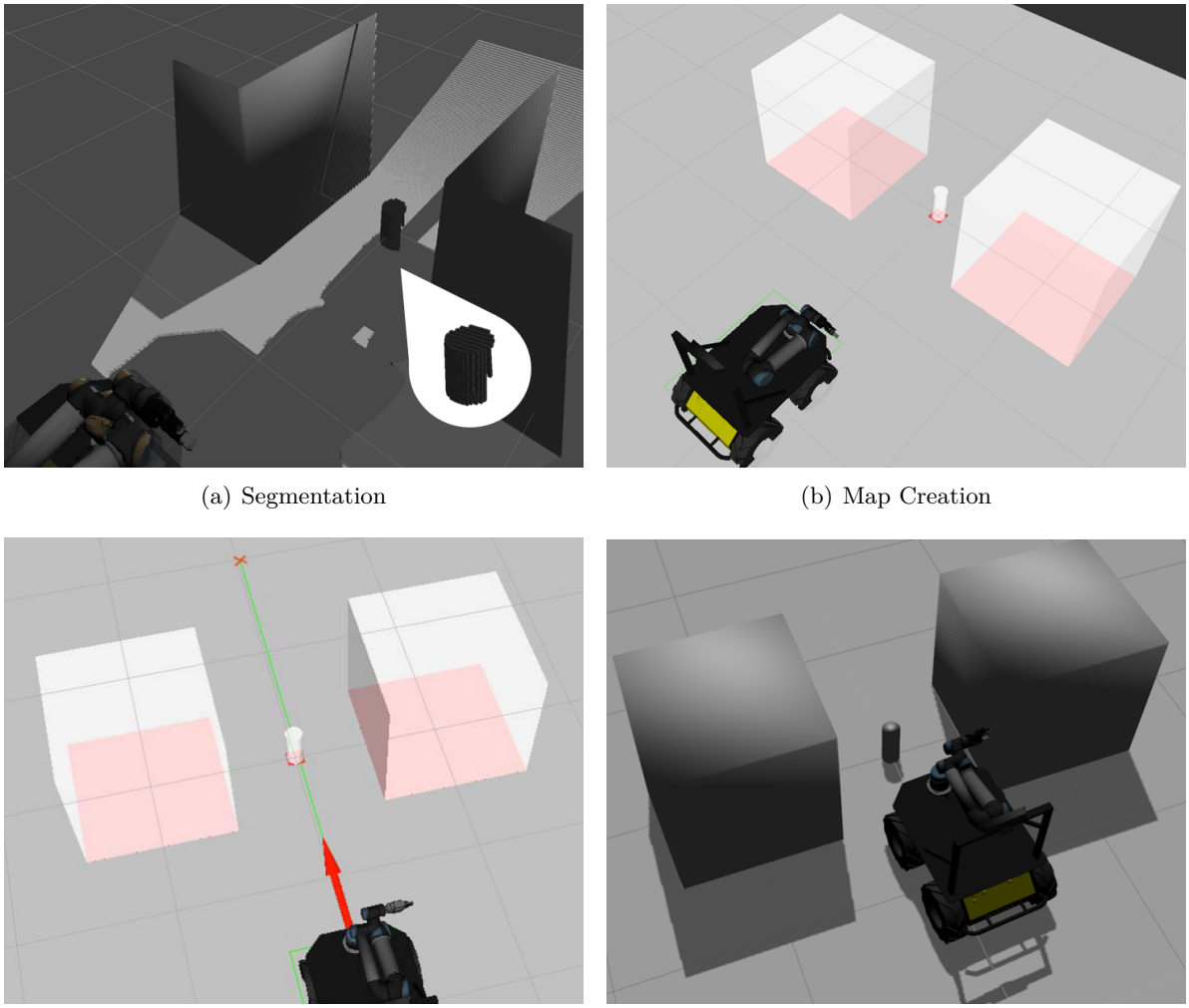
	2x2 Robot		3x4 Robot	
	Time	Removed	Time	Removed
	0.0258952	Y	0.0267104	Y
	0.0206904	Y	0.0236292	Y
	0.0213171	N	0.0252328	Y
	0.0217194	N	0.0280514	Y
	0.0199351	N	0.0311342	Y
	0.0200567	Y	0.0230287	Y
	0.0182324	Y	0.0226662	Y
	0.0168608	N	0.0261195	Y
	0.0209737	N	0.0286647	Y
	0.0203648	N	0.0267853	Y
Mean	0.0206046	40%	0.0262022	100%

In both cases, the execution time increases linearly with the growth of the number of obstacles but, as depicted in Figure 2.9, the *random* execution time increases faster than the other one. Figure 2.10 shows that the *random* algorithm time increases exponentially with respect to the change of the grid size. The one of the implemented algorithm, instead, increases more slowly and linearly.

Without loss of generality, we can state that the algorithm is scalable with respect to the size of the map and the number, weight, size and location of obstacles. Please note that weights are randomly assigned to obstacles and obstacles are randomly located on the map.

2.9.2 Experiment 2

Figure 2.11 includes two screen shots of the Debug phase of the experiments. Every square of the map has dimension equal to 1 pixel. Figure 2.11(a) depicts a 2x2 pixels robot footprint; Figure 2.11(b), instead, a 3x4 pixels one. Black rectangles refers to heavy obstacles, the gray ones depicts obstacles observing the robot payload. Screen shots show that the first robot easily circumnavigates the barrier; the second robot, instead, moves the lightest object. Table 2.3 details the results. On



(a) Segmentation (b) Map Creation

(c) Path Planning (d) Robot Motion

Figure 2.12: Experiment 3: The simulated routine.

10 attempts, the 2x2 pixels robot removes the lightest obstacle in 40 percent of the cases, for an average computational time of 20 ms. The 3x4 pixel robot, instead, decides to remove the lightest obstacle in 100 percent of the cases. The computational time is higher, about 0.26 ms, but does not influence the scalability of the system.

2.9.3 Experiment 3

Footprints of known obstacles were computed and projected on the ground. Every polygon was remapped into the cells it occupied. At the same time, the vision sensor detected the scene and *rail_segmentation* segmented the objects. The correspondence between the known scene and the detected one was tested, if some differences were found, the map was updated. The planner started. The robot had to decide if circumnavigating the barrier or cross it. At the end, it decided to cross the barrier and remove the lightest object (see Figure 2.12). It is noted that the mass of every known object is known a priori, as well as the robot average velocity. This means that efforts required by the robot to remove obstacles are proportional to the time required to place obstacles on the new computed positions. If an obstacle is new, its mass is imposed as negligible in order to arise curiosity.

2.10 Discussion

Experiments prove that our algorithm addresses a primary NAMO challenge: scalability. Its computational time makes it scalable with respect to the size of the map and the number, shape, and pose of obstacles. The algorithm does not impose restrictions on actions to be performed: the robot can both push and grasp every object. It works on a two-dimensional projection of a three-dimensional cluttered workspace letting consider both objects in contact and not in contact with the ground. It assumes full world knowledge but the environment is reconfigurable and the algorithm can be easily extended in order to solve NAMO problems in unknown environments. In fact, it is able to handle sensor feedbacks and correct uncertainties regarding obstacle poses and robot actions.

During trials, objects were known as well as actions necessary to manipulate them. The need of making robots Cognitive during the manipulation was born. Robots should be able to manipulate objects seen for the first time without the need of pre calculated poses.

3

Manipulation

This Chapter proposes some Cognitive Manipulation techniques.

Contents

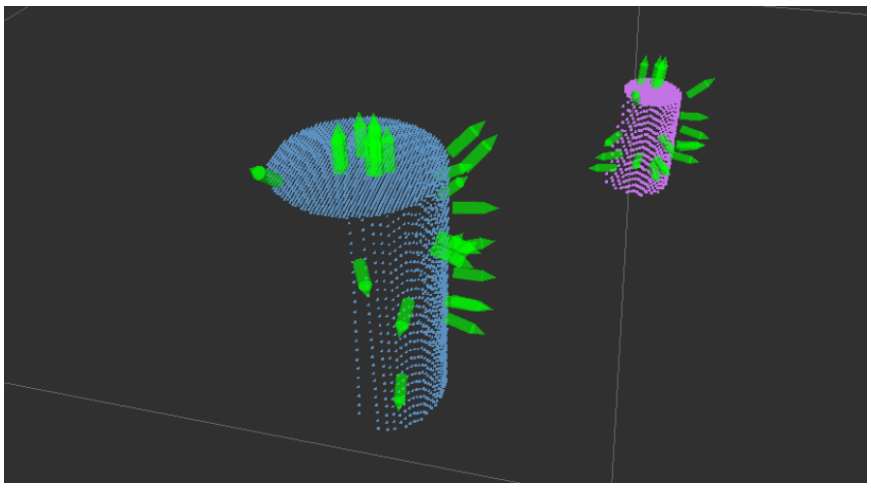


Figure 3.1: The AGILE output.

Once selected objects to be manipulated, the robot has to be able to manipulate them. Pre coding gripper poses for all existing objects and grippers is onerous or even impossible. Moreover, it does not make robots Intelligent.

This Chapter proposes some manipulation methods that make robots Cognitive: a RL technique and a RLfD framework. They make robots learn from their actions an from actions performed by other agents. These methods are not the main Research of the author but the author collaborated for their implementation. The RL technique was proposed by Gatto in his Master thesis [37]. RLfD is the main research of Michieletto [38]. The author collaborated for the development of a RLfD framework for manipulator robots, below summarized.

3.1 Reinforcement Learning

3.1.1 Overview

As stated in [39], RL enables a robot to autonomously discover an optional behavior through trial-and-error interactions with its environment. Instead of explicitly detailing the solution of a problem, the designer provides feedback in terms of a scalar objective function that measures the one-step performance of the robot. The *state* of the system is available, s, as well as the set of *actions*, a. A function that generates the movements necessary to accomplish that actions would be called the *policy*. A RL problem is to find a policy that optimizes the long term sum of rewards $R(s, a)$; a RL algorithm is the one designed to find such a (near)-optimal policy.

Gatto proposed a RL algorithm for the resolution of grasp problems for robots equipped with 2-(parallel)-fingers grippers. Rewards are given by humans evaluating the goodness of grasps (e.g., finger poses on the object, object stable grasped).

3.1.2 The Framework

States and actions are defined. States represent objects to be grasped, actions are grasps. In detail, after objects in the scene are segmented, SIFT extracts their features. Since SIFT features are large vectors, PCA is applied to reduce the vectors size. 5 components vectors result. They define objects states. Actions, instead, are defined as shown in Figure 3.2. The main idea is that

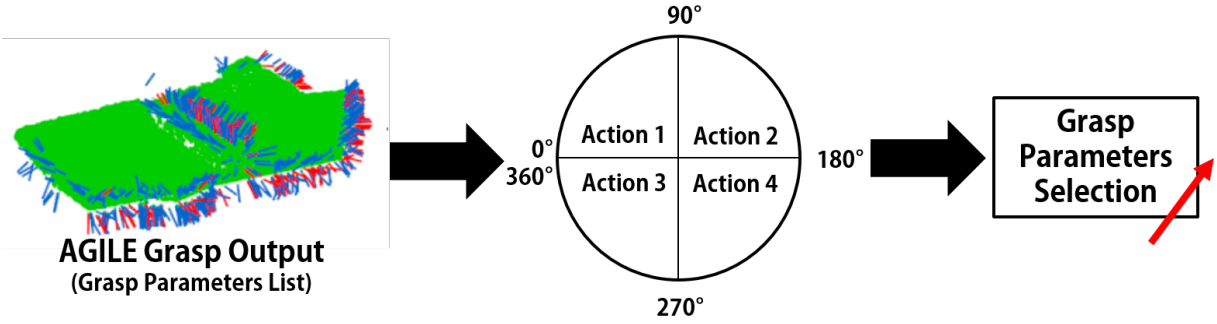


Figure 3.2: Action definition for the grasping task.

the relationship between actions and states should be simple. For this reason, the proposed RL algorithm calls the AGILE grasp algorithm. AGILE [40] detects grasps poses of a 2-fingers gripper directly from the geometrical features of the point cloud: it search for antipodal points inside the point cloud itself. An example of the AGILE output from a virtual Kinect is shown in Figure 3.1: Given the object to be grasped, it returns a list of possible planar grasps, (x, y, θ), with (x, y) the position of the grasp and θ its orientation. Instead of assigning a single action to every (x, y, θ) of the grasps list, the implemented RL algorithm works a follows:

- Only 4 actions are defined (see Figure 3.2). An action identify a grasp if only if its θ value lies in one of the orientation ranges;
- 4 ranges are defined: $0° - 90°$, $90° - 180°$, $180° - 270°$, and $270° - 0/360°$.

The expected result is that the algorithm will choose the action (the planar grasp) that has a high probability to be successful. Once a tuple of (x, y, θ) parameters is selected, the robot executes the action. A human operators assigned a reward evaluating the final result of the grasp. The RL engine waits for that feedback in order to understand if that grasp is the correct one or if another action should be executed in order to improve the obtained result.

3.2 Robot Learning from Demonstration

3.2.1 Overview

A RLfD framework lets a human demonstrator teach a robot how to accomplish a task: the robot will learn that task by imitating the teacher. Three processes are identified: sensing, understanding, and doing. They can be redefined as: observing the action, representing the action, and reproducing it.

There exists several modalities to collect data in order to control a robot by demonstration, e.g. motion sensors or kinesthetic teaching. Motion sensors must be worn by the teacher, they might be uncomfortable and prevent the human to act in a natural way. Kinesthetic demonstration, instead, involves robot motion guided by a human performer. The here presented RLfD framework uses a vision system. Vision sensors let achieve a sufficient accuracy while maintaining the motion naturalness and the comfort of the human actor performing the task.

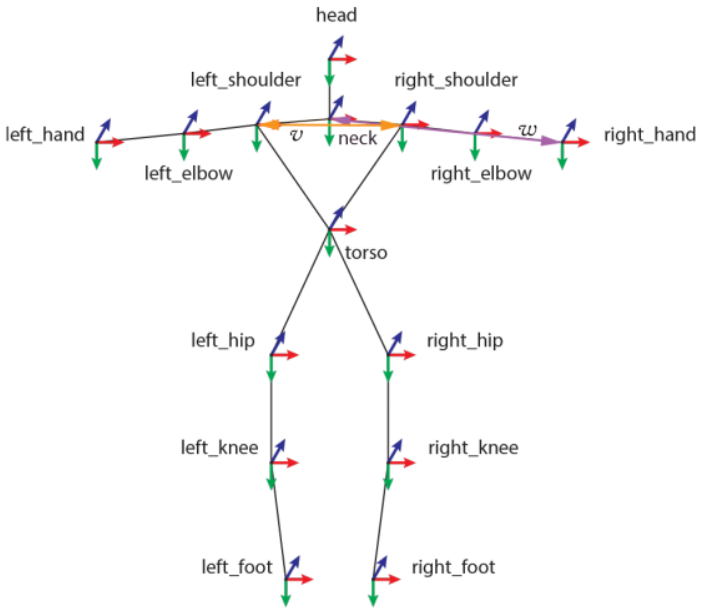


Figure 3.3: The skeleton joints provided by the tracker.

3.2.2 The Framework

A human demonstrator performs the task and an RGB-D sensor acquires the scene (human in action). A skeleton tracking algorithm extracts the useful information from the acquired images (positions and orientations of skeleton joints) and this information is given as input to the motion re-targeting system that remaps the skeleton joints into the manipulator ones. After the remapping, the collected data are used to train a Gaussian Mixture Model (GMM)[41] that retrieves a generalized motion trajectory by applying a Gaussian Mixture Regression (GMR). Two types of controllers are modeled: a position controller and a velocity one. The former has already been presented in [42]. The robot motion controller interprets the generated trajectory to make the manipulator fulfill movements similar to the human ones.

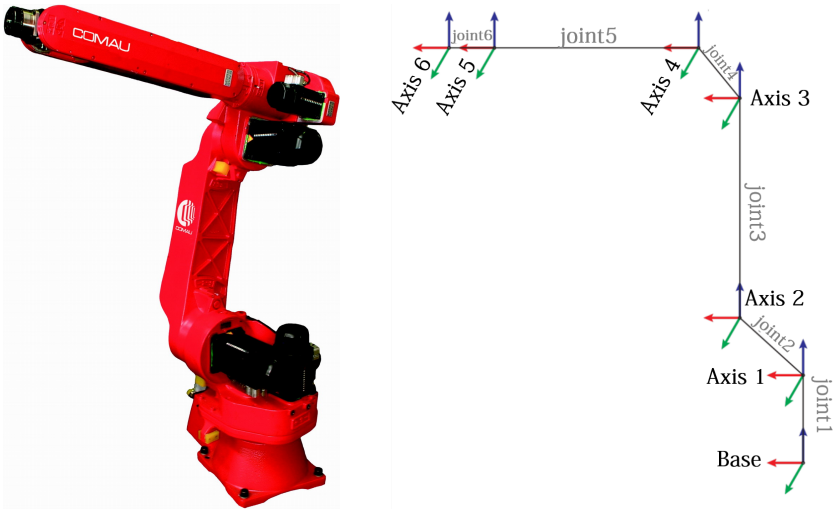
This Section focuses on the re-targeting of human joints into the first three robot joints. This lets the robot to perform only simple push actions. Bisson completed the job by re-targeting also the final three joints, those of the robot end-effector [43]. In this way, humans can teach the robot more complex actions. The same method can be adopted to remap human hands and teach the robot to grasp.

3.2.2.1 Data Acquisition

An RGB-D sensor (characterized by a frame rate of 30 fps and a resolution of 640x480 pixels) extracts the teacher's movements. A skeleton tracking algorithm (with a frame rate of 30 fps) educes the skeleton joints positions and orientations and stores them (see Figure 3.3).

3.2.2.2 Retargeting

The implemented retargeting system gives the teacher the possibility to control the manipulator using his all body. This is a novelty with respect to existing retargeting techniques. Most of them allows to caputre and remap only the motion of the arm of the actor, omitting his lower-body.



(a) Real robot (b) Robot specifications

Figure 3.4: Robot Comau Smart5 SiX Specifications.

The Euler angles of the 15 skeleton joints are extracted and remapped into the first 3 robot joints (*Axis1*, *Axis2*, *Axis3* of Figure 3.4):

- The *Axis1* rotation angle α is the projection P onto the XY plane of the robot coordinate system of the vector **v**(neck,right_hand) starting from the neck skeleton joint and ending in the right hand skeleton joint (see Figure 3.3):

$$\alpha = \begin{cases} \arccos(\frac{x_v}{\mathscr{P}_{XY}(v)}) & if \;\; y_v \geq 0 \\ -\arccos(\frac{x_v}{\mathscr{P}_{XY}(v)}) & otherwise \end{cases} \tag{3.1}$$

  where x_v and y_v are the x and y components of the considered vector v.

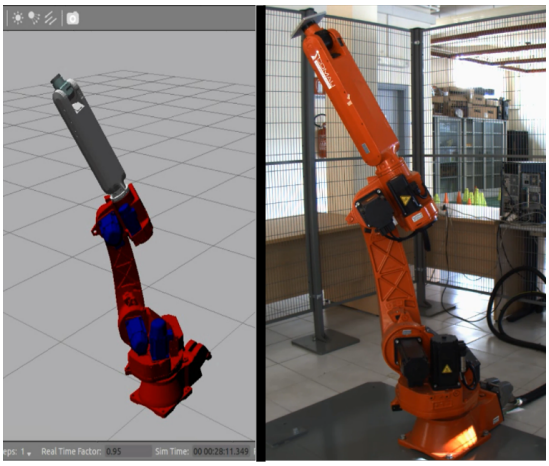
- The rotation β around the *Axis2* is referred to the relative motion of the demonstrator with respect to his initial position. In order to be robust to the sensor noise we considered the position of the segment **w**(left_shoulder, right_shoulder) from left shoulder skeleton joint to right shoulder skeleton joint:

$$\beta = \frac{\pi}{2} - \arccos \frac{\mathscr{P}_X(M_R^{-1} * d_w)}{l_{Axis2}} \tag{3.2}$$

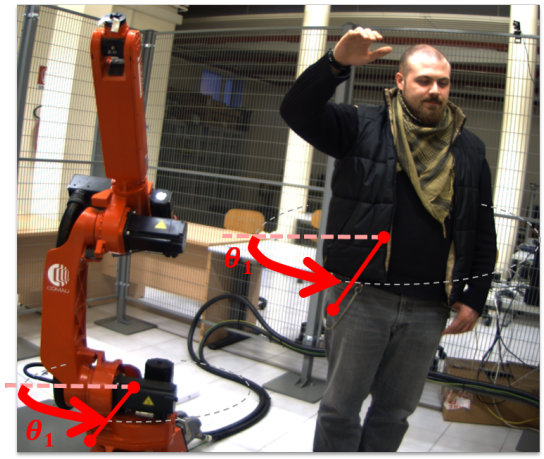
$$M = \begin{bmatrix} \frac{y_w}{\|w\|} & \frac{x_w}{\|w\|} & 0 \\ -\frac{x_w}{\|w\|} & \frac{y_w}{\|w\|} & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

  where

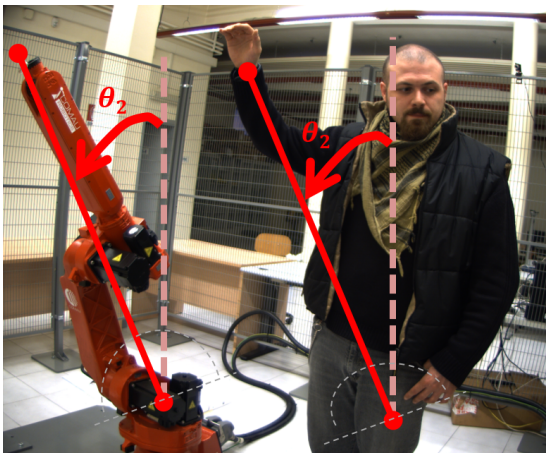
  - M_R R is the rotation matrix between initial and current w segment;
  - d_w is the distance between initial and current position of the w mean point $p_m \in w$;
  - $\mathscr{P}_X$ denotes the projection onto X;
  - l_{Axis2} is the length of the robot part involved in the motion.



(a) The System to be controlled: Real and Simulated Comau Smart5 Six Robot.



(b) Axis 1 Controller



(c) Axis 2 Controller

(d) Axis 3 Controller

Figure 3.5: System Controllers

- The rotation of *Axis3* represents the motion of the human arm remapped through a combination of both **v** and **w** (see Figure 3.5(c)). The obtained movement is similar to a composition of a translation and a rotation of the actor.

  The *Axis3* rotation γ really represents the human arm and it depends from both v and w:

$$\gamma = \begin{cases} \beta - \frac{\pi}{2} - \arccos \frac{\sqrt{x_v^2+y_v^2}}{\|v\|} & if \;\; z_v \geq 0 \\ \beta - \frac{\pi}{2} + \arccos \frac{\sqrt{x_v^2+y_v^2}}{\|v\|} & otherwise \end{cases} \tag{3.4}$$

3.2.2.3 Motion Controllers

Motion controllers have been defined by generalizing the forms of motion, positions $\hat{\xi} = \{\xi_s, \hat{\xi}_s\}$ and velocities $\hat{\dot{\xi}} = \{\dot{\xi}_s, \hat{\dot{\xi}}_s\}$ respectively. Starting from stored demonstrated movements, the generalization keeps in memory only the means and the covariance matrices of Gaussian components calculated through the Gaussian Mixture Model of acquired human joints positions and velocities. A detailed description follows.

3.2.2.3.1 Position Controller The position controller has been modeled using a GMM. The GMM input dataset consists of a sequence of robot joint angles $\xi_s \in \mathbb{R}^{D-1}$ recorded at a specific temporal instant $\xi_t \in \mathbb{R}$. D is the dimensionality of the problem, e.i. the number of DoFs of the robot. Equation 3.5 defines the input dataset and Equation 3.6 the resulting probability density function:

$$\xi = \{\xi_j\}_{j=1,\dots,N}, \xi_j \in \mathbb{R}^D \tag{3.5}$$

$$p(\xi) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\xi_j; \mu_k, \Sigma_k) \tag{3.6}$$

π_k is the prior probability and $\mathcal{N}(\xi_j; \mu_k, \Sigma_k)$ is the k-th Gaussian distribution. It is characterized by a mean μ_k and a covariance matrix Σ_k. The optimal mixture parameters (π_k, μ_k, Σ_k) are estimated using the Expectation Maximization (EM) algorithm [41] and the optimal number K of Gaussian components in the mixture is calculated through the Bayesian Information Criterion (BIC) [41]. Equation 3.7 describes the score assigned at each k.

$$S_{BIC} = -\mathscr{L} + \frac{n_p}{2} logN \tag{3.7}$$

where $\mathscr{L} = \sum_{j=1}^{N} log(p(\xi_j))$ is the model log-likelihood (it measures how well the model fits the data) and $n_p = (K-1) + K(D + \frac{1}{2}D(D+1))$ is the number of free parameters required for a mixture of K components with full covariance matrix. It aims to maintain the generality of the model while avoiding data overfitting. The last refinement is obtained by applying a Gaussian Mixture Regression that lets the retrieval of a smooth generalized version of the signal encoded in the associated Gaussian Mixture Model (GMM). The conditional expectation of robot joint angles $\hat{\xi_s}$ is calculated starting from the known a priori temporal value ξ_t. A Gaussian component k is defined by (π_k, μ_k, Σ_k), with:

$$\mu_k = \{\mu_{t,k}\mu_{s,k}\} \tag{3.8}$$

$$\Sigma_k = \begin{bmatrix} \Sigma_{t,k} & \Sigma_{ts,k} \\ \Sigma_{st,k} & \Sigma_{s,k} \end{bmatrix} \tag{3.9}$$

Then, the conditional expectation and its covariance can be estimated using Equations 3.10 and 3.11, respectively:

$$\hat{\xi}_s = E[\xi_s|\xi_t] = \sum_{k=1}^{K} \beta_k \hat{\xi}_{s,k} \tag{3.10}$$

$$\hat{\Sigma}_s = Cov[\xi_s|\xi_t] = \sum_{k=1}^{K} \beta_k^2 \hat{\Sigma}_{s,k} \tag{3.11}$$

where $\beta_k = \frac{\pi_k \mathcal{N}(\xi_t|\mu_{t,k}, \Sigma_{t,k})}{\sum_{j=1}^{K} \mathcal{N}(\xi_t|\mu_{t,j}, \Sigma_{t,j})}$ is the weight of the k-th Gaussian component through the mixture, $\hat{\xi}_{s,k} = E[\xi_{s,k}|\xi_t] = \mu_{s,k} + \Sigma_{st,k}(\Sigma_{t,k})^{-1}(\xi_t - \mu_{t,k})$ is the conditional expectation of $\xi_{s,k}$ given ξ_t, and $\hat{\Sigma}_{s,k} = Cov[\xi_{s,k}|\xi_t] = \Sigma_{s,k} + \Sigma_{st,k}(\Sigma_{t,k})^{-1}\Sigma_{ts,k}$ is the conditional covariance of $\xi_{s,k}$ given ξ_t.

3.2.2.3.2 Velocity Controller Through a similar approach, a GMM is created to model a velocity control. Each demonstration in input to the EM algorithm consists of a sequence of joint velocities $\dot{\xi}_s \in \mathbb{R}^{D-1}$. Equation 3.12 represents the input dataset, while Equation 3.13 is the probability density function:

$$\dot{\xi} = \{\dot{\xi}_j\}_{j=1,\dots,N}, \dot{\xi}_j \in \mathbb{R}^D \tag{3.12}$$

$$p(\dot{\xi}) = \sum_{k=1}^{K} \dot{\pi}_k \mathcal{N}(\dot{\xi}_j; \dot{\mu}_k, \dot{\Sigma}_k) \tag{3.13}$$

where $\dot{\pi}_k$ is the prior probability and $\mathcal{N}(\dot{\xi}_j; \mu_k, \Sigma_k)$ is the k-th Gaussian distribution modeling the input joint velocities. It is characterized by a mean $\dot{\mu}_k$ and a covariance matrix $\dot{\Sigma}_k$. Again, the number K of Gaussians composing the mixture is given by the Bayesian Information Criterion (BIC) (Equation 3.7). The GMR calculates the conditional expectation of the joint velocities $\hat{\dot{\xi}}_s$ and its covariance matrix $\hat{\dot{\Sigma}}_s$, as reported in Equation 3.14 and Equation 3.15.

$$\hat{\dot{\xi}}_s = E[\dot{\xi}_s|\xi_s] = \sum_{k=1}^{K} \dot{\beta}_k \hat{\dot{\xi}}_{s,k} \tag{3.14}$$

$$\hat{\dot{\Sigma}}_s = Cov[\dot{\xi}_s|\xi_s] = \sum_{k=1}^{K} \dot{\beta}_k^2 \hat{\dot{\Sigma}}_{s,k} \tag{3.15}$$

where $\dot{\beta}_k$ is the weight of the k-th Gaussian component through the mixture, $\hat{\dot{\xi}}_{s,k}$ is the conditional expectation of $\dot{\xi}_{s,k}$ given ξ_s, and $\hat{\dot{\Sigma}}_{s,k}$ is the conditional covariance of $\dot{\xi}_{s,k}$ given ξ_s.

3.3 Discussion

This Chapter proposed some Manipulation routines able to make robots Cognitive: a RL technique and a RLfD framework. Both routines make robots able to interact with humans, learn from their experiences. and from that performed by other agents.

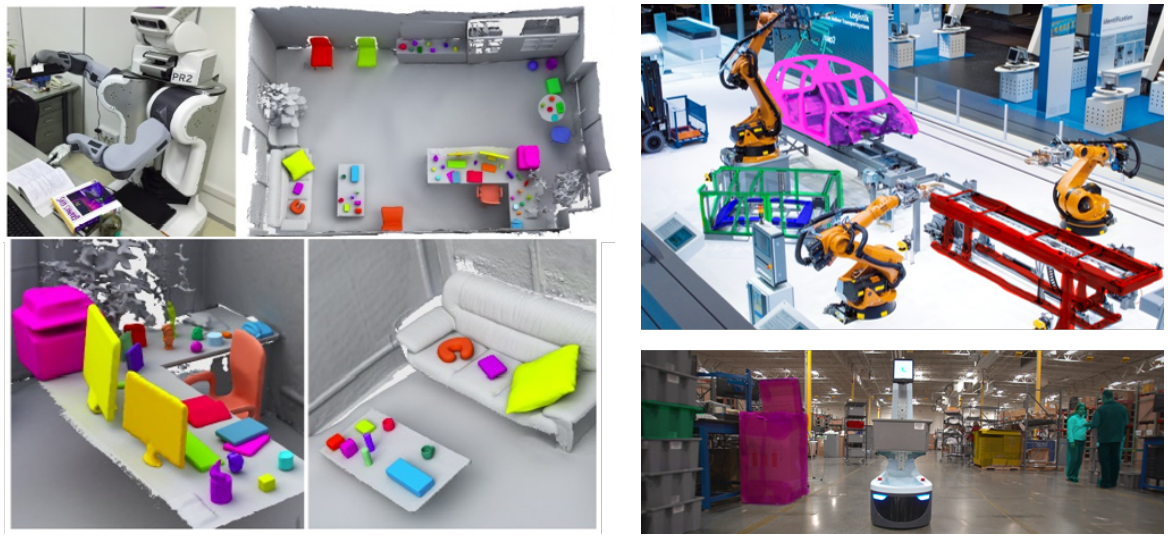
Experiments proving the validity of proposed methods were not reported in this Chapter. The ones proving the validity of the RL technique can be found in [37]. Gatto tested the algorithm on a NAO robot grasping objects on a table. The ones proving the goodness of the RLfD framework can be found in Chapter 5 and in [43].

4

Open Semantic Framework

This Chapter collects efforts done to tap into the Cloud and develop an Open Semantic Framework for Knowledge acquisition of Cognitive robots performing Manipulation tasks.

Contents



(a) Objects in an house.

(b) Objects in a firm.

Figure 4.1: Objects in the world

The robot is now able to autonomously navigate in cluttered workspaces and manipulate objects. It is able to interact with humans and learn from their demonstrations. Two constraints still limit its intelligence: the limit capacity of on board memory and the absence of a common medium to communicate and share knowledge between robots with different hardware and software components.

Focusing on objects manipulation: a large variety of objects exists and every object has to be manipulated in a certain way (see Figure 4.1). In order to let a robot manipulate all existing objects, its amount of on board memory has to be great enough for storage of vast resources of data. If no communication exists among robots, the same data could be stored several times. This is ineffective and inefficient.

Think about a common Knowledge Base where robots can save the acquired data and from which they can gleaned the information necessary to fulfill an assigned task. Think about an external Engine that elaborates this *Big Data*. Offloading robots from CPU-heavy tasks and from *Big Data* could speed up the computation. Storing data on a common Base and sharing it could speed up the Learning phase and let robots systems quickly perform complex tasks [44]. Focusing again on manipulation, this can facilitate incremental learning of grasp strategies [45] by matching sensor data against 3D CAD models stored on the Base. Examples of sensor data could include 2D image features [46], 3D features [44], and 3D Point Clouds [47], as well as demonstrations [48]. All data exceeds the processing capacity of conventional robotics systems. Other positives follow: robots could learn the stability of finger contacts from previous manipulations on the same object [3] or it could apply strategies used (by it or others) on some objects on similar parts encountered later [4].

The W3C[i] states that *the Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries.* According to this definition, the author thinks that tapping into the Cloud could be the solution. A Cloud server allows to offload robots from CPU-heavy tasks and to perform intensive computation while meeting the hard real-time constraints of operations. It provides, in fact, access to ten of thousands of remote processors for massively-parallel computation. A Semantic Web Ontology, instead, lets the

[i]World Wide Web Consortium: http://www.w3c.com/

definition of a common vocabulary that ensures a common understanding during the interaction as well as an efficient data transfer and integration.

This Chapter collects efforts done to tap into the Cloud and complete the conversion of simple robots into complex Cognitive robotics systems.

4.1 State of Art

As stated in [49], automation manufactures started their efforts in exploiting networking to connect machines over 30 years ago. The first was General Motors that, in the 1980s, developed the Manufacturing Automation Protocol (MAP) [50]. Since that data, manufactures started developing their adHoc communication protocols and in 1990, when the World Wide Web popularized the HTTP over IP protocols [51], the IP protocol was adopted as standard. In 1994, the first industrial robot was connected to the Web with an intuitive graphical user interface that allowed visitors to teleoperate the robot via any Internet browser [52]. In the mid and late 1990s, researchers developed a series of Web Interfaces to robots and devices to explore issues such as user interfaces and robustness that initiated the subfield of *Networked Robotics* [53]. In 1997, Inaba et al. described the advantages of remote computing for robot control [54]. In 2009, the RoboEarth project started: *a World Wide Web for robots: a giant network and database repository where robots can share information and learn from each other about their behavior and environment* [55]. It was mainly focused on service robotics and the need of saving and sharing on the Cloud heavy data such as 3D models of environments. In 2010, James Kuffner coined the term "***Cloud Robotics***" to define that branch of robotics which studies the use of Cloud technologies in order to make robots communicate with each other. Steve Cousins summarized the concept as "***No Robot is an Island***".

With the advent of Industrie 4.0 and the Internet of Things, the use of *Big Data* and *Cloud Computing* by all industrial robots begin to be consolidated requirements. In 2012, General Electric introduced the term *Industrial Internet* to describe new efforts where industrial equipment such as wind turbines, jet engines, and MRI machines connect over networks to share data and processing for industries including energy, transportation, and healthcare.

Many projects are emerging. We now focus on the work done to make the Cloud suitable for robots and that done to create Knowledge Bases suitable for robotics *Big Data*.

4.1.1 Cloud Computing

Different Could Platforms exist but most of them are not suitable for robotics applications. An example is the Google App Engine[ii]. It is an efficient and widespread Cloud Platform but it exposes only a limited subset of program APIs tailored specifically for Web applications, allows only a single process, and does not expose sockets. These limits make it unsuitable for robotic scenarios. Robotic applications are state-full, multi-processed, and require a bidirectional communication with the Client. ROS, for example, requires the existence of sockets. An example of efficient Cloud Robotics Platform, instead, is Rapyuta, the RoboEarth Cloud Engine [56].

[ii]Google, Inc. "Google App Engine". Online: https://cloud.google.com/appengine/ (2014)

4.1.2 Big Data

Many Robotics Knowledge Bases are available online. Examples are the SOMs of Rusu at al. [57], its extended version (SOM$^+$s) presented by Pangercic at al. [58], the Columbia Grasp dataset [59], the KIT object dataset [60], and the Willow Garage Household Objects Database [47]. KnowRob [61], the knowledge base of RoboEarth [55], is the most widespread. They store information about objects in the environment and their grasp poses. They can be used to evaluate different aspects of grasping algorithms, including grasp stability [3], robust grasping [62] and scene understanding [63]. The Household object database is a *simple* SQL database. The others are well-defined by an Ontology. RobotEarth models objects as 3D colored point clouds [64], the others store objects as triangular meshes. Stored items are of high quality but each object model consists of several recordings from different point of views; thus requiring either a lot of manual work or expensive scanning equipment.

4.2 Idea

The idea is that of creating a Robotics Cloud Engine and a Knowledge Base specialized in the computation, storing, and sharing of robotics manipulation data and actions [I, III, VI]. With respect to the State of Art, we aim to guarantee a quick Cloud data access through the adoption of an efficient Cascade Hashing algorithm. We aim also to avoid data duplication during the insertion of new data by using a novel powerful interlinking algorithm that finds the interlinking pattern of two data sets by applying two machine learning methods: the K-medoids [65] and the Version Space [66]. The algorithm was developed by Zhengjie Fan [67]. The Knowledge Base has to be scalable and should guarantee an intelligent data storage and access also thanks to the type of stored data. For this reason, we want to adopt and Ontology that defines data. Its name will be RTASK and will give users the possibility of storing multiple types of objects visual data: 2D Images, B-Splines, and Point Clouds. In this way, users will be free to save the data that they prefer and data access will be efficient: in order to search an object on the Base, the recognition process will be able to start from the comparison of the smallest features (e.g., the ones representing 2D Images) and eventually expand to the others (B-Splines and Point Clouds, in increasing order). With respect to the existing work, we do not want to impose an onerous manual work in order to store objects from different view points: an object should be stored even if there exist only a single registration of one of its views. The idea is to replace the onerous training with a human helping the robot in recognizing objects when viewed from other orientations. The teacher will exploit the connection between new view points and other properties, e.g., name and function. These new features will be stored in the Ontology and will gradually increment robots knowledge about the object itself. Moreover, RTASK will differ from existing ones because it will observe the IEEE Standards proposed by the IEEE RAS ORA WG by extending the Knowledge Base it proposed [21, 22, 23]. Balakirsky et al. [68] proposed a kitting ontology that observes these standards. RTASK will generalize the manipulation concept by introducing the notion of manipulation *task* and *action*. This means that any manipulation task can be represented, e.g., grasps and pushes.

4.3 Ontologies vs Database models

As stated in [69], the first Relational Database model was proposed by Codd in 1970. Its release caused a revolution in the management of mass data repositories. New technologies for data

representation and exploitation subsequently emerged: Object Oriented Databases for representing complex data [70], Multimedia Databases for storing multimedia files [71], Parallel or Distributed Databases for sharing the load of data or processes [72], XML Databases for interacting with the Web [73]. All these databases let the efficient and flexible data storage and querying. However, these technologies present several problems in a Semantic Web context because the information cannot be semantically annotated. On one hand, the content of these databases is only shown when a query is performed in the database, and on the other, the semantic description of databases is represented using its schema, often unavailable or even useless because it cannot be exploited depending on the format chosen to represent it.

An Ontology is the manifestation of a shared understanding of a domain that is agreed between a number of agents and such agreement facilitates accurate and effective communications of meaning, which in turn leads to other benefits such as interoperability, reuse, and sharing (Agarwal [74]). Ontologies allow to depict key concepts with their properties, relationships, and constraints making all pertinent knowledge about a domain explicit. Their format is computer-interpretable and allows instances to be understood and processed automatically by software agents without people's mediation.

Some pros and cons are detailed.

A Database schema is usually designed to meet the requirements of a particular application. When these requirements change, the schema also need to be modified. Ontologies, instead, are usually independent from text and implementation because they operate on a higher level of abstraction.

Ontologies do not need to distinguish between basic or complex data types and their properties have much more semantics than Database data types, since Ontologies do not need to be normalized. Databases, instead, are not able to represent axioms. This disadvantage could be solved with the use of logic rules. However, it is the way through which Ontologies represent data that is different and innovative: while Databases make a clear distinction between schema and data, Ontologies mix the schema specification with real data, named *instances*. Moreover, it is not necessary that Ontologies have data. As consequence, Ontologies are more flexible and instances of classes can be instanced again without corrupting the already stored information.

Ontologies do not follow any rule: a new instance definition does not require any defined constraint to be accomplished - these are simply added. A Database representation, instead, must accomplish all the requirements defined therein in order to ensure data integrity. More specifically, a Database tuple cannot be included in the Database if it does not satisfy all the relational model constraints, such as primary keys, foreign keys, null rules, etc. Consequently, the closed world assumption is applied to the Relational Database model, which represents a big semantic loss in the process of modeling information. On the other hand, Ontologies use reasoners to determine which instances belong to them. Each time an instance check is required, the reasoner is executed, contrary to Database system, in which information integrity is always ensured.

Ontology reasoners are also used to extract new information. In a similar way, query languages like SQL are used to develop such tasks in the Database area. The main difference between both technologies is that reasoners can find new information regardless of whether data is defined therein. From Databases, we can only obtain new information using the tuples stored in the system and new information related to the schema or mixing parts of the schema. Tuples are never obtained.

Ontologies languages, such as RDF or OWL, allow to express more semantic concepts than Database languages which only include constructs for defining or extracting data.

While Ontologies represent reality quite well, this technology is not efficient in managing instances since these are represented in OWL or RDF files. Consequently, when the number of instances increases, they need to be stored in a database environment, and then the Ontology provides an interface that enables them to be accessed.

This is what we did: an Ontology has been implemented instead of a simple Relational Database. However, a Database supports the Ontology in order to quickly retrieve data.

4.4 RTASK

RTASK formulates a common vocabulary for robotics manipulation. As proposed in [75], it separates the concepts of *tasks* and *tasks executions*. Tasks are *abstract* entities that describe goals to be reached; while tasks executions are *events* composed of *actions* that are performed by robots in the world in order to reach goals.

4.4.1 Design

Figure 4.2 depicts the Ontology design: a **Task** is assigned to an **Agent**, e.g., a **Robot**. It should be executed within a certain **time** interval and requires the fulfillment of a certain **Motion** in order to be performed. **Manipulation** is a sub-class of **Task**. Several types of manipulations exist, e.g., grasps and pushes. They involve the handling of an **Object** located at a certain **Pose** (**Position** and **Orientation**) through the execution of a **Manipulating** action. If the **Task** is assigned to a **Robot**, then the **Motion** will be represented by a **Robot Action**. In detail, the **Robot Manipulation Action** involves the activation of the robot **End Effector**. Studies have demonstrated that: i) placing the arm in front of the object before acting improves actions; ii) humans typically simplify the manipulation tasks by selecting one of only a few different prehensile postures based on the object geometry [76]. On this view, the **End Effector** is first placed at a **Pose** *p'* at a distance *d* from the **Object**, with its actuable **Joints** at a certain **Pre** manipulation **Posture**. Then, the **End Effector** is placed at a **Pose** *p* close to the **Object** and the effective manipulation **Posture** is assigned to its **Joints**. In order to retrieve the manipulation data of an object in the scene, the object should be recognized as an instance previously stored in the Ontology. For this purpose, every **Object** is characterized by an *id*, *name*, *function*, and the visual features obtained by the **Sensors**. For every **Object**, RTASK stores multiple types of visual features: **2D Images**, **B-Splines**, and **Point Clouds**.

4.4.2 Implementation details

Figure 4.3 depicts how RTASK follows the IEEE Standards by extending CORA [23]. In details:

- *SUMO* is a top-level Ontology that aims to define the main ontological categories describing the world;
- *CORAX* includes concepts and relations associated with *design*, *interaction*, and *environment*, which are not covered in SUMO;

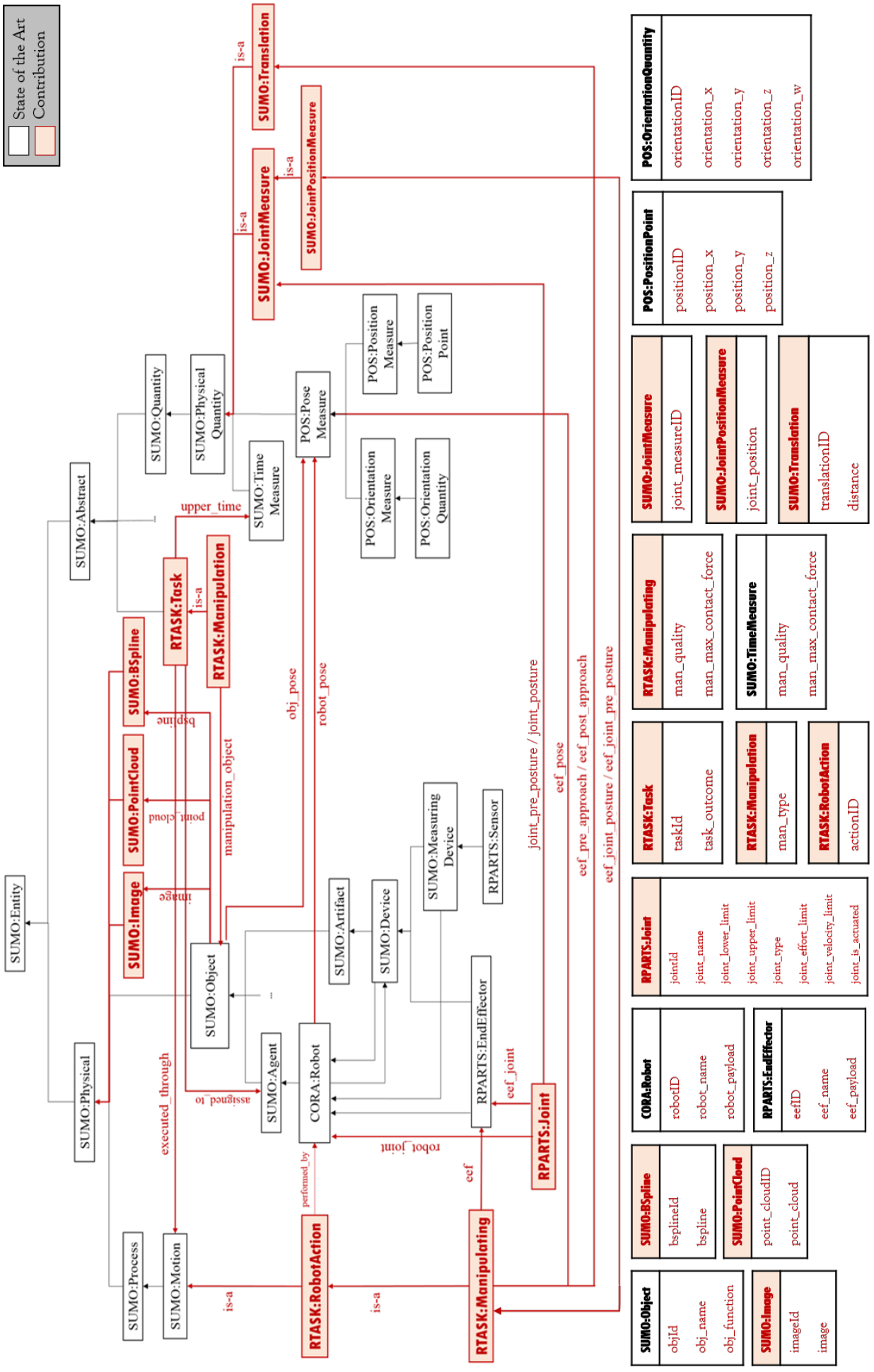


Figure 4.2: The Ontology.

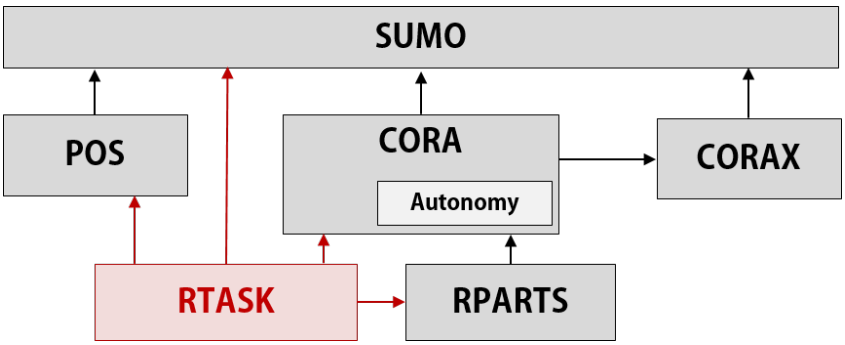


Figure 4.3: The RTASK extension to the IEEE Ontology for Robotics and Automation.

- *CORA* has to maintain the consistency among the different sub-ontologies in the Standard. It commits the existence of four main notions in the whole domain of R & A: *device*, *robot*, *robot group*, and *robotic system*. It defines robots as devices that exhibit agency (capability of acting by themselves), including devices ranging from clockwork robots to autonomous robots. Robots can form robot groups, which display their own agency. Finally, robots and robot groups, together with auxiliary equipment, can form robotic systems, such as robotic manufacturing systems;

- *RPARTS* is a sub-ontology of CORA that specifies the notions related to specific kinds of robot parts: robots are devices composed of other devices and *robot part* is the role played by devices while they are connected to robots. Robot parts can be: *robot sensing parts*, *robot actuating parts*, *robot communicating parts*, and *robot processing parts*;

- *POS* allows to describe positions, orientations, and poses in *coordinate systems*, which are *abstract* entities that are defined in relation to a single *reference object*. This ontology also provides means of describing positions and orientations *qualitatively* and *quantitatively*. Regarding the information about position, in the quantitative case, it can be described as a *position point* in a given coordinate system; and, in the qualitative case, it can be described as a *position region* defined as a function of a reference object. Besides these concepts, POS also specifies other important notions such as *transformation* and *spatial operator*.

A detailed overview of the final obtained schema can be found in Figure 4.2.

4.4.3 Languages

RTASK is represented thorugh the union of RDF[iii] and OWL[iv], namely OWL Full. RDF is used to define the structure of the data, OWL adds semantic to the schema and allows users to specify relationships among data. OWL Full allows an ontology to augment the meaning of the RDF vocabulary guaranteeing the maximum expressiveness of OWL and the syntactic freedom of RDF. Indeed, OWL is adopted by W3C and it is the representation language used by the RAS ORA WG. Protégé is used as ontology editor[v].

Queries allow robots to investigate the knowledge base and retrieve existing data. A robot able to query the database has the capability of efficiently and intelligently perform tasks. In our case,

[iii] Resource Description Format (RDF). Online: htpp://www.w3.org/RDF
[iv] Web Ontology Language (OWL). Online: http://www.w3g/TR/owl-features
[v] Protégé. Online: http://protege.stanford.edu/

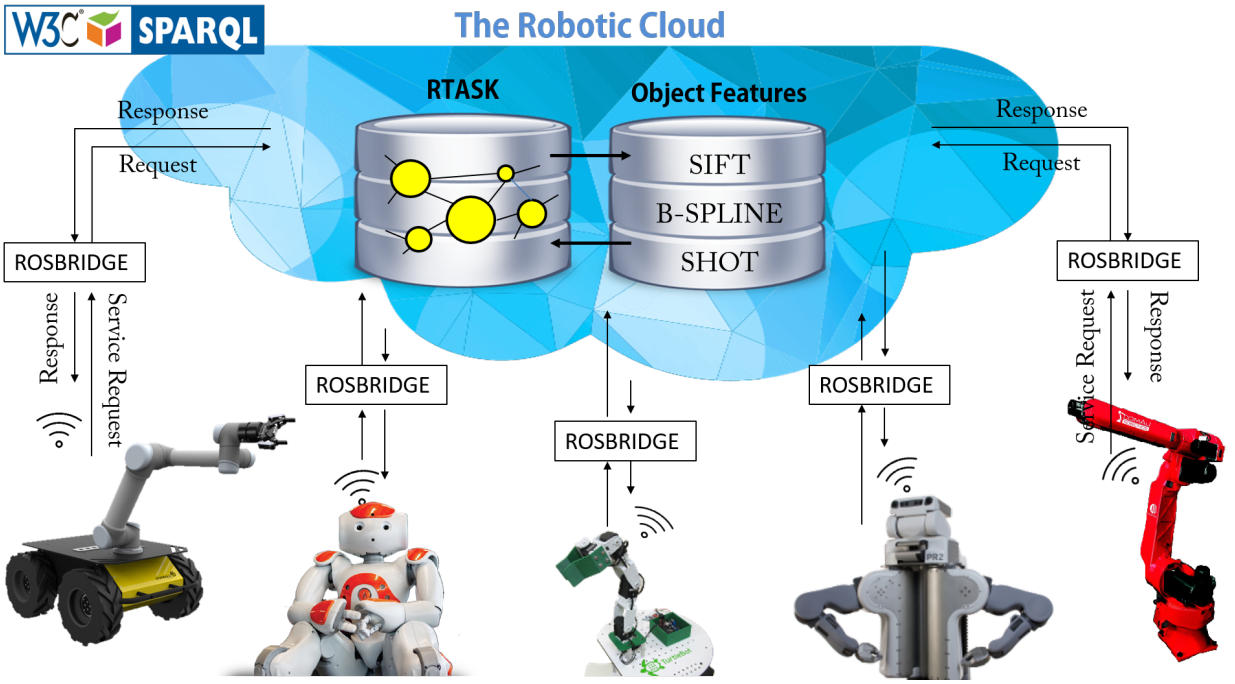


Figure 4.4: The Robotic Cloud.

a C++ interface lets ROS users query RTASK using SPARQL[vi]. Apache Jena Fuseki is used as SPARQL server[vii].

4.5 Engine

Given an object previously segmented from the scene, the Cloud-based Engine accesses RTASK and retrieves the manipulation actions necessary to manipulate it. If no ontological data exists, the Engine saves the new objects in RTASK together with its semantic and visual properties (e.g., name, function, 2D Images, Point-Clouds, and B-Splines). Then, it generates the relative manipulation actions.

The current implementation of the Cloud-based Engine is based on CORE [77]. The robot has an internal *ROS node* that receives the segmented objects and sends them to the Cloud-based Engine. The Engine is composed on another *ROS node* capable of reading the content of received messages. The communication is based on the **ros_bridge** interface, which provides a Web socket channel between nodes [15].

4.5.1 Data Retrieval

The robot asks the Cloud Server for the retrieval of the manipulation data of an object. It sends a message containing the type of its gripper and the compressed point cloud of the manipulable object. The compressed point cloud representation saves space and connection time. The compressed point cloud is encoded in order to be transmitted over the Web socket channel. After the encoding the whole message is represented as JavaScript Object Notation (JSON) objects. The ROS message being encoded on the Web socket request follows.

[vi] Simple Protocol and RDF Query Language (SPARQL). Online: http://www.w3.org/TR/sparql11-query
[vii] Apache Jena Fuseki. Online: https://jena.apache.org/documentation/fuseki2/index.html

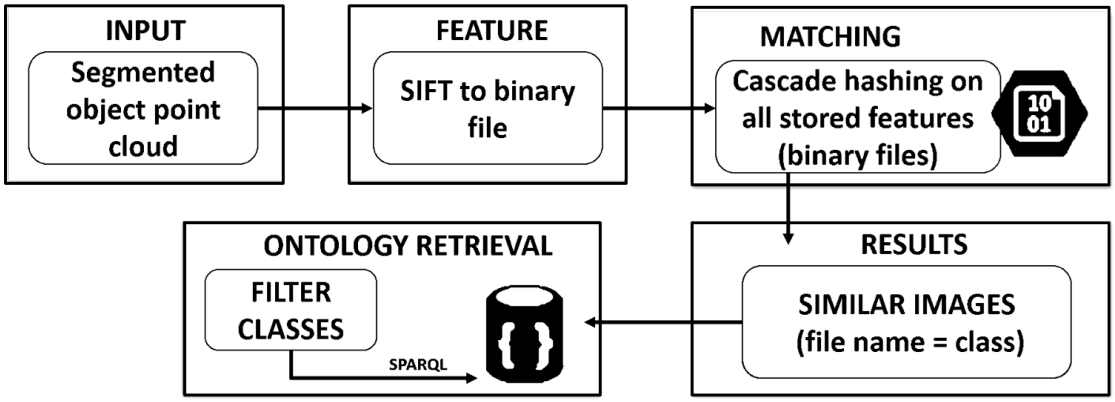


Figure 4.5: Image processing pipeline for the matching phase.

```
# Task name (pick, place for example)
string task
# Gripper name
string gripper_id
# Compressed point cloud (object representation)
string data
```

The Server receives the Client request and performs a super-fast search of the object inside RTASK. The search starts from the comparison of the object 2D images features (SIFT[viii]) (see Appendix B). In case of mismatch, it compares the B-Spline. Finally, it try to compare the point clouds features (SHOT) [20] (in increasing order of format size). Steps of the super-fast search of SIFT features follow (see Figure 4.5). Other comparisons work in a similar way.

1. Decode the message and decompress the point cloud;
2. Convert the point cloud to a color image using OpenCV [78] (OpenCV) functions;
3. Extract the image SIFT features and store them on a binary file of a Server folder;
4. Match the features with the ones stored in the Server;
5. Query RTASK through SPARQL;
6. The Server returns a *moveit_msgs::Grasp* ROS message containing the relative manipulation data.

The search is fast because of the novel and super-fast Cascade hashing algorithm adopted during the matching phase (Step 4) and because of the way in which features are stored. The algorithm allows constructing a dataset without a learning phase: there is not need to train the hashing function as in other ANN methods. Given the input image, the function returns the names of its most similar images features (according to the SIFT parameters). Names are in the form of integer numbers and define the object classes of RTASK. Note that features of stored objects are precalculated and stored on a Server folder (treated as a Database): they are not calculated at every data set access. Thanks to the combination of these two characteristics, only one query at the end of the matching process is needed in order to retrieve the similar objects stored on the Ontology.

[viii] Chris Sweeney, Theia Multiview Geometry Library: Tutorial & Reference. Online: http://theia-sfm.org

Algorithm 4 Interlinking Instances across Data Sets

Input: Two Data Sets
Output: Links accross Data Sets
1: The data set *D*, *D'*; /*two data sets to be interlinked*/
2: Similarity threshold *T*
3: **for** Each property/relation in the data set *D* **do**
4: **for** Each property/relation in the data set *D'* **do**
5: Match properties/relations that are corresponding to each other and store as the alignment *A*
6: **end for**
7: **end for**
8: **for** Each instance in the data set *D* **do**
9: **for** Each instance in the data set *D'* **do**
10: Compare instances' property values according to the correspondences of the alignment *A*;
11: Aggregate all similarities between property values as a similarity value *v*
12: **if** *v* >= *T* **then**
13: The two compared instances are interlinked with *owl:sameAs*.
14: **end if**
15: **end for**
16: **end for**

4.5.2 Data Insertion

An example of Client message aiming to create a new Object instance follows. It is encoded as a JSON string.

```
{ "op": "send_new_object",
"service": "insert",
"new_object":
 [
 {"new_point_cloud": "zc5p81H1cO8P+Ksmfdf...",
 "X": "0.5", "Y": "2.5", "Z": "1.5", "rad": "0.314"},
 ]
}
```

When the Server receives the message, it calculates the relative B-Splines, SIFT and SHOT features and inserts them both within the Server folder and RTASK.

During the insertion, duplication avoidance is desirable. To this end, Algorithm 4 is applied to automate the interlinking process. It was proposed in [67] and finds out the interlink pattern of two data sets by applying two machine learning methods: the K-medoids and the Version Space. Although interlinking algorithms require interactions with users for the sake of the interlinking precision, computations of comparing instances are largely reduced than manually interlinking. As the work-flow of Algorithm 4 shows, when interlinking two instances across two data sets *D* and *D'*, the algorithm first computes property/relation correspondences across two data sets (line 5). Then, instances property values are compared by referring to the correspondences (line 10). A similarity value *v* is generated upon all similarities of property values (line 11). If such a similarity is equal to or larger than a predefined threshold *T*, the two compared instances can be used to build a link with the relation *owl:sameAs* (line 12-14).

4.5.3 Human-Robot Interaction

Usually, an onerous a-priori human manual work is required to store objects visual features in a Robotics Knowledge Base: every object is represented by a large amount of registrations from different points of view. Objects representation is accurate but the a priori work is onerous. Our approach eliminates this prerequisite by introducing a human teacher that supports robots during their learning phase. First registrations will not be accurate, but knowledge will gradually increase until becoming absolute and giving robots autonomy.

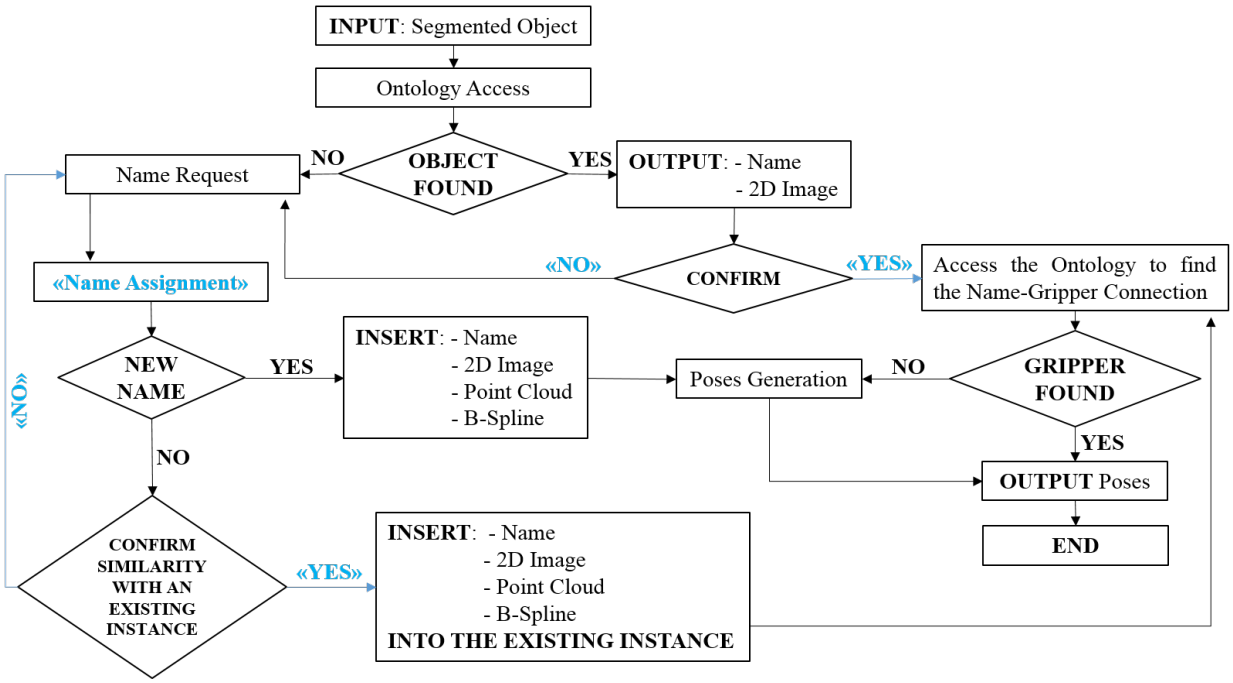


Figure 4.6: The Reasoning Pipeline. Blue parts represent human interventions.

Figure 4.6 highlights the reasoning at the bottom of this cooperation approach. Robots require human intervention to confirm the identity of a recognized object or to assign an identity to a new object. Moreover, humans help robots storing visual features of objects already stored in the Knowledge Base but seen from other points of view. To perform the connection, other objects properties are exploited, e.g., name and function.

This approach eliminates the a priori work currently done by humans by introducing a Cognitive and Social robot able to interact with other agents. A robot capable of learning becomes a flexible, adaptable, and scalable cyber-phisical system.

The message used for human-robot interaction follows.

```
# Operation type
string operation
# Object class name
string class_description
# Compressed point cloud (object representation)
string data
```

The human operator shows an object to the robot and gives it a description (e.g., coke, can, pen). The server seeks for a similar object filtering the Ontology by using the object description. If matches are found, the server return a message containing the classes of the similar objects found:

```
{ "op": "search_object_response",
"service": "query",
"similar_objects":
[
 {"class":"1", "X":"0.5","Y":"2.5","Z":"1.5","rad":"0.314"},
 {"class":"2", "X":"0.1","Y":"24.5","Z":"1.3","rad":"0.24"},
 {"class":"3", "X":"0.2","Y":"3.5","Z":"0.5","rad":"2.14"}
]
}
```

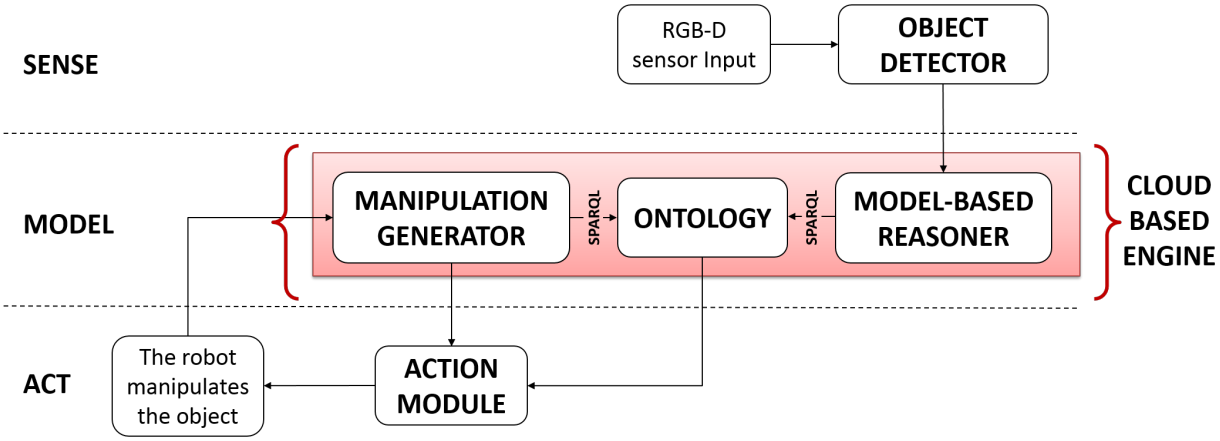


Figure 4.7: The Sense-Model-Act Framework.

A human feedback confirms the object class and visual features are added to it. Otherwise, a new instance is inserted in RTASK:

```
{ "op": "send_new_object",
"service": "insert",
"new_object":
 [
 {"new_point_cloud": "zc5p81H1cO8P+Ksmfdf...",
 "X": "0.5", "Y": "2.5", "Z": "1.5", "rad": "0.314"},
 ]
}
```

4.6 Experiments

Experiments aim to provide truthful temporal results on Cloud access and ontological data retrieval. For this purpose, RTASK and the Engine have been integrated inside the Cloud environment of CORE, which is available on the Wisconsin CloudLab cluster[ix] under the project "core-robotics". It consists of one x86 node running Ubuntu 14.04 with ROS Indigo installed. Moreover, RTASK has been integrated with OSD[x]: a data set of 726 MB currently containing 111 different objects, all characterized by a 2D color image and a point cloud.

In simulation, an Husky mobile robot equipped with a Universal Robot UR5 manipulator and a Robotiq 2 finger gripper has to solve a tabletop object manipulation problem: it has to grasp the nearest object located on the table in front of it. A Microsoft Kinect acquires the scene. Gazebo [79] is used as simulator. A ROS Sense-Model-Act framework has been implemented in order to give the robot the ability to detect objects on the table, access the Cloud server, and retrieve manipulation data (see Figure 4.7). Implementation details follow.

4.6.0.1 Sense

The Kinect acquires the RGB-D images of the environment. From the collected data, a compressed segmented 3D Point Cloud of each individual manipulable object is computed using PCL.

[ix] CloudLab. Online: www.cloudlab.us
[x] Object Segmentation Database. Online: http://www.acin.tuwien.ac.at/forschung/v4r/software-tools/osd/

Table 4.1: Elaboration time taken for features matching.

Features Matching		
	Time	*Matches on 6105 possible image pairs*
15 inliers	4.82 s	1592
300 inliers	4.48500 s	18

4.6.0.2 Model

On the Cloud, from the compressed Point Cloud, their 2D image is computed together with the B-Spline [80]. The relative visual features are computed, e.g., SIFTs and SHOT. A Cascade Hashing algorithm starts searching for a match between the features of the segmented object and that saved in the data set. The comparison starts from SIFTs and eventually expands to B-Splines (by computing the squared distance among points) and SHOTs. If a match exists, a reasoner accesses RTASK to find a match between the assigned action (e.g., *push* or *grasp*) and gripper. If a match exists, the gripper joints configuration is outputted. Otherwise, the Manipulation Data Generator computes the necessary poses by using the RL proposed by Gatto and detailed in Chapter 3.

4.6.0.3 Act

The module lets the robot move by activating its simulated engines. MoveIt![xi] generates the kinematic information required for the system to pass from the current to the goal configuration. During the motion, the information acquired by sensors is used to compare the system final state with the expected one. In case of mismatch, a trial and error routine starts correcting the joints configuration. The configuration that allows the task achievement is saved in RTASK.

4.6.1 Results

Figure 4.8 shows the system in operation: the robot detects the surrounding environment, segments the object in front of it, and grasps the nearest objects through the manipulation configuration retrieved from the Cloud.

Tables 4.2 and 4.1 reports the most significant time data. Table 4.2 depicts the time taken for the extraction of the descriptors of the 2D images stored in the data set. The table shows two types of extraction: the extraction of descriptors of all the 111 images contained in the data set and that of descriptors of a single image. For each type of extraction, tests were done on 1, 2, 3, and 4 threads respectively. The system employees 0.387422 s to extract descriptors of a single image using 4 threads. Table 4.1 focuses on the time taken for features matching. By using 300 inliers, the match is accurate and the computational time does not affect the real-time constraints of a robotics manipulation.

The reported computational times consider SIFT features. Times gradually increments if B-Splines or HOGs are considered. Moreover, the computational effort depends on the richness of the features: the more complex the point cloud is, the greater the extraction time will be. The segmentation helps maintaining computational times low.

[xi] Ioan A. Sucan and Sachin Chitta, "MoveIt!", Online: http://moveit.ros.org

Table 4.2: Elaboration time taken for features extraction.

Features Extraction				
	1 Thread	*2 Thread*	*3 Thread*	*4 Thread*
111 images	45.1554 s	21.8381 s	15.331 s	12.7545 s
1 image	0.379014 s	0.408295	0.396394	0.387422 s

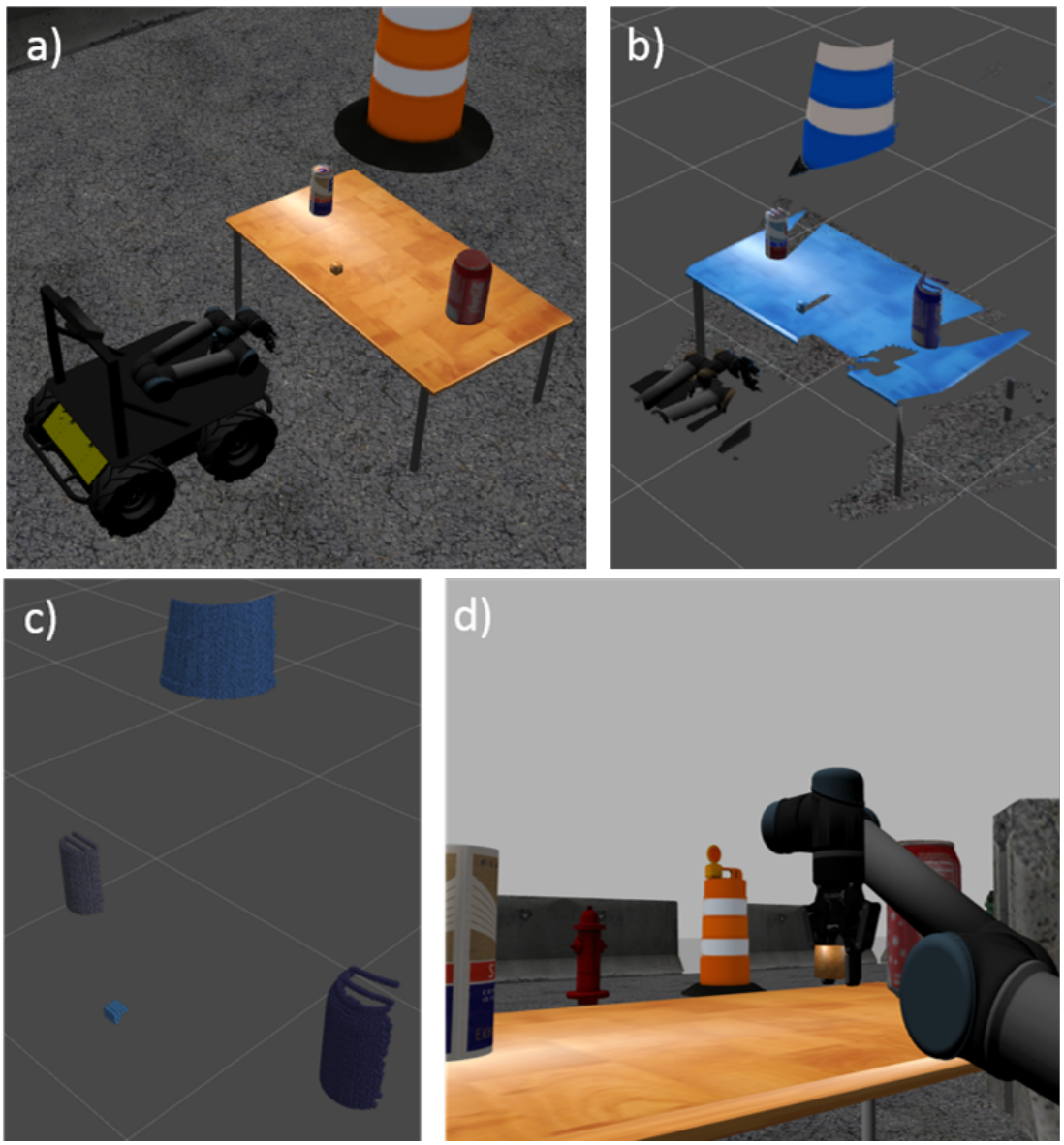


Figure 4.8: a) The robot in the scene; b) The detected scene; c) The segmented objects; d) The grasp of the nearest object.

During the experiments, MoveIt! took on average 3.765 s to find a feasible inverse kinematics solution (best case: 0.162757 s; worst case: 7.367439 s; number of trials: 100) on a Dell Intel Core i7-4470 CPU @ 3.40 GHz x 8, 15.6 GiB Memory, 970 GB Disk. In the best case, it completed the planning after 0.8486 s. Reported data proves that the intelligent and efficient structure of the proposed Open Semantic Framework does not adversely affect the time required to complete a manipulation: executing the features extraction and matching on 4 threads off-loads robots and increases system performances.

4.7 Discussion

The presented Open Semantic Framework increases robots knowledge and capabilities on objects manipulation. The Framework is composed of an OWL Ontology and a Cloud-based Engine. From the study of human actions when handling objects, the Ontology formulates a common vocabulary that encodes the robotics manipulation domain. The Engine, instead, was developed in order to transfer the computation on the Cloud: it off-loads robot CPUs and speeds up the robots learning phase. Given an object in the scene, the Engine retrieves its visual features and accesses the Ontology in order to extract the corresponding manipulation action. The Ontology respects the IEEE Standard by extending the existing CORA. The Engine minimizes visual data processing through an intelligent ontological data access and retrieval. During ontological data retrieval, a Cascade Hashing algorithm is adopted in order to optimize the comparison between saved and new visual features. In order to avoid data duplication during the insertion of new instances in the Ontology, a novel efficient interlinking algorithm has been adopted. The training for objects recognition and manipulation is replaced by a human-robot interaction.

We proved the efficiency and effectiveness of the proposed approach by building a ROS Sense-Model-Act framework able to associate manipulation actions to the features of the objects in the scene. Tests were performed in simulation.

5

Flexible robots: Open Controllers and ROS-I Interfaces

This chapter highlights efforts done and contributions made in order to make robots flexible. It focuses on the Opening of Industrial Robot Controllers and the development of a common control interface.

Contents

Usually, standard controllers allow users to control real robots thanks to the motion primitives formulated through the manufacturer motion planning library. In a Smart Factory users should be able to command robots through any software platform using any motion planning library. In order to avoid robot damages while testing new plans, every real robot should be accompanied by its simulated counterpart and this counterpart should be controlled in the same way of the real one.

Opening controllers and implementing for them a standard interface, compliant with industrial use and reliable, means achieving these objectives. It gives robots new features and capabilities such as sensory feedback control [81]. It increases ease of use, flexibility, and integrability of the controller.

5.1 State of Art

In the last decades, several attempts have been made in order to open industrial controllers. Please note [82] and [83]. Taking into account, for example, Comau Robotics and the efforts it done in order to implement Open features, the Company released its first open controller in 1991 [84], followed by more powerful implementations: C4G Open and C5G Open.

In order to increase ease of use, flexibility, and integrability of the open controller, a standard interface, compliant with industrial use and reliable, is mandatory. Joying ROS-Industrial seams the most appropriate way to develop this interface. ROS Industrial (ROS-I)[i] is the industrial extension of ROS [8]. It extends ROS capabilities, such as advanced perception and path/grasp planning, by enabling industrial robotic applications previously technically infeasible or cost prohibitive. The combination of ROS high-level functionalities with the low-level reliability and safety of industrial robot controllers lets the creation of a robust, reliable, and hardware-agnostic software. Many robot manufactures are involved in the ROS-I program: ABB, Adept, Fanuc, Universal Robot, Motoman, and, with the contribution described by this chapter, also Comau Robotics. To the author's best knowledge, only Motoman released production ready code. In the other cases the software is not yet production ready and it has only some levels of unit-testing. For Adept and Fanuc only joint position streaming is supported, with the controller currently overriding commanded velocity with a constant one. Trajectory downloading, velocity or force control are not yet supported. For ABB only trajectory streaming is supported [85]. The Motoman driver controller interface, instead, provides a more high-performance interface for controlling Motoman robots and it meets qualitative criteria for being used in production systems. The controller plans movements using an existing ROS-compliant motion planning library: MoveIt![ii]. Tt provides a low-level control of joint position, velocity, and path timing. Being ROS non real-time, the computational load decreases robot speeds down to 20% and 70% of robot capabilities.

5.2 Idea

The idea is to draw inspiration from what was done by robot manufactures that actually join the ROS-I community: opening an unopened robot controller and implementing its ROS-I interface. The opening should let the integration of external sensors. The interface should allow both ROS and non ROS-compliant platforms to give motion commands to the controller. Users should be able to choose the motion planning library through which controlling the robot and the open controller

[i] http://rosindustrial.org/
[ii] http://moveit.ros.org/

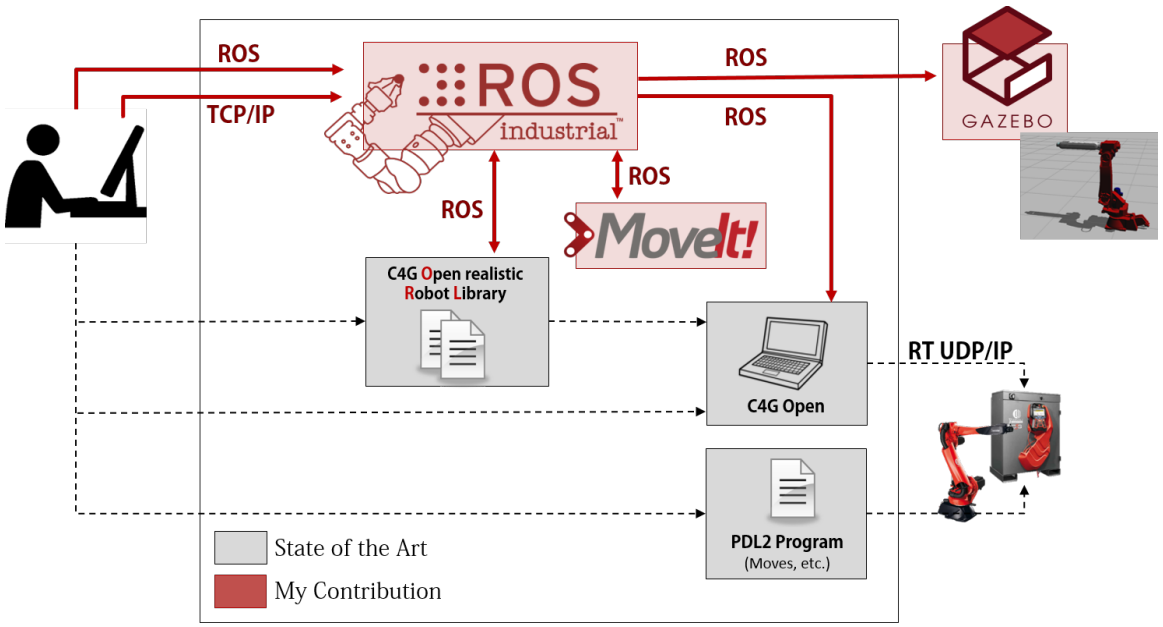


Figure 5.1: Architecture of the obtained system. It includes including the initial structure (gray) and the integration with ROS (red).

should be able to elaborate, at least, position and velocity controls. Appropriate correction factors should be implemented so that ROS does not affect the real-time capabilities of the controller. Moreover, simulated robots should be provided. They should faithfully reproduce their real counterparts by ensuring the correspondence of movements in case of both joint positions and velocities streaming. For this purpose, the Comau case has been considered [IV].

5.3 A ROS-I Interface for C4G Open Controller

When the interface was implemented, C5G Open was not yet released. For this reason, this work focuses on the opening of C4G and the development of its ROS-I interface. The interface implementation can be extended to C5G Open.

Figure 5.1 the architecture of the obtained system. It is composed of:

- C4G Open: the Comau industrial Open Robot Controller;
- ROS nodes implementing the communication with C4G Open;
- the Comau Smart5 SiX robot model in the Universal Robot Description File (URDF[iii]) format for the simulated representation of the automata;
- the Open realistic Robot Library (ORL) (the Comau motion planning library).

These tools allow either a ROS or a non ROS -compliant platform to move either a real or a virtual Comau Smart5 SiX robot. Both a position and a velocity controller are supported and motion commands can be issued using both ORL and any other ROS-compliant motion planning library. As above mentioned, other manufactures offer packages suitable only for MoveIt. Moreover, in our case both the real and the simulated automata are driven using C4G Open. Being it real-time, the faithful control of the robot and its 100% speed are guaranteed

[iii] http://wiki.ros.org/urdf

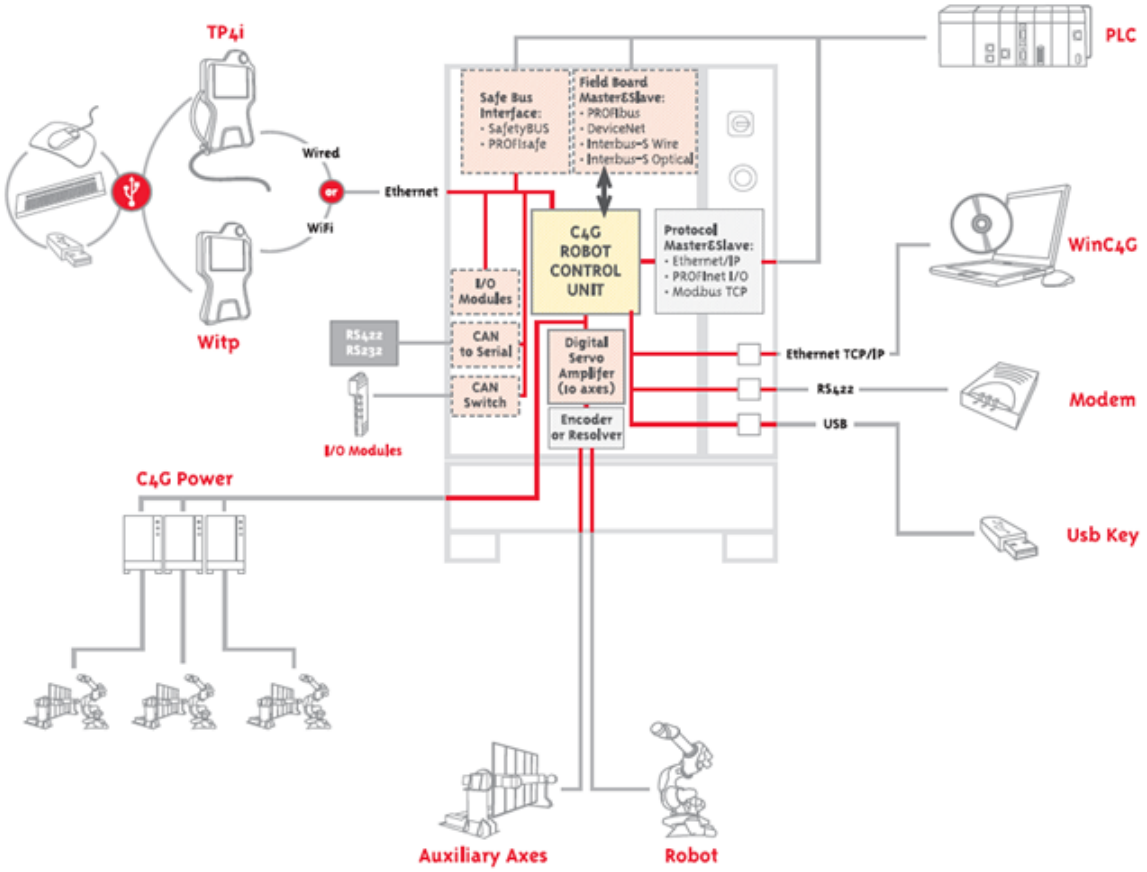


Figure 5.2: Standard architecture for the Comau C4G robot controller.

5.3.1 Comau C4G Open Controller

The standard COMAU C4G controller consists of a modular architecture with three different hierarchical hardware levels (see Figure 5.2). The first level is the SMP+ control board. At this level, all high level processes take place: interpretation of user programs, management of operator interfaces, network communications, trajectory generation, computation of dynamic model and management of assigned tasks, collisions detection, system diagnostics, high hierarchical level centralized adjustment process, axes synchronizing control, management of all I/O devices. The second level is the MPC board, inside: fine interpolation of the manipulator trajectory, robot position adjustment, real time system diagnostics, master-slave axes management. The lower level is the (DSP board, inside: control of electric motor currents and torque generation process for individual axes control, power stage management, position sensor and acquisition of motor angular measurements, high speed digital and analogical I/O management. The system architecture is based on a real time communication. It has a frame rate of *1ms* on an Ethernet network that uses an UDP protocol, between the SMP+ board (client) and the MPC board (server).

This controller is powerful but it does not allow sensors and devices connections to the robot. This restriction has been overcome opening the robot controller: a second server process has been inserted introducing an external PC in the network (see Figure 5.3). The PC adds power to the Robot Control Unit simplifying the implementation of complex manufacturing applications: programming automated robotics cells is easier and external sensors integration is possible. In this way, writing custom applications where standard control processes and trajectory generation interact with external sensors, devices or PCs is feasible. Mixing trajectories between open and standard modalities is also possible, together with the possibility of programming the robot

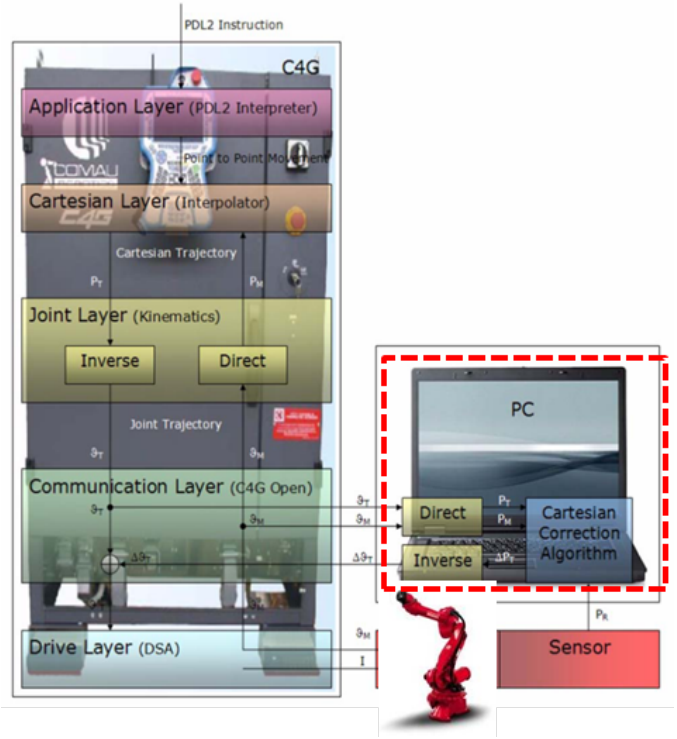


Figure 5.3: Open Control architecture for the Comau C4G robot controller.

using different open modalities such as: additional and absolute position control, additional current control, trajectory management and modification of pre-planned trajectory. Different Open architectures have been presented in several European Projects (SME Robot[iv] and ARFLEX[v]) to correct a real trajectory using vision systems or to integrate an inertial navigation tool. The C4G Open Control is based on GNU/Linux Operating System and the real-time is achieved by using the Real-Time Application Interface (RTAI) [13] module to extend the standard kernel with real-time capabilities. On the other hand, in order to provide a real-time communication layer, C4G Open architecture uses the Real-Time networking (RTnet) [14] stack, an open source hard real-time network protocol compatible with RTAI: a real time thread provides the robot status information from the external PC to the robot controller and vice versa.

5.3.2 Open Motion Planning Library

The Open realistic Robot Library (ORL) has been developed in order to perform operations on a virtual robot without having it. The motivation behind the realization of this library is that C4G Open forces the user to program at a low level in motor round references. The ORL allows the user to program at a high level in order to let him develop custom applications with C4G Open (or C5G Open, if we consider the new controller Comau is developing). The library allows to perform the following operations:

- Initialization of the virtual robot on Linux (complete Comau robot family) starting from a real configuration file
- Computation of Direct and Inverse Kinematics, for each Comau robot
- Error management of position and joint ranges

[iv] http://www.smerobot.org/
[v] http://www.arflexproject.eu/

- Integration of Comau Trajectory Generator and Interpolator
- Computation of the Dynamic Model and Jacobian

All these operations are realistic, i.e. they describe what the robot would do if they were performed via the Comau standard programming language (PDL2) on the real robot. Performing direct/inverse kinematics allows the user to control the real robot workspace in advance (e.g. joint stroke-ends, Cartesian avoided positions). Routines for the direct motor rounds control are included, together with motion functions able to convert and manage degrees or Cartesian references.

5.3.3 The ROS-I Interface

In order to make the controller ROS compliant without loosing its real-time capabilities, existing modules have been wrapped in ROS and the basic real-time communication thread has been extended by developing other parallel threads. This extension lets the decoupling of the real-time communication process from those that manage data input/output: the communication with the robot becomes independent from the data used to control it and to interpret its feedbacks.

The wrapper maintains existing features. The user can still directly program the robot through the PDL2 language or use ORL to formulate robot motion primitives that C4G Open converts into the equivalent machine code.

In addition, the user can impose motion directives in the form of ROS commands: a ROS message with the motion request is sent to the system, ROS redirects the message to the motion planning library, and the library answers with a ROS message containing the motion primitives the controller needs to move the robot. ROS integration makes ORL a simple module of the system that can be replaced at any time with any other ROS compliant motion library, e.g., MoveIt!.

A TCP/IP connection has been implemented in order to give non ROS compliant frameworks the possibility to interface to the proposed system. This type of protocol provides a reliable, ordered, and error-checked delivery of the data. The ordered data transfer guarantees the robot to execute the motion primitives in the same order specified by the user. Reliability ensures the retransmission of lost packets and the consequent execution of all motion primitives. As result, the robot will executes all the desired motion commands in the desired order.

In order to provide users with a faithful simulation environment, a virtual model of a Comau robot has been created. I can be used within any ROS compliant simulator (e.g., Gazebo [79] or VRep [86]). Moreover, a plug-in allowing the user to control the robot inside the Gazebo simulator has been developed.

A description of the process adopted to make the system usable both in the real and in the simulated environment follows.

5.3.3.1 Real robot

In order to command a real Comau robot using ROS, we have to take into consideration the fact that the robot communication is hard real-time (it has a frame rate of 2 ms) and ROS is non real-time. In order to face this problem, a communication node was developed. It includes both a real-time and a normal priority thread. The former establishes the connection between the controller and the real robot. The latter provides the controller with the motion data: if a ROS compliant platform aims to control the robot through specific motion directives, then the

thread will deliver these motion commands to the controller in the form of ROS messages. If a non ROS compliant processor has the same request, the thread will send the motion primitives to the controller in the form of TCP/IP messages. Performed tests show that the proposed structure ensures the absence of robot/controller connection errors and lets the user to control the robot both through a real and a non real -time platform.

5.3.3.2 Simulation

The simulated counterpart of a Comau Smart5 Six robot has been developed. The package contains robot meshes and the URDF file. The latter describes the geometry, kinematics and dynamics of the robot, such as joints masses and inertial matrices. In order to control the robot virtual model within the Gazebo simulation environment, a plug-in connecting the Comau Open Controller to Gazebo has been developed, together with and a wrapper for ORL. The latter ensures the same trajectories interpolation both for the virtual and the real robot, that results in the same robot movements execution both in the virtual and in the real world.

5.4 Experiments

5.4.1 Objectives

Experiments aim to prove the reliability and versatility of the system. The robot has to properly perform the imposed tasks both in the virtual and in the real world. The correctness of movements has to be guaranteed when both joints position and velocity commands are imposed. The system has to be independent from the motion planning library used to imprint motion primitives: any ROS-compliant library should be used, e.g., ORL, MoveIt!, other open-source or customized libraries.

In order to remain within the context of Industrie 4.0, experiments aim also to test the capability of the Open Controller to process the information obtained from integrated sensors, and the one of the robot to learn from human teachers. For these reasons, tests were performed through the RDfL framework proposed in Section 3.

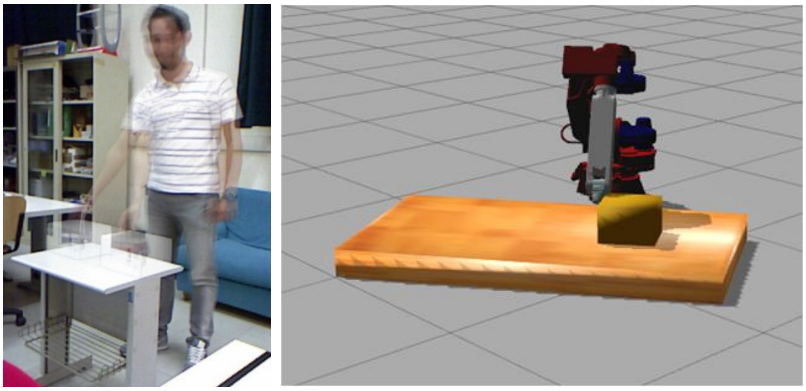
5.4.2 Experimental Setup

Every experiment is performed both in simulation and in the real world. Every experiment adopts a specific controller and a specif motion planning library. The same controller and the same motion planning library are used to move both the virtual and the real robot.

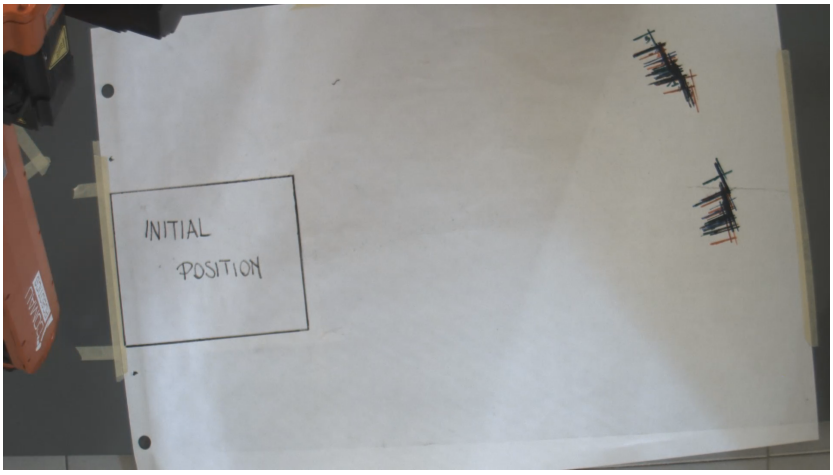
Data are acquired from 3 demonstrators repeating each task 15 times.

5.4.2.1 Scenario 1 - Position Controller

The first experiment aims to reproduce the displacement of a piece in the conveyor belt. The task requires very accurate robot movements, so that the manipulator is controlled by setting joint positions. The interpolation of robot motion is computed by ORL, that offers high reliable functions to command a robot in position. Accuracy and repeatability tests are performed by measuring the displacement of the piece and the consequent precision of movements.



(a) Human demo (b) Simulated experiment



(c) Real experiment

Figure 5.4: The first experiment: use of a Position Controller

5.4.2.1.1 Assignment A human demonstrator moves a box along a line from the beginning to the end of a 45 cm long table; the manipulator has to reproduce the task both in simulation and in the real world (see Figure 5.4).

5.4.2.1.2 Simulated Setup A virtual model of a Comau Smart5 Six robot is in the scene. A table, larger then that used by the demonstrator, is in front of it. A simulated box is on its top. The virtual box and the table have the same characteristics of to the real ones, e.g., they have the same friction coefficient. The box is equivalent to the real one less then the color. The table is larger in order to avoid robot singularities when the box reaches the end of the simulated table.

5.4.2.1.3 Real Setup A Comau Smart5 Six is on the floor. A white paper sheet is situated on the floor in front of it in order to simulate the table. A box is on the sheet. The box is the same used by the human demonstrator. The absence of a real table is due to the robot configuration (mounted on the ground).

5.4.2.2 Scenario 2 - Velocity Controller

The second experiment simplifies the arrest of a piece before its extraction from the conveyor belt. The action can be interpreted as the block of an object after a certain event. A velocity controller

(a) Human demo (b) Real experiment

Figure 5.5: The second experiment: use of a velocity controller

is required in order to make the robot moving within a certain time interval. For this reason, a custom interpolator has been used to compute velocities from joints positions. An ad-hoc motion planning library, different from ORL, is used. Tests aim to verify the observance of a time limit.

5.4.2.2.1 Assignment A subject stops a Lego Mindstorms NXT[vi] robot performing a uniform rectilinear motion on a 60 cm long table. The stop occurs when the NXT robot reaches a certain position. The manipulator has to reproduce the task (see Figure 5.5). It stands out that the NXT robot is performing a uniform motion, this means that the action of stopping the robot at a certain position is equivalent to the one of gripping the robot after a certain time interval.

5.4.2.2.2 Simulated Setup A simulated Pionner [vii] robot moves in front of a simulated Comau Smart5 Six robot. The manipulator robot has to stop the mobile robot.

5.4.2.2.3 Real Setup A Pioneer robot with a box on its top moves at a constant speed in front of a Comau Smart5 Six robot. The manipulator robot has to stop the box on the Pioneer within a certain period of time (see Figure 5.5).

5.5 Results

5.5.1 Scenario 1 - Position Controller

5.5.1.1 Simulation

The replica reflects the real human demonstration: on 25 attempts, the system obtained an average displacement of 45.011 +/- 0.402 cm. The 0.402 cm variance is negligible because of sensors and motors noises. Moreover, it may be due to the imprecise positioning of the box at the beginning of every test.

[vi] http://www.lego.com/en-us/mindstorms/?domainredir=mindstorms.lego.com
[vii] http://www.mobilerobots.com/ResearchRobots/PioneerP3DX.aspx

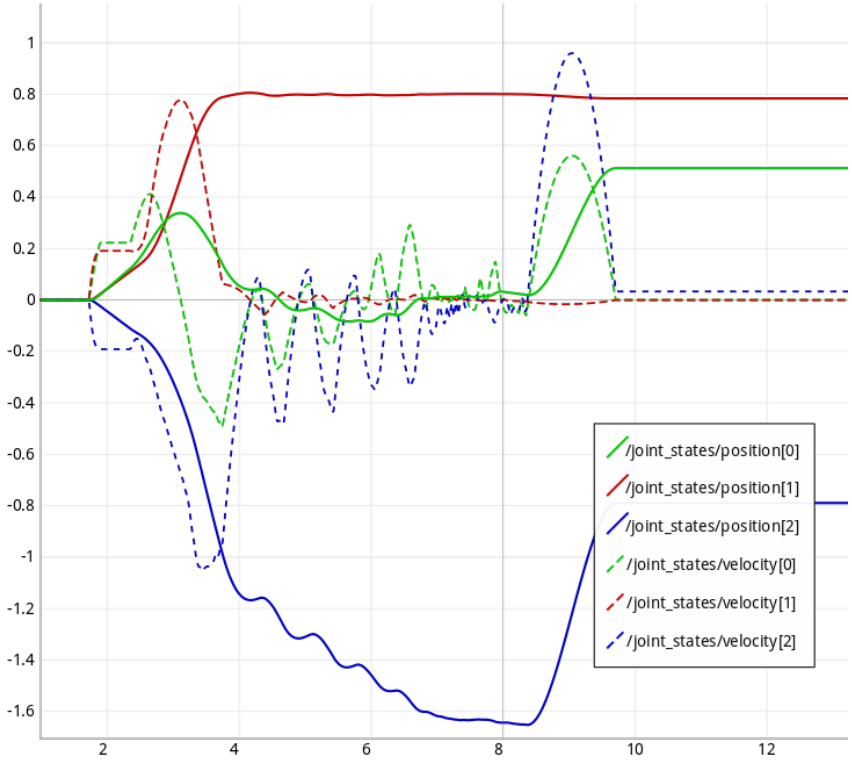


Figure 5.6: Robot joint positions and velocities during the second test.

5.5.1.2 Real World

On 25 attempts, the system obtained an average displacement of 54.332 +/- 0.7547667 (see Figure 5.4). The real robot motion leads to a gap of about 10 cm in the displacement. The alteration is due to the different frictional forces: the material of the support surface used in the real experiment has a friction coefficient lower then that of the tables used for the demonstration and for the simulated representation. Having a lower friction coefficient facilitates displacements. Thanking into consideration the different friction, sensors and motors noises, and box positioning inaccuracies, the resulting gap can be considered negligible.

5.5.2 Scenario 2 - Velocity Controller

5.5.2.1 Simulation

In this case, simulation is used as a tool to formulate the necessary ROS controls and coordinate the start of the two robots.

5.5.2.2 Real World

It is important to notice that the experiment does not require measurements because precision is not required. The robot stopped the box in time in all the 25 performed trials. Figure 5.6 shows positions and velocities of the three joints controlled by the RLfD framework during the experiment.

5.6 Discussion

This chapter described the methods adopted to Open an Industrial Robot Controller and develop its ROS-I interface. The final system lets users control both a real Comau robot and its simulated counterpart through both a ROS-compliant and a non ROS-compliant platform. Both joints position and velocity commands can be imprinted using the same controller both in the simulated and in the real world. Any ROS-compliant motion planning library can be used to generate the motion primitives.

In order to test the reliability and flexibility of the obtained system, a Comau Smart5 Six robot was asked to perform two different tasks, both in simulation and in the real world. The former involves the displacement of a static box and the latter considered the capture of a moving object. The experiments choice allows to test the system using two types of controllers and two types of motion planning libraries. A position controller is used in the first case and position primitives are formulated through ORL. A velocity controller is used in the second case and velocity primitives are formulated through an ad-hoc motion planning library. Experiments show that both the real and the virtual robot perform the same movements during the execution of both assignments. This proves that using the same controller in both environments guarantees the same trajectories interpolation. The detected movements precision proves the good functioning of the system itself, the proper development of the ROS-I interface, and the possibility of using any type of motion planning library. Having a ROS-I interface makes the Comau system re-usable by the entire ROS community, and having a replaceable motion planning library is an essential characteristic to make the system freely accessible and usable.

6

Conclusions and Future Work

This chapter highlights achievements reached and discusses future developments within the Cognitive Task Planning for Smart Industrial Robots.

Contents

6.1 Conclusions

This thesis presented a Cognitive Task Planning Framework for Smart Industrial Robots. Its aim was to contribute to the creation of Smart Robots to be inserted into the Smart Industries of the 4th Industrial Revolution. In details, it aims to make mobile industrial manipulator robots able to Autonomously Navigate Among Movable Obstacles while combining their Navigation and Manipulation capabilities. Robots were made Cognitive: able to learn from past experiences and from actions performed by other agents.

A novel NAMO planner was proposed. It extends the exploration strategy of KPIECE, which focuses the exploration on less explored areas, by removing the KPIECE constraint of obstacle avoidance: it assigns an importance to each cell by combining the efforts necessary to reach it and that needed to free it from obstacles. In this way, a path from a robot start configuration q_S to a goal configuration q_G is found by taking into consideration the possibility of moving objects if q_G cannot be reached or if moving objects may significantly shorten the path. This approach gives robots the capability of reasoning about the environment and choosing when manipulating obstacles. The algorithm has a computational time that makes it scalable with respect to the size of the map and the number, shape, and pose of obstacles. It does not impose restrictions on actions to be performed: the robot can both push and grasp every object. It works on a two-dimensional projection of a three-dimensional cluttered workspace letting consider both objects in contact and not in contact with the ground. It assumes full world knowledge but the environment is reconfigurable and the algorithm can be easily extended in order to solve NAMO problems in unknown environments. In fact, it is able to handle sensor feedbacks and correct uncertainties regarding obstacle poses and robot actions [II].

If a robot is able to choose when manipulating obstacles, it has also to be able to manipulate them. A large variety of objects exist. Pre computing grasp or push actions necessary to manipulate them is impossible or, however, inefficient. It does not do justice to efforts done to make robots Intelligent, Autonomous, Cyber-Physical systems suitable for the firm of Industrie 4.0. For these reasons, this thesis presented some alternative manipulation approaches based on the capability of robots to learn from their actions and from that done by human teachers. A Reinforcement Learning Approach is presented, together with a Robot Learning from Demonstration one. Both approaches make robots Cognitive and do not require an onerous pre calculation of all manipulation poses [V].

If a robot has to store all the information learned, a large amount of memory is necessary. This is not efficient. Moreover, only sharing the acquired Knowledge could speed up the Learning. An Open Semantic Framework was proposed. The Framework is composed of a Cloud-based Engine and an OWL Ontology. The Engine off-loads robot CPUs by transferring the computation on the Cloud. The Ontology defines the robots manipulation domain and can be considered both a data store for the efficient and reliable execution of repeated tasks, and a Web-based repository for the information exchange among robots. Given an object in the scene, the Engine retrieves its visual features and accesses the Ontology in order to extract the corresponding manipulation action. If no information is stored, the Engine generates the gripper manipulation poses that will be stored in the Ontology. The Ontology respects the IEEE Standard by extending the existing CORA. The Engine minimizes visual data processing through an intelligent ontological data access and retrieval. During ontological data retrieval, a cascade hashing algorithm is adopted in order to optimize the comparison between saved and new visual features. Instead, in order to avoid data duplication during the insertion of new instances in the Ontology, a novel efficient interlinking algorithm has been adopted. It finds the interlink pattern of two data sets by applying two machine

learning methods, the K-medoids and the Version Space. The efficiency and effectiveness of the proposed Framework was proved by building a ROS Sense-Model-Act framework able to associate manipulation actions to the features of the objects in the scene [I, III, VI].

To ensure the flexibility and reusability of the final system, it has been coded as a modular Framework where every block is based on ROS. Not all Industrial Controllers are ROS-compliant and let the integration of external devices such as vision sensors (necessary in this thesis to detect objects to be manipulated). A brief description on efforts done to Open Industrial Controllers and develop their ROS-I interface was outlined. We focused on the Comau case. The final Comau system lets users control both a real Comau robot and its simulated counterpart through both a ROS-compliant and a non ROS-compliant platform. Both joints position and velocity commands can be imprinted. The same controller can be used both in simulation and in the real world. Any ROS-compliant motion planning library can be used to generate the motion primitives. The use of ROS makes the system easily adaptable to changes, reusable, and flexible [IV].

6.2 Future Work

The first future goal is that of proving the goodness of the proposed NAMO planner on unknown workspaces. This will let us study robotics behaviors in more realistic situations. Then, in order to let robots better learn from human demonstrations, a more accurate gesture recognition system will be integrated inside the proposed Learning from Demonstration Framework. This will allow to perform manipulation in a real, no simplified industrial setting. Finally, we aim to extend the proposed Ontology by defining other tasks and actions, e.g., we would like to include the Navigation domain. In fact, it is useful for a robot, that has to navigate inside an environment seen for the first time, to have information from other robots that have already passed through that area. We aim to assign robots new tasks and execute the relative new actions (Navigations and Manipulations) in order to increase the capabilities of the Cloud-based Engine.

A

Appendix

Kinodynamic Motion Planning by Interior-Exterior Cell Exploration

Kinodynamic motion Planning by Interior-Exterior Cell Exploration (KPIECE) [17, 18] is a sampling-based motion planning algorithm, specifically designed for use with physics-based simulation. KPIECE is able to handle high dimensional systems with complex dynamics: it reduces both runtime and memory requirements by using the information collected during the planning process to decrease the amount of forward propagation the algorithm needs.

Problem: *Find a collision free path from a robot Start configuration q_S to a robot Goal configuration q_G.*

Solution: A projection of the state space is discretized into cells. Multiple levels of discretization are defined depending on the size of the cells (see Figure A.1). Cells on one level all have the same size and they are created when needed, to cover the tree of motions as it grows. Coarser discretizations (larger cells) are used to grossly identify areas of the space that are less covered. Finer discretizations (smaller cells) can be subsequently used to more accurately identify less covered areas, within the grossly identified ones. In addition, cells are distinguished into *Interior* and *Exterior*:

- *Interior* cells are ones that have *2n* neighbors with n the dimension of the state space;
- *Exterior* cells are ones that cover the tree of motions but are not *Interior*.

When searching for a tree connecting q_S and q_G, KPIECE prefers to start the expansion from *Exterior* cells (P(*Exterior*) = 0.9). Once selected the type of cell from which expanding the tree (*Exterior* or *Interior*), a cell i is selected according to the formula:

$$Importance(i) = \frac{\log_{10}(I) * score}{S * N * C} \tag{A.1}$$

where:

- I is the iteration in which i is created;
- *score* is the estimated distance to the goal;

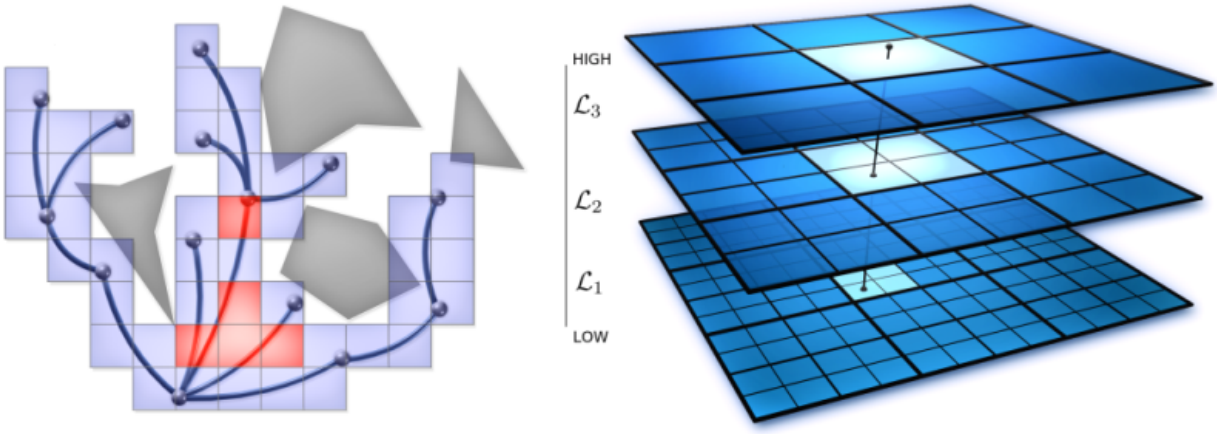


Figure A.1: Kinodynamic Motion Planning by Interior-Exterior Cell Exploration: Cells Discretization.

Algorithm 5 KPIECE

```
Create a grid G = Cell(q_S)
Create a list Cell(q_S).list = q_S
while time < t_max do
    type ← Select a cell type (Exterior or Interior)
    cell ← Select the most Important cell of that type
    q_old ← Select one of the visited states of cell
    Sample a new state q_rand with probability P such that distance(q_old, q_rand) ≤ ε
    if (q_old, q_rand) is valid then
        Add Cell(q_rand) to G
        Add q_rand to Cell(q_S).list
        Save the path between q_old and q_rand in the solution tree
        if q_rand = q_G then
            return the path between q_S and q_G
        end if
    end if
end while
```

- S is the number of times in which i has been selected for expansion;
- N is the number of instantiated neighboring cells;
- C is the measure of coverage for i (the sum of its visited states).

Once selected the cell, one of its states, q_{old}, is selected and inserted inside the list of visited states of that cell, according to an half normalization distribution. From q_{old}, the expansion starts. With a certain probability P, a new state q_{rand} is sampled. The state should be at a distance of at most ε from q_{old}. The edge joining the two states is evaluated. If it is valid:

- the cell of q_{rand} is added to the grid (if not already present);
- q_{rand} is added to the list of visited states of that cell;
- the search tree is updated by adding the new edge.

Otherwise:

- the last valid state q_{new} belonging to the edge is extracted;
- $q_{rand} = q_{new}$.

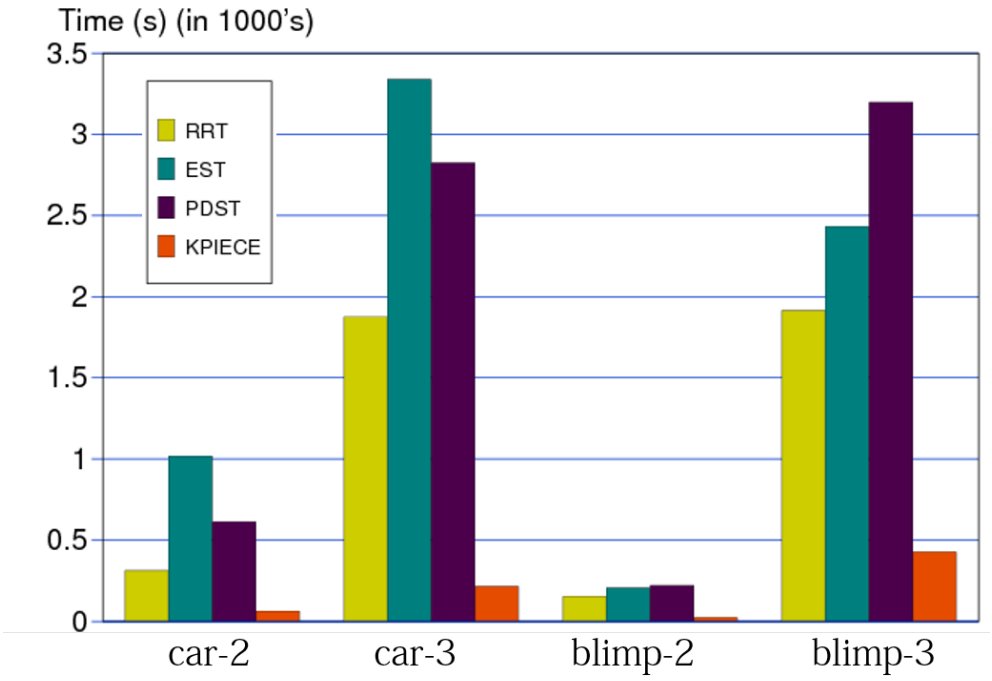


Figure A.2: Computational statistics.

The process iterates until q_G is reached (see Algorithm 5).

Computational improvement over other Motion Planning methods is observed (see Figure A.2).

B

Appendix

Open Source Computer Vision Library

Open Source Computer Vision (OpenCV) is a library of programming functions mainly aimed at real-time computer vision, originally developed by Intel's research center in Nizhny Novgorod (Russia), later supported by Willow Garage, and now maintained by Itseez [78]. The library is cross-platform and free for use under the open-source BSD license.

Images are described by *visual features*, or *descriptors*. Descriptors are compact descriptions of the images. Two types of descriptors exist:

- *Global descriptors*: they encode the visual content of the entire image, e.g., the overall appearance, color, intensity, etc. Since they encode a single, global image description, they are fast to compute;
- *Local descriptors*: they describe an image region, i.e. the *patch*, around a point of the image, i.e. the *interest point*. First, a set of image *interest points* are detected, then local descriptors are extracted with respect to each point.

We review the 2D image descriptor included in OpenCv, publicly available, and used in this thesis for Image Matching.

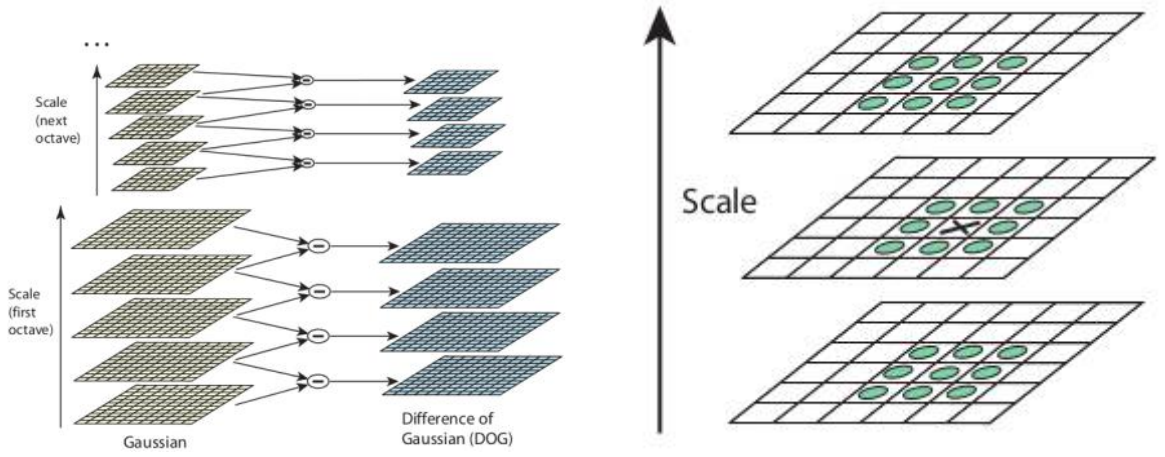
Scale-Invariant Feature Transform

SIFT is an algorithm in computer vision to detect and describe local features in images. The algorithm was published by David Lowe in 2004 [i].

SIFT is composed of five steps[ii]:

1. *Scale-space Extrema Detection*: A scale-space filtering is used to detect key points with different scale. First, DoG is found. DoG is an an approximation of LoG. LoG acts as a blob detector which detects blobs in various sizes due to change in σ, with σ a scaling parameter. DoG is obtained as LoG but it blurs an image with two different scaling parameters, σ and $k\sigma$. This process is done for different octaves of the image in Gaussian Pyramid (see Figure B.1(a)). Once DoG is found, images are searched for local extrema over scale and space. E.g., one pixel in an image is compared with its 8 neighbors as well as 9 pixels in next scale

[i] Chris Sweeney, Theia Multiview Geometry Library: Tutorial & Reference. Online: http://theia-sfm.org
[ii] http://docs.opencv.org/3.1.0/da/df5/tutorial_py_sift_intro.html#gsc.tab=0



(a) Difference of Gaussians (b) Search of local extrema over scale and space

Figure B.1: Scale-invariant feature transform: the Scale-space Extrema Detection phase.

Figure B.2: Scale-Invariant Feature Transform

and 9 pixels in previous scales. If it is a local extrema, it is a potential key point. It basically means that key point is best represented in that scale (see Figure B.1(b)).

2. *Keypoint Localization*: Once potential key points locations are found, they have to be refined to get more accurate results. The Taylor series expansion of scale space is used to get more accurate location of extrema. If the intensity at this extrema is less than a threshold value, i.e. the *Contrast Threshold* (0.03 as per the paper), it is rejected. DoG has higher response for edges, so edges also need to be removed. For this, a concept similar to Harris corner detector is used: a 2x2 Hessian matrix computes the principal curvature and if a key point is located above that threshold, called *Edge Threshold*, then that key point is discarded. The process removes both low-contrast key points and edge key points; only strong interest points remain;

3. *Orientation Assignment*: An orientation is assigned to each key point to achieve invariance to image rotation. A neighborhood is taken around the key point location depending on the scale. The gradient magnitude and direction is calculated in that region. An orientation histogram with 36 bins covering 360 degrees is created. (It is weighted by gradient magnitude and Gaussian-weighted circular window with σ equal to 1.5 times the scale of key point). The highest peak in the histogram is taken and any peak above 80% of it is also considered to calculate the orientation. It creates key points with same location and scale, but different directions. It contribute to stability of matching;

4. *Keypoint Descriptor*: The key point descriptor is created. A 16x16 neighborhood around the key point is taken. It is divided into 16 sub-blocks of 4x4 size. For each sub-block, a 8 bin orientation histogram is created. So a total of 128 bin values are available and organized as a vector in order to form the key point descriptor;

5. *Keypoint Matching*: Key points between two images are matched by identifying their nearest neighbors. Because of noise, the second closest-match may be very near to the first. In that case, ratio of closest-distance to second-closest distance is taken. If it is greater than 0.8, they are rejected. This choice eliminates around 90% of false matches while discards only 5% correct matches.

Figure B.2 shows the resulting set of key points after the application of the SIFT algorithm on a 2D image.

C

Appendix

Point Cloud Library

The Point Cloud Library (PCL) is a collection of stateof-the-art algorithms and tools to process 3-D data, which can be combined to efficiently solve common problems such as object recognition, registration of point clouds, segmentation, and surface reconstruction [19]. The library is open source and licensed under BSD terms and, therefore, free to use for everyone.

As for OpenCv, both *Local* and *Global descriptors* exist. The describe the content of point clouds. In this thesis, SHOT is used. A brief description follows. It has been taken from [20].

Signature of Histograms of Orientations

The signature of histograms of orientation (SHOT) descriptor is a local descriptor that encodes a signature of histograms representing topological traits, making it invariant to rotation and translation and robust to noise and clutter. The descriptor for a given key point is formed by computing local histograms incorporating geometric information of point locations within a spherical support structure. For each spherical grid sector, a one-dimensional histogram is constructed by accumulating point counts of the angle between the normal of the key point and the normal of each point belonging to the spherical support structure. The final descriptor is formed by orderly juxtaposing all histograms together according to the local reference frame.

Discrete quantization of the sphere introduces a boundary affect when used in combination with histograms, resulting in abrupt changes from one histogram bin to another. Therefore, quadrilinear interpolation is applied to each accumulated element, resulting in an evenly distribution into adjacent histogram bins. Finally, for better robustness toward point-density variations, the descriptor is L_∞-normalized. The dimensionality of the used signature is 352.

The pipeline used to compute SHOT descriptors in PCL follows. An object of the class *pcl::SHOT-Estimation* is created using the template parameters of our input data (cloud and normals). A kdtree is provided to the estimator to perform NN searches. Indices represent the key points from the input cloud where SHOT descriptors should be computed. The radius search represents the size around each key point that will be described. Finally, calling the compute method returns a point cloud with as many descriptors as key points.

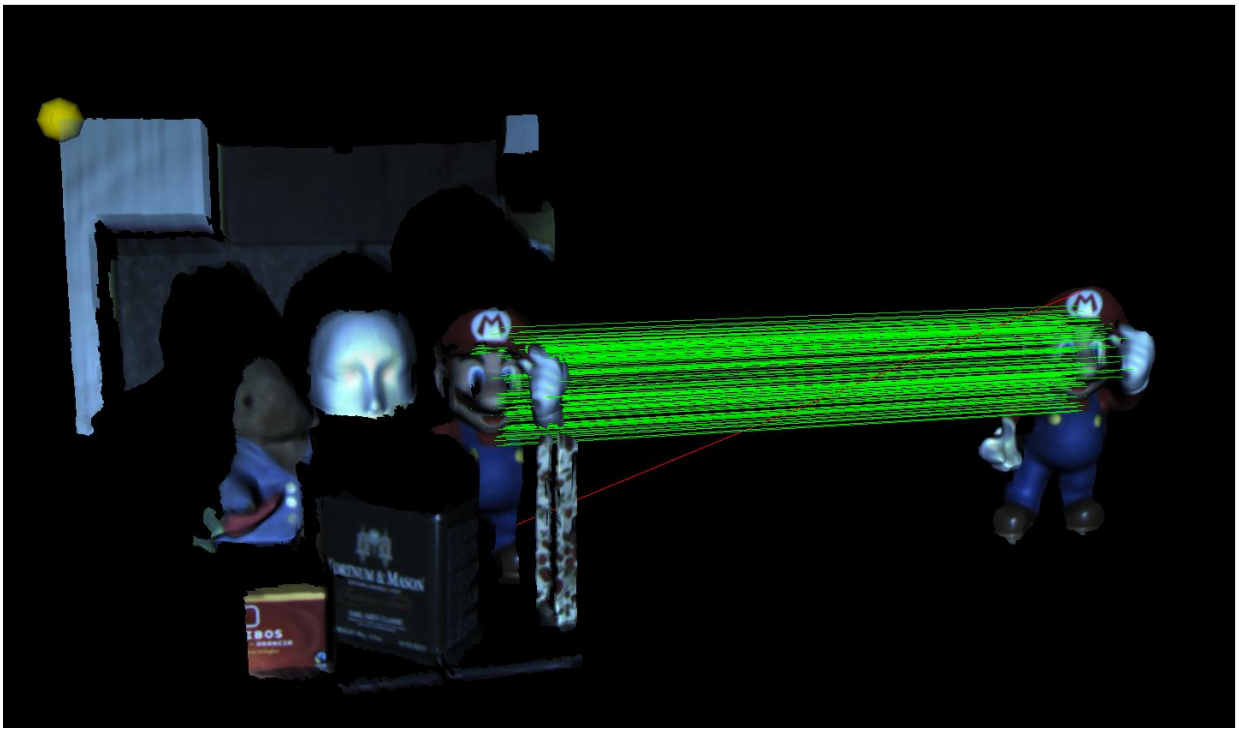


Figure C.1: Signature of Histograms of Orientations.

```
pcl::SHOTEstimation< ... > shot;
shot.setSearchMethod (tree); //kdtree
shot.setIndices (indices); //keypoints
shot.setInputCloud (cloud); //input
shot.setInputNormals(normals);//normals
shot.setRadiusSearch (0.06); //support
shot.compute (*shots); //descriptors
```

Figure C.1 depicts the working principles of the algorithm.

rail_segmentation

Segmentation is the process by which a scene Point Cloud is divided, i.e. segmented, into different Point Clouds. Every Point Cloud depicts a detected object.

rail_segmentation is a ROS segmentation tool proposed by the RAIL Lab[i]. It let users perform a tabletop object segmentation, i.e. the segmentation of objects on a plane. Its pipeline follows.

1. *Table Plane Estimation*: this phase finds the table plane. The RANdom SAmple Consensus (RANSAC) algorithm is used [87]. RANSAC is an iterative method to estimate the parameters of a mathematical model from a dataset containing outliers. Given an input dataset, it usually returns those points that fits a certain model. In the case in analysis, the input model is the table's plane. The points of the table are detected by first estimating a plane in the point cloud, all the points belonging to such a plane will be the points of the table. Once detected these points, they are separated from objects and removed.

2. *Euclidean Cluster Extraction*: the Euclidean Clustering method is used to divide the remaining, unorganized Point Cloud P. Assuming that we use a Kd-tree structure for finding the nearest neighbors, Algorithm 6 follows.

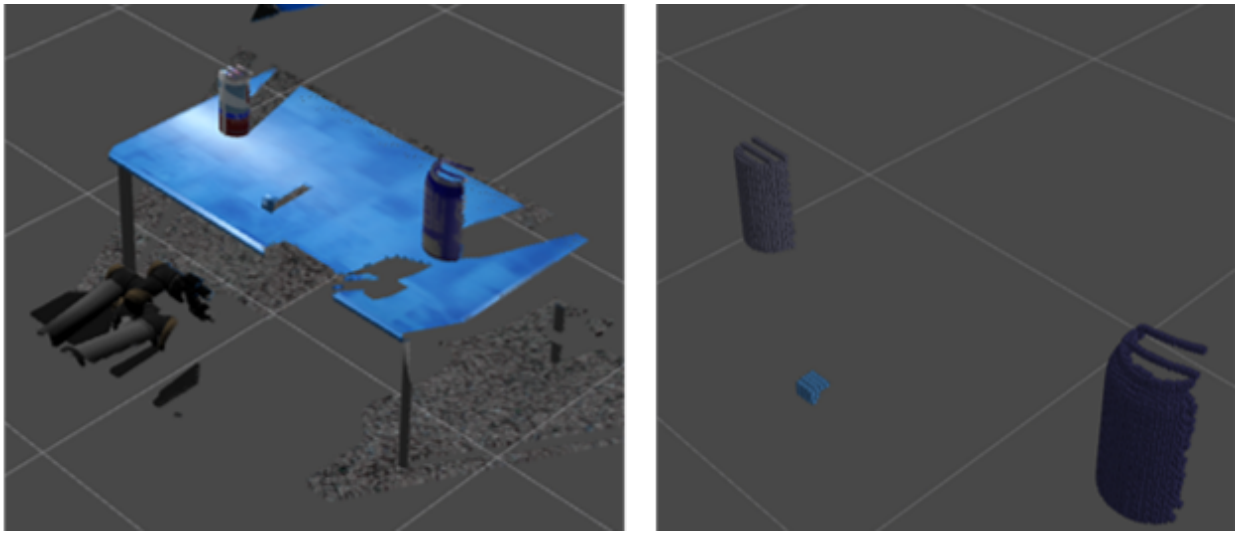


Figure C.2: The rail_segmentation tool.

Algorithm 6 Euclidean Cluster Extraction

1: Create a Kd-tree representation for the input point cloud dataset P
2: Set up an empty list of clusters C and a queue of the points that need to be checked Q
3: **for** every point $\boldsymbol{p}_i \in P$ **do**
4: Add $\boldsymbol{p}_i$ to the current Q
5: **for** every point $\boldsymbol{p}_i \in Q$ **do**
6: Search for the set P_k^i of point neighbors of $\boldsymbol{p}_i$ in a sphere with radius $r < d_{th}$
7: **for** for every neighbor $\boldsymbol{p}_i^k \in P_i^k$ **do**
8: Check if the point has already been processed, and if not add it to Q
9: **end for**
10: **end for**
11: When the list of all points in Q has been processed, add Q to the list of clusters C, and reset Q to an empty list
12: **end for**
13: the algorithm terminates when all points $\boldsymbol{p}_i \in P$ have been processed and are now part of the list of point clusters C

Figure C.2 depicts an example of obtained results.

[i] http://wiki.ros.org/rail_segmentation

Bibliography

[1] Aude Billard, Sylvain Calinon, Rüdiger Dillmann, and Stefan Schaal. *Robot Programming by Demonstration*, pages 1371–1394. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[2] M. Stilman and J. Kuffner. Navigation among movable obstacles: Real-time reasoning in complex environments. In *Proceedings of the 2004 IEEE International Conference on Humanoid Robotics (Humanoids '04)*, volume 1, 2004.

[3] H. Dang and P. K. Allen. Learning grasp stability. In *in Proc. Int. Conf. Robot. Autom. (ICRA)*, pages 2392–2397, Saint Paul, MN, 2012.

[4] J. Glover, D. Rus, and N. Roy. Probabilistic models of object geometry for grasp planning. In *Proceedings of Robotics: Science and Systems (RSS)*, Zurich, Switzerland, 2008.

[5] G. Metta, G. Sandini, D. Vernon, D. Caldwell, N. Tsagarakis, R. Beira, J. Santos-Victor, A. Ijspeert, L. Righetti, G. Cappiello, G. Stellin, and F. Becchi. The RobotCub project - an open framework for research in embodied cognition. *Humanoids Workshop, IEEE RAS International Conference on Humanoid Robots*, 2005.

[6] A. W. M. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain. Content-based image retrieval at the end of the early years. *IEEE Trans. Pattern Analysis and Machine Intelligence*, pages 1349–1380, 2000.

[7] J. Pauli and G. Sommer. Perceptual organization with image formation compatibilities. *Pattern Recognition Letters*, pages 803–817, 2002.

[8] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[9] A. Oliveira, E. De Pieri, and U. Moreno. An Open-architecture Robot Controller applied to Interaction Tasks. In E. Hall, editor, *Advances in Robot Manipulators*. InTech, 2010.

[10] P. Lutz and W. Sperling. Osaca the vendor neutral control architecture. In *Proceedings European Conference Integration in Manufacturing*, pages 247–256, Dresden, DE, 1997.

[11] Martin Levihn. *Autonomous Environment Manipulation to Facilitate Task Completition*. PhD thesis, Georgia Institute of Technology, 2015.

[12] Mike Stilman. *Navigation Among Movable Obstacles*. PhD thesis, Carnegie Mellon University, 2007.

[13] L. Dozio and P Mantegazza. Real time distributed control systems using rtai. In *Sixth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, Hakodate, Hokkaido, Japan, 14-16 May 2003.

[14] Rtnet – a flexible hard real-time networking framework. In *10th IEEE International Conference on Emerging Technologies and Factory Automation*, Italy, 2005.

[15] C. Crick, G. Jay, S. Osentoski, B. Pitzer, and O. C. Jenkins. Ros-bridge: Ros for non-ros users. In *Proceedings of the 15th International Symposium on Robotics Research*, 2011.

[16] Jian Cheng, Cong Leng, Jiaxiang Wu, Hainan Cui, and Hanqing Lu. Fast and accurate image matching with cascade hashing for 3d reconstruction. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[17] I. A. Şucan and L. E. Kavraki. Kinodynamic Motion Planning by Interior-Exterior Cell Exploration. *Algorithmic Foundation of Robotics VIII: Selected Contributions of the Eight International Workshop on the Algorithmic Foundations of Robotics*, 57:449–464, 2010.

[18] I. A. Şucan and L. E. Kavraki. A Sampling-Based Tree Planner for Systems with Complex Dynamics. *IEEE Transactions on Robotics*, 28(1):116–131, 2012.

[19] R. B. Rusu and S. Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[20] Federico Tombari Walter Wohlkinger Christian Potthast Bernhard Zeisl Radu Bogdan Rusu Suat Gedikli Aitor Aldoma, Zoltan-Csaba Marton and Markus Vincze. Point Cloud Library - Three-DimensionalObject Recognition and 6 DoF Pose Estimation. 2012.

[21] C. Schlenoff, E. Prestes, R. Madhavan, P. Goncalves, H. Li, S. Balakirsky, T. Kramer, and E. Miguelanez. An IEEE standard Ontology for Robotics and Automation. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 1337 – 1342, Vilamoura, Algarve, Portugal, 2012.

[22] Craig I. Schlenoff. An Overview of the IEEE Ontology for Robotics and Automation (ORA) Standardization Effort. In *Standardized Knowledge Representation and Ontologies for Robotics and Automation, Workshop on the 18th*, pages 1–2, Chicago, Illinois, USA, 2014.

[23] E. Prestes, S. R. Fiorini, and J. Carbonera. Core Ontology for Robotics and Automation. In *Standardized Knowledge Representation and Ontologies for Robotics and Automation, Workshop on the 18th*, pages 7–9, Chicago, Illinois, USA, 2014.

[24] S. Kagami M. Stilman, K. Nishiwaki and J. Kuffner. Planning and executing navigation among movable obstacles. In *IEEE/RSJ Int. Conf. On Intelligent Robots and Systems (IROS 06)*, 2006.

[25] M. Levihn H. Wu and M. Stilman. Navigation among movable obstacles in unknown environments. In *IEEE/RSJ Int. Conf. On Intelligent Robots and Systems (IROS 10)*, 2010.

[26] M. Stilman M. Levihn and H. Christensen. Locally optimal navigation among movable obstacles in unknown environments. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, 2014.

[27] G. Wilfong. Motion planning in the presence of movable obstacles. In *SCG '88: Proceedings of the fourth annual symposium on Computational geometry*, New York, USA, 1988.

[28] J. O'Rourke E. Demaine and M. L. Demaine. Pushpush and push-l are np-hard in 2d. In *Proceedings of the 12th Canadian Conference on Computational Geometry*, 2000.

[29] Yohei Kakiuchi et al. Working with movable obstacles using on-line environment perception reconstruction using active sensing and color range sensor. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on. IEEE*, page 1696–1701, 2010,.

[30] Jean-Claude Latombe. *Robot motion planning*. 1991.

[31] Kevin M. Lynch and Matthew T. Mason. Stable pushing: Mechanics, controllability, and planning. *The International Journal of Robotics Research*, 15(6):533–556, 1996.

[32] S. M. LaValle. Planning Algorithms. Technical report, Cambridge University Press, available at http://planning.cs.uiuc.edu/, 2006.

[33] J. T. Schwartz and M. Sharir. On the piano movers' problem: General techniques for computing topological properties of real algebraic manifolds. *Communications on Pure and Applied Mathematics*, 36:345–398, 1983.

[34] J.J. Kuffner and S. M. LaValle. RRT-Connect: an efficient approach to single-query path planning. In *IEEE Intl. Conf. on Robotics and Automation*, 2000.

[35] J.-C. Latombe D. Hsu, R. Kindel and S. Rock. Randomized kinodynamic motion planning with moving obstacles. *Intl. Journal of Robotics Research*, 21(3):233–255, 2002.

[36] Jack E Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems journal*, 4(1):25–30, 1965.

[37] Alejandro Gatto. A Cloud based Reinforcement Learning Framework for humanoid grasping, 2016.

[38] Stefano Michieletto. *Robot Learning by observing human actions.* PhD thesis, Università degli Studi di Padova.

[39] J. Kober, J. Andrew (Drew) Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *International Journal of Robotics Research*, 2013.

[40] R. Platt. and A. Pas. Using geometry to detect grasp poses in 3d point clouds. *International Symposium on Robotics Research (ISRR)*, 2015.

[41] D.A. Reynolds. *A Gaussian Mixture Modeling Approach to Text-Independent Speaker Identification.* PhD thesis, Georgia Institute of Technology, 1992.

[42] S. Michieletto, N. Chessa, and E. Menegatti. Learning how to approach industrial robot taska from natural demonstrations. In *IEEE Workshop on Advanced Robotics and its Social Impacats (ARSO2013)*, 2013.

[43] Andrea Bisson. Development of an interface for intuitive teleoperation of Comau manipulator robots using RGB-D sensors, 2014.

[44] C. Goldfeder and P.K. Allen. Data-driven grasping. *Autonomous Robots*, 31(1):1–20, 2011.

[45] K. Hsiao G. Bradski P. Brook M. Ciocarlie, C. Pantofaru and E. Dreyfuss. A side of data with my robot. *IEEE Robot. Autom. Mag.*, 18(2):44–57.

[46] M. Przybylski N. Vahrenkamp T. Asfour K. Huebner, K. Welke and D. Kragic. Grasping known objects with humanoid robots: A boxbased approach. In *Proc. Int. Conf. Adv. Robot.*, 2009.

[47] M. Ciocarlie, K. Hsiao, E. Jones, S. Chitta, R. Rusu, and I. Sucan. Towards reliable grasping and manipulation in household environements. In *in Proc. Int. Symp. Experimental Robotics*, pages 1–12, Delhi, India, 2010.

[48] D. Kent and S. Chernova. Construction of an object manipulation database from grasp demonstrations. In *Proc. Int. Conf. Intell. Robot. Syst. (IROS)*, 2014.

[49] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg. A survey of research on cloud robotics and automation. *IEEE Transactions on Automation Science and Engineering*, 12(2):398–409, April 2015.

[50] J. D. Irwin. The industrial electronics handbook. *CRC Press*, 1997.

[51] Y. Kato Y. Murakwa K. Okabayashi M. Narita, S. Okabe and S. Kanda. Reliable cloud-based robot services,. In *Proc. Conf. IEEE Ind. Electron. Soc.*, page 8317–8322, 2013.

[52] K. Goldberg. Beyond the Web: Excavating the real world via mosaic. In *Proc. 2nd Int. World Wide Web Conf.*, page 1–12, 1994.

[53] K. Goldberg and R. Siegwart. Beyond webcams: An introduction to online robots. *MIT Press*, 2002.

[54] M. Inaba. Remote-brained robots. In *Proc. Int. Joint Conf. Artif. Intell.*, page 1593–1606, 1997.

[55] M. Waibel, M. Beetz, J. Civera, R. D'Andrea, J. Elfring, D. Galvez-Lopez, K. Haussermann, R. Janssen, J.M.M. Montiel, A. Perzylo, B. Schiessle, M. Tenorth, O. Zweigle, and R. van de Molengraft. RoboEarth - A World Wide Web for Robots. *Robotics & Automation Magazine, IEEE*, 18(2):69–82, 2011.

[56] G. Mohanarajah, D. Hunziker, R. D'Andrea, and M. Waibel. Rapyuta: A cloud robotics platform. *IEEE Transactions on Automation Science and Engineering*, 12(2):481–493, 2015.

[57] R. B. Rusu, Z. C. Marton, N. Blodow, A. Holzbach, and M. Beetz. Model-based and Learned Semantic Object Labeling in 3D Point Cloud Maps of Kitchen Environments. In *Procceding of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, St. Louis, MO, USA, October 11-15 2009.

[58] D. Pangercic, M. Tenorth, B. Pitzer, and M. Beetz. Semantic Objects Maps for Robotic Housework - Representation, Acquisition and Use. In *Procceding of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012.

[59] C. Goldfeder, M. Ciocarlie, and P. Allen. The Columbia grasp dataset. In *Proc. Int. Conf. Robot. Autom. (ICRA)*, pages 1710–1716, Kobe, Japan, 2009.

[60] A. Kasper, Z. Xue, and R. Dillman. The KIT object models database: an object model database for object recognition, localization and manipulation in service robotics. *International Journal of Robotics Research (IJRR)*, 31(8):927–934, 2012.

[61] M. Tenorth and M. Beetz. KnowRob: A Knowledge processing infrastructure for cognition-enabled robots. *International Journal of Robotics Research (IJRR)*, 23(5):566–590, 2013.

[62] J. Weisz and P. K. Allen. Pose error robust grasping from contact wrench space metrics. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 557–562, St. Paul, MN, USA, 2012.

[63] M. Popovic, G. Koostra, J. A. Jorgensen, D. Kragic, and N. Kruger. Grasping unknown objects using an early cognitive vision system for general scene understanding. In *Proc. Int. Conf. Intell. Robot. Syst. (IROS)*, pages 987–994, San Francisco, California, USA, 2011.

[64] D. Di Marco, A. Koch, O. Zweigle, K. Haussermann, B. Schiessle, P. Levi, D. Galvez-Lopez, L. Riazuelo, J. Civera, J. M. M. Montiel, M. Tenorth, A. Perzylo, M. Waibel, and R. Van de Molengraft. Creating and using RoboEarth object models. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3549–3550, 2012.

[65] L. Kaufman and P.J. Rousseeuw. *Statistical Data Analysis Based on the L1-Norm and Related Methods*, chapter Clustering by means of Medoids, pages 405–416. North-Holland, 1987.

[66] V. Dubois and M. Quafafou. Concept learning with approximation: Rough version spaces. In *Rough Sets and Current Trends in Computing: Proceedings of the Third International Conference (RSCTC)*, pages 239–246, Malvern, PA, USA, 2002.

[67] Zhengjie Fan. *Concise Pattern Learning for RDF Data Sets Interlinking*. PhD thesis, University of Grenoble, 2014.

[68] S. Balakirsky, Z. Kootbally, T. RKramer, A. Pietromartire, C. Schlenoff, and S. Gupta. Knowledge driven robotics for kitting applications. *Robotics & Autonomous Systems*, 61(11):1205–14, 2013.

[69] Carmen Martinez-Cruz, Ignacio J. Blanco, and M. Amparo Vila. Ontologies versus relational databases: Are they so different? a comparison. *Artificial Intelligence Review*, 2011.

[70] Kim W. Introduction to object-oriented databases. Technical report, MIT Press, Cambridge, 1990.

[71] Subrahmanian VS Marcus S. Foundations of multimedia database systems. *J ACM*, 43(3):474–523, 1996.

[72] Valduriez P Özsu MT. Distributed database systems: Where are we now? *Computer*, 24(8):68–78, 1991.

[73] Goldfarb CF Graves M. Designing xml databases. Technical report, Prentice Hall PTR, Upper Saddle River, 2001.

[74] Agarwal P. Ontological considerations in giscience. *Int J Geogr Inf Sci*, 19(5):501–536, 2005.

[75] Sandro Rama Fiorini, Joel Luis Carbonera, Paulo Gonçalves, Vitor A.M. Jorge, Vítor Fortes Rey, Tamás Haidegger, Mara Abel, Signe A. Redfield, Stephen Balakirsky, Veera Ragavan, Howard Li, Craig Schlenoff, and Edson Prestes. Extensions to the core ontology for robotics and automation. *Robot. Comput.-Integr. Manuf.*, 33(C):3–11, 2015.

[76] M. R. Cutkosky. On grasp choice, grasp models, and the design of hands for manufacturing tasks. In *Robotics and Automation, IEEE Transactions on*, volume 5(3), page 269–279, 1989.

[77] W. J. Beksi, J. Spruth, and N. Papanikolopoulos. Core: A cloud-based object recognition engine for robotics. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 4512–4517, 2015.

[78] G. Bradski. *Dr. Dobb's Journal of Software Tools*.

[79] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *roc. of IEEE/RSJ International Conference on Intelligent Robots and Systems(IROS 2004)*, pages 2149–2154, 2004.

[80] T. Mörwald and M. Vincze. *Object modelling for cognitive robotics*. PhD thesis, Vienna University of Technology, 2013.

[81] V. Lippiello, L. Villani, and B. Siciliano. An open architecture for sensory feedback control of a dual-arm industrial robotic cell. *Industrial Robot: An International Journal*, 34(1):46–53, 2007.

[82] K. Oonishi. The open manipulator system of the mhi pa-10 robot. *Proceeding of International Symposium on Robotics*, 1999.

[83] A. Blomdell, G. Bolmsjo, T. Brogardh, P. Cederberg, M. Isaksson, R. Johansson, M. Haage, K. Nilsson, M. Olsson, T. Olsson, A. Robertsson, J. Wang, and Department of Automation of Lund University. Extending an industrial robot controller: implementation and applications of a fast open sensor interface. *Robotics and Automation Magazine*, 12, 2005.

[84] F. Dogliani, G. Magnani, and L. Sciavicco. An open architecture industrial controller. *Newsl. of IEEE Robotics and Automation Soc.*, 7(3):19–21, 1993.

[85] E. Venator. Hardware and software architecture of abby: An industrial mobile manipulator. In *9th IEEE International Conference on Automation Science and Engineering*, August 2013.

[86] M. Freese, S. Singh, F. Ozaki, and N. Matsuhira. Virtual robot experimentation platform v-rep: a versatile 3d robot simulator. In *Proceedings of the Second international conference on Simulation, modeling, and programming for autonomous robots, SIMPAR'10*, page 51–62, Berlin, Heidelberg, 2010. Springer-Verlag.

[87] M. Fischler and R. Bolle. Random sample consensus. a paradimg for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6), 1981.