UNIVERSITÀ DEGLI STUDI
DI GENOVA

ISTITUTO ITALIANO
DI TECNOLOGIA

UNIVERSITY OF GENOA

DOCTORAL THESIS

Toward Robots with Peripersonal Space Representation for Adaptive Behaviors

---

Author:
Nguyen Dong Hai Phuong

Supervisors:
Prof. Giorgio Metta
Dr. Ugo Pattacini
Dr. Matej Hoffmann

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

iCub Facility
Istituto Italiano di Tecnologia (IIT)

Declaration of Authorship

I, Nguyen Dong Hai Phuong, declare that this thesis titled, "Toward Robots with Peripersonal Space Representation for Adaptive Behaviors" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at Istituto Italiano di Tecnologia and University of Genoa.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Abstract

Nguyen Dong Hai Phuong

Toward Robots with Peripersonal Space Representation for Adaptive Behaviors

The abilities to adapt and act autonomously in an unstructured and human-oriented environment are necessarily vital for the next generation of robots, which aim to safely cooperate with humans. While this adaptability is natural and feasible for humans, it is still very complex and challenging for robots. Observations and findings from psychology and neuroscience in respect to the development of the human sensorimotor system can inform the development of novel approaches to adaptive robotics.

Among these is the formation of the representation of space closely surrounding the body, the Peripersonal Space (PPS) , from multisensory sources like vision, hearing, touch and proprioception, which helps to facilitate human activities within their surroundings.

Taking inspiration from the virtual safety margin formed by the PPS representation in humans, this thesis first constructs an equivalent model of the safety zone for each body part of the iCub humanoid robot. This PPS layer serves as a distributed collision predictor, which translates visually detected objects approaching a robot's body parts (e.g., arm, hand) into the probabilities of a collision between those objects and body parts. This leads to adaptive avoidance behaviors in the robot via an optimization-based reactive controller. Notably, this visual reactive control pipeline can also seamlessly incorporate tactile input to guarantee safety in both *pre-* and *post-*collision phases in physical Human-Robot Interaction (pHRI). Concurrently, the controller is also able to take into account multiple targets (of manipulation reaching tasks) generated by a multiple Cartesian point planner. All components, namely the PPS, the multi-target motion planner (for manipulation reaching tasks), the reaching-with-avoidance controller and the human-centred visual perception, are combined harmoniously to form a hybrid control framework designed to provide safety for robots' interactions in a cluttered environment shared with human partners.

Later, motivated by the development of manipulation skills in infants, in which the multisensory integration is thought to play an important role, a learning framework is proposed to allow a robot to learn the processes of forming sensory representations, namely visuomotor and visuotactile, from their own motor activities in the environment. Both multisensory integration models are constructed with Deep Neural Networks (DNNs) in such a way that their outputs are represented in motor space to facilitate the robot's subsequent actions.

Acknowledgements

The last three years have been one of the most amazing experiences of my life. I am grateful to everyone who has helped me to walk through this path and enjoyed it with me.

First of all, I would like to thank my family, who have always been supportive and kind to me. In particular, I want to thank my father, who gave me the passion for automation and robotics and encouraged me to pursue knowledge; I want to thank my mum, who has devoted her whole life for her family and given me moral advice for life; I want to thank my wife, who has been with me all the time without hesitation, shared with me the difficulties, encouraged me and offered me useful recommendations; I want to thank my parents-in-law, who have encouraged and been supportive me; and I want to thank my brother, whose different views have trained me to think through my decisions carefully at all times.

I also want to thank all of my friends who have made the time enjoyable: thank you Lugo, Prashanth, Vishal, Anand, Ninad for all the time we have shared together, especially our BBQ, Indian, Mexican or Vietnamese dinners. Thank you to all my friends in the SECURE network for being an important part of my last three years: Sven, Alexis, Francoise, Marie, Alexa, Egor, Mohammad, Chandrakan, Owen, Bruno. Thank you to my close friends who are living far away but who remain always supportive and helpful: Phuoc, Nguyen.

I am also extremely grateful to all the members of the iCub Facility, IIT, for their support, discussion, inspiration and friendliness. Thank you to all the mechanical design, electronics and support teams who helped to create and maintain our iCub, one of the most complex robotics systems in the world; thank you Matteo for keeping our ICT system working well; thank you Julien–iCub doctor–for treating the Blackie well every time I or my friend "hit" him hard, and when we asked him to run overtime for our deadlines; thank you Vadim for help with the iCub simulator and python-binding; thank you Roncone for all the help and the initial contributions on PPS, react-control and iCub; thank you Rafaello for discussions and references about model learning; thank you Silvio, Romano and Nori for giving me the opportunity in the first place; thank you Andrea, Lucia, Grazia and Chiara for all your help and support with admin and paperwork stuffs. Thank you Arman for all the nonsense discussions we had together; thank you Tanis for early help and recommendations to understand iCub and YARP; thank you Giulia P for the advice and explanations regarding the complicated iCub computer vision systems and Caffe; thank you Nicolo and Ali for helping me to understand YARP better and for solving all its bugs. Especially, thank you Alex, Massimo and Alessia for our friendships and joys; thank you Bertrand, my chitchat mate, for all the discussions we have had about Deep learning, cognitive sciences, neurosciences and the world; thank you Vale for being my coffee and "pausa" mate, who helps me know more about Italy, pizza and coffee; thank you Elisa for our Deep learning discussions, for sharing the GPU servers and the corner with me all the time; thank you Giulia V for the "annoying" event organizations; thank you Max, Fabrizio and Marco to bring new energy sources to the lab.

I also want to thank members of HRII lab, IIT, especially Arash, Wansoo, Pietro, Marta, who spent the unforgettable period of the KUKA Innovation Award with me. It was a tough but enjoyable and rewarding time. I also want to thank all members of WYSIWYD project from SPECS, Imperial, Sheffield and Lyon who spent an enjoyable time with me and taught me a lot about cognitive sciences and project integration.

I want to thank friends at the Personal Robotics Lab, Imperial College London, who gave me an amazing time during my internship in the lab. In particular, I am grateful to Prof. Yiannis Demiris for giving me the chance to be there; thank you Tobi, Fan, Mark and Ahmed for teaching me to play squash; thank you Tobi, Ahmed and Cat for coffee time. And extremely thank you Tobi and Hyung Jin for all the discussions and help on making my robot learn. It has been my pleasure to work with both of you.

Lastly, I would like to thank the members of my supervisory committee, not just for being my supervisors but for having been my mentors along the way. Thank you Matej for introducing me to PPS and neurosciences and for the contributions on PPS and react-control. In particular, thank you Ugo for teaching me iCub, YARP, github; and for pushing me to make things better, but with care; and for giving me helpful advice and support during the long journey. Thank you Prof. Giorgio Metta for giving me the opportunity to undertake this journey and for providing me with continuous support.

Again, a big thank to all the people that I did not mention here personally. This thesis could not have been completed without a lot of help, it is dedicated to all of you.

Finally, this thesis has been supported by a Marie Curie Fellowship (H2020-MSCA-ITA, SECURE 642667) from the European Commission.

List of Publications

1. Phuong D. H. Nguyen, Tobias Fischer, Hyung Jin Chang, Ugo Pattacini, Giorgio Metta, and Yiannis Demiris. 2018. "Transferring Visuomotor Learning from Simulation to the Real World for Robotics Manipulation Tasks." In 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, 6667–6674. IEEE.

2. Phuong D. H. Nguyen, Fabrizio Bottarel, Ugo Pattacini, Matej Hoffmann, Lorenzo Natale, and Giorgio Metta. 2018. "Merging Physical and Social Interaction for Effective Human-Robot Collaboration." In Humanoid Robots (Humanoids), 2018 IEEE-RAS 18th International Conference On, 710–717. IEEE.

3. Phuong D. H. Nguyen, Matej Hoffmann, Alessandro Roncone, Ugo Pattacini, and Giorgio Metta. 2018. "Compact Real-Time Avoidance on a Humanoid Robot for Human-Robot Interaction." In Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction, 416–424. ACM.

4. Phuong D. H. Nguyen, Matej Hoffmann, Ugo Pattacini, and Giorgio Metta. 2016. "A Fast Heuristic Cartesian Space Motion Planning Algorithm for Many-DoF Robotic Manipulators in Dynamic Environments." In Humanoid Robots (Humanoids), 2016 IEEE-RAS 16th International Conference On, 884–891. IEEE.

5. Phuong D. H. Nguyen, Matej Hoffmann, Ugo Pattacini, and Giorgio Metta. 2019. "Reaching development through visuo-proprioceptive-tactile integration on a humanoid robot – a deep learning approach" (submitted)

6. Tobias Fischer, Jordi-Ysard Puigbò, Daniel Camilleri, Phuong D. H. Nguyen, Clément Moulin-Frier, Stéphane Lallée, Giorgio Metta, Tony J. Prescott, Yiannis Demiris, and Paul F. M. J. Verschure. 2018. "iCub-HRI: A Software Framework for Complex Human–Robot Interaction Scenarios on the ICub Humanoid Robot." Frontiers in Robotics and AI 5, 22.

7. Clément Moulin-Frier, Tobias Fischer, Maxime Petit, Gregoire Pointeau, Jordi-Ysard Puigbò, Ugo Pattacini, Sock Ching Low, Daniel Camilleri, Phuong D. H. Nguyen, Matej Hoffmann, Hyung Jin Chang, Martina Zambelli, Aanne-Laure Mealier, Andreas Damianou, Giorgio Metta, Tony J. Prescott, Yiannis Demiris, Peter Ford Dominey, and Paul F. M. J. Verschure. 2017. "DAC-H3: A Proactive Robot Cognitive Architecture to Acquire and Express Knowledge About the World and the Self." IEEE Transactions on Cognitive and Developmental Systems.

8. Kim Wansoo, Marta Lorenzini, Pietro Balatti, Phuong D. H. Nguyen, Ugo Pattacini, Vadim Tikhanoff, Luka Peternel, et al. 2018. "A Reconfigurable and Adaptive Human-Robot Collaboration Framework for Improving Worker Ergonomics and Productivity." IEEE Robotics and Automation Magazine (accepted).

Contents

List of Figures

List of Tables

Acronyms

Notation	**Description**
AI	Artificial Intelligence. 65, 67
ANN	Artificial Neural Network. 31, 33
DNN	Deep Neural Network. vi, xxvi, 16, 33, 35, 47, 69, 70, 116, 118, 119, 130, 134, 136, 137, 146, 152, 153, 154, 155
DoF	Degree-of-Freedom. 20, 21, 36, 55, 63, 69, 81, 83, 94, 101, 111, 134
FoR	Frame-of-Reference. xxiii, 5, 11, 70, 90, 91, 92, 134, 136, 160
GPU	graphics processing unit. 47, 137
HRI	Human-Robot Interaction. 43, 55, 66, 70, 79, 80, 152
LSTM	Long Short-term Memory. 146, 154
pHRI	physical Human-Robot Interaction. v, 24, 43, 45, 46, 65, 66, 68, 127, 151, 152
PPS	Peripersonal Space. v, xix, xx, xxi, xxvii, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 15, 23, 43, 45, 48, 49, 52, 53, 55, 57, 60, 61, 68, 71, 72, 82, 127, 134, 151, 152, 153, 154, 155

Notation	**Description**
PRM	Probabilistic Roadmap. 7, 11, 99, 100, 124, 146
RGB	Red-Green-Blue. 11, 36, 111
RGB-D	Red-Green-Blue-Depth. 11, 45
RRT	Rapidly-exploring Random Tree. 83, 84, 86, 91, 95, 98, 99, 101, 124, 146
sHRI	social Human-Robot Interaction. 24, 65, 67, 68
SPA	Sense-Plan-Act. 17
UML	Unified Modeling Language. xix, 24
YARP	Yet Another Robot Platform. 39, 46, 47, 84, 95, 99

Part I

Introduction

Chapter 1

Introduction

1.1 Motivation

The abilities to adapt and act autonomously in an unstructured and human-oriented environment are vital for the next generation of robots, which aim to safely cooperate with humans. In other words, we want robots to operate in the same situations and conditions as humans do, to use the same tools, to interact with and to understand the same world in which humans' daily lives take place. In achieving this, we want robots to be able to learn autonomously how to behave appropriately in our world and thus to find smart ways to cope with the challenges posed by our dynamic and unstructured environment, rather than providing a rigid set of rules to drive their actions. Robots should be able to deal efficiently with unexpected changes in the perceived environment as well as modifications of their own physical structure. While this adaptability is natural and feasible for humans, it is still very complex and challenging for robots. In this regard, exploiting findings from psychology and neuroscience on the development of the human sensorimotor system can open the way to novel approaches to adaptive robotics. Following this principle, my thesis focuses on investigating and advancing the representation of the space near the robot body – i.e. the so-called Peripersonal Space (PPS) – that is where the interaction with the surroundings occurs, and how that PPS can purposely modulate the humanoids' sensorimotor capabilities.

1.2 Peripersonal Space

1.2.1 Peripersonal space and psychological findings

Many neuroscientific findings show that various multisensory integration processes occur in humans to represent the space close to the body, in

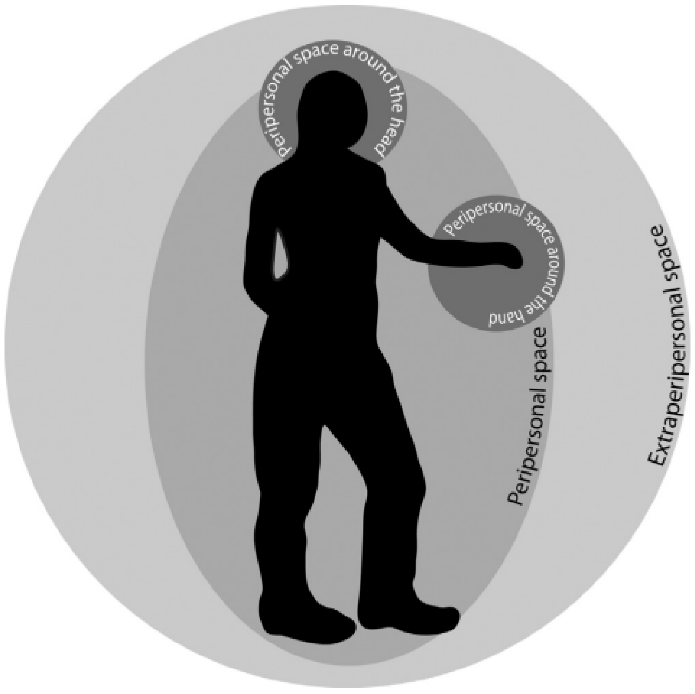

Peripersonal space around the head
Peripersonal space around the hand
Peripersonal space
Extraperipersonal space

FIGURE 1.1: Peripersonal space of body parts, from [Cléry et al., 2015] with permission.

which humans can directly act. This space is termed Peripersonal Space (PPS) [Cléry et al., 2015]. The PPS serves as a "safety margin" to facilitate the manipulation of objects [Holmes and Spence, 2004, Goerick et al., 2005] and to ease a variety of human actions such as reaching and locomotion with obstacle avoidance [Holmes and Spence, 2004, Làdavas and Serino, 2008]. Notably, this is not the case for the space farther from the human body [Farnè et al., 2005].

The defensive PPS representation is maintained by the brain in the form of a network of neurons with visuo-tactile receptive fields (RFs) attached to different body parts, following them as they move (see e.g. [Cléry et al., 2015] for a recent survey). This forms a distributed and very dense coverage of the "safety margin" around the whole body. It should be explicitly stated that the PPS representation is composed of many different sub-representations corresponding to different body parts, i.e. hands, arms, trunk, face (as shown in Figure 1.1); rather than a unique space for the whole body. The PPS representation of a body part is closely coupled with that part even in movement – a very useful property for obstacle avoidance. When a body part moves, its PPS representation will be modified independently from other body parts' representations, thus eliciting adaptive behaviors (e.g. reactive mechanisms) for only that specific body part within its environment. That said, Cléry et al. [2015] suggests that the separated PPS representations of body parts can interact and merge, depending on their relative positions. Besides, this protective safety zone is dynamically adapted to the action that the agent is performing—e.g. reaching vs. grasping [Brozzoli et al., 2010] and also modulated by the state of the agent or by the identity and the "valence" (positive or negative) of the approaching object —e.g. safety zones centered around empty vs. full glasses of water [de Haan et al., 2014] or reaction times to spiders vs. butterflies [de Haan et al., 2016]. Furthermore, the social and emotional cues of interaction contexts also cause dynamic adjustment of the PPS representation [Teneggi et al., 2013, Lourenco et al., 2011].

Moreover, this spatial representation is incrementally trained and adapted (i.e. expanded, shrunk, enhanced, etc.) through motor activities, as reported in, among others [Cléry et al., 2015, Làdavas and Serino, 2008, Serino et al., 2015a].

These results suggest that by exploiting motor activities in exploratory tasks, agents can, on the one hand, develop their perception of the space around their bodies, and on the other hand use the spatial representation they have built to improve the quality of their motor skills.

1.2.2 A possible neuronal pathway of Peripersonal Space

As collectively reported in [Cléry et al., 2015], the neuronal network of *parieto-premotor* areas of the cortex plays a vital role in PPS representation. In fact, PPS encoding neurons are found to be stimulated in the following regions in primate brains (when there is tactile stimulus triggered on the skin or visual stimulus close to a body part). It is worth noting that in neurosciences, most of data for the brain-related experiments comes from monkeys.

- Ventral intraparietal area (VIP): where the multisensory (i.e. audio-visuo-tactile) integration takes place. In this particular case, this area encodes visual information in an eye-to-head FoR and tactile information in a head-centred FoR;
- Parietal area 7b: where the sensory homunculus [1] is presented. About a third of the cells in this area process bimodal–integrating information from visual and tactile receptive fields;
- Premotor cortex: Including the areas F4 and F5.

[1]an area in the brain where each sub-region corresponds with an area in the agent's body

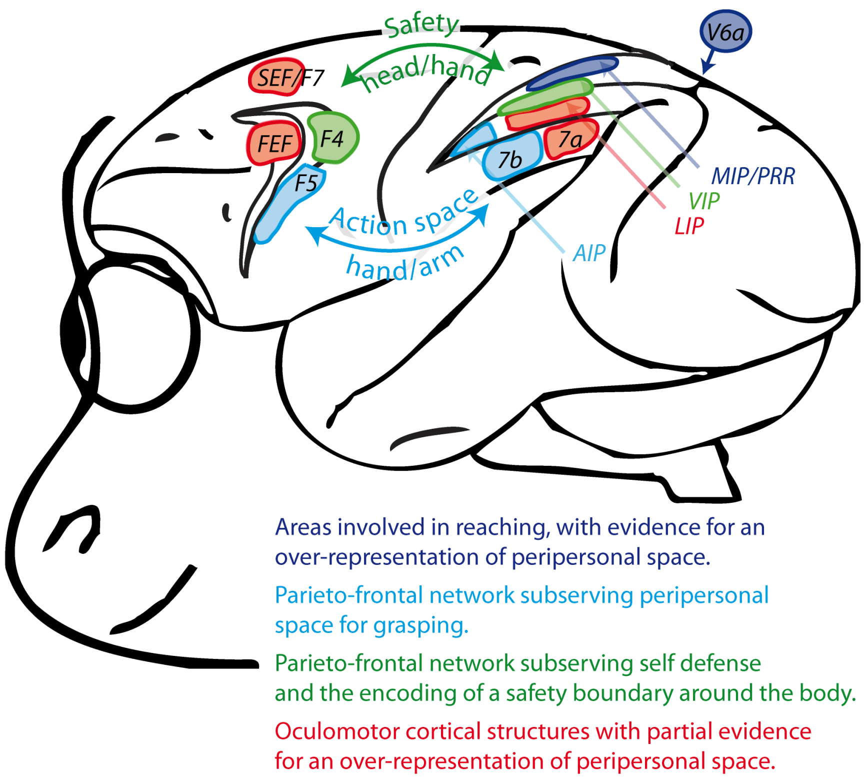

Safety
head/hand
SEF/F7
FEF
F4
F5
Action space
hand/arm
V6a
7b
7a
MIP/PRR
VIP
LIP
AIP
Areas involved in reaching, with evidence for an over-representation of peripersonal space.
Parieto-frontal network subserving peripersonal space for grasping.
Parieto-frontal network subserving self defense and the encoding of a safety boundary around the body.
Oculomotor cortical structures with partial evidence for an over-representation of peripersonal space.

FIGURE 1.2: Neuronal pathways form the PPS representation, from [Cléry et al., 2015] with permission.

The interactions between these different regions in the brain create distinct (but not fully isolated) networks for sub-PPS representations corresponding to different body parts. The face representation is mainly related to neurons in the parietal VIP and premotor F4 network (green areas and path in Fig. 1.2), while the arm/hand representation is more marked in parietal 7b and the premotor F5 network (red areas and path in Fig. 1.2).

The VIP-F4 network processes all the information necessary to bind together the localization of objects around the body, particularly around the head, with actions towards these objects aimed generally at defence and avoidance. While the parietal area is more involved in the construction of the perception of the surroundings, however, the premotor area focuses more in generating defensive motor responses. Notably, observations of neuron firing also suggest that the PPS defence network is also involved in the prediction of intrusive impacts to the body.

The 7b-F5 network shares sensory visual, tactile and motor (i.e. hand-related) properties, and serves a core role in planning and executing actions (e.g. grasping) within the reachable space. In particular, this network specializes in visuomotor transformation to effect actions in the environment.

In addition to the core PPS representation, which is formed via training and learning, there is a dynamically modulated PPS representation that is adjusted subject to the cognitive context. The following are aspects that contribute to this dynamic PPS modulation:

- Action dependence: Involving the 7b-F5 parieto-premotor network;
- Inferred sensations: Involving the VIP-F4 parieto-premotor network;
- Positional interactions: Involving the VIP-F4 parieto-premotor network;
- Social and emotional context: modulation of other PPS networks.

1.3 A review of computational and robotics models of PPS

The following section provides an overview of the research related to computational and robotics models of the PPS representation. The main differences between the approaches considered here are outlined in Table 1.1, which is constructed accounting for the following criteria: computation model for the PPS representation, sources of sensory information, agent's body, and learning approach (i.e. model-based or model-free, autonomous or not).

Model	Sensory information	Means to formulate PPS	Agent's body	Learning method
Magosso et al. [2010b]	visual & tactile	unimodal & multimodal neural networks	no	no
Magosso et al. [2010a], Serino et al. [2015a]	audio & tactile	unimodal & multimodal neural networks	no	synthesized (only in the former)
Straka and Hoffmann [2017]	visual & tactile	RBM & fully-connected neural network	no	synthesized

Roncone et al. [2016]	visual & tactile	distributed visual RFs	iCub humanoid robot	model-base, human support & self-touch
Juett and Kuipers [2016, 2018]	visual & proprioceptive	Probabilistic Roadmap (PRM)-like graph	Baxter robot	model-free, motor babbling
Antonelli et al. [2013], Chinellato et al. [2011]	visual & proprioceptive	RBF network	Tombatossals humanoid robot	model-base vision & model-free robot's actions, gazing & reaching
Ramírez Contla [2014]	visual & proprioceptive	fully-connected neural network	iCub simulator	model-base vision & model-free robot's actions, body modification

TABLE 1.1: Comparison between previous PPS models

1.3.1 Computational models of PPS

Magosso et al. [2010b] propose and analyse a neural network-based model to deal with visuotactile stimuli for the PPS representation. This model is composed of two identical networks, corresponding to the two hemispheres of the brain (i.e. left vs. right), where each is composed of unimodal neurons for input (i.e. visual and tactile stimuli) and multimodal neurons for multisensory integration. Feedforward synapses connect unimodal to multimodal

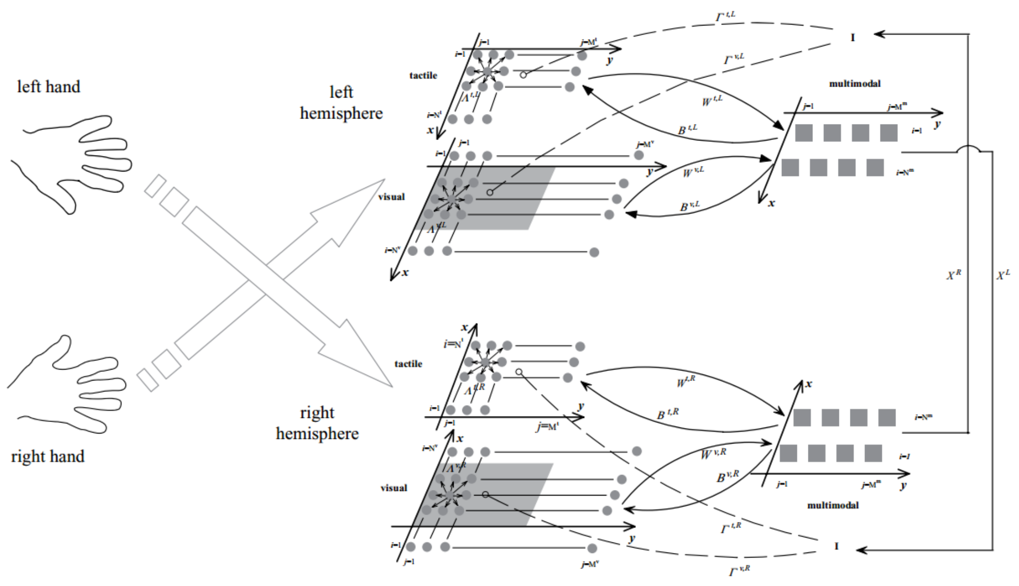


FIGURE 1.3: Magosso et al's artificial neural network for PPS, from [Magosso et al., 2010b]

neurons whereas feedback synapses establish connections in the reverse direction. Then, weights are assigned to these synapses so as to model the neuronal connections (see Fig. 1.3). It is worth noting that inhibitory connections also exist between the left and right hemisphere networks so as to model their mutually inhibiting relations. In other words, when one hemisphere activates, the other one will be to an equal extent inhibited. This brain-like construction allows to model the behaviour of the PPS at physical level and to be compared with data collected from humans. Similar models are proposed for the case of audiotactile stimuli in [Serino et al., 2015a] and [Magosso et al., 2010a].

These models are validated by considering the similarities between their computational responses with stimuli and the responses in humans' neural circuitry. Although the reported results are significant, the models were only tested without a body and only in a simple static scenario, assuming body parts to be still. Moreover, the authors did not design a training procedure, except for the tool-use case presented in [Magosso et al., 2010a].

Similarly, Straka and Hoffmann [2017] construct a computational model for PPS representation to associate visual and tactile stimuli in a simulated 2D scenario, where a synthesized object approaches a simulated "skin area". The inputs of the model are position and velocity, with the uncertainties surrounding the visually detected object encoded as a Gaussian distribution by probabilistic population code [Ma et al., 2006], while the output is the point

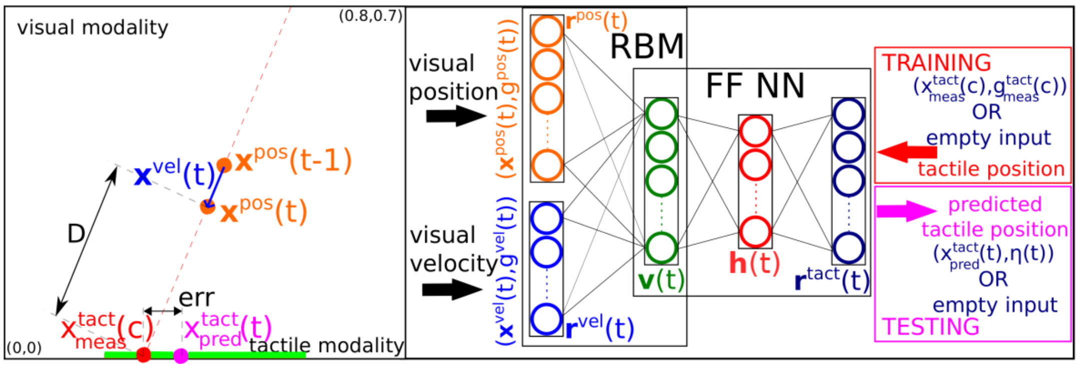


FIGURE 1.4: Straka et al.'s PPS model (right) and experiment scenario (left), from [Straka and Hoffmann, 2017]

on the body surface that will be hit. Authors propose a model composed of Restricted Boltzmann Machine [Smolensky, 1986] for object properties association (i.e. position and velocity) and a two-layer fully-connected artificial neural network for "temporal" prediction. As a result, the model is capable of predicting the collision position, given the visual stimulus. The prediction properties are also analysed, namely the error and confidence w.r.t. the stimulus distance. The designed scenario remains quite simple, however, since it boils down to simply a simulation in 2D space: the skin area is a line and there is no concept of the body, hence no transformation between sensory frames are taken into account. Besides, the velocity input is derived from the observed position of a object, thus it can be highly affected by the sensor noise in real scenarios.

1.3.2 Robotics models of PPS

Modeling PPS as a collision predictor with iCub's vision and skin

Roncone et al. [2015, 2016] propose a model to investigate an integrated representation of the artificial visual and tactile sensors (see Fig. 1.5) in the iCub humanoid robot (cf. Section 2.3). This is a representation of the protective safety zone for the iCub. The authors chose a distributed representation, in which every skin taxel (i.e. tactile element of the robot's artificial skin, Fig. 2.13) learns a collection of probabilities regarding the likelihood of objects from the environment coming into contact with that particular taxel. This is achieved by making associations between visual information, as the objects are seen approaching the body, and actual tactile information as the objects eventually physically contact the skin. A volume was chosen to represent the visual receptive field (RF) around every taxel: a spherical sector

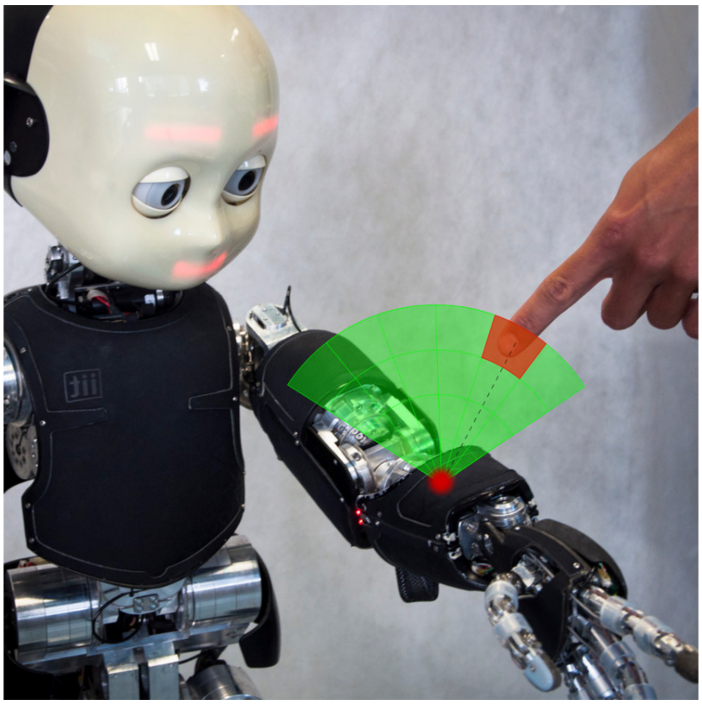


FIGURE 1.5: Schematic illustration of PPS receptive fields on the robot. There are five such receptive fields on the palms/hands and 24 around each of the forearms. From [Roncone et al., 2015] with the permission of authors.

growing out of every taxel along the normal to the local surface (presented in Fig. 1.6). The outcome is a visual collision predictor for objects close to a robot's body, which is constructed by visuo-tactile contingency. This model can be used for a simple reaching/avoidance controller.

This methodology, however, relies on a well-structured visual tracker for data collection, and on the *a priori* knowledge of the robot kinematic model in order to transform the FoR between the different sensory sources (e.g. the transformation from the camera or taxel FoR to the robot's Root FoR), rather than via autonomous learning from the raw signals.

Modeling PPS as a graph

In contrast, Juett and Kuipers [2016, 2018] model the PPS representation as a PRM-like graph (including nodes connected by edges) in the robot's reachable space through the robot's controlled motor babbling. Each node in the graph is composed of inputs from joint encoder values- (i.e. proprioceptive input) and images (i.e. visual input; from three different RGB cameras in the former work or from a single RGB-D camera in the latter one). The sampling size is chosen so that each edge is a feasible transition of the robot arm

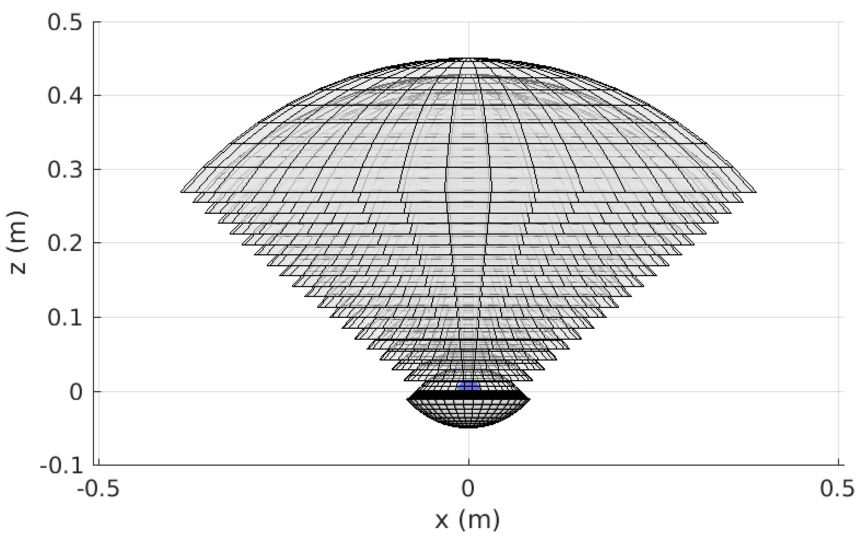


FIGURE 1.6: Schematic illustration of a single PPS receptive field on the robot.

between two connected nodes (state). Based on the learned graph, a search algorithm can be applied to find the shortest path connecting the current and the final state. The final state is defined as the reaching state if it is a learned node in the graph that contains stored image(s) overlapping with the input image(s) perceived at the planning stage. In their most recent work, the final state search algorithm is extended to allow grasping objects.

Although the learning procedure enables the robot to learn the graph without any kinematics knowledge, the authors utilize some image segmentation techniques to locate the robot's gripper (during the learning phase) and the targets (in the action phase) from input image(s). Requiring each node in the graph to store images is a memory intensive solution, however, especially when the size of the graph increases (i.e. a more densely sampled graph). Such a dense graph can also increase the search time to find the optimal solution.

Implicit mapping of PPS in a humanoid robot

Antonelli et al. [2013] and Chinellato et al. [2011] adopt radial basis function networks [Broomhead and Lowe, 1988] to construct the mapping (forward and inverse transformations) between stereo visual data and proprioceptive

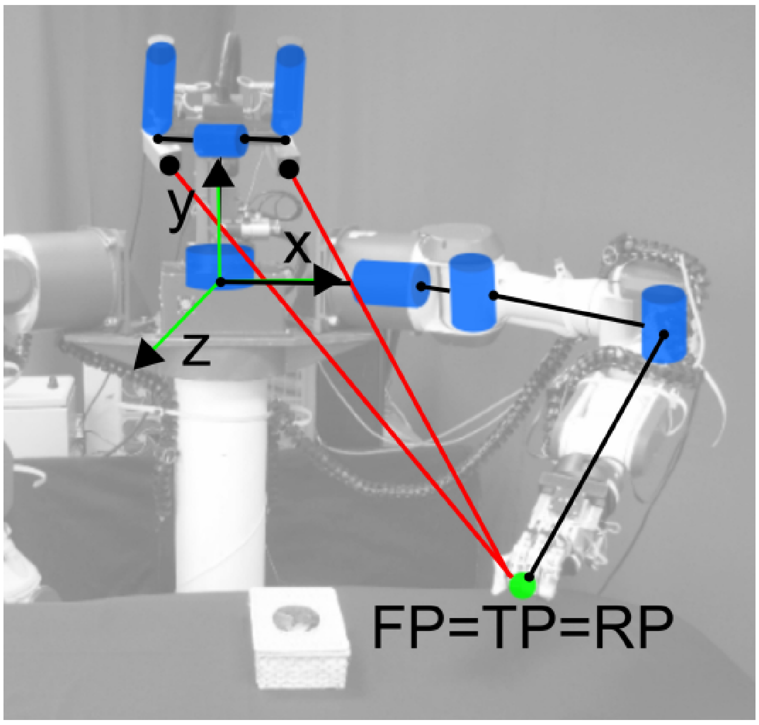


FIGURE 1.7: Robot learns a visuomotor transformation by gazing and reaching the same object, from [Antonelli et al., 2013], [Chinellato et al., 2011].

data in a robot platform (as shown in Fig. 1.7). This is conducted through the robot's gazing and reaching activities within the reachable space, which they define as the PPS. Their mapping, however, requires markers to extract features with known disparity (rather than being estimated from raw signals), and is apparently beneficial only for multi-sensory transformation and not as a spatial perception of the body's surroundings.

Modelling PPS plasticity in a humanoid robot

On the other hand, Ramírez Contla [2014] focuses on the plastic nature of PPS representation to account for the modification the body undergoes, and the impact of this plasticity on the confidence levels in respect to reaching activities. In their experiments, the author first assesses the contribution of visual and proprioceptive data to reaching performance, then measures the contribution of posture and arm-modification to reaching regions. The modifications applied to the arm (i.e. the changes in the arm's length, shown in Fig. 1.8) have similar effects on tool-use in terms of the extension of the PPS representation. The hypothesis is only validated in a simulated environment, however.

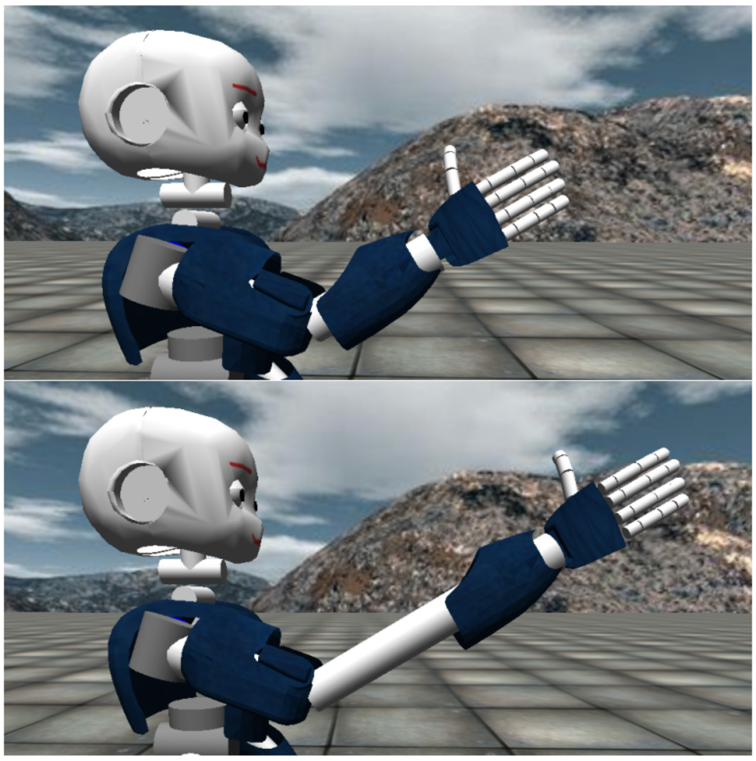


FIGURE 1.8: Simulated iCub with deformed arm, from [Ramírez Contla, 2014]

1.4 Contributions of this thesis

The discussion of the influences of the PPS representation on human motor activities, and the formation of this representation from a variety of sensory information poses two key questions to robotics researchers regarding the synthesis that needs to be investigated:

- How does a well-developed PPS representation help robots interact with the environment naturally and comfortably? In other words, can a representation enriched with the integration of sensory information facilitate robot movements in cluttered and dynamic environments?
- How can the PPS representation be constructed from multisensory information, i.e. vision, tactile and proprioception, through the robots' own actions performed in the environment? Are robots able to learn the association between their different sensors from events happening during the interaction with the environment?

In the remaining part of this thesis, I will focus on finding the answers for these two core questions, which are then aggregated into the two main contributions of this thesis:

- Following the integration technique to construct a robotics control system (see Section 2.1.1), I propose an implementation utilizing a PPS representation as a core perception layer. This control system aims to provide robots with the flexibility for safe interaction (with humans) in a cluttered and unstructured environment. It is conducted under the assumption that the robots' kinematics models are *a-priori* known, thus such a design falls under the category of model-based approaches.
- Pursuing the embodiment approach (see Section 2.2.3), the second contribution is a framework to allow robots to learn the PPS representation from a variety of sensory sources in their bodies through their own motor activities. Unlike the first contribution, the learning framework is constructed without prior knowledge of the robots' kinematics models.

These two contributions are organized into two parts, namely **Part II: Model based approach – PPS in interaction** and **Part III: Learning approach – Towards learning PPS through robot actions**, respectively. While the former part contains three chapters, 3, 4 and 5, aiming to build the control system with PPS representation as a core perception layer, the latter part consists of two chapters, 6 and 7, pursuing a learning framework to allow learning PPS representation through robots' actions. The following are the main contents of each chapter:

- **Chapter 2: Background and Experimental setup** provides an overview of the theory underlying my two main contributions, namely the robotics control system and learning by interaction. This chapter also briefly discusses the robot platform, the iCub humanoid robot, which I utilize for the experiments in this thesis.
- **Chapter 3: Compact real-time avoidance for a humanoid robot for human-robot interaction** presents a framework on iCub robot that dynamically maintains a protective safety zone around the robot's body against visually detected obstacles, composed of the following main components: (i) a human keypoints estimation pipeline employing a deep learning-based algorithm and the visual disparity of the stereo-vision system; (ii) a distributed PPS representation around the robot's body parts; (iii) a reactive controller that incorporates into the task all obstacles entering the robot's safety zone on the fly. This framework ensures safe robot movement during the *pre*-collision phase–i.e. predicting the possibility of collisions and avoiding them.

- **Chapter 4: Merging physical and social interaction for effective human-robot collaboration** extends components of the framework proposed in Chapter 3 further to detect objects within the vision pipeline (besides human detection) and to incorporate physical interaction through the robot's artificial skin system. The complete framework enables the robot's safe interaction in both *pre-* and *post-*collision phases, as well as in both physical and social interaction.

- **Chapter 5: Motion planning algorithm for robotic manipulators** proposes a fast heuristic planning method, designed for humanoid robots, which employs the sampling-based RRT* algorithm directly in the Cartesian space for all representative controlled points in the robot's manipulator. The motion path for the manipulator consists of paths for all controlled points, which can guide a multiple target reaching controller to safely bring the manipulator to its goal, e.g. reaching an object on a table. It is worth noting that this is an extension of the same reactive controller developed in Chapter 3 and Chapter 4.

- **Chapter 6: Learning Visuomotor mapping and Transferring learning from Simulation to the Real World** solves the hand-eye coordination task using a visuomotor DNN predictor that estimates the arm's joint configuration given a stereo image pair of the arm and the underlying head configuration. Since there are various unavoidable sources of sensing error on the physical robot, the predictor is trained on data obtained by robot motor babbling in simulation. The learned knowledge is then transferred from simulation to real robot with a domain adaptation framework, based on an image-to-image translation method.

- **Chapter 7: Learning multisensory representation for manipulation tasks** extends the visuomotor DNN model in Chapter 6 to also include tactile information and solves the multisensory integration for spatial representation task as a multitask learning problem. Consequently, given a pair of stereo-vision images of an object and the robot's head configuration, the output of the multisensory DNN is a set of arm-chain configurations allowing the robot to reach the object at different positions. Similar to the previous chapter, the model is learned with a dataset generated in simulated environment with robot's own motor movements, and then aims to be transferred to real robots by applying domain randomization technique.

Chapter 2

Background and Experimental setup

2.1 Overview of robot control system theory for manipulation tasks

2.1.1 Robot control system architecture

Robot control is the process of perceiving information from the environment with the sensors equipped in the robot systems (Sensing), building an internal model of the environment, reasoning about the actions that need to be conducted in order to accomplish the task (Planning), and then executing actions in the environment (Acting) [Matarić and Michaud, 2008]. The more complex the environment is, the higher the complexity of the control system.

Although there are very many ways to construct a robot control system, it is possible to classify them into four main types, namely Reactive systems, Deliberative systems, Hybrid systems and Behaviour-based systems [Matarić and Michaud, 2008].

The most common approach is *Deliberative*, which divides the control architecture into three functional elements of Sensing, Planning and Acting. This approach is also called the Sense-Plan-Act (SPA) paradigm (see Fig. 2.1), and its main feature is the single direction of the information flow from Sensing to Acting, i.e. the robot's actions are generated based on the internal model that has been built by the sensing data, instead of directly from the sensors themselves. Thus the internal model needs to be kept continuously up-to-date and accurate. If sufficient time is given, the Deliberative approach allows the robot to reason about the optimal solution to act for a certain situation. For complex tasks, however, robot systems need to interact with uncertainty and a dynamic environment in an asynchronous and real-time manner. This poses a problem for the SPA paradigm, which (i) requires a long planning time, leading to a suspension of the robot's action, and (ii) cannot guarantee safety in executing actions in a dynamic world due to the lack of sensory updates to the internal model [Kortenkamp and Simmons, 2008].

Unlike the Deliberative approach, *Reactive* control relies directly on the sensed information to generate corresponding actions for the robot, allowing the robot to respond quickly to the dynamic changes in the environment. In this architecture, there is no internal model of the environment constructed from sensing, thus this approach trades off the optimality of the planned actions (in the Deliberative approach) for a more prompt and rapid response. This is achieved by constructing the robot control with sets of minimal internal state and computation condition-action rules. Nevertheless, this construction drastically reduces the ability of the system to improve its performances over time through learning from past experiences.

Combining the advantages of both Deliberative and Reactive approaches, *Hybrid* control is designed in such a way that it can guarantee the optimality of the former and the responsiveness of the latter. In other words, deliberative components act on a longer time scale in respect to global tasks, utilizing the internal representation of robot systems about the world, whereas reactive components ensure that robots react immediately to the changes in the environment occurring within short time frames. An example of this hybrid approach is the manipulation control system (described in detail in the next Section), where the manipulation planning modules are the deliberative component and the controller is the reactive one. Bridging these two naturally conflicting approaches, however, often requires an intermediate component serving as coordinator between them, which is known as a *three-layer architecture*, for example 3T [Bonasso et al., 1996], ATLANTIS [Gat, 1992] and LAAS [Alami et al., 1998].

The *Behaviour-based* control system, meanwhile, is constructed by a collection of distributed modules, each receiving sensory inputs, maintaining its own representations and reasoning about action to feed to actuators or other modules [Matarić and Michaud, 2008]. Also, each module is designed to form a certain behaviour or functionality of the robot system. The interaction between different behaviours in this control architecture helps the robot system achieve its goal in dynamic environments. It is worth noting that each

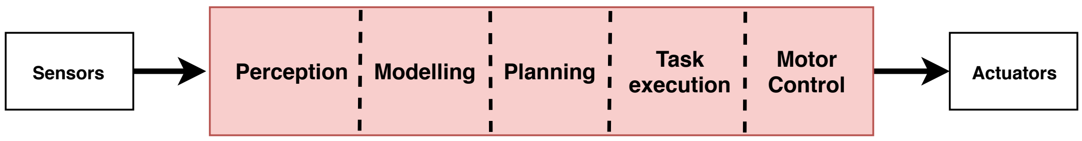


FIGURE 2.1: Sense-Plan-Act paradigm, inspired from [Brooks, 1986]

behaviour maintains its own internal representation, therefore allowing the ability to learning.

More details and discussions can be found for example in [Kortenkamp and Simmons, 2008], [Matarić and Michaud, 2008].

2.1.2 Common motion control approach for manipulation tasks

The common control approach for robot manipulation tasks [Brock et al., 2008] is composed of two complementary sub-tasks, namely manipulation planning in configuration space and task-level control in operational space (usually the Cartesian space). While the planner reasons about the global and low-frequency constraints of the task (e.g. where the goal is located, how to reach it, what to avoid), the controller deals with the local and high-frequency requirements of the manipulator motion (e.g. position, orientation, velocity, collision detection, etc.). The following section goes through these two sub-tasks briefly:

Task space control for a manipulator

The dynamical model of a rigid robot manipulator (with n link and vector $\mathbf{q}$ of joint variables) can be described by the Euler-Lagrange formulation as follows:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) = \boldsymbol{\tau}, \tag{2.1}$$

where $\mathbf{M}(\mathbf{q}) \in \mathbb{R}^{n \times n}$ is the symmetric, positive-definite inertia matrix; $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) \in \mathbb{R}^n$ is the centripetal and Coriolis vector; $\mathbf{g}(\mathbf{q}) \in \mathbb{R}^n$ is the gravity vector; and $\boldsymbol{\tau} \in \mathbb{R}^n$ is the applied motor torque.

Considering the end-effector, an operational point $\mathbf{x}$ on the robot's manipulator, the joint velocity ($\dot{\mathbf{q}} \in \mathbb{R}^n$) can be converted to the task velocity ($\dot{\mathbf{x}} \in \mathbb{R}^n$) through the Jacobian matrix $\mathbf{J}(\mathbf{q})$ of the manipulator, as follows:

$$\dot{\mathbf{x}} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}}. \tag{2.2}$$

Assuming that the Jacobian matrix is invertible, the operational space dynamics can be presented as:

$$\mathbf{F}_{ext} = \mathbf{\Lambda}(\mathbf{q})\ddot{\mathbf{x}} + \mathbf{\Gamma}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{x}} + \boldsymbol{\eta}(\mathbf{q}) \tag{2.3}$$

where $\mathbf{F}_{ext}$ is the external applied force (and moments) at point $\mathbf{x}$, which is represented as:

$$\boldsymbol{\tau} = \mathbf{J}^{\top}(\mathbf{q})\mathbf{F}_{ext}; \tag{2.4}$$

and the operational space inertial matrix $\mathbf{\Lambda}(\mathbf{q})$, the operational space centripetal and Coriolis forces $\mathbf{\Gamma}(\mathbf{q}, \dot{\mathbf{q}})$, and the gravity forces $\boldsymbol{\eta}(\mathbf{q})$ are computed as:

$$\begin{aligned}
\mathbf{\Lambda}(\mathbf{q}) &= \mathbf{J}^{-\top}(\mathbf{q})\mathbf{M}(\mathbf{q})\mathbf{J}^{-1}(\mathbf{q}), \\
\mathbf{\Gamma}(\mathbf{q}, \dot{\mathbf{q}}) &= \mathbf{J}^{-\top}(\mathbf{q})\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\mathbf{J}^{-1}(\mathbf{q}) - \mathbf{\Lambda}(\mathbf{q})\mathbf{J}(\mathbf{q})\mathbf{J}^{-1}(\mathbf{q}), \\
\boldsymbol{\eta}(\mathbf{q}) &= \mathbf{J}^{-\top}(\mathbf{q})\mathbf{g}(\mathbf{q}).
\end{aligned} \tag{2.5}$$

The main goal of the operational space control is to design a feedback controller that allows the execution of an end-effector motion $\mathbf{x}(t) \in \mathbb{R}^n$ that tracks the desired end-effector motion $\mathbf{x}_d(t)$ as closely as possible.

The tracking control problem of the end-effector can also be solved with an inverse dynamic control technique [Khalil, 2002], where the external applied force is defined differently as:

$$\mathbf{F}_{ext} = \mathbf{\Lambda}(\mathbf{q})\mathbf{v} + \mathbf{\Gamma}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{x}} + \boldsymbol{\eta}(\mathbf{q}) \tag{2.6}$$

where $\mathbf{v}$ is an auxiliary control input that needs designing. An option for $\mathbf{v}$ can be in PD controller form:

$$\mathbf{v} = \ddot{\mathbf{x}}_d + \mathbf{K}_V\dot{\mathbf{e}}_x + \mathbf{K}_P\mathbf{e}_x, \tag{2.7}$$

with tracking error $\mathbf{e}_x = \mathbf{x}_d - \mathbf{x}$. From Eq. (2.3), (2.6) and (2.7), the dynamics of error become:

$$\ddot{\mathbf{e}}_x + \mathbf{K}_V\dot{\mathbf{e}}_x + \mathbf{K}_P\mathbf{e}_x = 0, \tag{2.8}$$

which is asymptotically stable given an appropriate choice for the control gain matrices $\mathbf{K}_P$ and $\mathbf{K}_V$.

When the task space control comes to a redundant context, where the number of Degree-of-Freedoms (DoFs) in the manipulator is higher than requirements of the task, it is often decomposed into two components. The first component aims to satisfy the task requirement, specified by forces (and

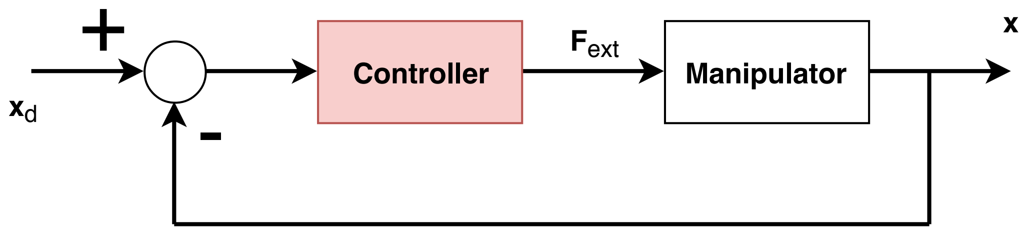


FIGURE 2.2: Diagram of closed loop control for a manipulator

moments) $\mathbf{F}_{task}$, whereas the second one is expressed by an arbitrary torque $\boldsymbol{\tau}_{posture}$, which accounts for a secondary objective (often a postural task) and, as such, is projected into the nullspace $\mathbf{N}(\mathbf{q})$ of the Jacobian $\mathbf{J}(\mathbf{q})$ operator with the aim of preventing the additional torque from affecting the first primary task. Overall, these tasks can be expressed as follows:

$$\begin{aligned} \boldsymbol{\tau} &= \mathbf{J}^{\top}(\mathbf{q})\mathbf{F}_{ext} + \mathbf{N}^{\top}\boldsymbol{\tau}_{posture}, \\ \text{and} & \begin{cases} \mathbf{N}(\mathbf{q}) = \mathbf{I} - {}^{-}\mathbf{J}(\mathbf{q})\mathbf{J}(\mathbf{q}) \\ {}^{-}\mathbf{J}(\mathbf{q}) = \mathbf{M}^{-1}(\mathbf{q})\mathbf{J}^{\top}(\mathbf{q})\boldsymbol{\Lambda}(\mathbf{q}); \end{cases} \end{aligned} \tag{2.9}$$

where $\mathbf{I}$ is identity matrix.

A more detailed discussion can be found in literature such as [Chung et al., 2008, Brock et al., 2008].

Configuration space planning for a manipulator

The manipulation planning problem is the process of searching for a collision-free path to move the manipulator from an initial pose (position and orientation) $\mathbf{x}_I$ to a final pose $\mathbf{x}_G$, where the main difficulty consists of the computationally intensive verification in the Cartesian space of all the possible collisions between the robot body and the environment. The introduction of configuration space, or *C-space* [Lozano-Perez, 1983], facilitates this procedure by mapping the complex geometric structure of the manipulator into a single point in the *C-space*. The *C-space* is the space of all possible transformations that can be applied to the manipulator given its kinematics. The dimension of the *C-space* is the number of DoFs of the manipulator n. In the *C-space*, the original planning problem corresponds to finding the sequence of configurations $\mathbf{q}$ of the manipulator connected from the initial configuration $\mathbf{q}_I$ to the final configuration $\mathbf{q}_G$.

Let denote $\mathcal{A}$ and $\mathcal{O}$ as the geometric descriptions of the robot manipulator and obstacles in the Cartesian space, then $\mathcal{C}^{\mathcal{A}}$ represents the n-dimensional *C-space* of $\mathcal{A}$. At a its generic configuration $\mathbf{q}$, the manipulator occupies the set of points in the the Cartesian space represented as $\mathcal{A}(\mathbf{q})$. Thus, the obstacle region can be defined as:

$$\mathcal{C}_{obs} = \{\mathbf{q} \in \mathcal{C} | \mathcal{A}(\mathbf{q}) \cap \mathcal{O} \neq \emptyset\} \tag{2.10}$$

The admissible set of configurations for collision free movements of the manipulator, $\mathcal{C}_{free}$, becomes:

$$\mathcal{C}_{free} = \mathcal{C} \backslash \mathcal{C}_{obs} \tag{2.11}$$

For a more complex context, where the manipulator has to transport a movable object, denoted as $\mathcal{P}$ and having a part configuration space $\mathcal{C}^{\mathcal{P}}$, the admissible configuration space reduces to:

$$\mathcal{C}_{free} = \mathcal{C} \backslash (\mathcal{C}^{\mathcal{A}}_{obs} \cup \mathcal{C}^{\mathcal{P}}_{obs} \cup \mathcal{C}^{\mathcal{PA}}_{obs}) \tag{2.12}$$

where $\mathcal{C}^{\mathcal{A}}_{obs}$, $\mathcal{C}^{\mathcal{P}}_{obs}$ and $\mathcal{C}^{\mathcal{PA}}_{obs}$ are the constraints set by collision-free conditions between the manipulator $\mathcal{A}$ and obstacles $\mathcal{O}$, between the the object $\mathcal{P}$ and obstacles $\mathcal{O}$, and the interaction condition between the manipulator $\mathcal{A}$ and the part $\mathcal{P}$, respectively, as in the following:

$$\begin{aligned}
\mathcal{C}^{\mathcal{A}}_{obs} &= \{(\mathbf{q}^{\mathcal{A}}, \mathbf{q}^{\mathcal{P}}) \in \mathcal{C} | \mathcal{A}(\mathbf{q}^{\mathcal{A}}) \cap \mathcal{O} \neq \emptyset\} \\
\mathcal{C}^{\mathcal{P}}_{obs} &= \{(\mathbf{q}^{\mathcal{A}}, \mathbf{q}^{\mathcal{P}}) \in \mathcal{C} | int(\mathcal{P}(\mathbf{q}^{\mathcal{P}})) \cap \mathcal{O} \neq \emptyset\} \\
\mathcal{C}^{\mathcal{PA}}_{obs} &= \{(\mathbf{q}^{\mathcal{A}}, \mathbf{q}^{\mathcal{P}}) \in \mathcal{C} | \mathcal{A}(\mathbf{q}^{\mathcal{A}}) \cap int(\mathcal{P}(\mathbf{q}^{\mathcal{P}})) \neq \emptyset\}
\end{aligned} \tag{2.13}$$

With the condition of $\mathbf{q}_I \in \mathcal{C}_{free}$ and $\mathbf{q}_G \in \mathcal{C}_{free}$, the final outcome of this planning sub-task is the path τ, which is defined as:

$$\begin{aligned}
&\tau : [0, 1] \rightarrow \mathcal{C}_{free} \\
&\text{s.t} \begin{cases} \tau(0) = \mathbf{q}_I \\ \tau(1) = \mathbf{q}_G \end{cases}
\end{aligned} \tag{2.14}$$

to which many searching algorithms can be applied.

More details and discussion can be found in some literature, such as [LaValle, 2006, Brock et al., 2008, Kavraki and LaValle, 2008]

2.1.3 A novel approach - A hybrid manipulation control architecture

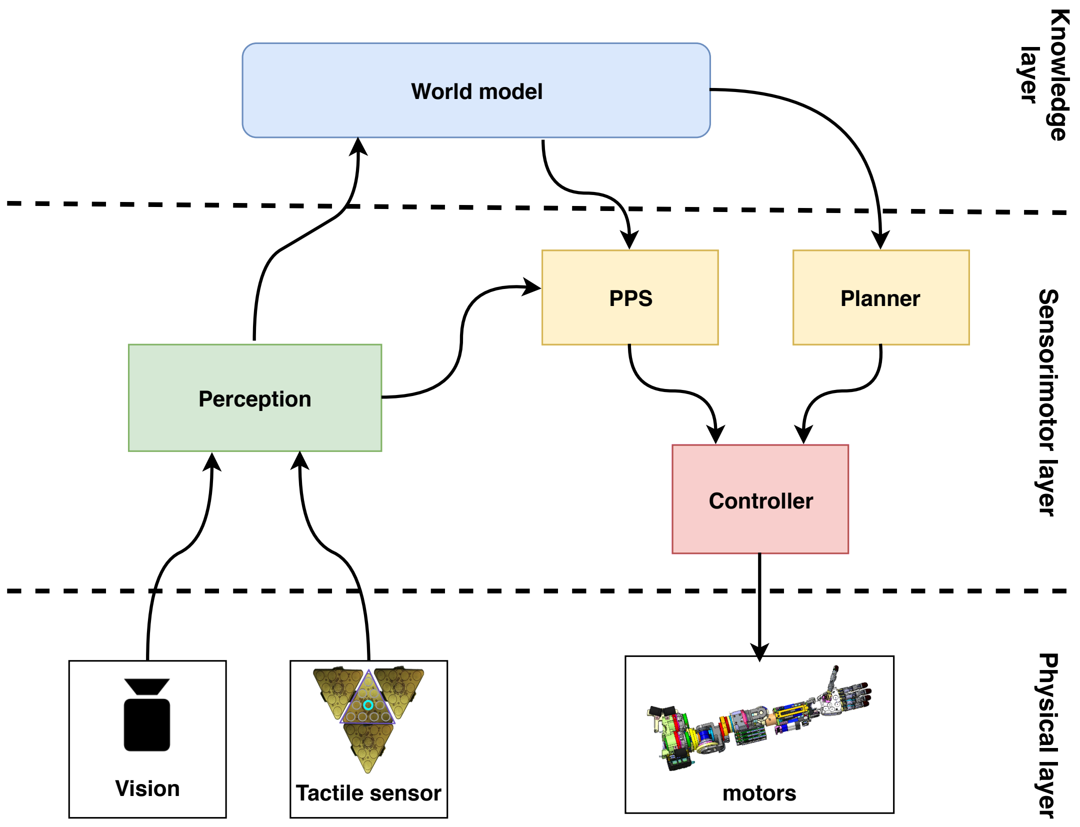


FIGURE 2.3: Simplified diagram of our hybrid control system for a manipulator. *Perception* modules process and integrate sensory information to build the *World model* and provide updates for reaction behaviour integrated in the *Controller*. The *World model* feeds global information to the *Planner* for reasoning about the motion plan needed to complete the task, which is also executed by the same *Controller*.

My proposed control architecture, as shown in Fig. 2.3, falls under the category of Hybrid control (cf. Section 2.1.1), which includes (i) a motion planner and reaching controller to deal with the global reaching movement of a robot's manipulator, considering the static environment (in the long timescale of a single task); and (ii) a visuotactile reactive controller to ensure that manipulator avoids local obstacles within its dynamic environment. The data flow for the global reaching task goes through *Sensors* (i.e. Visual or Tactile), *Perception* to construct the *World model*, then continues to *Planner*, *Controller* and the robot's *Motors*. For the local reactive task, it connects *Perception* and *Controller* through *PPS* modules instead of going through the

World model and *Planner*. While vision based reaction serves mainly in the *pre*-collision phase and in the middle timescale of the task (with a data update rate of 30 Hz), the tactile-based reaction yields responses in the *post*-collision stage and in the shortest timescale of the task (with the highest update rate of 100 Hz). This control architecture allows robots not only to "think" strategically about an optimal solution for the task at hand, but also to respond adaptively to any change in the environment. An example of recent work on this hybrid approach is [Kappler et al., 2018], although these authors only use visual rather than tactile sensing.

Furthermore, unlike the approach commonly adopted for doing manipulation control (see Section 2.1.2), the proposal consists of both planning and controlling parts defined in the robot operational space. The former component, the *Multiple Cartesian point planner* (see Chapter 5), aims to provide collision-free motion paths for robot manipulators that are made of multiple controlled points (e.g. end-effector, elbow) obtained by reasoning directly in the operational space. The latter component, the *Multiple target reaching-with-avoidance controller* (see Chapter 3, 4 and 5), is defined in terms of a local constraint minimization problem whose solution can at the same time coordinate and satisfy both the multiple-target reaching task and the dynamic obstacle avoidance.

This architecture allows the robot to execute its actions safely in dynamic environments (pHRI), and also eases socially oriented interaction with its human counterparts (social Human-Robot Interaction (sHRI)). The details of these discussions can be found in Chapter 5.

In addition, this control architecture guarantees seamless integration with existing cognitive architectures, such as DAC-h3 [Moulin-Frier et al., 2017a]. The UML diagram in Fig. 2.4 presents an example of interaction between a human and a robot (equipped with DAC-h3 architecture) in a table-top scenario, where the human asks the robot to push a toy towards his own side. In the diagram, the *OPC* module includes the *World model* of the architecture, whereas the *KARMA* module serves as a task scheduler, sequencing the pushing task into a collection of sub-tasks, such as reaching, moving object, restoring to home position. Remarkably, in this context, the proposed control architecture enables the robot to complete each sub-task in the presence of a cluttered dynamical environment, hence facilitating a safe interaction with the human partner.

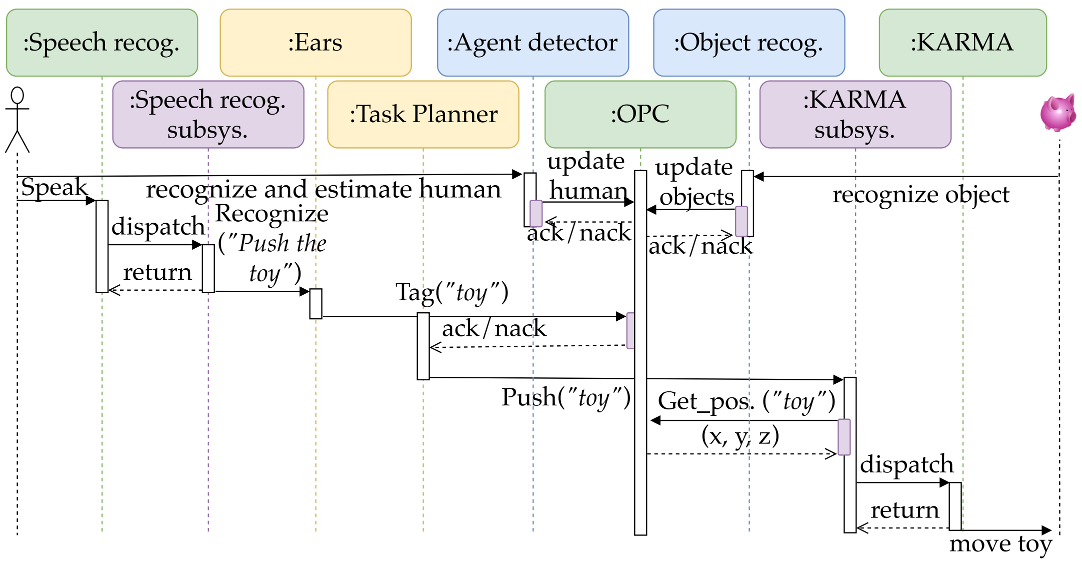


FIGURE 2.4: Temporal UML diagram for an interaction where a human gives a vocal command to the robot to push an object named "toy". The diagram depicts the modules and subsystems involved, and shows the data flow. After converting the vocal command into an action plan, the robot first localizes the desired object, and subsequently pushes that object; an action that is conducted with my proposed robot control architecture.

2.2 Overview of learning techniques

Learning is the process of an agent building a model of how the world functions by observing the effects of its own interaction with the world so that it can improve its future actions through "stages". Such a model (e.g. "world models" in [Ha and Schmidhuber, 2018]) should be referred to as a sensory representation or a sensorimotor mapping (in behaviour-based learning) that agents construct (implicitly or explicitly) during the learning process. This idea gives rise to applications of learning methodologies in robotics where the complexity of mechanical design and the properties of unstructured environments hinder the practical possibility of the classical analytical approaches.

Depending on the different types of feedback that agents receive from the environment, it is possible to distinguish three types of learning: namely, supervised learning, unsupervised learning and reinforcement learning.

In *supervised learning*, the agent is provided with pairs of input-output examples in order to infer the underlying relationships by minimizing the error between the desired and the observed output. By employing the learned input-output mapping, the agents can finally predict the outcome when presented with a new perceived input.

One special case of supervised learning is *Reinforcement learning*, where the agents receive reinforcement signals – rewards or penalties – corresponding to the success or failure of the actions they have conducted. By attempting to maximize the cumulative sum of rewards, agents learn the optimal way to complete the action.

Instead, *unsupervised learning* is concerned with the task of learning the underlying structure of the input data, when the agent is only supplied with a stream of input signals.

Since Part III of this thesis mainly relies on *supervised learning*, the next section will discuss the details of this specific learning methodology in more depth.

2.2.1 Supervised learning

Given a set of independently and identically distributed (i.i.d) examples

$$\mathcal{D} = \{\mathbf{x}_i, \mathbf{y}_i\}_i^N, \tag{2.15}$$

where $\mathbf{x}_i$, $\mathbf{y}_i$ denote a sample in the set of input $\mathbf{X}$, and the set of labelled output $\mathbf{Y}$ respectively; i denotes the index of the i-th example in the dataset $\mathcal{D}$. The objective of *Supervised learning* is to find a good approximation f^* of the unknown function $f : \mathbf{X} \mapsto \mathbf{Y}$, which can accurately predict the output of a novel input $\mathbf{x}$. To this end, it is necessary to gauge the predictive accuracy of the learning agent by evaluating the so-called *objective function* or *loss function*, denoted by $\mathcal{L}$, which accounts in some given way for the error between the prediction $\mathbf{y}^*$ and the actual values $\mathbf{y}$.

There are two common types of supervised learning task, namely *Classification* and *Regression*. The former deals with the task of specifying the category (in a list of K different items) that a given input $\mathbf{x}$ most likely fits into, or in other words, the output $\mathbf{y}$ belonging to a finite set of values ($\mathbf{Y} \in \mathbb{N}$). An example of this task is object recognition from input images, in which the output is the numeric value corresponding to the class of the shown object. On the other hand, the Regression task generates output $\mathbf{y}$ as continuous values ($\mathbf{Y} \in \mathbb{R}$). An example of regression is that of an agent asked to estimate the position of an object in the Cartesian space, having been provided stereo-vision images of that particular object as an input.

Different learning tasks use different types of loss function. The most common loss function is negative log-likelihood, wherein minimizing the loss means maximizing the likelihood estimation. In regression, the loss function is usually coded as the difference between the real output $\mathbf{y}$ and the estimate $f^*(\mathbf{x})$:

- Squared loss:
$$\mathcal{L}(\mathcal{D},\boldsymbol{\theta})_{MSE} = \frac{1}{N}\sum_{i}^{N}\Big(\mathbf{y}_i - f^*_{\boldsymbol{\theta}}(\mathbf{x}_i)\Big)^2$$

- Absolute loss:
$$\mathcal{L}(\mathcal{D},\boldsymbol{\theta})_{MAE} = \frac{1}{N}\sum_{i}^{N}|\mathbf{y}_i - f^*_{\boldsymbol{\theta}}(\mathbf{x}_i)|$$

where $\boldsymbol{\theta}$ denotes learnable parameters.

The following types of loss functions are often utilized for classification:

- Hinge loss:
$$\mathcal{L}_i(\mathcal{D},\boldsymbol{\theta})_{Hinge} = max\Big(0, 1 - \mathbf{y}_i f^*_{\boldsymbol{\theta}}(\mathbf{x}_i)\Big)$$

- Logistic regression loss:
$$\mathcal{L}(\mathcal{D},\boldsymbol{\theta})_{logit} = -\frac{1}{N}\sum_{i}^{N}\Big[\mathbf{y}_i \log f^*_{\boldsymbol{\theta}}(\mathbf{x}_i) + (1-\mathbf{y}_i)\log\Big(1 - f^*_{\boldsymbol{\theta}}(\mathbf{x}_i)\Big)\Big]$$

- Multi-class classification loss:
$$\mathcal{L}(\mathcal{D},\boldsymbol{\theta}) = -\log P(\mathbf{Y}|\mathbf{X};\boldsymbol{\theta})$$

With optimization techniques such as *gradient descent*, it is possible to find the best set of parameters $\boldsymbol{\theta}$ that minimizes the loss function of the task:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{D},\boldsymbol{\theta}) \tag{2.16}$$

Gradient descent is a gradient based optimization technique, where the derivative of a function, e.g. loss function $\mathcal{L}$, w.r.t a variable, e.g. learnable parameters $\boldsymbol{\theta}$, is employed to bring about changes in $\boldsymbol{\theta}$, thus making small changes in $\mathcal{L}$. By choosing the moving direction of $\boldsymbol{\theta}$ at every iteration as opposite to the derivative $\nabla_{\boldsymbol{\theta}}\mathcal{L}$ of function $\mathcal{L}$ as:

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} - \alpha\nabla_{\boldsymbol{\theta}}\mathcal{L},$$

where α denotes step size, it is possible to reduce the value of function $\mathcal{L}$. More detailed discussion about gradient descent in machine learning and deep learning can be found for example in [Goodfellow et al., 2016, Chapter 5, 8]

Depending on the hypothesis space $\mathcal{H}$ of the learning task, the function f can have different forms. If f is a linear function $f(\mathbf{x}) = \mathbf{X}^\top\boldsymbol{\theta}$, the related learning problem is called *linear regression*, for which the squared loss is usually employed as the objective function. Analogously, the problem becomes a *polynomial regression* if f is a polynomial. Further, replacing the linear term with the sigmoid function

$$\sigma(z) = \frac{1}{1 + exp(-z)} \tag{2.17}$$

and applying the linear transformation $\mathbf{z} = \mathbf{X}^\top\boldsymbol{\theta}$ leads to *logistic regression* for a binary classification problem, for which the logistic loss $\mathcal{L}(\mathcal{D}, \boldsymbol{\theta})_{logit}$ can be used. For the multi-class case, the sigmoid function is replaced with the softmax function

$$\sigma(z) = \frac{exp(z_k)}{\sum_j exp(z_j)},$$

then we have *softmax regression*. The function $\sigma()$ in the last two cases is termed the *activation function*, which can convert a result of a linear function to a probability.

In summary, every learning problem should include four main factors, namely the (i) *dataset* to provide examples for the (ii) *Optimizer* to search the (iii) *Hypothesis space* under the guide of the (iv) *Objective function* [Domingos, 2012]. It is worth noting that by replacing the last three components independently we can obtain a wide range of different *learning algorithms*. This is also the case for unsupervised learning.

Moreover, as noted in [Russell et al., 2010] when discussing *computational learning theory*, "any hypothesis that is consistent with a sufficiently large set of training examples is unlikely to be seriously wrong: that is, it must be probably approximately correct", any learning algorithm returning *probably approximately correct* hypotheses is therefore called a probably approximately correct (PAC) learning algorithm.

If the algorithm is allowed to see N example, where:

$$N \geq \frac{1}{\epsilon}\Big(ln\frac{1}{\delta} + ln|\mathcal{H}|\Big), \tag{2.18}$$

and if a learning algorithm returns a hypothesis that is consistent with this N example, it has an error of at most ϵ with a probability of at least $1 - \delta$. In

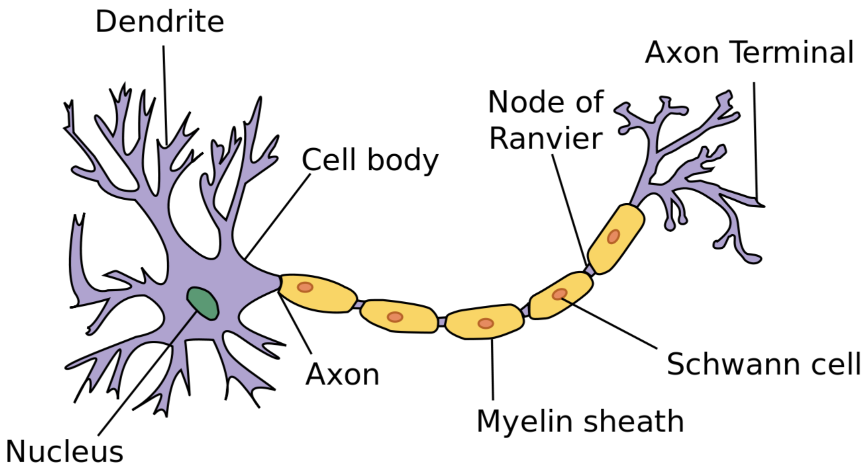


FIGURE 2.5: Illustration of a neuron, source Wikipedia

other words, it is probably approximately correct. The number of required examples N, as a function of ϵ and δ, is called the sample complexity of the hypothesis space $\mathcal{H}$.

2.2.2 Artificial neural networks and Deep learning

Humans have approximately 100 billion neurons, or nerve cells, the most essential cell in the nervous system. Each neuron's main body, called a *soma*, contains the genetic material in its *nucleus*. Many branches, termed *dendrites*, extend out from the neuron's body, which are where the cell receives signals from other cells. One extension, an *axon*, is different to others in that it allows the neuron to transmit a signal to other dendrites through the axon ending, or the *synapse*.

Computationally, a neuron can be modelled (in simplified form) as in Fig. 2.6, where the learnable synapse strengths, i.e. w_i, allow the signals from other neurons' axons (e.g. x_0) to interact with dendrites of the neuron. The synapse weights can effectively control the influences of other neurons through the multiplication $w_i x_i$. In the cell's body, signals from all the dendrites are summed up, and ultimately cause a *neuron firing* when the sum exceeds a threshold. This interaction can be modelled by applying an *activation function* g on the sum of the dendrites. The original choice of activation function is the sigmoid function (cf. Eq(2.17). That is to say each neuron conducts a dot product of input with its weights, adds bias and applies a non-liner transformation, e.g. $g(\cdot) = \sigma(\cdot)$, as

$$\sigma(\sum_i w_i x_i + b).$$

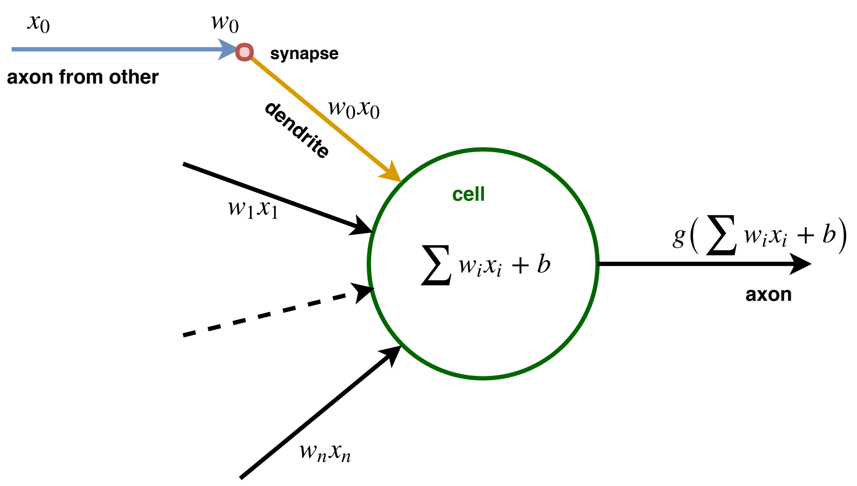


FIGURE 2.6: Illustration of an artificial neuron

As this point, we realize that there are similarities between the computational model of a neuron and the classifier discussed above. Notably, the original purpose of the neuron model was to find explanations for the biological neural system, but since then the computational model has diverged and it has been widely used to construct artificial learning models.

There are many different choices of activation function, namely sigmoid, softmax, Hyperbolic tangent, Rectified linear unit, etc., each creating different dynamics for the neuron's output. Although neurons with a sigmoid activation function can nicely model the firing rate of a biological neuron, the sigmoid function has two main drawbacks, namely saturation and a non zero-centred output (see Fig. 2.7). The saturation at the tails of 0 or 1 causes the gradient at the regions to become almost zero, hence there will be almost no signal flowing through the neuron. Whereas the non zero-centred output leads to zig-zagging dynamics in the gradient updates for the weights.

Similarly to the sigmoid function, the Hyperbolic tangent $tanh(z) = 2\sigma(2z - 1)$, which transforms the input to the range $[-1, 1]$ (instead of $[0, 1$ in sigmoid), also saturates at the tails, but it has zero-centred output (see Fig. 2.8). Moreover, the tanh activation function is similar to the identity function when close to 0, making it easier to train. Consequently, it is more preferable to use tanh than sigmoid in practice.

The rectified linear unit applies the function $ReLU(z) = max(0, z)$ on the weighted sum of the neuron's input (see Fig. 2.9). As a result, it is less expensive in computational terms than sigmoid or tanh (which requires exponential calculation), and is easier to optimize with gradient based methods [Goodfellow et al., 2016]. Some variations of this type of activation are

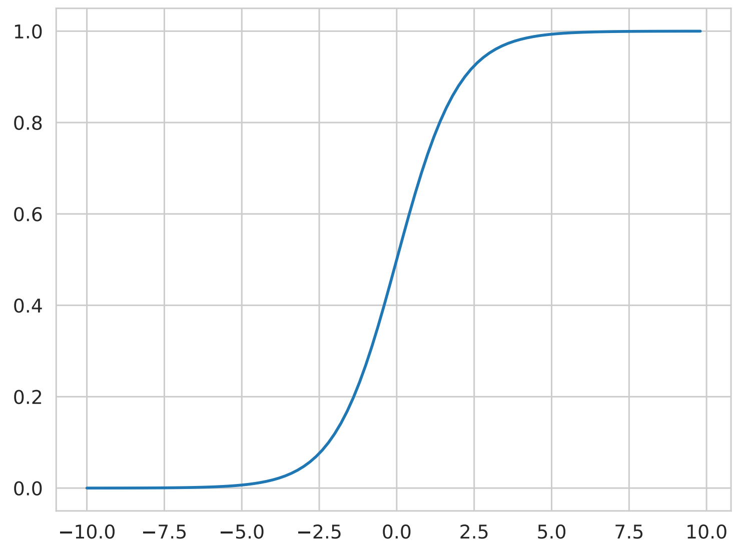


FIGURE 2.7: Sigmoid activation function

Leaky ReLU [Maas et al., 2013], *parametric ReLU* [He et al., 2015] and *Maxout* [Goodfellow et al., 2013].

If neurons are organized in the form of a graph, which often includes a number of neuron layers, we call it an Artificial Neural Network (ANN). In an ANN, the output of some neurons constitute the input of other neurons, but there is no connection between neurons in the same layer. The most common type of neural network layer is *fully-connected*, where all neurons in two successive layers are connected pairwise. The network becomes deeper when we increase the number of layers. The illustration in Fig. 2.10 presents an example of a two-layer neural network, which includes one *input layer*, one *hidden layer* and one *output layer*.

Denote the input of the above ANN as $\mathbf{x}$, then the output at every layer of the network can be computed as follows:

- Hidden layer:
$$\mathbf{h} = g^{(1)}(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$

- Output layer:
$$\mathbf{y} = g^{(2)}(\mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)})$$

where $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ are the weight matrix and bias vector of the l-th layer, respectively; and g denotes the activation functions of the neuron layers.

The above example reveals that an ANN can define a nonlinear approximation function f to map between input $\mathbf{x}$ and $\mathbf{y}$, and is parameterized by the weights of the network. The properties of the approximator depend on

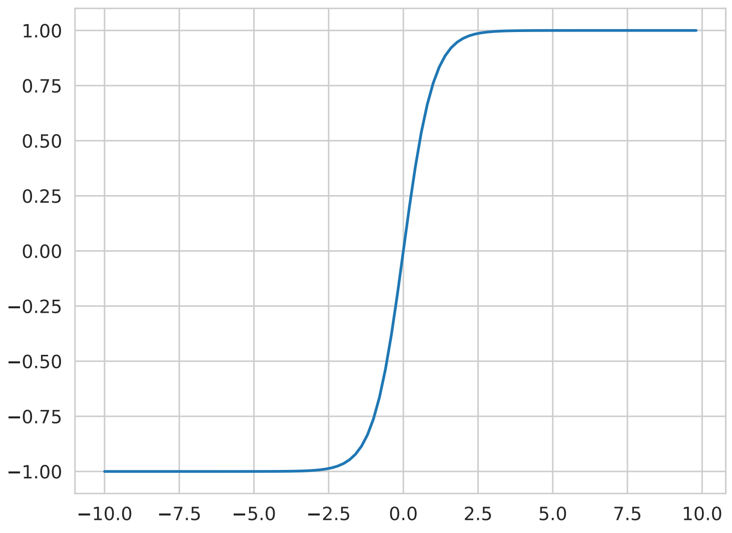


FIGURE 2.8: Tanh activation function

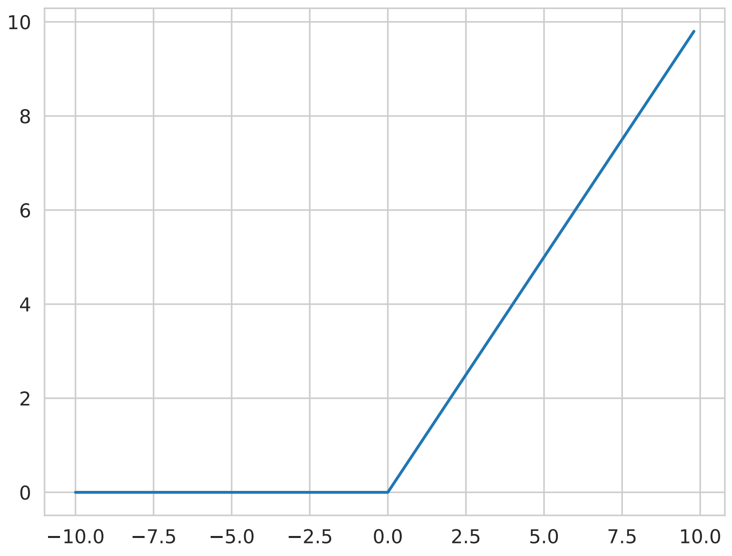


FIGURE 2.9: ReLU activation function

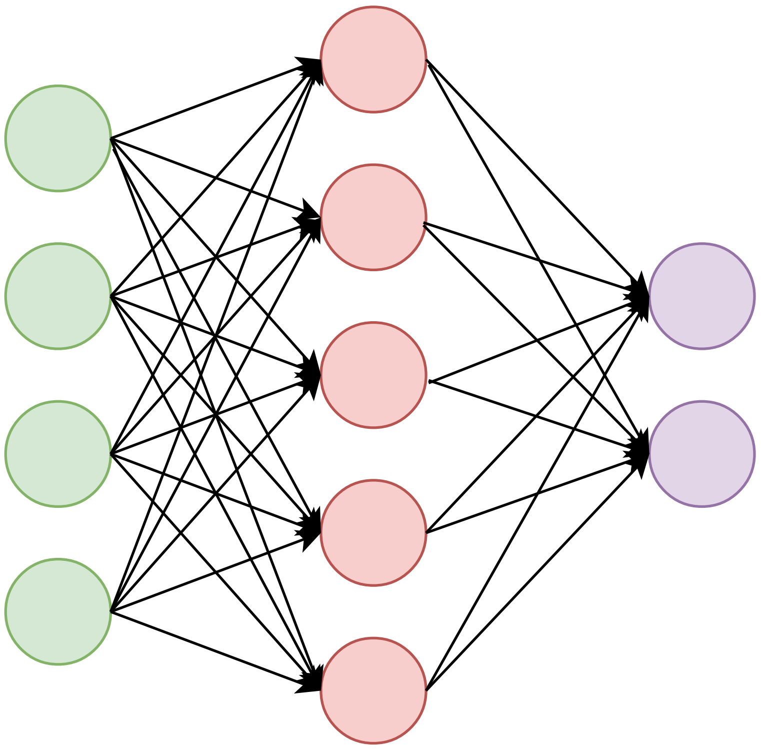


FIGURE 2.10: A 2 layer Neural network. Green neurons belong to input layer, Red ones are in hidden layer and output layer contains purple neurons

the choice of network architecture (e.g. number of layers, number of neurons in each layer) and activation function of each layer. As discussed in [Goodfellow et al., 2016], even an ANN with a single layer is sufficient to represent any function (universal approximation theorem [Hornik et al., 1989]). In other words, given a function, an ANN exists that can approximate that function. The layer may be infeasibly large, however, and may fail to learn and generalize correctly. In many circumstances, using deeper models can reduce the number of units required to represent the desired function, and can reduce the amount of generalization error. Nevertheless, how deep a network should be can only be determined by experimentation and depends on the learning task.

For a complete overview of learning theory, ANN and DNN, it is advised to review the literature, for example [Russell et al., 2010, Chapter 18-21] and [Goodfellow et al., 2016].

2.2.3 From motor skills development in infants to sensorimotor learning in robotics - the Developmental approach

Development is a process to create complexity by accumulating changes. – [Cangelosi and Schlesinger, 2014]

Development is a process in an agent to expand its repertoire of possible actions. In humans, it is well-known in literature, e.g. [Thelen and Fogel, 1989, Bushnell and Boudreau, 1993, Adolph, 1997, Berthier and Keen, 2006, Gerber et al., 2010] that there are developmental processes of motor skills (alongside other skills like language acquisition and expression), e.g. reaching and grasping, balancing and walking. During these processes, there are gradual changes in the properties of motor behaviours, from simple primitive reflexes to increasingly precise task-oriented actions. Specifically, these processes mostly take place in the first years of the human lifespan, e.g. 0–24 months old in the development of reaching and grasping skills [Berthier and Keen, 2006, Gerber et al., 2010], 6–12 months old in the development of locomotion skills [Vereijken and Adolph, 1999]. In detail, developmental milestones of infants' reaching skill (in the period preceding successful grasping) is summarized below [von Hofsten, 1984, Bushnell, 1985, Gerber et al., 2010, Cangelosi and Schlesinger, 2014]:

- After birth–2 months old: Spontaneous object-oriented movement of hand and arm–prereaching movements;
- 2–3 months old: Prereaching frequency declines;
- 3–4 months old: Robust and organized object reaching behaviours emerges, e.g. corrective movements, pre-grasping shape of hand with hand-eye coordination.

Noticeably, the prereaching movements in early age infants can be considered as being like motor babbling [Cangelosi and Schlesinger, 2014]. This exploring movement takes place since newborns do not know what muscle movements are needed to achieve the goal associated with some particular action. In order to learn this association, they engage in random movements or trial-and-error learning. This procedure allows them to generate a map linking movements (motor commands) and the resultant action [Vernon, 2014].

The hypothesis is that through exploration activities, i.e. motor babbling, infants can learn the association between different sensory information (i.e.

sensory representation), which then lays a foundation for developing further motor skills (rather than learning skills directly).

In the development of infants' reaching skills, the substitution of initial spontaneous movements (where there is no association between motor control and vision [Sandini et al., 1997]) with visually guided movements (where multisensory integration is required) suggests the above idea of learning the basic multisensory integration, e.g. vision and proprioception in eye-hand coordination, before developing new motor skills [Bushnell, 1985, Bushnell and Boudreau, 1993, Cangelosi and Schlesinger, 2014].

Instead of modelling the whole developmental process of emerging reaching skills on robots (e.g. from prereaching to grasping in [Savastano and Nolfi, 2012, 2013]), since these normally take place over a long period of time (e.g. months to years in humans), the learning section in this thesis focuses on a much shorter process, i.e. the prereaching phase, where the agents' motor babbling could bring about multisensory integration in its reachable space.

Taking inspiration (but not mimicking) from this development phase, the iCub robot is allowed to conduct random movements in its arm and head simultaneously in two different environments, without and with a randomly generated object. In the former scenario, the robot learns the visuomotor mapping–i.e. it associates the visual and proprioceptive information of its own arm (see Chapter 6), whereas in the latter case, the robot also incorporates the tactile signal (along with vision and encoders) when interacting with an object (see Chapter 7). The motor movements serve as data collection (both input and label) for the succeeding supervised learning (see the above Section 2.2.1 and 2.2.2) to train the DNNs model built for multisensory association purposes. The semi-supervised learning strategy, in which the robot generates its own training data through random movements, illustrates the phenomenon of motor babbling.

My sensorimotor learning method reflects the idea of a *developmental approach* in robotics, which focused on learning the motor skills by trial and error exploration, rather than by computing a desired movement trajectory in advance through forward or inverse kinematics with *a priori* knowledge about agents' models [Cangelosi and Schlesinger, 2014].

Nevertheless, the proposed learning method is in line with the *embodiment* (embodied cognition) approach, in which the agent constructs and develops its own understanding of the world (the space of its possible action and where it is embedded within it) through actions it can engage in. The acquired knowledge is specific to the agent itself (i.e. its body structure) in such a way that agents with different types of bodies understand the world differently. This is due to the differences in the physiology of agents' perception and motor systems. It also means that agents develop the understanding of their environment in terms of their embodied action capabilities, i.e. different physical capabilities bring about different interactive experiences [Vernon, 2014]. Hence, this learning approach encourages the adaptability of agents to the environment,

It is worth noting, however, that this approach mainly focuses on the learning aspect rather than the development aspect, recognizing that development and learning are related but are not the same. While learning is based on an agent adapting the parameters of a built model, development focuses on that agent discovering and generating a model of how the world works by itself through its own interaction [Vernon, 2014]. Moreover, development relates to the changes in the agent's morphology or structure, and in the sense of the emergence of sensorimotor skills, these changes consist of modification in the agent's sensory system, brain and body, i.e. similar to the improvement in visual acuity or body structure strength in infants.

It is important to state that there are a number of applications of learning in robotics, from perception to manipulation, grasping, etc., and covering all of them is not in the scope of this thesis. Specific reviews of related work will be discussed in the context of the particular learning tasks of this thesis (see Section 6.2, Section 7.1). Broader reviews, however, can be found in literature, such as for model learning [Nguyen-Tuong and Peters, 2011], reinforcement learning in robotics [Kober et al., 2013], deep learning in robotics [Sünderhauf et al., 2018].

2.3 Experimental setup - iCub humanoid robots and Simulator

The iCub [Metta et al., 2010] is a child-sized full humanoid robot (see Fig.2.11) that designed to study embodied cognition and autonomous exploration. In total, the iCub has 53 DoFs: six DoFs for the head and eyes system, 16 DoFs for each arm (including three shoulder joints, two elbow joints, two wrist joints and nine hand joints; see Fig. 2.12), three DoFs for the torso, and six

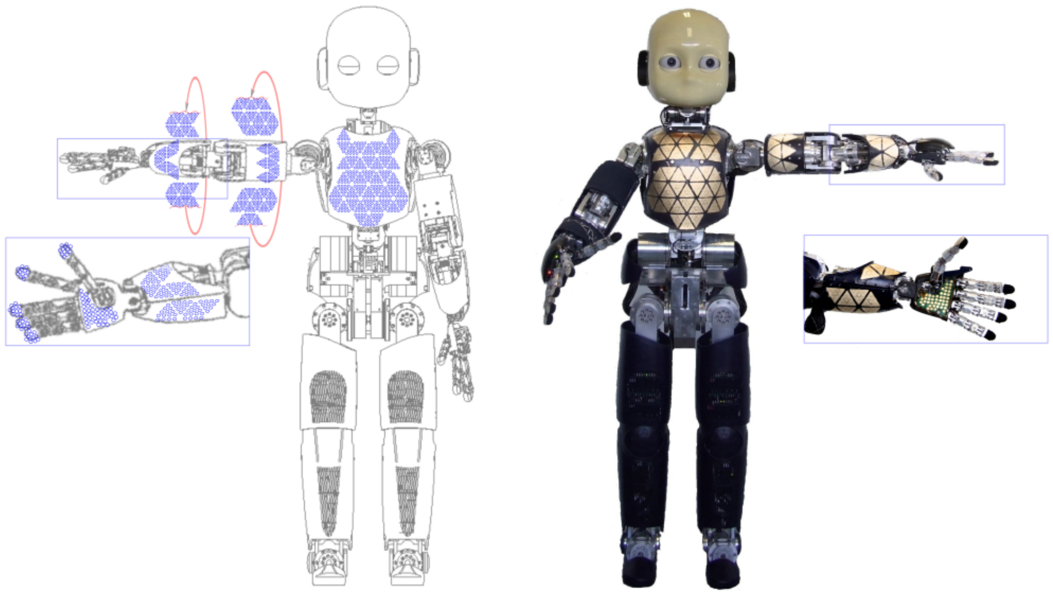


FIGURE 2.11: iCub humanoid robot with whole body tactile sensing, from [Hoffmann et al., 2018].

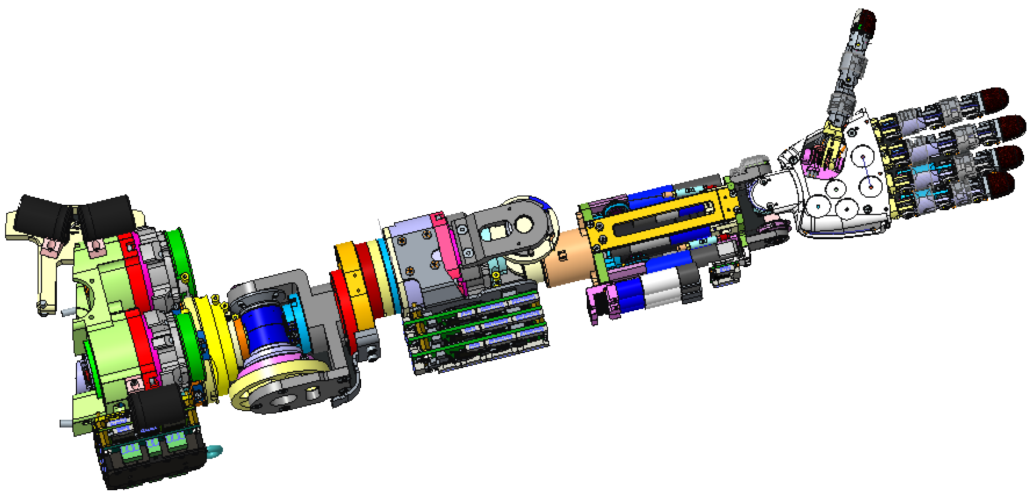


FIGURE 2.12: CAD drawing of the iCub left arm.

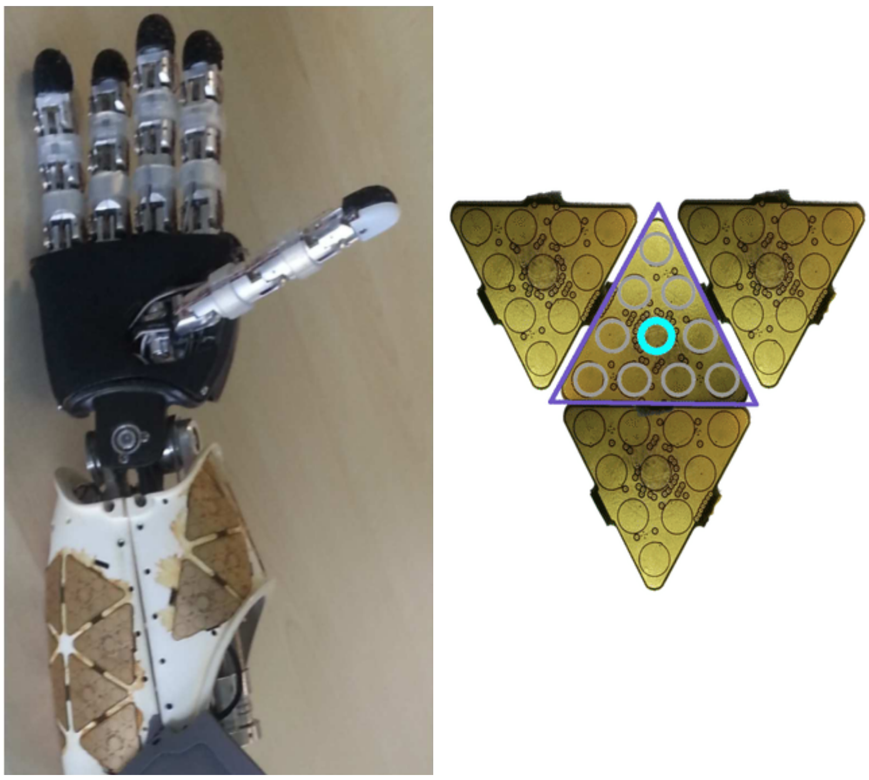


FIGURE 2.13: Close-up of iCub's skin taxels, from [Roncone et al., 2016]. Left–Skin taxels on the right arm and hand (with black covers); Right–A skin "super-taxel" (a combination of ten individual taxels inside the purple triangle) at scale 1-1

DoFs for each leg. The iCub head features a three DoF neck and a binocular stereo system composed of two identical RGB cameras in a human-like arrangement, with three DoFs allowing mechanically coupled tilt motion, and independent version and vergence movements. In addition, the iCub is equipped with a variety of other sensors, of which the joint encoders, and the artificial skin are relevant here. An array of artificial electronic skin taxels (Fig.2.13) covers large areas of the iCub body [Maiolino et al., 2013] (Fig. 2.11), which allows iCub to sense touch—applied pressure. The poses of the skin taxels are calibrated with respect to the kinematic model of the robot, and are kept updated during robot movements. Thus, physical contacts with the iCub skin can be sensed and localized.

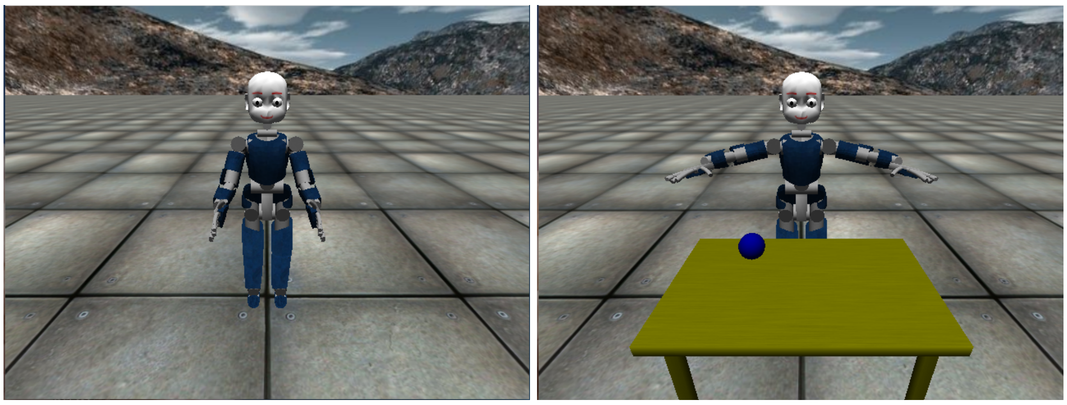


FIGURE 2.14: iCub simulator in different simulated scenarios

A physics-based simulator of the robot, the iCub simulator [Tikhanoff et al., 2008], is also available (see Fig. 5.8), which was originally developed as part of ITALK European project[1]. When it was created, this was one of the few simulated robotics platform aiming to create a 3D dynamic robot environment with all available sensors like the physical robots, and fully based on non-proprietary open source libraries. The ODE (Open Dynamic Engine) physical engine [2] is utilized for the purposes of simulating rigid bodies and computing physical interactions (based on collision detection algorithms). Moreover, the simulator makes use of OpenGL and the open-source multimedia library SLD [3] as rendering engines, which aims to facilitate and speed up the simulation of complex environments. This simulated platform plays a core role in the learning algorithms developed in this thesis (see Part III for more details and discussion).

[1] `italkproject.org`
[2] `http://opende.sourceforge.net/`
[3] `http://www.libsdl.org`

Most of the software for this thesis has been written using Yet Another Robot Platform (YARP) [Metta et al., 2006b, Fitzpatrick et al., 2008], an open-source software framework that supports distributed computation under different operating systems (Windows, Linux, MacOS) with the main goal of achieving efficient robot control. YARP facilitates code reuse and modularity by decoupling the programs from the specific hardware (using device drivers) and operating systems, and by providing an intuitive and powerful way to handle inter-process communication (using Ports objects). Furthermore, YARP provides mathematical (vector and matrix operations) and signal (image, sound) processing libraries. This software platform helps to construct the robot's "brain" as a collection of interconnected independent modules, running on different machines, exchanging data and control signals.

Part II

Model based approach – PPS in interaction

Chapter 3

Compact real-time avoidance for a humanoid robot for human-robot interaction

3.1 Introduction

Safety plays a fundamental role in the context of Human-Robot Interaction (HRI), with the avoidance of collisions between the robot and humans being the most important aspect. This is the subject of so-called pHRI, where the handling of collisions between machines and humans can be divided into two phases: *pre-impact* and *post-impact* [De Luca et al., 2006].

Research in the *post-impact* direction relies typically on joint torque or force/torque sensors, the measurements from which work together with a robot model to allow for contact localization (cf. [Haddadin et al., 2017a] for a recent survey). The redundancy of the robot's manipulator can then be employed still to accomplish the manipulation task while not exerting forces on the obstacle using some contact detection techniques. One example of such techniques is the residual method, proposed by [De Luca et al., 2006, De Luca and Ferrajoli, 2008]. Combining the above results with the trajectory scaling strategy, Haddadin et al. [2008] presents a framework in which the robot switches between different control strategies upon the detection of collisions. The roots of *pre-impact* strategies, meanwhile, lie in the use of motion planning to find collision-free end-effector trajectories, leveraging on the full knowledge of the robot body and the environment. In dynamic scenarios, planning needs to be complemented with reactive strategies such as the potential field approach [Khatib, 1990]. In this respect, the elastic strips framework [Brock and Khatib, 2002] combines reaching for a goal configuration (global behaviour) with reactive obstacle avoidance (local behaviour) through incremental modification of a previously planned motion. Flacco

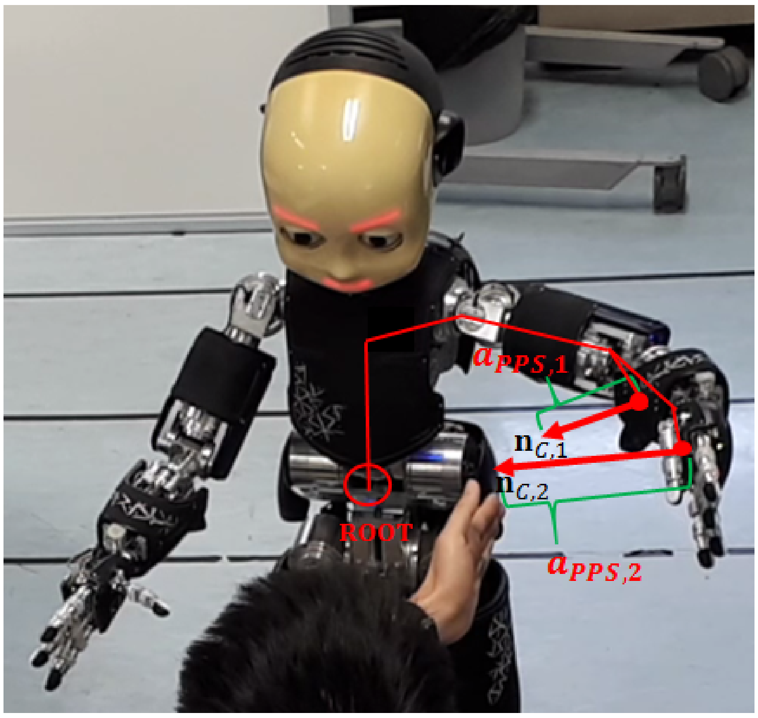


FIGURE 3.1: Experimental scenario. The proposed system is able to detect the presence of humans close to the robot's body thanks to a keypoint estimation algorithm combined with a PPS representation. Prior-to-contact activations are translated into a series of distributed control points (a_{PPS} in figure) for *pre-impact* avoidance. See text for details.

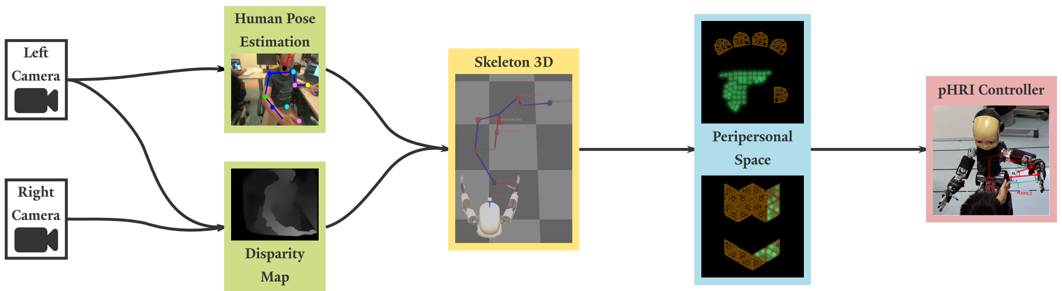


FIGURE 3.2: Software architecture for physical human-robot interaction. In this work, I develop a framework composed of: i) a human pose estimation algorithm, ii) 2D to 3D disparity mapping, iii) a peripersonal space collision predictor, iv) a pHRI robot controller for distributed collision avoidance. See text for details.

et al. [2012] employ a so-called depth-space approach, in which they use an external Kinect sensor and a 2.5D space projection to obtain distances between obstacles and interest points on a robot arm. These are in turn used to generate corresponding repulsive vectors that are remapped into the joint space, effectively preventing joint movement in the collision direction. Magnanimo et al. [2016] do not address the control problem but, interestingly, they do provide a framework for the dynamic construction of warning and protective safety fields around a manipulator, relying on laser scanners to sense the environment, and proprioception to sense the manipulator's own velocity.

An essential component o guarantee safety in pHRI is being able reliably to perceive the human. This usually translates to methods that segment the human body parts from the background and localizing them with respect to the robot. This is known as human pose/keypoints estimation or skeleton extraction—refer to recent surveys such as [Sarafianos et al., 2016] for 3D pose estimation, and [Helten et al., 2013, Ye et al., 2013] for 3D pose estimation from a RGB-D camera. In a robotics context, particle filters [Azad et al., 2007], or Iterative Closest Points (ICP) approaches [Droeschel and Behnke, 2011] have been used for human pose estimation.

This chapter proposes a compact, flexible and biologically inspired solution for safe pHRI in general, and collision avoidance in particular, that departs from the body of work reviewed above in the following aspects. First, for human keypoints estimation, no external sensor and no depth sensor is employed. Instead, a real-time pipeline that leverages deep learning methods for 2D pose estimation (e.g. [Cao et al., 2017, Insafutdinov et al., 2016]), in combination with disparity map computation from a binocular humanoid robot head, is presented. This choice of human estimation facilitates the deployment of robots in unstructured non-industrial environment where equipping external sensors is difficult or expensive. Second, I move significantly beyond solutions that consider the robot's end-effectors exclusively or those in which the robot's body is modelled by a set of geometrical collision primitives (e.g. spheres). The methods based on geometrical collision primitives often requires much computational effort, making it difficult to meet the real-time requirement. Instead, taking inspiration from the defensive PPS representation for the whole body in humans, I capitalize on the PPS representation developed by [Roncone et al., 2015, 2016]) (see Section 1.3.2) around the artificial pressure-sensitive skin of the iCub humanoid robot and provide extensions for the purposes of this work. Lastly, a novel robot controller is proposed. This combines end-effector reaching in 3D Cartesian space with simultaneous obstacle avoidance, with control points created dynamically on the fly on the robot's hands and forearms based on the PPS activations (as illustrated in Fig. 3.1).

This chapter is structured as follows. The next section discusses the Materials and Methods. This is followed by the Experiments and Results in Section 3.3, and finally the Discussion in Section 3.4.

The main content of this chapter has been published in [Nguyen et al., 2018b] except Section 3.3.1, where the vision based human pose estimation quality is compared with the tracking results from a wearable sensor suit.

3.2 Methodology

3.2.1 pHRI architecture

The software architecture of the pHRI framework, presented in Fig. 3.2, is implemented in C++ and Python with YARP [Metta et al., 2006a, Fitzpatrick et al., 2008]. Each node in the diagram represents a module in the pipeline.

- **Human pose estimation** processes images from a camera and generates human keypoints, presented in Phase 1 of Section 3.2.2;
- **Disparity mapping** builds the depth map of the environment from images from a stereo-camera. This is the result of previous work [Pasquale et al., 2016, Fanello et al., 2014];
- **Skeleton3D** constructs the 3D human pose estimation from 2D computations and the depth map of the environment (see Phase 2 of Section 3.2.2);
- **Peripersonal Space** serves as a visual-based collision predictor, and is described in Section 3.2.3;
- **pHRI Ctrl** (physical Human-Robot Interaction Control) translates the spatial perception of the robot into motion for safe interaction. The controller is detailed in Section 3.2.4.

3.2.2 Human keypoints estimation

In general, the purpose of human pose estimation algorithms is to provide the configuration of the human body from input image(s) or video. In this case, the input is the set of two images (with resolution of 320×240 pixels) coming from the iCub cameras. The iCub head-eye plant differs from common stereo camera systems in that the eyes/cameras are not fixed and move independently in space. This has two consequences: i) it allows for a compact, self-contained system that does not need any external device to perceive depth information, and ii) it is not possible to pre-calibrate the plant for the purposes of a disparity map computation–rather, calibration needs to be performed on the fly. Because of the latter, the 3D pose estimation algorithm is separated into two successive phases: (i) 2D pose estimation, and (ii) mapping of the 2D pose into the 3D Cartesian space of the robot. These are detailed below.

Phase 1 – 2D Pose Estimation

The 2D Pose estimation algorithm has the goal of computing the highest probability pixel locations of human keypoints in single camera frames; in this case: head, shoulders, elbows, hands, hips, knees and ankles. These locations are denoted as *keypoint pixels*—e.g. $[u_H, v_H]$ for the head. The positions of these body parts represent a simplified human model, which is suitable for the task at hand: The keypoints act as obstacles for the robot control algorithm. This work adopts the *DeeperCut* approach [Insafutdinov et al., 2016], a state-of-art DNN model for multi-person pose estimation. The main component of the algorithm is the parts detector constructed by *ResNet* [He et al., 2016], a deep learning architecture for object detection. In addition, DeeperCut uses an incremental optimization approach with integer linear programming [Pishchulin et al., 2016] and an image-conditioned pairwise term between body parts to improve the quality of the pose estimation. The model is then trained on the *Leeds Sports Poses* dataset [Johnson and Everingham, 2010] for multiple persons, and the *MPII Human Pose* dataset [Andriluka et al., 2014] for a single person. The implementation of this algorithm in our system (using Tensorflow[1] with a single NVIDIA [2] GTX1080i graphics processing unit (GPU)) can provide body parts poses at a frame rate of 30 *ms*. An example of the results from this algorithm can be seen in Fig. 3.6, panel A.

[1] `https://www.tensorflow.org/`
[2] `https://www.nvidia.com`

Phase 2 – 2D to 3D Pose Mapping

The second step of the human keypoints estimation algorithm is to reconstruct 3D poses of body keypoints thanks to the single-camera 2D information from Phase 1 and a disparity map computed from both cameras (as shown in Fig. 3.6, subplot B). For the reasons detailed above, the disparity map computation does not rely on a pre-existing camera calibration, but needs to rectify both cameras in real-time. For this reason, it is composed of an initial rectification algorithm followed by a disparity estimation step. The rectification algorithm aligns the two images to a common plane and keeps this transformation up to date with the robot's motion (neck, eyes and torso) [Fanello et al., 2014]. The disparity estimation step evaluates pixel displacements between the two rectified images [Pasquale et al., 2016], making use of the Efficient Large-Scale Stereo (ELAS) Matching algorithm [Geiger et al., 2010]. The outcome of this 2D to 3D mapping is to complement all the pixels from the left and right cameras with additional depth information in real time. As a result, it is possible to estimate the 3D Cartesian coordinates of each keypoint pixel estimated in Phase 1 from the depth computed above—that is, 3D keypoints coordinates $[x_H, y_H, z_H]$ can be computed from 2D keypoints pixels $[u_H, v_H]$. More specifically, the estimated 3D positions in the 7×7 pixel neighbourhood of each keypoint are averaged in order to improve robustness. In addition, biomechanical constraints of human body size and median filters on keypoints' 3D poses are applied in order to reduce the noise of estimation results. Importantly, this computation is performed in parallel with Phase 1 thanks to the YARP distributed software architecture. The human 3D pose estimation that results from this phase is shown in Fig. 3.6, panel C.

Moreover, the visual pipeline not only detects the body parts, but identifies them as well. The recognized body part identities (e.g. head vs. hands) are then exploited to modulate the robot's safety margin and, finally, to regulate robot behaviour. The software developed for this module is freely available [3].

3.2.3 Peripersonal space representation

For the purposes of this work, the visual receptive field (RF) size of the PPS representation, developed by [Roncone et al., 2015, 2016] (see summary and discussion in Section 1.3.2), is extended to a maximum of 45cm away from

[3] `https://github.com/robotology/skeleton3D`

every taxel, motivated by new findings regarding the PPS around human hands–the peri-hand PPS [Serino et al., 2015b], where the safety margin is suggested to be as large as 45cm in human. Fig. 1.5 schematically illustrates one such RF on the robot (in total, there are five RFs on every palm and 24 around each forearm).

In contrast to the work of Roncone et al., the visual RFs were not trained here; instead, they were designed uniformly for all taxels.The advantages of this approach are that uniform representation leads to similar responsive behaviour for every taxel, allowing the robot to initiate avoidance w.r.t. objects approaching from different directions. On the other hand, one loses the adaptive, learning element of the trained RFs. To preserve compatibility with the original implementation, the taxel RFs have a discrete representation divided in 20 bins that relate the distance of the stimulus/obstacle to activation, which in turn corresponds to the probability of eventual collision—see Fig. 3.3. The discrete representation is then interpolated using a Parzen window estimation algorithm, giving rise to the dark green curve in Fig. 3.3. Furthermore, in this implementation, the PPS representation can handle multiple objects in the environment concurrently, with every taxel deriving its response from the closest object. When the RFs of individual skin taxels are combined, a "safety margin" volume around the respective body parts is constructed. Such a "protective zone" around the forearm is visualized in Fig. 3.4 (a modulated/attenuated version is chosen for visualization—see Section 3.2.3 below). The change in the activation w.r.t. the distance (from closest to farthest) is denoted by the change in colour from red to light yellow, while the robot body is sketched in grey and black.

PPS Modulation

Inspired by the human PPS and its modulation (see Section 1.3.2), e.g. smaller safety zones around empty vs. full glasses of water during reaching [de Haan et al., 2014], a similar mechanism is implemented here. Such a case is illustrated schematically in Fig. 3.5: An overall increase/modulation of activation values changes the distance at which a certain activation point is reached by an oncoming stimulus, and hence the effective safety margin secured by the robot's responses is also adjusted. In this case, the modulation will pertain to the "sensitivity" of human body parts: for example, while it may be acceptable to come into contact with the hands of the human, the head should be avoided with a much larger safety margin.

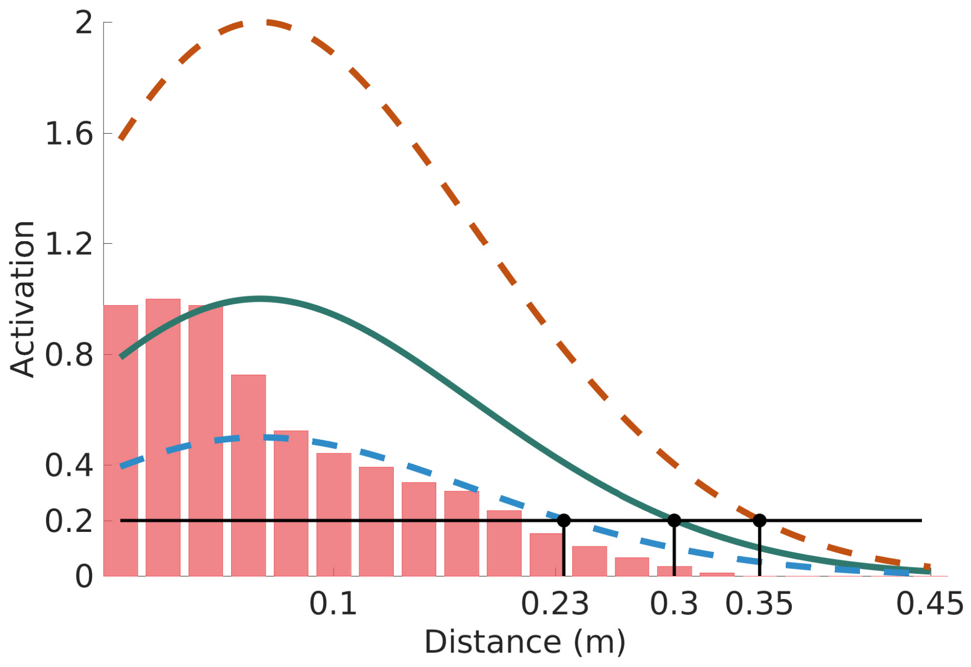


FIGURE 3.3: Activation curve of individual taxels' RFs, and effect of modulation. Pink bars represent the discretized representation stored into the taxel. The green curve is the result of the Parzen window interpolation technique, whereas the blue and brown dashed curves show the effect of modulation—response attenuation by 50% and positive modulation by 100%, respectively. The black line marks an activation threshold of 0.2, which roughly corresponds to distances from the origin of the RF of 23*cm*, 30*cm* and 35*cm* in attenuated, normal, and expanded cases respectively.

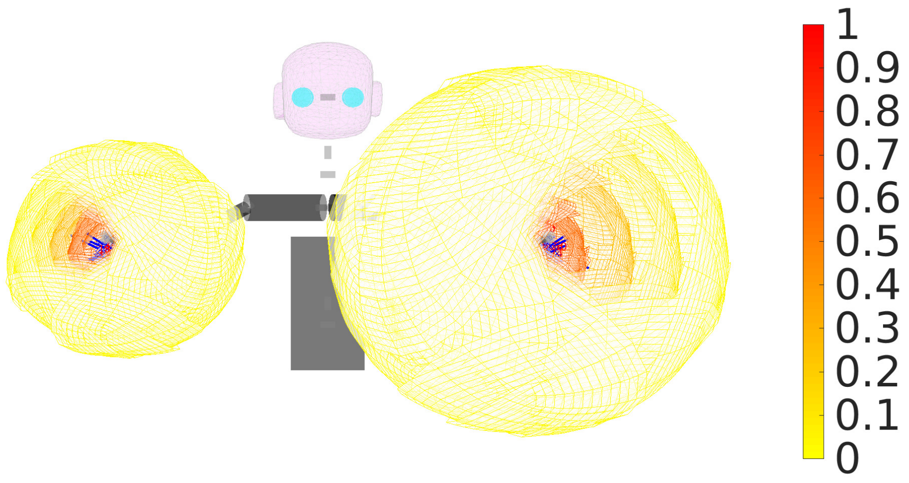


FIGURE 3.4: PPS visualization of the iCub's two forearms (front view). The picture shows two negatively modulated/attenuated PPS RFs (75% for the left forearm and 50% of the nominal RF for the right forearm). The colour scheme represents the magnitude of the activation in the volume surrounding the robot's arm, according to the scale in the sidebar.

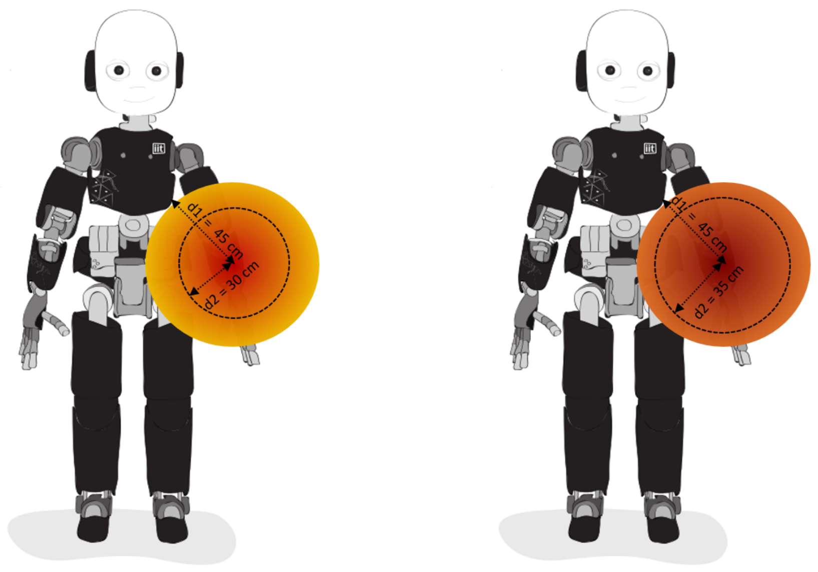


FIGURE 3.5: Peripersonal space modulation illustration. Receptive field extending 45cm and its Gaussian-like distribution of activations (highest at $d = 0cm$, lowest at the periphery). (a) Nominal RF. (b) Positively modulated RF.

A value between $[-1,1]$ was associated to each object as the "valence" $\theta(t)$, with negative values for stimuli where a smaller safety margin is allowed, and positive modulation for stimuli that should be avoided with a bigger margin—threatening or fragile objects for example. The final modulated PPS activation $a_{m,i}(t)$ of the i-th taxel w.r.t. an object with valence $\theta(t)$ is then calculated as follows:

$$a_{m,i}(t) = a_i(t)\big[1+\theta(t)\big] \tag{3.1}$$

where $a_i(t)$ is the PPS activation of i-th taxel at instant t. The mechanism is further illustrated in Fig. 3.3: the modulation simply translates the activation curve (y-axis). If associated with a particular activation threshold to trigger behaviour (say $a = 0.2$ which will be used here), this will be reached at different distances depending on the modulation—for example at $d2 = 30cm$ in the nominal case and $d2 = 35cm$ when subject to positive modulation – see Fig. 3.5. This gives rise to effective expansion/shrinking of the aggregated safety margin (composed of multiple RFs), as shown in Fig. 3.4. The software developed for this module is freely available online.[4]

3.2.4 Reaching with avoidance on the whole arm surface

The proposed controller has its roots in the Cartesian controller of [Pattacini et al., 2010] who proposed an inverse kinematics solver and minimum-jerk controller for the iCub robot. There, the solver was formulated as a nonlinear constrained optimization problem expressed in the joint position space, and makes use of the IpOpt library [Wächter and Biegler, 2006]. It is therefore decoupled from the controller part. In this work, a solution is proposed that unifies the inverse kinematics and the robot control problems into a single formulation. The problem is directly expressed in the joint velocity space and its solutions—joint velocities—can be directly used to control the robot. More specifically, at every time step $t = \bar{t}$ (with a period $T_S = 20ms$), the desired joint velocities $\dot{\mathbf{q}}^*(\bar{t})$ are computed by solving the following:

$$\begin{aligned}\dot{\mathbf{q}}^* &= \underset{\dot{q}\in\mathbb{R}^n}{arg\,min}\Big[\left\|\mathbf{x}_{EEd} - \left(\bar{\mathbf{x}}_{EE} + T_S\mathbf{J}(\bar{\mathbf{q}})\dot{\mathbf{q}}\right)\right\|^2\Big] \\ \text{s.t.} &\begin{cases}\mathbf{q}_L < \bar{\mathbf{q}} + T_S\dot{\mathbf{q}} < \mathbf{q}_U \\ \dot{\mathbf{q}}_L < \dot{\mathbf{q}} < \dot{\mathbf{q}}_U\end{cases}\end{aligned} \tag{3.2}$$

[4] https://github.com/robotology/peripersonal-space

where $\bar{\mathbf{q}} = \mathbf{q}(\bar{t})$ represents the instantaneous configuration of the n joints of the robot arm ($n = 7$ in this work), $\bar{\mathbf{x}}_{EE} = \mathbf{x}_{EE}(\bar{t})$ is the 6D pose of the end-effector in the Cartesian space (comprising of position and orientation), $\mathbf{x}_{EEd}$ is the desired end-effector 6D pose, and $\mathbf{J}(\bar{\mathbf{q}})$ is the Jacobian matrix. Eq. (3.2) models the end-effector reaching task as an optimization problem, specifically as a minimization of the distance between the desired end-effector pose $\bar{\mathbf{x}}_{EEd}$ and one-step-ahead prediction, giving the future pose that can be computed from the current pose $\bar{\mathbf{x}}_{EE}$, current joint configuration $\mathbf{q}(\bar{t})$, and joint velocities $\dot{\mathbf{q}}$ (the unknown). It is well established that, given a sufficiently small T_S, this can be approximated by the Jacobian map $\mathbf{J}(\bar{\mathbf{q}})$ multiplied by the vector of joint velocities $\dot{\mathbf{q}}$, which are the unknown of the problem. To ensure the feasibility of the optimal solution, the minimization needs to be carried out under a set of constraints that confine the one-step-ahead estimated joint position $\bar{\mathbf{q}} + T_S\dot{\mathbf{q}}$ within the feasible joint range $[\mathbf{q}_L, \mathbf{q}_U]$ (first row); furthermore, the estimated joint velocity $\dot{\mathbf{q}}$ is limited to be within the maximum and minimum ranges $[\dot{\mathbf{q}}_L, \dot{\mathbf{q}}_U]$ as per specifications of the respective robot actuators (second row). Other non-linear constraints can be conveniently added for further specialization of the control loop (see below). Again, the IpOpt library [Wächter and Biegler, 2006] is employed here.

The reaching task has to be reconciled with simultaneous obstacle avoidance, where the PPS representation can be capitalized. The idea ([Roncone et al., 2015, 2016]) is to aggregate the distributed PPS activations of any visually detected obstacle into a single locus and strength per body part (forearm or hand), using a weighted average of position $\mathbf{P}_C$, normal direction $\mathbf{n}_C$ (to the skin at the individual taxels), and activation $\mathbf{a}_{PPS}$ as follows:

$$\begin{aligned}
\mathbf{P}_C(t) &= \frac{1}{k}\sum_{i=1}^{k}\left[a_i(t)\cdot\mathbf{p}_i(t)\right] \\
\mathbf{n}_C(t) &= \frac{1}{k}\sum_{i=1}^{k}\left[a_i(t)\cdot\mathbf{n}_i(t)\right] \\
\mathbf{a}_{PPS}(t) &= \max_{i=1}^{k}\left[a_i(t)\right]
\end{aligned} \tag{3.3}$$

with subscript i denoting the i-th taxel, $i = 1 \ldots k$. The idea of PPS activation aggregation is illustrated in Fig. 3.6, where the high-resolution activations on the forearm and hand (panels 3 and 4) are combined into single vectors per body part – the red arrows in panel C. These aggregated vectors acting along the normal are schematically illustrated in Fig. 3.1. The weighted average position $\mathbf{P}_C$ is employed as a new control point C_i that can then be used to bring about "reaching" or avoidance behaviours along the normal $\mathbf{n}_C$. Only one task—either reaching or avoidance—can be accomplished at a time for a single control point, however. In this work, a novel solution is introduced to enable reaching with simultaneous obstacle avoidance by incorporating these additional control points into the controller described in Eq. (3.2) as additional joint velocity constraints. This remapping of the Cartesian "repulsive vectors" into joint space constraints is described by the following equations:

$$\begin{aligned} \mathbf{s} &= -\mathbf{J}_C^\top \cdot \mathbf{n}_C \cdot V_C \cdot a_{PPS} \\ \dot{\mathbf{q}}_{L,j} &= max\{V_{L,j}, s_j\}\,, \quad s_j \geq 0 \\ \dot{\mathbf{q}}_{U,j} &= min\{V_{U,j}, s_j\}\,, \quad s_j < 0 \end{aligned} \tag{3.4}$$

where C is a control point belonging to a generic robot link, $\mathbf{J}_C$ is its associated Jacobian, V_C is a gain factor for avoidance, and V_L, V_U are a predefined set of bounding values of joint velocity, i.e. $\pm 25 deg/s$ in this work. When projecting repulsive vectors into joint space, it is possible to obtain the value s_j, whose component $\mathbf{s}$ represents the "degree of influence" of the Cartesian constraint on the j-th joint. From these, the admissible upper ($\dot{\mathbf{q}}_U$) and lower ($\dot{\mathbf{q}}_L$) velocity limits of those joints influenced by the risk of collision are reshaped. In contrast to [Flacco et al., 2012], which inspired this approach, and where joints that would move towards the obstacle are stopped, my approach brings about an active avoidance behaviour. The avoidance action is thus proportional to the "threat level", a_{PPS}, and, for individual joints, to how much each joint can contribute in the current configuration. To improve smoothness, the desired target velocities are fed to a minimum-jerk filter; velocities are then integrated to compute target joint positions, which are directly fed to low-level position-direct motor controllers. This last step is standard to most robotic platforms—see [Pattacini et al., 2010] for details on this for in respect to the iCub.

The proposed approach is relevant to the robot control community in that it uses a constrained non-linear optimization technique for inverse kinematics and control. By sidestepping the computation of an analytical solution to inverse kinematics, the system is automatically immune from singularities; however, it may incur sub-optimal local minima. This problem is mitigated in practice, however, by the large number of degrees of freedom available. In future work, the framework can be complemented by Cartesian planning algorithms (cf. Section 3.4). To my knowledge, this is the first attempt at developing a robot control software that yields velocity profiles adopting nonlinear optimization. In addition, the novelty of this approach has been further enhanced by the integration of avoidance capabilities. Software developed for this module is also freely available online.[5] The details of the underlying computation are described in the Appendix A.

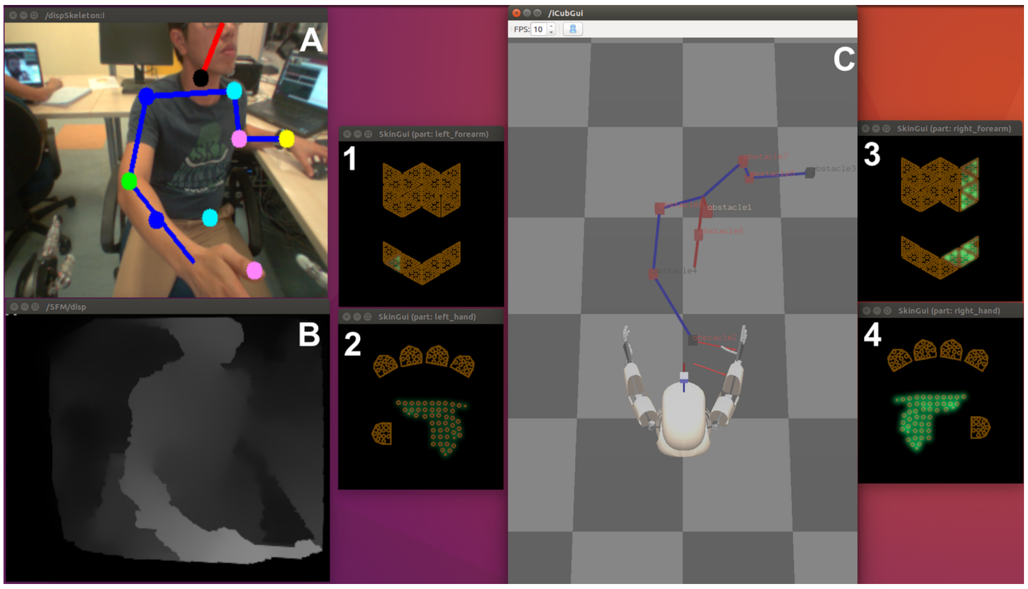
3.3 Experiments and Results

This section describes experimental results regarding three different HRI scenarios in which the robot executes pre-defined tasks (reaching for a position, following a trajectory) and the human experimenter interferes. For the purposes of this work, pilot experiments were performed in which trained human participants interact with the robot; these prototypical experiments (described in Section 3.3.2, 3.3.3 and 3.3.4) lay the groundwork for future user studies, which are out of the scope of this work. In all experiments, a minimum threshold of $a_{PPS} = 0.2$ was set for the avoidance behaviour to be triggered; also, the robot was commanded to either static or moving target positions, with a fixed orientation (palms pointing inwards). Results are reported using one arm of the robot with PPS around its hand and forearm, and control of seven joints of the arm; however, the framework operates in the same way for both arms and three torso joints could be toggled on using the very same controller. While the robot was being operated, data from several software modules (*skeleton3D*, *Peripersonal Space*, *pHRI Ctrl*) and robot sensors (joint encoders, cameras) were recorded and later analysed in Matlab.[6] Fig. 3.6 provides a static overview of the perception part of the pipeline. Please refer to the accompanying video for an overview of the setup and a qualitative evaluation of the performance. [7] This framework has been released under the LGPL v2.1 open-source license, and is freely accessible on Github; the control architecture is readily available for any iCub robot, and can be extended to other platforms.

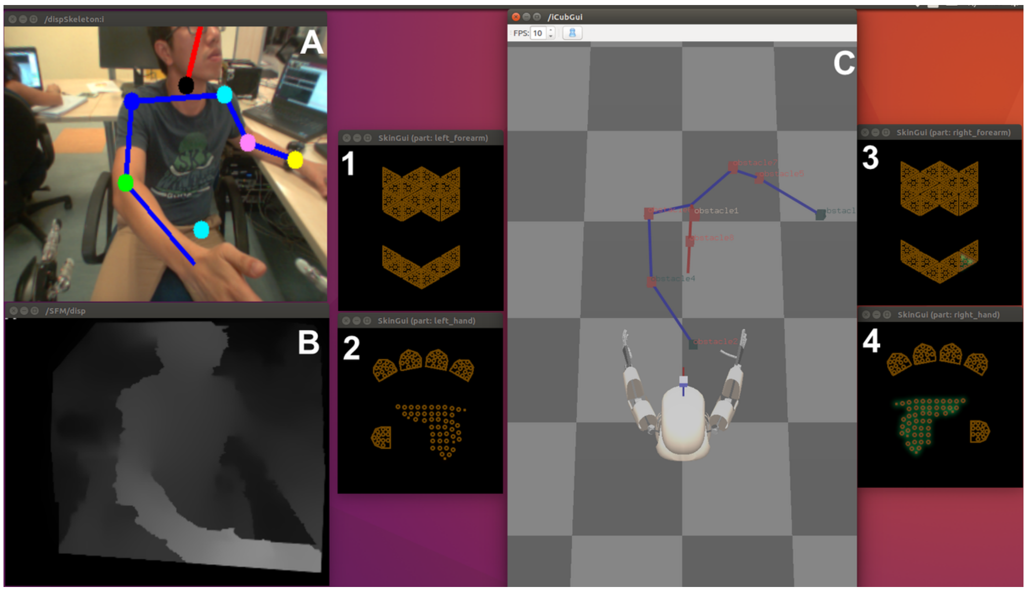
3.3.1 Human keypoints estimation benchmark

Before getting into the HRI experiments, the performance of the human tracking conducted by the vision system was evaluated by comparing the

[5]https://github.com/robotology/react-control
[6]https://www.mathworks.com/products/matlab.html
[7]https://youtu.be/A9Por3anPJ8



Normal (unmodulated) PPS. A human hand triggers high activation of the right palm and the inner part of the right forearm PPS.



Modulated PPS with attenuated response for hands and arms.

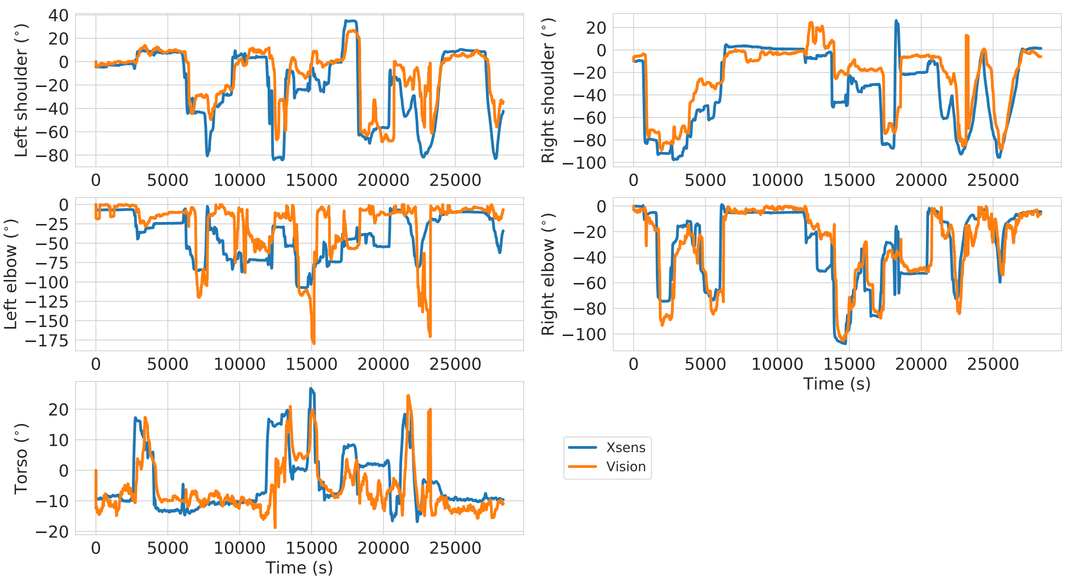
FIGURE 3.6: Perception, PPS representation, and its modulation on the iCub during interaction with humans. Panels 1,2,3,4: PPS activations on left forearm, left hand, right forearm and right hand, respectively visualized using iCub *skinGuis*. Taxels turning green express the activation of the corresponding PPS representation (proportional to the saliency of the green). Panel A: the human skeleton in 2D. Panel B: disparity map from stereo-vision. Panel C: estimated human skeleton in 3D alongside the iCub robot. The red arrows in this panel show the direction and magnitude of aggregated PPS activations on the iCub body parts w.r.t. the obstacle (a human right hand).

human configuration estimated by the proposed vision-based human tracking and that measured by the wearable sensory suit.[8] For this benchmarking procedure, the joint position measurements were converted to the relative angle between two successive limbs. In addition, the movable 6-DoF stereo-vision system of the iCub was replaced with a static stereo-vision camera, *rc_visard 160* [9], while the processing algorithms were kept unchanged. The root-mean-square error (RMSE) between the joint angles provided by the vision system and the wearable MVN Biomech suit was $6.93 \pm 3.86°$across all ten tracked joints (hips, knees, ankles, shoulders and elbows on both the left and right side). The detailed comparisons are shown in Fig. 3.7.

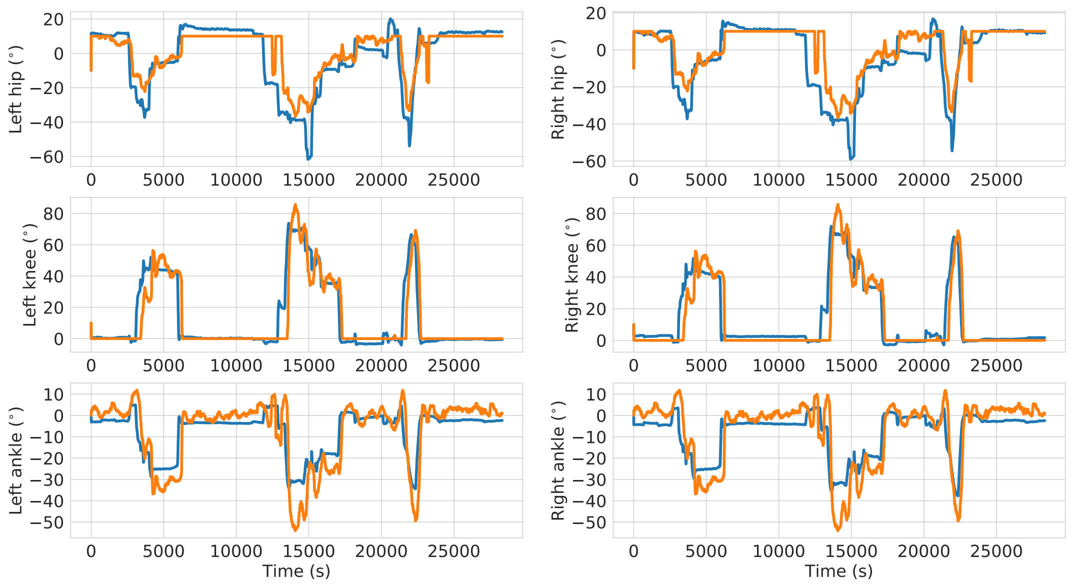
3.3.2 Reaching for a static target with simultaneous avoidance

In this experiment, the iCub was tasked with maintaining its end-effector at a predefined position (i.e. the control target was a static 3D point), while avoiding collisions when the human approached the robot body. Note that collision avoidance always has priority, since it is a constraint for the controller and needs to be satisfied at all times, whereas the reaching task is expressed as a criterion to be minimized. That is, when the human interferes, the robot should be able to avoid contact with the human at any given moment, departing from its predefined static target when necessary. Results from this experiment are shown in Fig. 3.8. The human body parts activate the PPS when they enter their RFs ($45cm$ zone from the skin surface), and increase the activations if they continue to get closer to the robot's arm. There is no effect on joint velocities, however, until the activations reach the threshold of 0.2, which corresponds to approximately $30cm$ away from the skin surface (shown by the dashed green straight line in top two panels; cf. Fig. 3.3). The end-effector error is minimal there. After $t \simeq 2.7s$ into the experiment, the human body parts induce super-threshold PPS activations at the robot hand and, partially, at the forearm. This propagates into the robot control algorithm which adaptively tunes the joint velocity limits for all affected joints, as specified by Eq. (3.4). As shown in Fig. 3.8, panels 3 and 4, the range of velocity limits is reduced and the joint velocities are consequently constrained such that avoidance is generated (only two joints out of seven are shown for clarity). The activations on the robot's left hand influence all the joints in

[8]Xsens Technologies BV, https://www.xsens.com/products/xsens-mvn-animate/
[9]Roboception, https://roboception.com/en/rc_visard-en/



(A) Upper body



(B) Lower body

FIGURE 3.7: Comparison results of human keypoints estimation between vision system and Xsense wearable suit

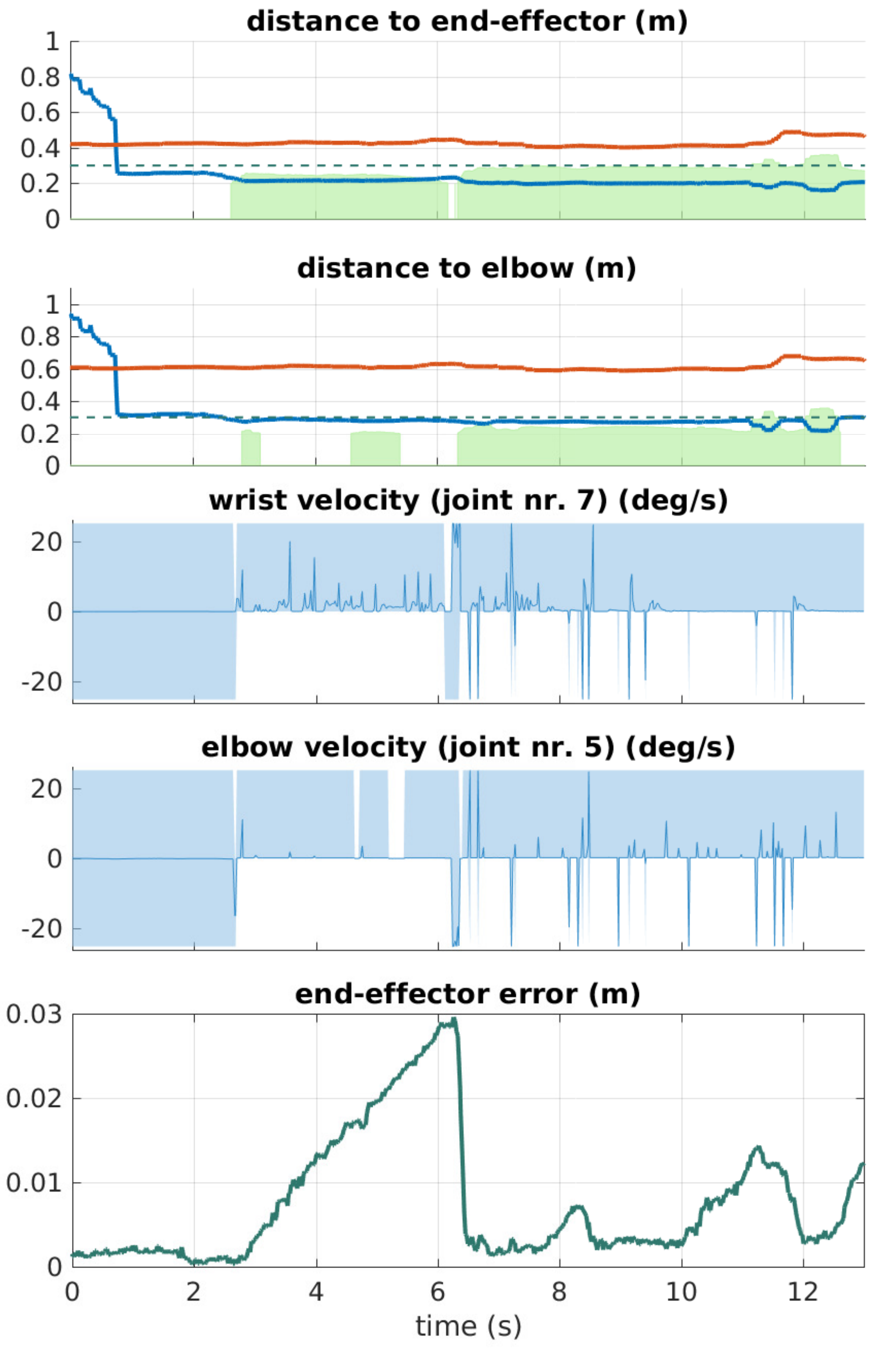


FIGURE 3.8: Reaching for a static target with avoidance. Top two panels (*Distance-Activation*) for robot end-effector/elbow: blue and orange lines – distance from left hand and head of human respectively; light green areas: PPS activations $\mathbf{a}_{PPS}$ on robot body parts (hand – top panel, forearm – 2nd panel); green dotted line – distance at which *PPS* activation exceeds 0.2 and avoidance is activated (cf. Fig. 3.3); Panels 3-4 (*Joint velocity*): Joint velocity (in blue) and their adaptive bounds (light blue band) – two selected joints only. Bottom panel (*End-effector error*): Euclidean distance between reference and actual position of the end-effector.

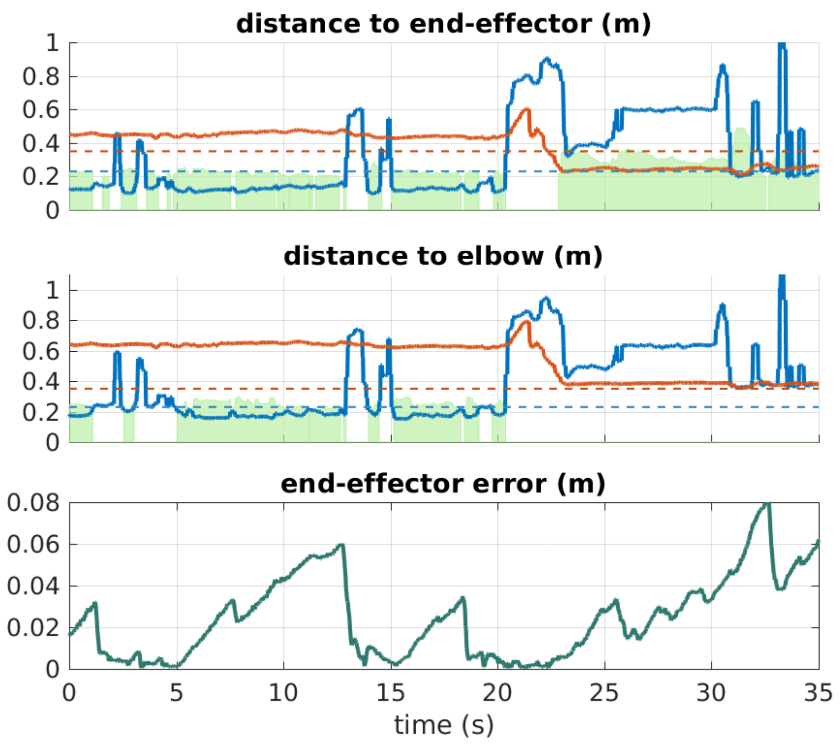


FIGURE 3.9: Static target and PPS modulation: human hand vs. head. See Fig. 3.8 for explanation of individual panels and text for details.

the chain, while forearm activations influence only those from the elbow up (more proximal). Eventually, the robot's end-effector cannot stay at the desired position but avoids the human body parts when they approach, shown by the growing error in the bottom panel of Fig. 3.8 (e.g. after 2.7*s*).

3.3.3 Reaching with modulation for different body parts (human head vs. hand)

The setup for this experiment was similar to Section 3.3.2, the main difference being that different valances were assigned for the different human body parts, as specified in Section 3.2.3. This directly relates to a realistic human-robot interaction in which the safety of some body parts (e.g. the head) should be guaranteed with a bigger margin than for others (e.g. arms). To illustrate the principle, a 50% PPS attenuation was applied at the hands (i.e. $\theta = -0.5$ in Eq. (3.1)); see also blue dashed curve in Fig. 3.3 and left forearm PPS in Fig. 1.1), while the PPS pertaining to the human head was positively modulated (valence 1.0; red dashed curve in Fig. 3.3). A potential interaction scenario that can take advantage of PPS modulation is physical human robot cooperation in a shared environment, where the robot may need to come into contact with the human hands to receive or hand-over objects (active contacts), but must always avoid any collisions with their head. Results from the experiment are reported in Fig. 3.9 and structured similarly to Section 3.3.2, with the exception that, for the sake of clarity, joint velocity plots are not reported. Due to the reduced safety margin, the human left hand (blue line in panels 1 and 2 of Fig. 3.9) could get close to the robot's end-effector and elbow, respectively, while it only activated the robot's PPS slightly, just above 0.2 (before $t \simeq 5s$). As a consequence of this, only small regulations were applied to the joint velocity bounds, and the robot could still perform the task successfully (as shown by the small error in panel 3 of Fig. 3.9, $t \in [0,5]s$). At $t \simeq 22.5s$, the human head entered the PPS of the end-effector and triggered a strong response within the PPS representation. In this circumstance, therefore, in order to preserve safety, the robot could not maintain the reaching task (the end-effector error in panel 3 increases) but was successful in maintaining a safe margin from the human head.

3.3.4 Following a circle while avoiding human

In this experiment, the robot was commanded to follow a circular trajectory with the left arm, while the human interfered with this task, hence triggering the avoidance behaviour. The valences of human body parts were kept the same as in Section 3.3.3 (attenuation for hand; boosting for head). Results are shown in Fig. 3.10, with four joint velocity subplots (two for the elbow and two for the wrist). Similar to the static reaching case, when the human parts approached close enough to the robot's arm ($t \simeq 8s$), the controller chose to avoid the human rather than continuing to follow the desired path. This behaviour can be recognized by the relationship between distances-activations (in the *Distance-Activation* panels in Fig. 3.10) and the changes in the joint velocity bounds (in panels 3 to 6). Without the interference of the human (e.g. before $t = 8s$), the bounds of the joint velocities remained at the preset values ($\pm 25 deg/s$), indicating the successful tracking behaviour of the robot's end-effector on the desired trajectory (small error shown in the panel 7). Conversely, the joint velocity bounds were dynamically adapted when the human approached (the blue band reduces), thus causing the robot to deviate from the demanded trajectory (error increases cyclically after $t \simeq 8s$).

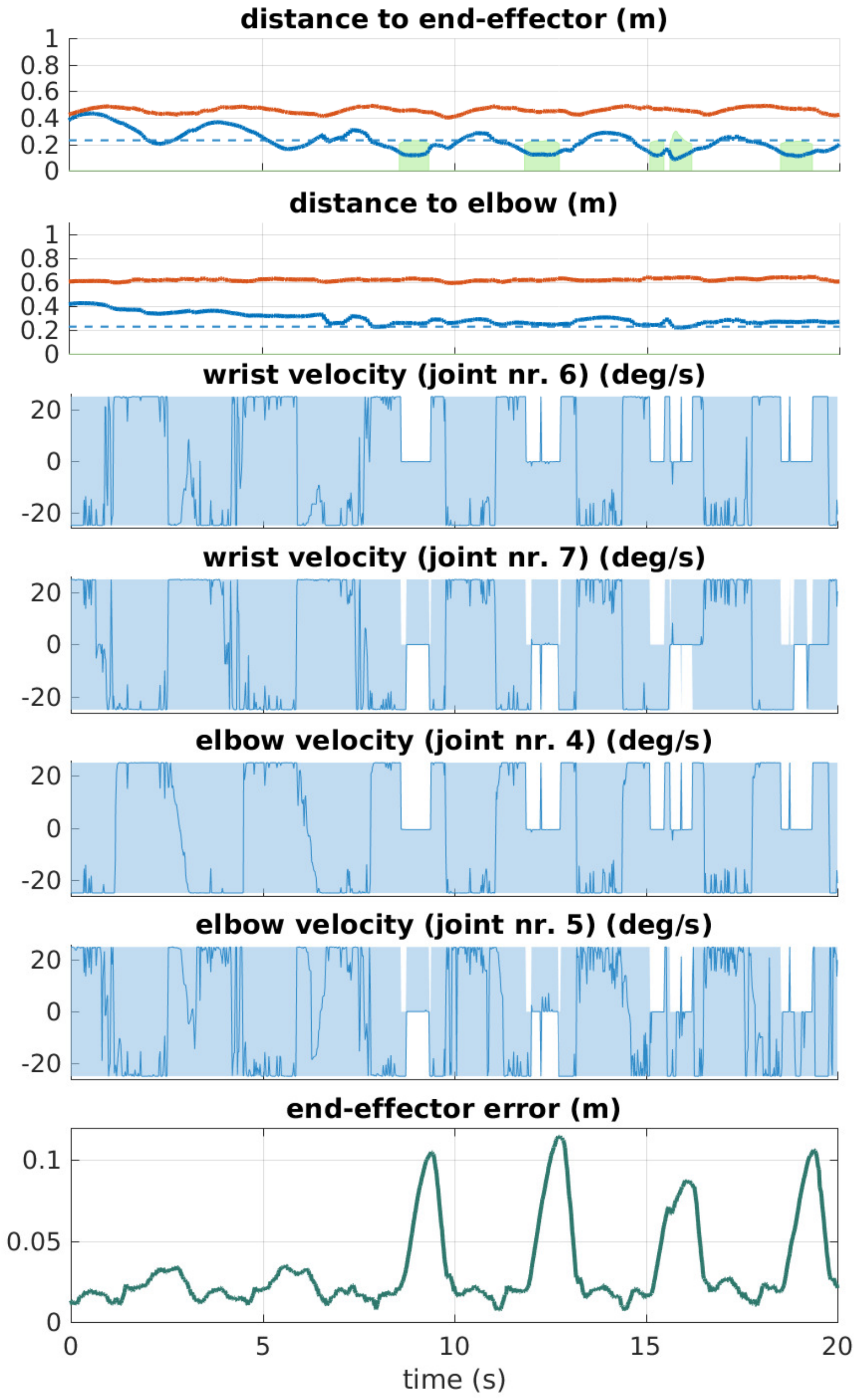


FIGURE 3.10: Moving target on a circle. See Fig. 3.8 for explanation of individual panels and text for details.

3.4 Discussion

This chapter develops and tests a new framework for safe interaction between a robot and a human. The framework is made of the following main components: (i) a human 2D keypoints estimation pipeline employing a deep learning-based algorithm, extended here into 3D using disparity; (ii) a distributed PPS representation around the robot's body parts; (iii) a new reactive controller that incorporates all obstacles entering the robot's protective safety zone into the task on the fly. The main novelty lies in the formation of the protective safety margin around the robot's body parts—in a distributed fashion and adhering closely to the robot structure—and its use in a reactive controller that dynamically incorporates threats in its PPS into the task. The framework was tested in real experiments that reveal the effectiveness of this approach in protecting both human and robot against collisions during the interaction. The proposed solution is compact and self-contained (onboard stereo cameras in the robot's head being the only sensor) and flexible, since different modulations of the defensive PPS are possible—here I demonstrate stronger avoidance of the human head compared to the rest of the body.

Relying solely on visual perception of the human is, however, not enough to warrant safe interaction under all circumstances. Additional safety layers would naturally fall into the *post*-impact phase. In the iCub humanoid robot, this could be contacts perceived on the artificial skin or from the force/torque sensors located in the upper arm. Such contacts can be seamlessly integrated into the controller presented here, making the whole framework multimodal and more robust (see the next Chapter 4). At the same time, the proposed solution is not restricted to the iCub humanoid robot, and its adaptation to other platforms (with RGB-D sensors instead of stereo cameras; without artificial skin; with a different number of DoF etc.) would be straightforward. The "pHRI controller" presented here is unique in that it combines a local inverse kinematics solver with a controller in a single module. This controller will be further extended to enable processing of multiple targets in Cartesian space—for different control points on the robot body—and to couple with a global whole-body planner (cf. Chapter 5). Finally, the controller need not only consider static distances between the robot and the human, but both human and robot velocities could be taken into account (as dealt with by [Roncone et al., 2016] and [Magnanimo et al., 2016], respectively).

Chapter 4

Merging physical and social interaction for effective human-robot collaboration

4.1 Introduction

Removing the safety cages around robots and bringing them to work alongside human workers requires new capabilities that would eventually enable robots to analyse and assess both the physical (e.g. position and movement of humans, and the presence of objects and useful tools) and the social properties of the environment (e.g. emotions and intentions of humans). These requirements translate into the development and, occasionally, further improvement of fundamental "skills", ranging from accurate perception (including human actions) up to the representation of the acquired information in the form of a "shareable knowledge" across different skills. The ultimate goal is to plan and execute generic tasks safely and effectively on the factory floor, as well as to assist humans in their daily chores. More pragmatically, given the current level of technological development, we need to resort to a variety of computational techniques such as symbolic and sub-symbolic Artificial Intelligence (AI), machine learning, vision, planning and control. The task complexity that is addressed in this chapter is still beyond reach of a single technique "end-to-end", i.e. a comprehensive approach that connects the raw sensory input down to the motor control output. Notable exceptions can be found in the work of [Gu et al., 2017] albeit deployed in simulated environments.

Instead, this chapter takes the humbler, but extremely practical, approach of combining methods from the pHRI and sHRI domains. It must be noted that technology is indeed mature enough to deploy markerless perception of the human body in 3D (similar to what has been shown in the previous

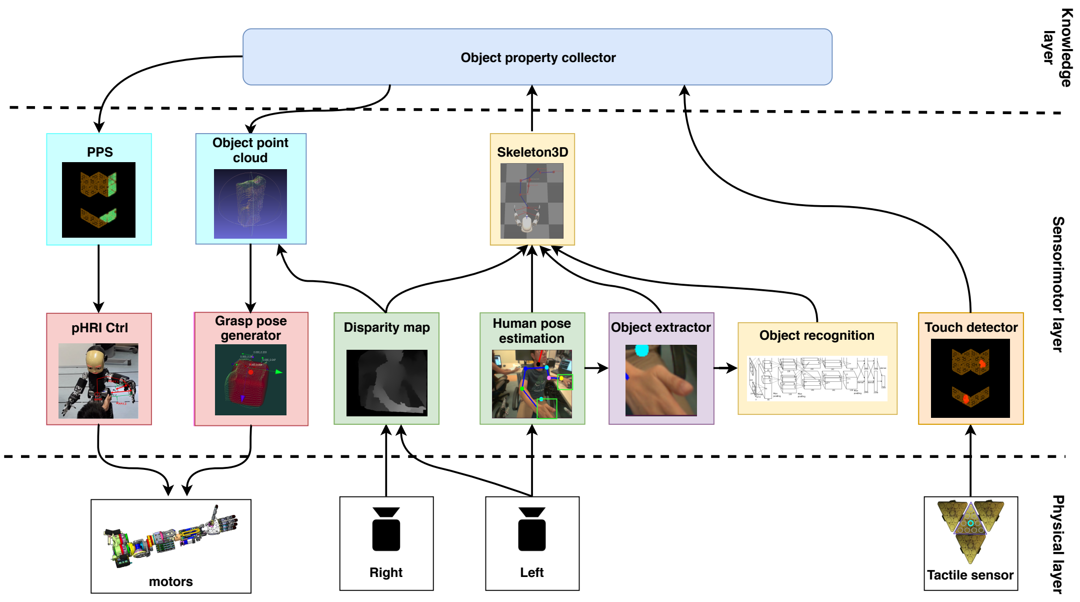


FIGURE 4.1: Overview of the overall system comprising perception (right side) and action (left side) pathways. At the physical level, perception includes vision and touch. The robot's visual system allows for stereoscopic vision. Low-level motor control makes it possible to specify the position trajectories of the joints, exploiting as feedback a combination of pressure (from the tactile sensors) and force information (from a number of 6-axis force-torque sensors located on the robot's structure). The sensorimotor layer transforms raw sensory data into symbolic tokens (e.g. object identities, posture, 3D shape, human body posture, etc.) that can be easily stored into the "object property collector" database. This symbolic knowledge is used to control action, for example to avoid contacts rather than to grasp objects, through reasoning modules (i.e. *PPS, Object point cloud, pHRI Controller, and Grasp pose generator*).

Chapter 3), to recognize and model generic objects for grasping (e.g. [Vezzani et al., 2017]), seamlessly integrating visual perception with whole-body force/torque control, tactile sensing and speech-based communication, to name a few. In addition, machine learning provides the ability to teach the robot about new objects and tools, whose descriptions can be stored and organized – at least for the scope of these experiments – into a standard database. To set the stage for this work, I start by briefly reviewing the strengths and limitations of some recent pHRI and sHRI architectures.

Most pHRI frameworks focus on "low-level" interaction (e.g. contact detection, avoidance and control) and consider humans merely as other objects in the workspace that the robot needs to deal with. In their architecture, De Luca and Flacco [2012] integrate residual-based collision detection and reaction (gathered from the proprioceptive layer) with collision avoidance (leveraged on depth information provided by a Kinect sensor). Since this work models the environment by considering only obstacle-to-robot distances, it is arguable that this approach is not flexible enough to scale up to more complex collaborative tasks. Haddadin et al. [2011] outline a different solution where the robot simply switches between different functional modes (e.g. autonomous task execution with/without humans, cooperation with humans) based on the state of the humans in the robot's workspace, as detected through proximity sensors surrounding the robots. Although this approach combines different control techniques (e.g. impedance control, residual-based collision detection, and task relaxation reaction) as well as several sensory modalities (e.g. laser-based imaging, visual-based edge filtering) to offer safe HRI, it lacks a knowledge-based communication channel and thus it is not seamlessly extendable to other applications. It does, however, do a very good job in implementing flexibility in the physical interaction layer.

At the other end of the spectrum, sHRI frameworks often overlook the physical aspects of the interaction (e.g. safety and physical contacts) or simply resort to path planning methods to deal with static or slowly changing environments. For example in a recent work, Lemaignan et al. [2017] present a cognitive architecture for service robots that supports human actions and decisions. It only utilizes path planning based methods [Sisbot and Alami, 2012, Mainprice et al., 2011], however, to guarantee that the path of the robotic manipulator is collision free, which is difficult to satisfy in highly dynamic environments, and in particular when interacting with a human partner.

Moulin-Frier et al. [2017a] and Fischer et al. [2018] have developed the so-called DAC-h3 cognitive architecture. One of DAC-h3's main strengths is its implementation, which is an ensemble of functional modules. Functional modules can be mapped one-to-one to the software modules of typical middleware systems. The authors validate DAC-h3 by experimenting with human-robot and robot-object interaction to acquire and express procedural knowledge. Although the robot can execute a wide repertoire of actions, such as waving, pointing, pulling and pushing objects in a table-top setting, DAC-h3 exclusively employs predefined motor primitives and does not address the problem of safety (e.g. avoiding human and moving objects). Nonetheless, it is a very good reference implementation for sHRI. Along the same lines, in [Moulin-Frier et al., 2017b], the authors integrate diverse AI techniques into a single cognitive architecture that combines symbolic reasoning with embodied behaviours, yet they do not consider the "low-level" details of physical interaction. Notably, all the aforementioned systems employ fiducial markers and/or exteroceptive sensors to enhance the robot perception. Nonetheless, certain elements of DAC-h3 are readily combined into this system. In Section 4.4, for example, I speculate about a possible integration with DAC-h3.

To summarize, the main contribution of this chapter is the design of a control system that merges elements of pHRI and sHRI, namely:

- A compact human-centred visual perception system for humanoid robots;
- A visuotactile reactive controller that allows the robot to react safely in both *pre*- and *post*-collision phases;
- A simple symbolic "storage" of information about humans, objects and tools, supporting social interaction.

Some parts of the system presented in this chapter were developed and analyzed in the previous Chapter 3, namely the human keypoints estimation, the reactive controller and the PPS. Here I demonstrate that my approach can handle different types of interaction effectively. Two main experiments are developed, where the robot is given an object by the human partner and it is subsequently asked to grasp another object from a table top to perform a handover task.

The remainder of the chapter is organized as follows: the method is presented in detail in Section 4.2, and experiments and results are analysed in Section 4.3. Finally, in Section 4.4, the possibility of integrating this work into existing cognitive architectures is explained in detail.

The main content of this chapter is published in [Nguyen et al., 2018c].

4.2 Methodology

4.2.1 General architecture

The underlying architecture of this framework is shown in Fig. 4.1, where *functional modules* are classified into the three different layers described below:

- The **physical layer** consists of the low-level systems of the iCub humanoid [Metta et al., 2010]: the *stereo-vision*, the *artificial skin* covering the robot body, and the *joint actuators*. This layer allows the robot to perceive the surroundings as well as act on the environment.

- The **sensorimotor layer** encompasses those modules responsible for processing the raw signals produced by the physical layer in order to yield meaningful internal representations: the *touch detector*, the *disparity map*, the *human pose estimation*, the *object extractor*, the *object recognition* and the *skeleton3D* for visual input. Components responsible for the computation of control signals are also listed here, such as the *Peripersonal Space*, the *pHRI controller*, the *Object point cloud* and the *Grasp pose generator*.

- The **knowledge layer** contains the *Object properties collector* module (discussed in detail in Section 4.2.3), whose task is to store and manage the properties of the entities perceived from the environment. In the human-oriented designed environment, this perceived sensory information not only constructs the internal model of the world (recall Fig. 2.3), but also contains the common knowledge. This element plays an essential role for high-level social interactions of robots with humans, laying down the fundamentals for social skills, e.g. language communication, emotion perception, usage of natural cues, etc. [Dautenhahn, 2007].

4.2.2 Environment acquisition and perception:

Human detection and tracking

Chapter 3, proposed a real-time framework to estimate the 3D pose of humans from the six DoF stereo vision system mounted in the iCub head. The framework was composed of two steps: (1) a 2D human pose detection given as a set of keypoint pixels $[u_i, v_i]$ extracted from the raw images using the *DeeperCut* model [Insafutdinov et al., 2016]; (2) a 3D human pose reconstruction from 2D information and a depth map. The latter was performed by averaging the spatial projection of each 2D keypoint along with its neighbours through the depth map. The output set of 3D coordinates $[x_i, y_i, z_i]$ was then refined by applying median filtering.

Context-aware object detection and tracking

To provide the robot with a context-aware ability during the collaboration with the human partner, the above human tracking framework was extended to incorporate object recognition. To this end, I adopt the method developed by [Pasquale et al., 2015], which turned out to be simple yet efficient in our setting, where the system must detect and recognize objects held by the human. The proposed image recognition system utilizes a *CaffeNet* [Krizhevsky et al., 2012] DNN model, pre-trained on the ImageNet dataset [Russakovsky et al., 2015], to extract features from the input images, and a *Regularized Least Squares* method for the classification stage. The algorithm also allows partners to train the robot with novel objects via verbal annotation.

Unlike [Pasquale et al., 2015], who employed a heuristic motion detector to acquire cropped images for the DNN, I propose an accurate and flexible solution specifically designed for the HRI context. By resorting to the keypoint pixels obtained from the 2D human pose computation, the bounding boxes containing the human hands (with or without objects) can be precisely estimated in real-time, and are successively passed on to the image recognition system for labelling purposes. The size of the bounding boxes are constantly adapted based on the distance of the human hands, as retrieved from the depth map.

Physical collision detection through artificial skin

As introduced in Section 2.3, the iCub body is covered with a layer of artificial skin composed of capacitive tactile sensors [Maiolino et al., 2013], termed *skin taxels*. The poses of the skin taxels are calibrated with respect to the kinematic model of the robot, and are kept updated during robot movements. Thus, physical contacts with the iCub skin can be accurately sensed and localized w.r.t the robot's Root FoR. Notably, this approach differs from other recent methods (e.g. [Haddadin et al., 2017b]) that typically rely on proprioceptive inputs instead. To reduce spiking effects, multiple adjacent tactile contacts firing concurrently over a preset threshold can be aggregated into one representative *super contact*, whose activation a^t_{PPS} corresponds to the highest pressure value measured at the relative taxels. The *super contact* is also parameterized in terms of its location $\mathbf{P}^t_C$ and normal vector $\mathbf{n}^t_C$, which also encodes, to a first approximation, the collision direction (Fig. 4.2).

4.2.3 Centralized knowledge representation through Object Properties Collector (OPC):

In order to cooperate effectively with humans, robots not only need to perceive the surroundings through their sensors, but have to convert these representations into a "common knowledge" that can be shared with their partners to support reasoning and task planning. For this purpose, an ontology based framework [Lallee and Verschure, 2015] is adopted for knowledge representation. These representations can be considered to be the centralized working memory of the robot during the interaction with the environment. Thereby, the framework partially solves the grounding problem [Harnad, 1990] of pure symbolic cognitive systems, where environment stimuli are firstly transformed into lower dimensional representations with machine learning methods, and then are mapped into symbols (given *a priori* or through interactive learning [Pasquale et al., 2015]) in a database.

In this regard, the working memory makes use of the *Entity* and the *Relation* to identify basic concepts and the connections between different entities, respectively. Thus, the perceived objects are denoted as *Objects*, an abstract type of *Entity*, composed of some physical properties such as position, dimension or valence (further properties like objects' affordances can be added easily). On the contrary, objects that have self-motion abilities are deemed as *Agents*: humans and robots fall under this class. A structured hierarchy of classes can be constructed along the same lines, comprising e.g. *Bodies*, *Emotions*, *Beliefs* etc., as described in [Lallee and Verschure, 2015].

Leveraging on this knowledge management, the human partner can be represented within the OPC memory as an *Agent* whose *Body* parts are localized in 3D through vision (see Section 4.2.2). A similar process applies to the visually recognized objects along with their properties (e.g. location, colour, valence) that are relevant to the task at hand.

4.2.4 Learned Peripersonal Space (PPS) as an adaptive embodied perception layer

This chapter continues to rely on the adaptable PPS representation from the previous Chapter 3 (cf. Section 3.2.3). It is worth recalling that this representation serves as a distributed safety zone around the robot body and can be modulated (expanded or shrunk), depending on the identity of the visually-detected objects. Formally, the modulated PPS signal $a_{m,i}(t)$ occurring at the *i*-th taxel w.r.t. the valence–threatening value $\theta_k(t)$ of the *k*-th object at time instant *t* is given by:

$$a_{m,i}(t) = a_i(t) \cdot [1 + \theta_k(t)] \tag{4.1}$$

where $a_i(t)$ represents the original PPS activation.

Also, this representation serves as a mapping of visual stimuli to body parts of the robot, parametrized in terms of location $\mathbf{P}^v_C$ and normal vector $\mathbf{n}^v_C$ of magnitude a^v_{PPS}.

Furthermore, this adaptable protective layer can be exploited for different collaboration contexts. Human body parts that are meant to be contactable during the cooperation (right hand holding an object) would entail lower activations than other parts (left hand). This allows the robot to approach the right hand for a close interaction while handing over the object and to avoid collisions with other parts (i.e. left hand, head). This contextual modulating mechanism can be synthesized as follows:

$$a_{m,i}(t) = \begin{cases} \min\Big(a_i(t),\, a_i(t)[1+\theta_k(t)]\Big) & k \text{ contactable} \\ \max\Big(a_i(t),\, a_i(t)[1+\theta_k(t)]\Big) & \text{otherwise} \end{cases} \tag{4.2}$$

4.2.5 Controllers:

Extension of bio-inspired reactive controller for safe physical interaction

Most robot movements in the interaction scenarios can be formalized as reaching with obstacle avoidance. To this end, a reactive controller tasked with solving a nonlinear constrained optimization problem was proposed earlier in 3.2.4, which is recalled here as follows:

$$\dot{\mathbf{q}}^* = \arg\min_{\dot{\mathbf{q}} \in \mathbb{R}^n} \left\| \bar{\mathbf{x}}_{EEd} - \left(\bar{\mathbf{x}}_{EE} + T_S \cdot \mathbf{J}(\bar{\mathbf{q}}) \cdot \dot{\mathbf{q}}\right) \right\|^2 \tag{4.3}$$

In this setting, visually perceived objects elicit PPS activation, thus reshaping the movements of the robot's parts through the PPS representation by adapting the joint velocity limits in real-time . Remarkably, the controller can respond in a similar manner to tactile stimuli, hence dealing with *post*-collision scenarios. Fig. 4.2 depicts the occurrence of a physical contact, eliciting the activation of a skin taxel on a robot body part. As a result, the bounding velocity values of the corresponding joints (e.g. mainly the elbow joint, as visible in Fig. 4.2) are adapted accordingly. In formal terms, the joint

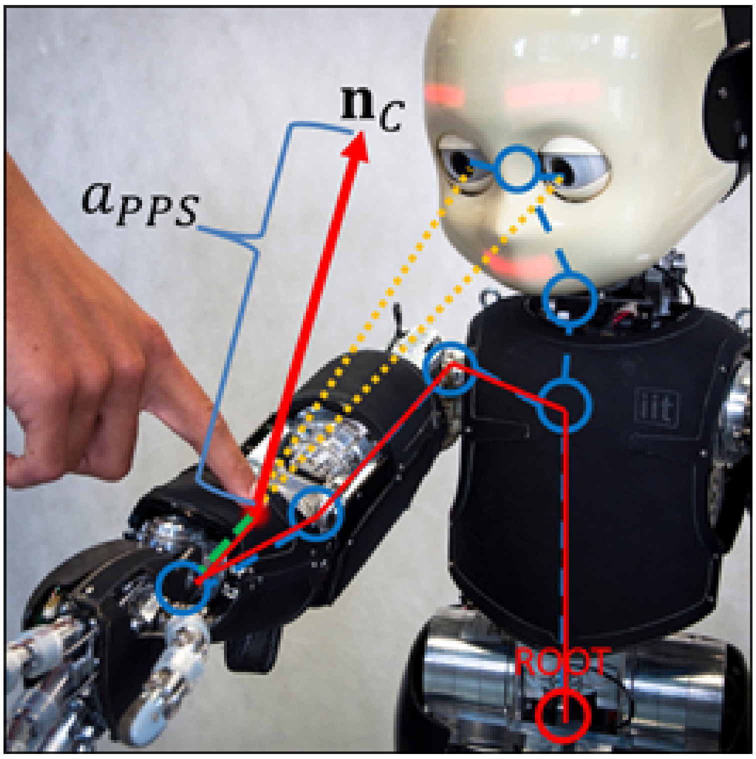


FIGURE 4.2: React-control dealing with a tactile stimulus. The diagram shows the quantities involved when a control point is elicited upon the detection of a real contact (physical contact triggered by tactile sensors).

velocity constraints of Eq. (4.3) relative to both visual and tactile inputs are expressed by:

$$\begin{aligned} \mathbf{s} &= -\mathbf{J}_C^\top \cdot \mathbf{n}_C \cdot V_C \cdot K \cdot a_{PPS} \\ \dot{\mathbf{q}}_{L,j} &= \max\left\{V_{L,j}, s_j\right\}, \quad s_j \geq 0 \\ \dot{\mathbf{q}}_{U,j} &= \min\left\{V_{U,j}, s_j\right\}, \quad s_j < 0 \end{aligned} \tag{4.4}$$

where subscript C denotes a control point attached to a generic robot link, represented by either a *mapped PPS locus* or a *super contact*, depending on the nature of the input signal (i.e. visual or tactile, respectively). The difference compared to Eq.(3.4) in Chapter 3 is the tuning gain K, which is intentionally introduced for the purpose of multimodal integration. In particular, the gain K is set to be higher for tactile events than visual ones (i.e. three times in this implementation), reflecting the notion that a physical collision detected by the skin system is more critical than a collision predicted from the visual input.

4.3 Experiments and Results

In this section, the proposal is evaluated in two different hand-over experiments.

- *Safe human-robot hand-over task*, where the robot receives an object from the human.
- *Safe robot-human hand-over task*, where the robot has to pick up an object from a table to then perform hand-over to the human.

The experimental setup is presented in the Fig. 4.3, where there are two tables, denoted as *Table 1* and *Table 2*. The human partner sits next to *Table 1* and cannot reach objects located on *Table 2*, whilst the robot can reach and grasp objects lying on *Table 2*. This setup is designed to simulate the situation where the robot and the human need to cooperate to complete a shared task, such as moving an object from *Table 1* to *Table 2* or *vice versa*. Intentionally, the hand-over phase of the robot's action is set long enough (at least 15s) to enable any possible physical interaction with the partner. Please refer to the accompanying video[1] for an overview of the setup and the performance of the tasks.

4.3.1 Safe human-robot handover task

In this experiment, when engaged by the human partner, the robot has to look for the requested object the human holds in his hand, take it, and place it on the table. During this interaction, the robot's movements can be interfered with by the human, and hence the robot needs to react to possible collisions and adjust its planned motion in order to guarantee a safe cooperation. The dialogue between the human and the robot can be scripted as below:

PARTNER : Hi iCub, help me put the DUCK in the basket!

ICUB : I don't have the DUCK. You have the DUCK. Please give it to me!
(PARTNER *shows the* DUCK *to* ICUB*, and* ICUB *moves its hand to receive the* DUCK *from* PARTNER)

In detail, the iCub locates the human hand and the object with its visual system, then moves its hand to approach the object. As is evident in Fig. 4.4 and Fig. 4.5, the valances of the human right hand holding the object, as well

[1]https://youtu.be/zNbLCC10qX4

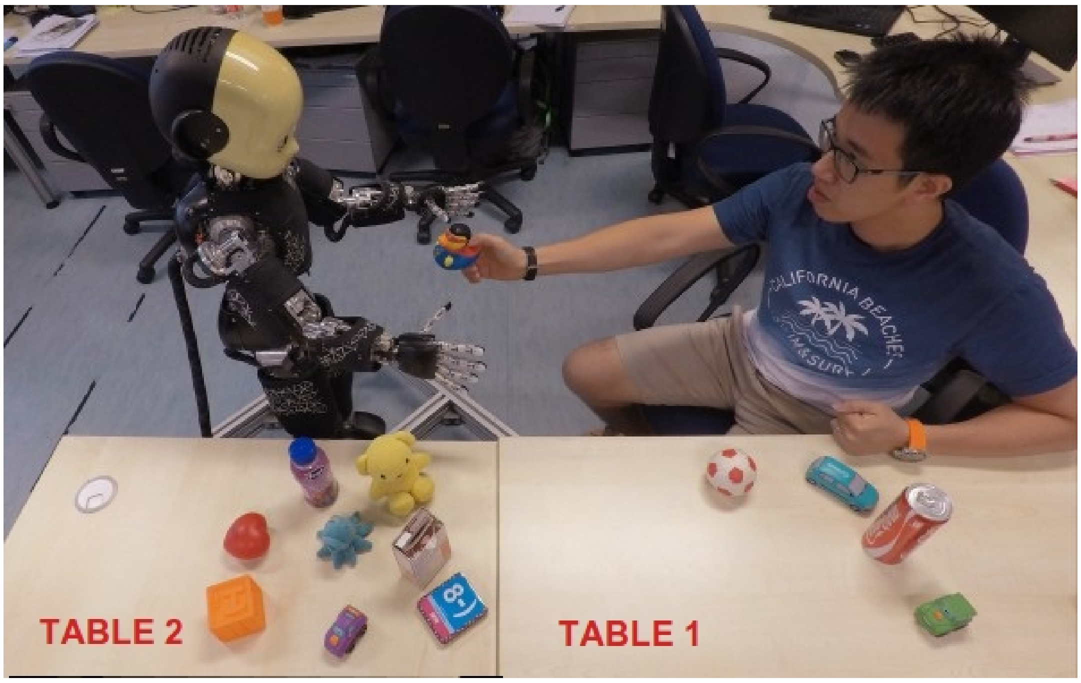


FIGURE 4.3: Our experimental setup with the human and the iCub sharing the workspace. The human is sitting next to *Table 1* while the iCub is located near *Table 2*.

as the object itself, are reduced (at $t \simeq 9s$) from normal to approachable values, while the value of the human left hand remains unchanged. As a result, the visual PPS activation on the robot right hand is almost null (barring the short time range $t \in [11.3, 11.5]s$), even though the human right hand is very close (see Fig. 4.4-Top). The robot can therefore move directly to reach for the target object as long as it does not detect any approaching obstacle, as illustrated by the quickly decreasing distance between the robot end-effector and the object in Fig. 4.4 ($t \in [10, 12]s$). This is not the case presented in Fig. 4.5, where the human moves his left hand to interfere with the robot's movements; in fact, the relative distance (blue profile) becomes very close to 0 at $t \simeq 11s$, causing very high PPS activation. The iCub correctly reacts by anticipating its planned movement for safety reasons until the instant the human moves his left hand away ($t \simeq 13s$). Afterwards, iCub continues to move its hand to approach the human right hand to receive the object safely. A detailed analysis of the joint velocities commanded at the robot arm during the interaction can be found in the previous Chapter 3.

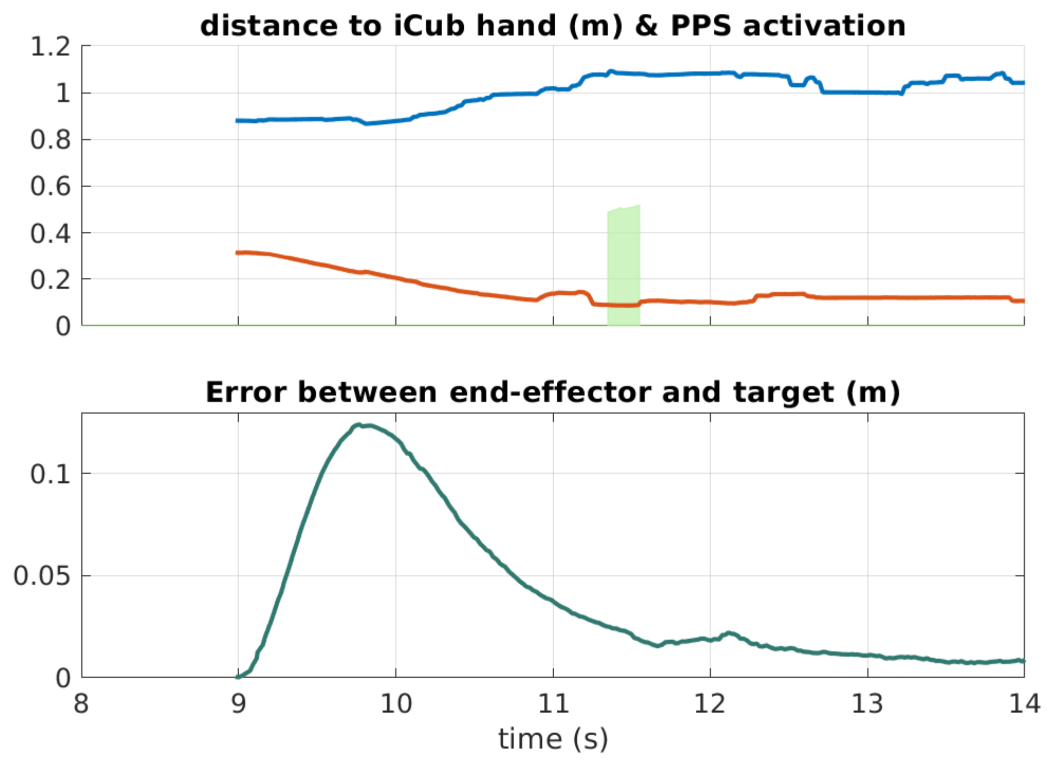


FIGURE 4.4: Experimental results when iCub is approaching the human right hand holding the object. Top: the distance from the robot right hand to the human hands are shown (red for right, blue for left; aggregated PPS visual activation on the robot right hand are shown by the green shaded areas. Bottom: distance between the end-effector and the target object.

4.3.2 Safe robot-human handover task

In this experiment, the robot has to find the object lying on *Table 2*, grasp it properly and finally perform the handover to the human partner. The co-operation task is initialized with the following dialogue between the two agents:

PARTNER : Hi iCub, can you give me the OCTOPUS?

ICUB : I have the OCTOPUS on my table. I will give it to you.
(ICUB *then grasps the* OCTOPUS *from* TABLE 2 *and shows the* OCTOPUS *to* PARTNER)

More specifically, the iCub looks for the requested object lying on *Table 2* and calculates the best grasping pose using the superquadric-based [Vezzani et al., 2017] grasping method extended in [Nguyen et al., 2018c, Section II.E.2]. If a suitable grasping pose is found, iCub reaches for the object to perform a power grasp. With the object in hand, the robot brings the object to the handover location that best suits the estimated human pose. As displayed in

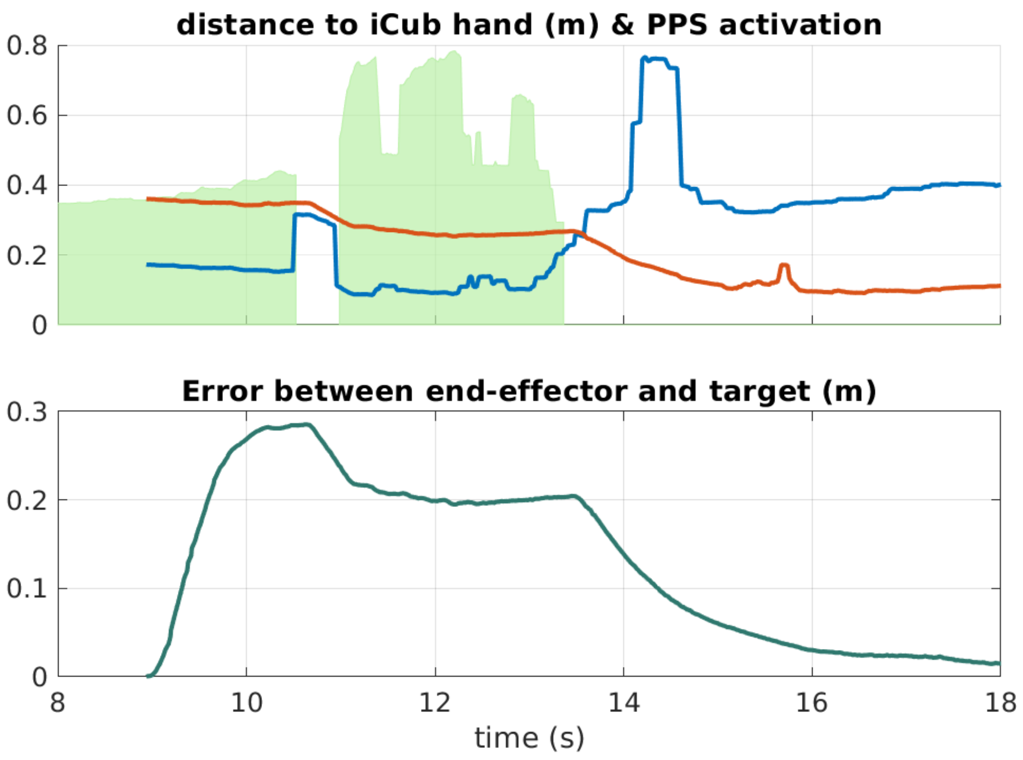


FIGURE 4.5: Experimental results when iCub is approaching the human's right hand holding the object and the human interferes with the robot movements using his left hand. Relevant quantities are explained in the Caption of Fig. 4.4.

Fig. 4.6, the iCub's movements towards the desired pose are continuously adapted to accommodate for any incoming visual obstacle, represented in this case by the human left hand ($t \in [38, 40]s$), as well as any unexpected physical contacts ($t \in [31.2, 34.2]s$ and $t \in [36.5, 39]s$). Both visual and tactile events constrain the admissible range of joint velocities, generating avoiding behaviour on the part of the robot, as is visible in Fig 4.6-Bottom in terms of the distance between the end-effector and the handover position (blue profile). This behaviour guarantees a safe physical interaction between the human and the iCub in *pre-* and *post-*collision phases.

4.3.3 Quantitative assessment of the interactions

This section reports the success rates of the two handover experiments carried out with a set of different objects (cf. Fig. 4.7). This quantitative analysis focuses on the detail of four sub-tasks per interaction, which need to be completed in sequence. In particular, the sub-tasks {*Recognize, Localize, Receive, Drop*} and the sub-tasks {*Detect, Plan, Grasp, Give*} are identified for the

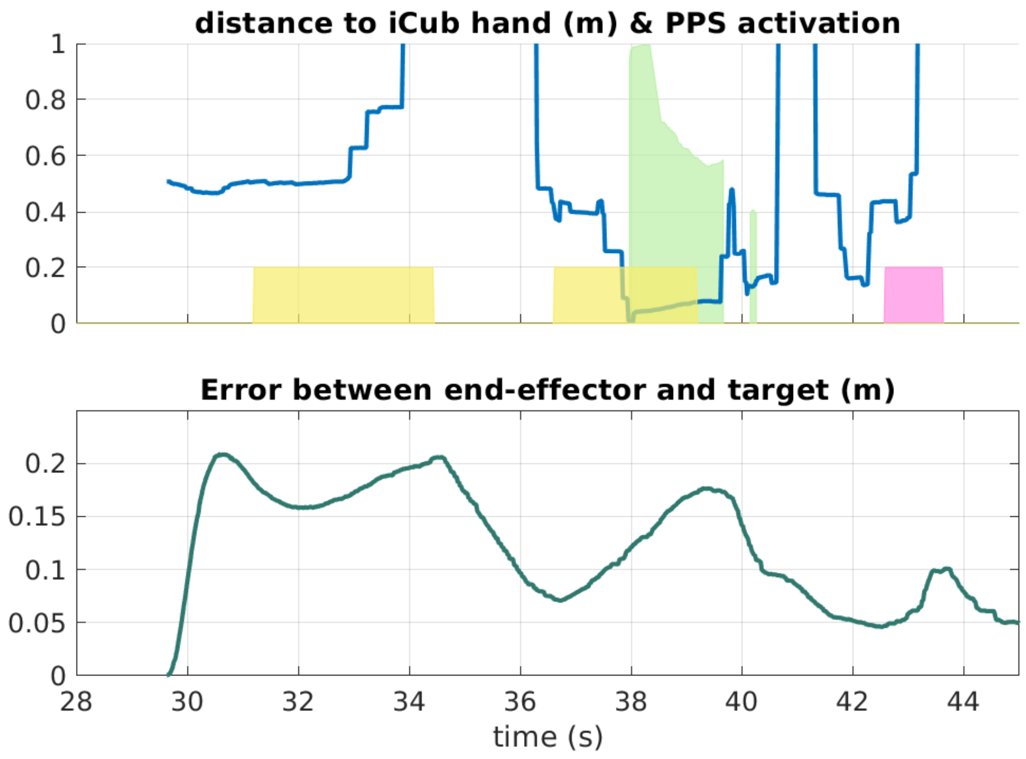


FIGURE 4.6: Experimental results when iCub is approaching the human right hand to hand over the object. Top: we show only the distance from the human left hand (blue) for sake of simplicity; tactile contacts with the robot right forearm (yellow) and right hand (pink) are also depicted.

human-robot and the *robot-human* hand-over sequences, respectively. For each object, the whole experiment was repeated ten times. Note that the sub-task corresponding to the control of the robot movements was omitted since the reactive controller guarantees the safety of the action at all times (as shown in the previous experiments).

TABLE 4.1: Success rates of human-robot hand-over task

Object	Sub-tasks			
	Recognize	Localize	Receive	Drop
Octopus	100%	90%	100%	100%
Duck	100%	100%	100%	90%
Bottle	100%	90%	100%	100%

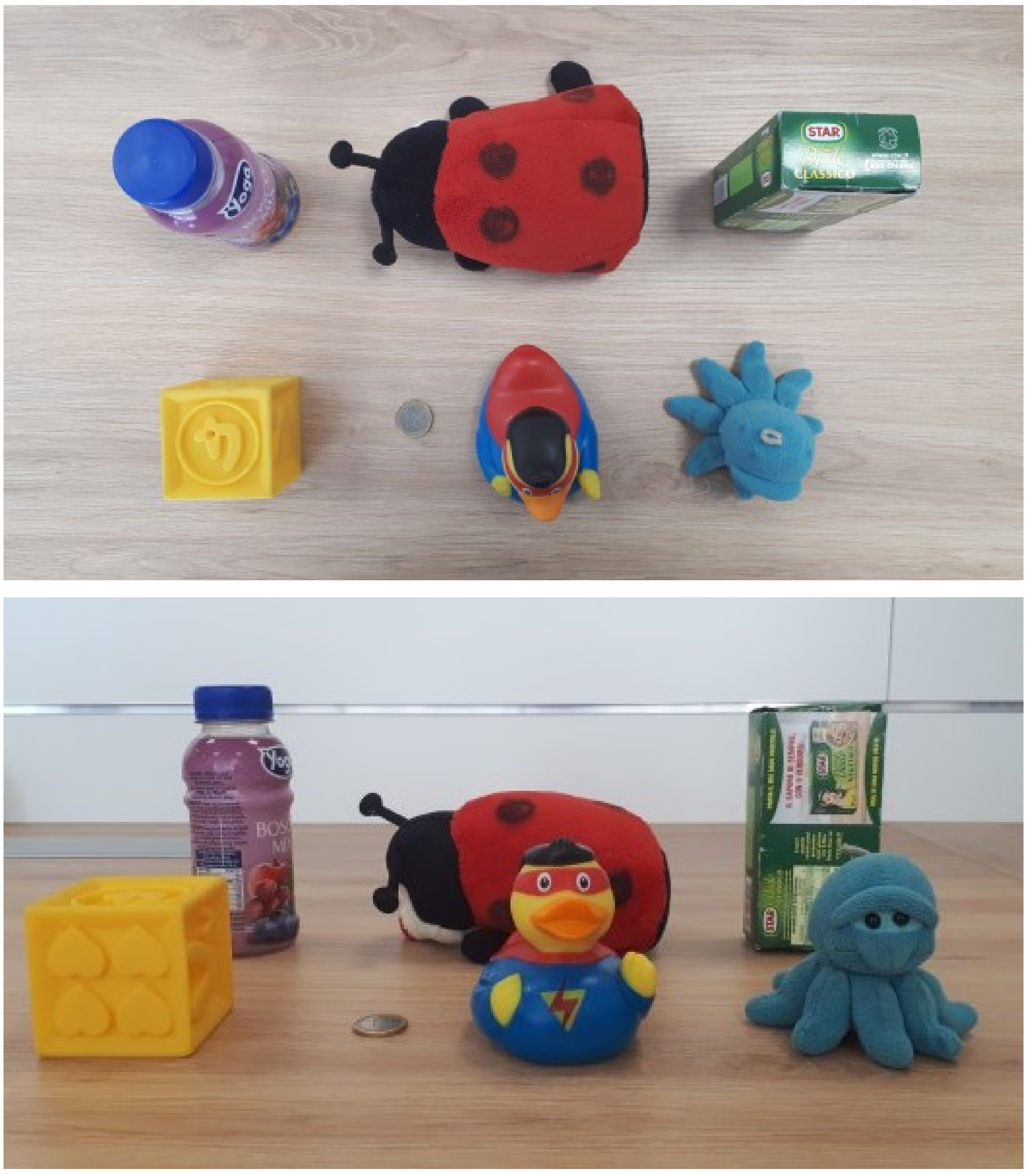


FIGURE 4.7: Objects used in the assessments

TABLE 4.2: Success rate of robot-human hand-over task

Object	Sub-tasks			
	Detect	Plan	Grasp	Give
Ladybug	100%	80%	100%	100%
Box	100%	90%	90%	90%
Bottle	100%	90%	80%	100%

The high success rates recorded in the two handover experiments and reported in Table 4.1 and Table 4.2 demonstrate the effectiveness of the proposed solution in scenarios where both the physical and social properties of the interaction are relevant during the human-robot collaboration.

4.4 Discussions

A compact, fully integrated and scalable architecture has been introduced in this chapter, aiming to fill in the gap between physical and social HRI with the following key features: (i) a markerless 3D context-aware visual perception system, (ii) a multi-modal visuotactile reactive controller, and (iii) a simple database for storing symbolic knowledge. It is shown that the complete system works in real-time to control a robot in the human-robot and robot-human object handover tasks while guaranteeing safety for the human experimenter. Moreover, it is argued that this architecture can be feasibly adopted with different robots equipped with a similar set of sensors (i.e. stereo-vision, tactile and/or force/torque sensing).

Future work will include integration with a state-of-the-art cognitive architecture. In particular, safe behaviours generated through this visuotactile component recall and advance the mechanisms of the *Somatic* and *Reactive* layers of the DAC-h3 architecture, as described in [Moulin-Frier et al., 2017a, Fischer et al., 2018, Moulin-Frier et al., 2017b]. Likewise, the visual pipeline tightly connect to the *World* and *Action* layers available in DAC-h3, at least within the limits of this simplified task planning. Thereby it can be incorporated almost seamlessly in further DAC-h3 functional modules such as the *Synthetic Sensory Memory*, the *Perspective Taking* and the *Autobiographical Memory*, in order to enrich the current repertoire of capabilities, such as by adding action recognition skills. Comparing with the architecture of [Lemaignan et al., 2017], the one proposed here does not share functional modules as such, but the overall structure is similar to [Lemaignan et al., 2017] at the symbolic layer. This similarity may pave the way to a future integration of functionality that is missing in this design but readily accessible in [Lemaignan et al., 2017], such as the *human-aware task planning*.

In conclusion, it is intended now to develop the present system further with the goal of implementing a general and principled cognitive architecture by taking advantage of the integration with other existing approaches. Paramount for effective HRI is the need to improve action planners to tackle fast dynamic environments (see Chapter 5), while taking into account ergonomics, as discussed, for example, in [Kim et al., 2018].

Chapter 5

Motion planning algorithm for robotic manipulators

5.1 Introduction

The problem of motion planning is relevant in a range of disciplines, with applications in areas such as machining using numerically controlled tools, assembly, or robot motion planning, including both mobile robots and manipulators (see e.g. [LaValle, 2006]). The problem consists in finding a collision-free path for a rigid object in an environment containing other rigid objects (obstacles), connecting the start and goal configurations and, possibly, respecting other constraints and optimizing certain criteria (such as length of the path). In its generality, the path planning problem is almost intractable [Reif, 1985]. In practice, efficient and even complete[1] solutions can be found depending on the problem at hand. Considering the classical "piano-mover"[2] type of problems, the difficulty consists primarily in how the collision-checking is performed in the Cartesian space (or *workspace*). For reasons of computational efficiency, the 2D or 3D objects (robot and obstacles) need to be approximated by more simple objects such as polytopes (polygons in 2D, polyhedra in 3D; e.g. [Gilbert et al., 1988]); yet, this step will still be computationally expensive. Here, the introduction of a configuration space (*C-space*) representation has been fundamental [Lozano-Perez, 1983], as discussed in Section 2.1.2. In this, typically higher-dimensional, space (with dimensionality corresponding to the number of robot DoFs), the complete robot configuration can be described as a single point. The obstacles are then remapped into the same space, forming regions (*C-space obstacle*) in which the obstacles collide with the body of the robot in the particular

[1]An algorithm is considered complete if for any input it correctly reports whether there is a solution in a finite amount of time.

[2]see [LaValle, 2006, Chapter 1]

configuration. After this transformation, collision free paths for the robot can be searched more easily in this space, and complete and optimal[3] algorithms exist (they are sometimes referred to as exact motion planning algorithms; see e.g. [LaValle, 2006, Chapter 2]).

In practice, the exact transformation of obstacles from the workspace into the configuration space is often intractable or even theoretically impossible [Bialkowski et al., 2016] while real-time performance is desirable. For this purpose, [Barraquand et al., 1992] presented a technique that performs faster by connecting local minima of potential fields in the *C-space*.

More recently, *sampling-based* motion planning algorithms have become very popular, which trade absolute completeness and optimality for probabilistic completeness and asymptotic optimality. Sampling is performed in the robot configuration space and the points sampled are mapped into the workspace and checked for collisions with obstacles. As mentioned above, the collision checking becomes a primary computational bottleneck of these approaches, and this has motivated modifications of the approach to warrant more favourable computational complexity (most recently [Bialkowski et al., 2016]). Finally, the resulting plan may undergo—often also computationally costly—postprocessing, such as interpolation and smoothing. The output of this stage is a smooth collision-free path in configuration space that can be transformed into a trajectory (which includes the temporal dimension) to command the robot in joint space.

The setting here is quite different, however. I are seeking a solution for reaching tasks in the iCub robot in dynamic environments, while, importantly, being able to reuse existing Cartesian solvers and/or available controllers of the robot. An example is shown in Fig. 5.1 – the scene may be cluttered and humans may also constitute dynamic obstacles. Thus, guaranteeing collision-free trajectory execution at all times will not be feasible and is not the main goal. Instead, the requirement on the planner is to obtain an approximate collision-free path for the hand and arm of the robot in real time. Then, other layers of the proposed architecture will then be relied upon to ensure safe interaction of the robot with its environment. These are: (i) local avoidance reflexes for the whole robot body triggered by a safety margin—PPS representation (see Section 1.3.2, 3.2.3 and 4.2.4); (ii) responses to collision on contact relying on the artificial pressure-sensitive skin (and/or force/torque sensors) mounted on the robot (as described in Section

[3]returning the shortest path

2.3). Similar frameworks have been presented by [De Luca and Flacco, 2012, Flacco et al., 2012, Kappler et al., 2018], for example.

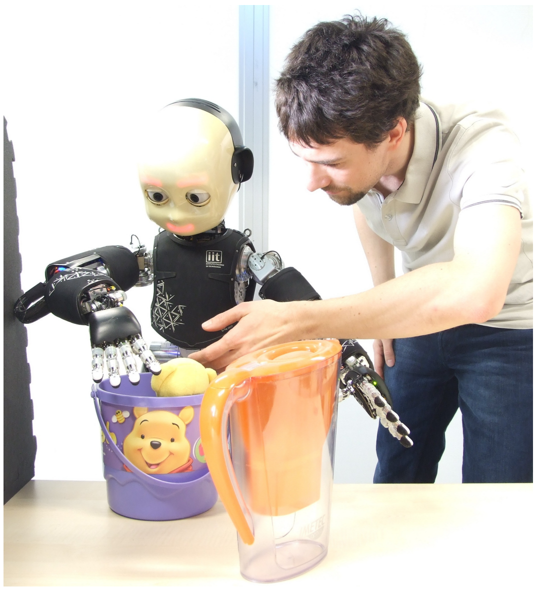


FIGURE 5.1: Illustration of the scenario involving robot interaction with a realistic environment. The scene may be cluttered and moving obstacles (including humans) may be present.

Thus, the key characteristics of the scenario investigated here, and their implications for the planning problem, are:

1. Obstacles may appear and disappear dynamically, and may even be moving. This precludes their mapping into *C-space*, and also renders any caching of their positions w.r.t. the robot (such as the safety certificates introduced in [Bialkowski et al., 2016]) less efficient.

2. The output of the planner needs to be only coarse—a sequence of via points in the workspace connected by straight lines. A smooth trajectory in joint space (with the minimum-jerk property) will be a result of the application of the Cartesian solver and controller to the via points.

3. The redundancy of the manipulator is not part of the planning problem (which happens in the workspace), but will be taken care of by the Cartesian solver.

4. The number of DoFs available to the robot may change dynamically – there will be 7 arm joints employed; in addition 1-3 torso joints (pitch, roll, yaw) may be recruited.

Given this context, a relatively simple planner is developed that matches these requirements. The basic component is an Rapidly-exploring Random

Tree (RRT)* sampling-based motion planner in the 3D workspace to obtain the path for the robot end-effector. The robot reachable space is simply superimposed onto the workspace; the obstacles are represented with polyhedra and collision-checking with the robot end-point can be done very quickly using only inequalities.

A collision-free plan only for the end-effector does not suffice, however, since it is necessary to consider the whole robot body. At the same time, it would be preferable to avoid representing the whole body of the robot using meshes or polyhedra since this would render the collision checking much more difficult. A coarse but effective approximation has therefore been introduced to consider the forearm occupancy: including the elbow and the forearm midpoint into the planning problem. The planner is then applied hierarchically—for every two via points of the end-effector, RRT* is triggered to provide a corresponding local plan for the forearm midpoint. Finally, the elbow is checked for collisions with obstacles too (somewhat similarly to [Ralli and Hirzinger, 1996]).

The chapter is structured as follows. In Section 5.2, the overall framework, the proposed algorithm, and the experimental setup are described. This is followed by Results in Section 5.3 and finally the Discussion in Section 5.4.

The main content of this chapter has been published in [Nguyen et al., 2016], except Section 5.2.4 and 5.3.4, where the novel multiple target reaching controller is presented, and the planner-controller integration results are discussed.

5.2 Methodology

5.2.1 Overview of overall control architecture

A schematic of my overall framework is shown in Fig. 5.2. The central node, “Multiple-Cartesian-point planner” is the subject of this chapter. The context of the other modules is essential to understand its working principle, however. In short, the planner provides a path for several points on the manipulator when queried by the supervisor, which in turn relays this information to a solver and a controller that can handle multiple Cartesian points. The planner has direct access to the perceptual module to locate the goal and obstacles in the workspace, as well as to the “kinematic chain” to retrieve the current robot configuration, using forward kinematics, and the starting position for the control points.

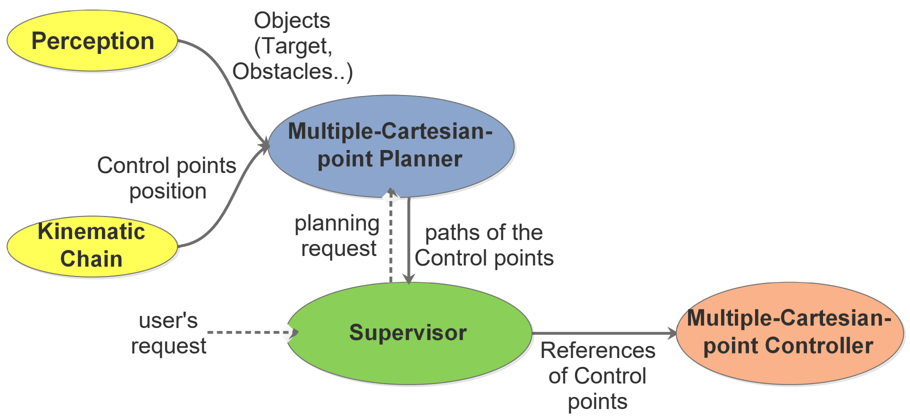

Perception
Objects (Target, Obstacles..)
Multiple-Cartesian-point Planner
Control points position
Kinematic Chain
planning request
paths of the Control points
user's request
Supervisor
References of Control points
Multiple-Cartesian-point Controller

FIGURE 5.2: Overview of the overall control framework.

1. **Perception**: Interfaces to various modules available in the YARP and iCub software repositories that deal with 3D object segmentation and tracking, relying on stereoscopy from the robot's two cameras (e.g. [Ciliberto et al., 2011, Fanello et al., 2014]). These can be used to stream the current positions of both target and obstacles in the scene.

2. **Kinematic chain**: provides the position of all control points on the robot (i.e. points in the robot's manipulator that the planner has to consider when computing the collision-free path), in this case: end-effector, midpoint of forearm, and elbow by applying forward kinematics to the current configuration of the robot.

3. **Multiple-Cartesian-point planner**: This module uses information about target, obstacles and the current state of control points as input, then generates collision-free motion paths for all control points, composed of via points connected by straight lines. The details of the algorithm will be described in Section 5.2.3.

4. **Supervisor**: This module receives a planning request from the user (human or another module) and forwards it to the planner. Then, after receiving the motion paths from the planner, it uses simple linear interpolation to create a trajectory (reference points in time) in Cartesian space that is in turn forwarded to the multiple Cartesian point solver and controller.

5. **Multiple-Cartesian-point controller** is an extension of the end-effector reaching controller (described in Section 3.2.4 and extended in Section 4.2.5) that can accommodate multiple control points. In addition,

both the solver and the controller are combined into a single optimization problem in velocity space, using the IpOpt library [Wächter and Biegler, 2006].

5.2.2 RRT* algorithm

Algorithm 1 RRT

```
1: $N \leftarrow x_{init}$
2: $E \leftarrow \emptyset$
3: for all i=1...n do
4:     $x_{rand} \leftarrow SAMPLE-RANDOMLY$
5:     $x_{closest} \leftarrow FIND-CLOSEST((N,E),x_{rand})$
6:     $x_{new} \leftarrow CREATE-NEW(x_{closest},x_{rand})$
7:     if $OBSTACLE-FREE(x_{closest},x_{new})$ then
8:         $N \leftarrow N \cup \{x_{new}\}$
9:         $E \leftarrow E \cup \{x_{closest},x_{new}\}$
10:    end if
11: end for
12: $V,E$
```

The RRT algorithm [LaValle, 2006] is one of the most common incremental sampling-based algorithms for robotics, generating a path connecting the initial state and goal state by randomly growing a search tree (cf. Algorithm 1). The algorithm is summarized as follows: The tree starts from its root x_{init}, a single node in the list of nodes N of the graph, and no edge in the list of edge E. At every iteration, a random node is sampled from the free space with the $SAMPLE-RANDOMLY$ procedure; then $FIND-CLOSEST$ looks for the nearest node $x_{closest}$ to the newly sampled x_{rand} in the available list N, which can be expressed as:

$$FIND-CLOSEST((N,E),x) := argmin_{n\in N} \|x-n\|; \tag{5.1}$$

and the $CREATE-NEW$ procedure attempts to find a new node x_{new} closer to x_{rand} than $x_{closest}$, which aims to grow the tree as much as possible. Finally, if the edge connecting $x_{closest}$ and x_{new} passes the test of $OBSTACLE-FREE$, the tree will grow with new node x_{new} and the new edge $\{x_{closest},x_{new}\}$.

The RRT algorithm is probabilistically complete but most of the time converges to a non-optimal solution. This sub-optimal problem is solved by the new RRT* algorithm proposed by [Karaman and Frazzoli, 2011] (see Algorithm 2). Unlike RRT, the RRT* algorithm introduces a cost function, e.g.

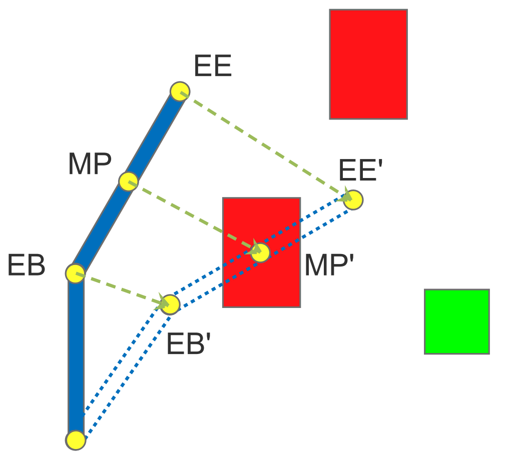
Algorithm 2 RRT*

```
1: $N \leftarrow x_{init}$
2: $E \leftarrow \emptyset$
3: for all i=1...n do
4:     $x_{rand} \leftarrow SAMPLE - RANDOMLY$
5:     $x_{closest} \leftarrow FIND - CLOSEST((N, E), x_{rand})$
6:     $x_{new} \leftarrow CREATE - NEW(x_{closest}, x_{rand})$
7:     if $OBSTACLE - FREE(x_{closest}, x_{new})$ then
8:         $\mathbf{X}_{close} \leftarrow FIND - CLOSE((N, E), x_{new}, r)$
9:         $N \leftarrow N \cup \{x_{new}\}$
10:        $x_{min} \leftarrow x_{closest}$
11:        $c_{min} \leftarrow COST(x_{closest}) + DIST(x_{closest}, x_{new})$
12:        for all $x_{close} \in \mathbf{X}_{close}$ do
13:            if $OBSTACLE - FREE(x_{close}, x_{new})$ then
14:                $c' \leftarrow COST(x_{close}) + DIST(x_{close}, x_{new})$
15:                if $c' < COST(x_{new})$ then
16:                    $x_{min} \leftarrow x_{close}$
17:                    $c_{min} \leftarrow c'$
18:                end if
19:            end if
20:        end for
21:        $E \leftarrow E \cup \{x_{min}, x_{new}\}$
22:        for all $x_{close} \in \mathbf{X}_{close} \backslash \{x_{min}\}$ do
23:            if $OBSTACLE - FREE(x_{close}, x_{new})$ then
24:                $c' \leftarrow COST(x_{new}) + DIST(x_{close}, x_{new})$
25:                if $c' < COST(x_{close})$ then
26:                    $x_{parent} \leftarrow FIND - PARENT(x_{close})$
27:                    $E \leftarrow E \backslash \{x_{close}, x_{parent}\}$
28:                    $E \leftarrow E \cup \{x_{close}, x_{new}\}$
29:                end if
30:            end if
31:        end for
32:    end if
33: end for
34: $V, E$
```

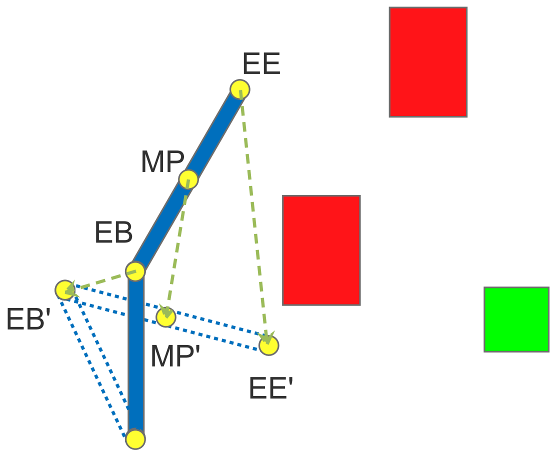
function of Euclidean distance, $COST(x)$ and $DIST(x, x_{new})$ to guide the extension of new nodes of the tree in such a way that the algorithm can find the shortest path. The former function computes the cost from the root of the tree to a node x, whereas the latter finds the cost between the two nodes x and x_{new}. While its computational complexity is a constant factor of RRT, it is also asymptotically optimal—guaranteed to converge to an optimal solution as the number of samples approaches infinity. The lines 12-20 in Algorithm 2 aim to make connections along the minimum cost path, while the lines 22-31 attempt to rewire the tree. In contrast to $FIND - CLOSEST$, the $FIND - CLOSE((N, E), x, r)$ procedure looks for all nodes in the list N that lie within a sphere of radius r and centered at x. The new function $FIND - PARENT(x)$ finds the node x_{parent} in the current list N such that there exists an edge of x_{parent} and x in the list current E. Furthermore, Perez et al. [2011] have further augmented the algorithm with a sparse sampling procedure.

5.2.3 Multiple Cartesian point planning algorithm

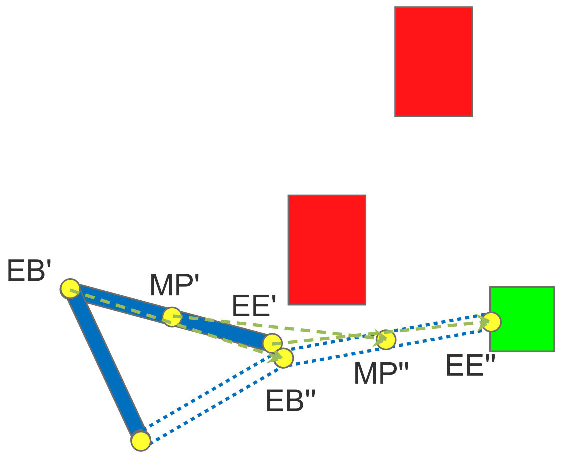
The main idea of the Cartesian planning algorithm proposed here is to use individual (modular) planners in the workspace for a few selected control points on the manipulator. Imagine taking the end-effector (EE) and the elbow (EB) as control points. At first, the planner of the EE may use the information from the environment (target and obstacles), together with the EE's position, to generate a collision-free motion path for the EE only. Based on this path, local planners for the next control point, the elbow, can be applied to find collision free paths for the elbow for every segment of the end-effector's path. Yet, there is a problem with this approach, as illustrated schematically in a 2D scenario in Fig. 5.3a. Although, according to the new position of the end-effector at EE′, the local planner of the elbow can find a new position for it at EB′ so that both paths are free of collision, but in doing this, the occupancy of the forearm (wrist-elbow link) has not been taken into account, which may lead to collisions with small obstacles in the course. Thus, a new control point in the middle of the link (MP) can be introduced, leading, for example, to the situation in Fig. 5.3b. This may still not guarantee a collision-free path in every case, but in most situations and given the size of obstacles the robot in this study will encounter, it is a good enough approximation.



(A)



(B)



(C)

FIGURE 5.3: Schematics of a two DoF planar manipulator with obstacles (red) and target (green). (a) Part of a plan for End-Effector and elbow (EB) illustrating that a collision-free path for the two control points does not guarantee that no collisions will occur for the whole occupancy of the manipulator (b) Introduction of another control point in the forearm link helps to avoid collisions, eventually leading to a collision-free path from start to goal, as shown in c).

The algorithm proposed is hierarchical and modular in nature: First, a plan (a set of via points connected by straight lines) is generated for the end-effector. Then, for every segment of this path, local collision-free paths are searched for the forearm midpoint. Finally, a check for collisions with the elbow is also performed. If it is not possible to construct a collision-free plan for all the control points, re-planning from the top (end-effector) is triggered. The details of the algorithm are described in pseudo-code in Algorithm 3.

Algorithm 3 Multiple Cartesian point planning Algorithm

```
$ESTIMATE-WORKSPACE$
$UPDATE-MANIPULATOR-POSE$
$OBTAIN-SCENE$
repeat
    $clear(EE-path)$
    while ($!EE-path$) do
        $MODULAR-PLANNER(EE, EE-pos, GOAL)$
    end while
    if ($EE-path$) then
        for all $w_i \in (EE-path)$ do
            $DILATE-OBSTACLE(w_i)$
        end for
        repeat
            $i = 0$
            $s_{MP} = MP-pos$
            repeat
                $PICK-VIAPOINT(w_i, w_{i+1})$
                $g_{MP} = goal_{MP} \leftarrow w_{i+1}$
                $MODULAR-PLANNER(MP, s_{MP}, g_{MP})$
                if ($size(MP-path) > 2$) then
                    $sucess \leftarrow PAD-VIAPOINT(EE)$
                end if
                $i \leftarrow i+1$
                $s_{MP} = g_{MP}$
            until $size(EE-path) = size(MP-path)$
            $FIND-ELBOW$
            $sucess \leftarrow CHECK-COLLISION-ELBOW$
        until ($!EB-collisions$)
    end if
until ($success$)
```

The key components of the algorithm are explained below. All coordinates are w.r.t. the Root FoR of the iCub, located around its waist.

- *ESTIMATE–WORKSPACE* uses an inverse kinematics solver (using the IpOpt library) to estimate the reachable space for the end-effector, as visualized in Fig. 5.4. This procedure is computationally expensive but can be precomputed offline. A corresponding approximation of the reachable space of the forearm midpoint is derived by simply shrinking the end-effector reachable space by a constant corresponding to the EE-MP distance. This automatically guarantees compliance with the kinematic constraints: for far away targets, the elbow will naturally assume poses "inside"—between the torso and the end-effector.

- *UPDATE–MANIPULATOR–POSE* uses forward kinematics at the current joint configuration (three torso joints and seven arm joints) to calculate the position of the end-effector and the elbow, and then infers the position of the forearm midpoint (see Fig. 5.5).
- *OBTAIN–SCENE* retrieves information about objects, i.e. position and size, from corresponding perceptual modules. The coordinates are converted to the iCub's root FoR in the process.
- *MODULAR–PLANNER(control point, start, goal)* applies the RRT* algorithm to generate a path free of collision from "start" to "end". This can be a global plan in cases dealing with the end-effector, or a local plan that seeks the forearm midpoint positions corresponding to two end-effector via points.
- *DILATE–OBSTACLE(via point)* dilates the size of all obstacles which are close to each via point of the previous control point's motion path, as illustrated in Fig. 5.6. Only obstacles (red) inside the area swept by the wrist-midpoint (EE-MP) link, namely obstacles 1, 2, 4 and 5, are dilated. For the case of the forearm midpoint planning, obstacles will be dilated w.r.t. to the via points of the end-effector path. This procedure prevents midpoint local planners generating unfeasible paths or via points.
- *PICK–VIAPOINT*(w_i,w_{i+1}) obtains two successive via points (w_i,w_{i+1}) from the generated path of the previous control point—e.g. planned path of the end-effector in case of planning for the forearm midpoint. These via points are then used as starting and ending positions for the modular planner, using fixed geometric relations between the end-effector and forearm link.
- *PAD–VIAPOINT(control point, via point position)* is used to generate additional via points in a top-level path if triggered by the lower-level modular planner. As illustrated in Fig. 5.6, the presence of obstacle

3 requires the creation of an additional via point (MP'') for the forearm midpoint. To ensure an equal number of via points for all control points, the top-level (end-effector) path needs to be "padded" with an additional via point (EE'').

- *FIND–ELBOW* computes the set of the elbow's positions corresponding to the (end-effector, midpoint) pairs (the via points in the already constructed plans), as determined by the forearm geometry.
- *CHECK–COLLISION–ELBOW* examines whether the newly generated path (and all via points) of the elbow is collision free.

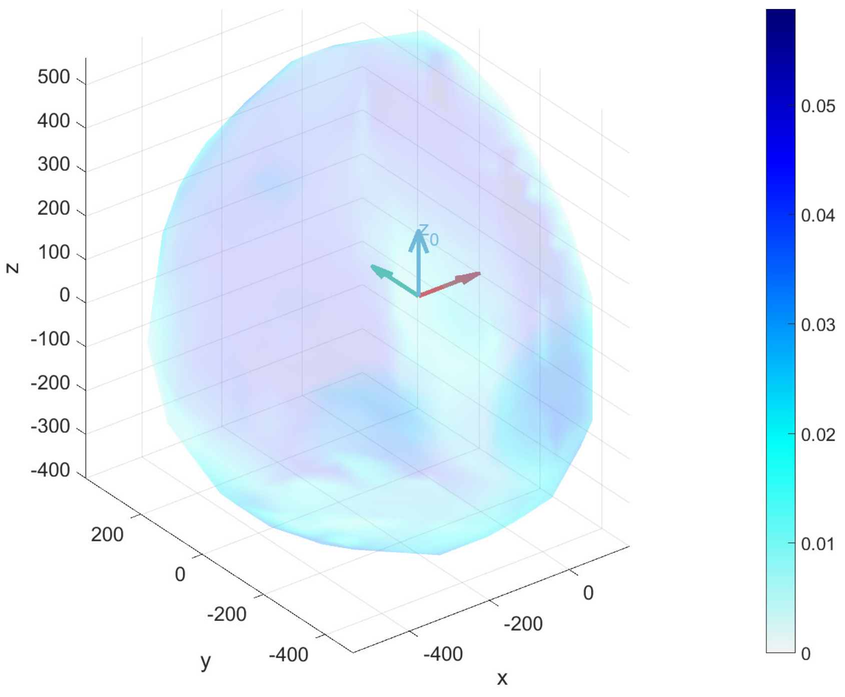


FIGURE 5.4: Visualization of the reachable space of the left arm of the iCub, starting from the Root FoR in the robot's waist area. The colour map depicts the manipulability of the different poses.

5.2.4 Multiple-target reaching controller

As briefly described above (cf. Section 5.2.1), with the proposed multiple Cartesian point planner, the reaching task requires a controller that can augment not only the robot's end-effector but other selected control points

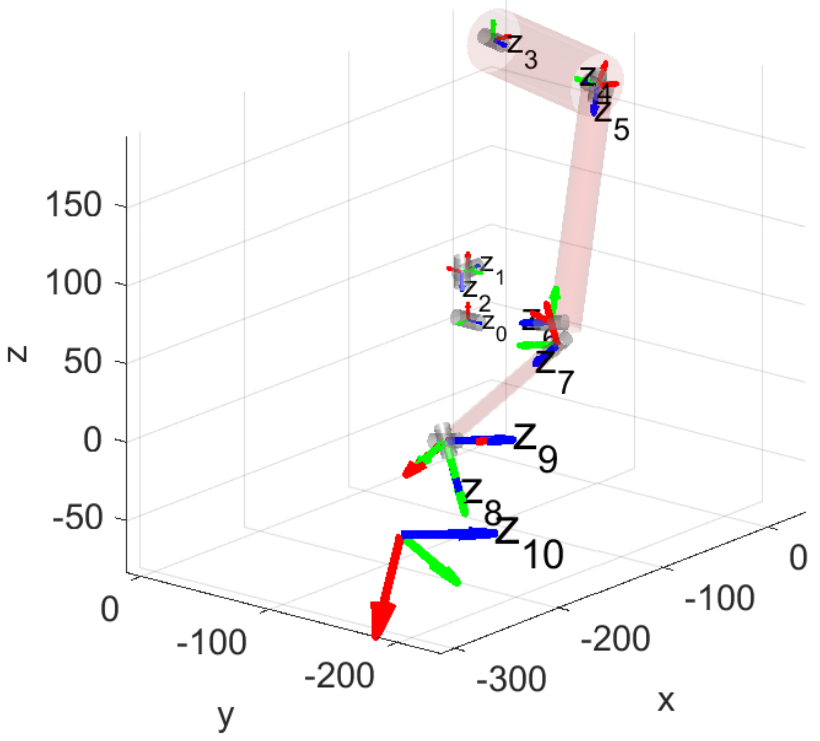


FIGURE 5.5: Schematic of the kinematic chain of iCub – left arm and torso. Reference frames have their z-axes labelled. The 0th reference frame is the iCub Root; the last ("z_{10}") corresponds to the end-effector.

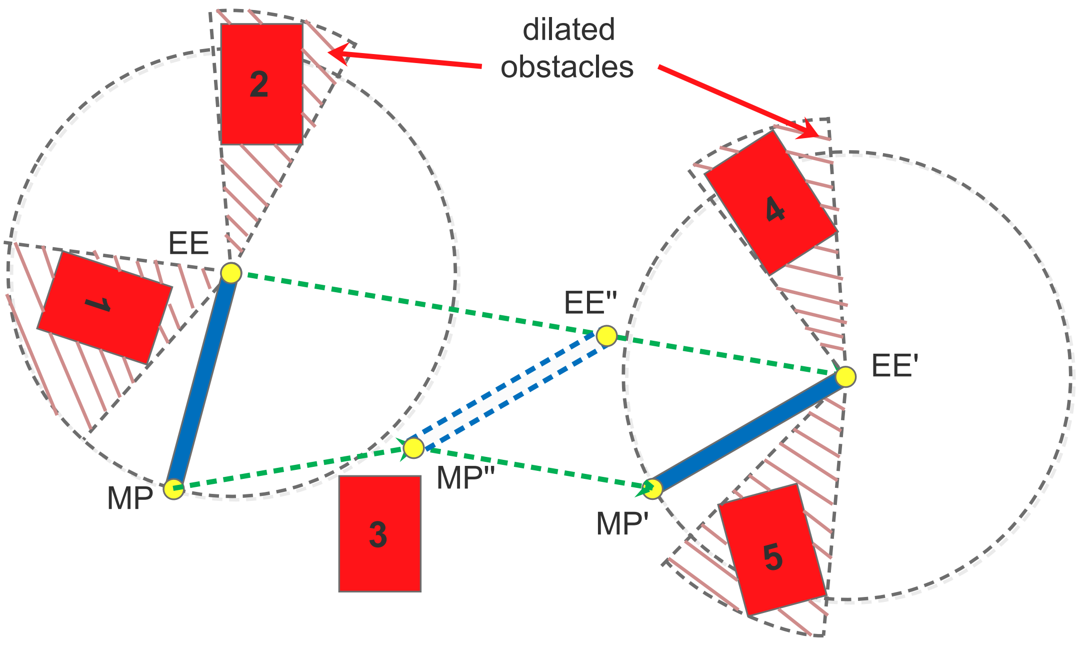


FIGURE 5.6: 2D illustration of *DILATE-OBSTACLES* and *PAD-VIAPOINT*. See text for details.

as well. Thus, the proposed reactive controller (cf. Section 3.2.4 and Section 4.2.5) is here extended by introducing extra constraints to adapt to the new task, multiple-target reaching, as follows:

$$\dot{\mathbf{q}}^* = \underset{\dot{q}\in\mathbb{R}^n}{arg\,min}\left[\left\|\mathbf{x}_{EEd} - \left(\bar{\mathbf{x}}_{EE} + T_S \cdot \mathbf{J}(\bar{\mathbf{q}}) \cdot \dot{\mathbf{q}}\right)\right\|^2\right]$$
$$\textbf{s.t.}\begin{cases} \mathbf{q}_L < \bar{\mathbf{q}} + T_S\dot{\mathbf{q}} < \mathbf{q}_U \\ \dot{\mathbf{q}}_L < \dot{\mathbf{q}} < \dot{\mathbf{q}}_U \\ e_{L,k} < e_{C,k} = \left\|\mathbf{x}_{d,k} - \left(\bar{\mathbf{x}}_k + T_S \cdot \mathbf{J}_{C,k}(\bar{\mathbf{q}}) \cdot \dot{\mathbf{q}}\right)\right\|^2 < e_{U,k} \end{cases} \tag{5.2}$$

where $e_{C,k}$ represents the Euclidean distances (i.e. errors in Cartesian space) between the referenced $\mathbf{x}_{d,k}$ and one-step predicted values of the chosen controlled point C_k in the robot's manipulator; $e_{L/U,k}$ stands for the allowable Euclidean distance of the controlled point C_k. For example, if two controlled points, the end-effector and the elbow, are chosen, there will be two corresponding constraints added to the optimization problem.

The idea is to find the optimal joint velocities $\dot{\mathbf{q}}^*$ at each time instant to minimize, not only the distance between the desired end-effector pose $\bar{\mathbf{x}}_{EEd}$ and the one-step prediction of the robot current pose $\bar{\mathbf{x}}_{EE}$, but also the same distances $e_{C,k}$ of other controlled points C_k (the elbow for example).

5.2.5 Experimental setup

Robot and Simulator

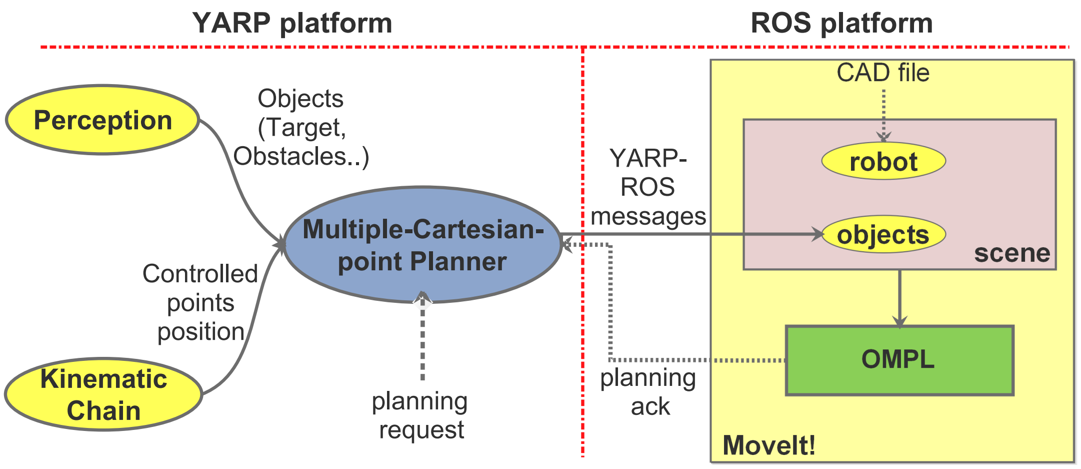


FIGURE 5.7: Communication between YARP and ROS for scenario synchronization.

This work focuses only on using seven DoFs of the arm (three shoulder joints, two elbow joints, two wrist joints) and three DoFs of the torso of the iCub robot (see Section 2.3). All experiments are run on its simulator (see Fig. 5.8).

The ROS[4] (Robot Operating System) is an open-source robotics software framework, which has been developing broadly with the contribution of the robotics community. MoveIt! [Şucan and Chitta] is a robotics software library implemented in the ROS environment, which provides tools like motion planning, manipulation, 3D perception, kinematics, control and navigation. Specifically, MoveIt! use the Open Motion Planning Library (OMPL, [Şucan et al., 2012]) as the core for the motion planning model, providing many different sampling-based planners, namely RRT*, PRM*, RRTConnect, etc. In this chapter, this motion planning environment is utilized to benchmark the performance of the proposed algorithm.

In order to simulate the iCub robot in the ROS-MoveIt! environment, the Unified Robot Description Format (URDF)[5] file was obtained from the CAD design files of iCub, and then the Setup Assistant tool provided by MoveIt! was used to generate MoveIt! compatible ROS packages which can be called by MoveIt! during run-time (see [Şucan and Chitta] for details). Fig. 5.7 shows the communication mechanism between the iCub simulator in the YARP environment and the iCub simulator in the ROS-MoveIt! environment. This connection was used to synchronize scenarios between the two simulators so that the planning performance could be compared.

The cost function

In these experiments, the cost was defined as the total motion distance for a path of a representative point, as in Eq.(5.3). In particular, the representative points in the experimental scenarios were the end-effector and the elbow.

$$cost_{path} = \sum_{i=0}^{N-1} ||w_i - w_{i+1}|| \tag{5.3}$$

where:

- w_i, w_{i+1} are successive via points on the path;
- N is the number of via points on the path;
- $||.||$ is the Euclidean distance calculation between two via points

[4]http://www.ros.org/
[5]http://wiki.ros.org/urdf

Experiments

The experimental scenario is shown in Fig. 5.8: The goal was to construct a motion plan for the robot to reach an object, the target, on a table (table-top scenario), while avoiding other objects as obstacles. While the target was fixed, the position of the obstacles was randomly changed for every trial in such a way that they did not overlap with the target. The default number of generated obstacles was ten, but if a generated obstacle was placed at the target position, that obstacle would be omitted. Thus the number of obstacles on the table was variable. The obstacles were slightly taller than the target, but a collision-free plan still always existed because the robot could reach for the target from above. Such a plan would have a higher cost than approaching from the side, however.

All the experiments were run on a standard PC (Processors: 4 × Intel Core i5-4310U CPU 2.00GHz, graphics: Intel Haswell Mobile, RAM: 8Gbs, OS: Ubuntu 14.04LTS 64-bit). Hence, the planning times reported in the Section 5.3 are referring to this PC.

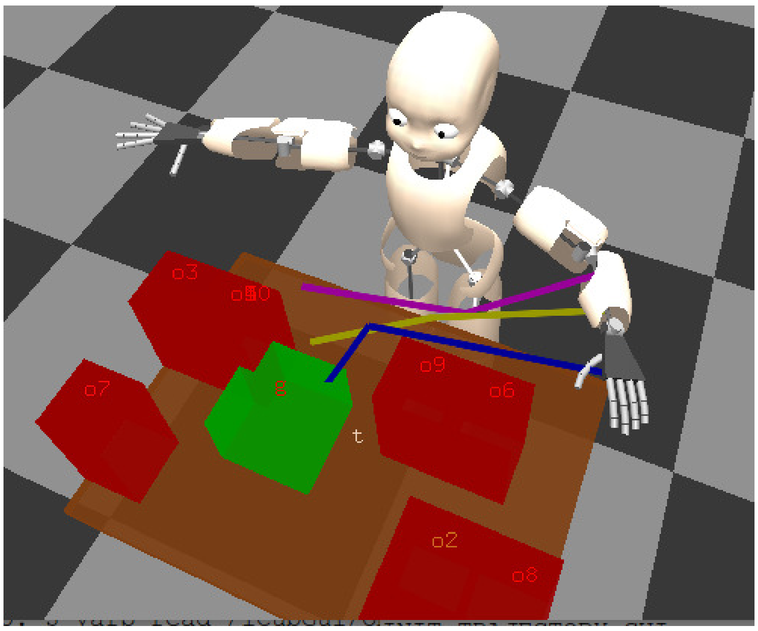


FIGURE 5.8: Experimental scenario with iCub simulator, target (green), randomly generated obstacles (red), and motion plans for end-effector (blue), forearm midpoint (yellow), and elbow (violet).

5.3 Results

Three types of experiments were conducted. First, a batch experiment was conducted with the maximum allowed planning time for individual planners being fixed and the performance statistics in 1000 randomly generated environments studied. Second, the environment was kept fixed, while the maximum planning time was varied in order to investigate the asymptotic optimality of the planner. Finally, some characteristics of the performance of this planner were compared with other state-of-the-art planners in a benchmark experiment.

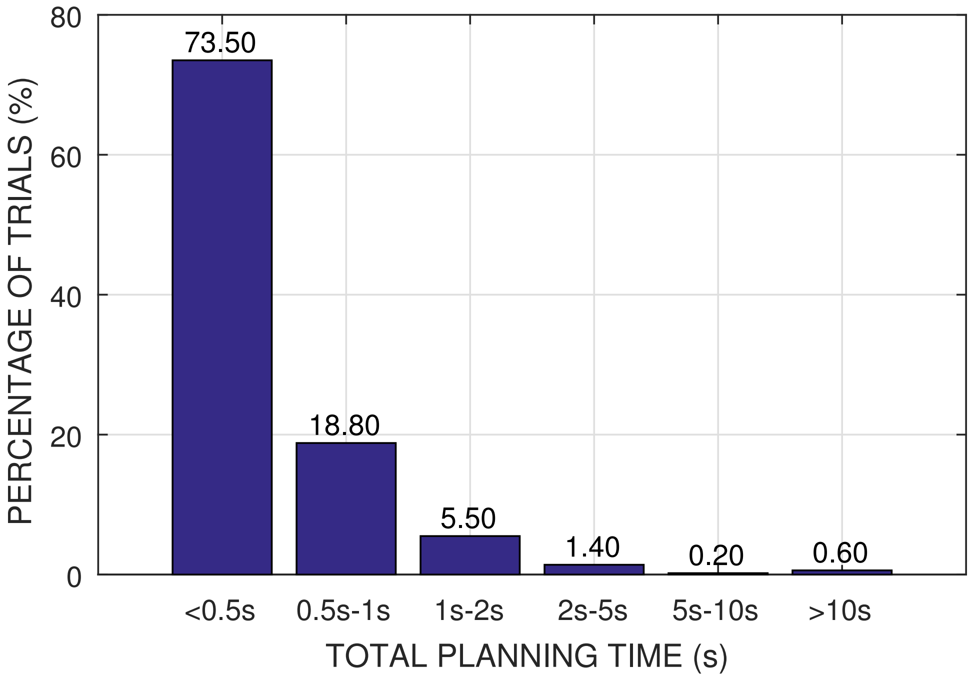


FIGURE 5.9: Summary of a batch experiment with 1000 trials.

5.3.1 Run-time batch experiment

In this experiment, 1000 trials were conducted successively, each with new randomly generated obstacle positions. As explained in Section 5.2.3, the proposed algorithm has a hierarchical structure and involves a series of calls to individual "modular" planners. In these experiments, the time for any of these modular planners was fixed to $0.1s$, but the number of calls to the individual planners was not known in advance and depended on the environment: the number of via points was determined dynamically by the planner. Furthermore, if the local planners for the forearm midpoint or the final elbow collision checking failed, replanning from the top-level would be triggered.

Thus, the total computation time is the sum of the time taken to execute all the individual planners.

The bar graph in Fig. 5.9 summarizes the results of the batch experiment in terms of this total planning time. It shows that 73.50% of trials completed with a computation time of less than half of second, and a further 18.80% trials finished with between 0.5*s* and 1*s* of computation time. 5.5% of trials, meanwhile, had a computation time of between 1*s* and 2*s*. In summary, the overwhelming majority of trials (97.8%) finished within 2*s*, which is suitable for this experimental setting. The fact that all percentages sum up to 100% confirms that a solution was found in all cases.

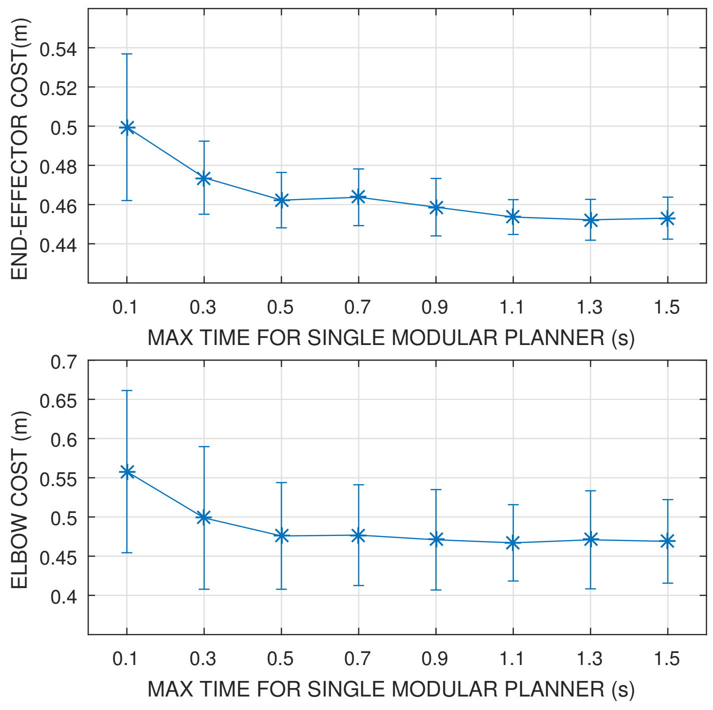


FIGURE 5.10: Cost of control point as a function of the defined planning time of modular planners (horizontal axis). The vertical bars show the standard deviation over 50 trials while the stars indicate the mean values of motion cost. Top: end-effector; Bottom: elbow.

5.3.2 Asymptotic optimality experiments

For this experiment, the obstacles were randomly created, but their positions kept unchanged from then on. Here, the timeout for every individual planner was systematically varied: after every 50 trials, the planning time was increased by 0.2*s*. After each trial, the planning results, namely total computation time and cost of motion plan (see Section 5.2.5), were stored. The experiments were deemed to be finished when the simulator had completed all 50 trials at the setting of 1.5*s* timeout for each modular planner. The results in Fig. 5.10 indicate that the motion costs reduced when the modular planning time was increased, for each of the two representative points of the forearm (end-effector and elbow). Please note that only the end-effector used a unique global planner (RRT*) while the elbow path (calculated through the forearm midpoint) was a result of the application of local planners to the end-effector via points. Nonetheless, the results suggest that the asymptotic optimality of the original RRT* algorithm was transferred to the additional control points. A formal proof will, however, remain the subject of future work.

5.3.3 Comparison with state-of-the art C-space algorithms

As explained in Section 5.2.5, the aim was to compare these results, with state-of-the-art planners available through the OMPL library (using the setup with OMPL and MoveIt!). To this end, for every trial of this experiment, information about all obstacles was synchronized to the ROS-MoveIt! module to update the environment. Then, different motion planning algorithms (RRT*, PRM* and RRTConnect) were applied to the new scenario, with the maximum computation time set to 10*s*. When the whole planning computation had finished in ROS-MoveIt!, an acknowledgment was sent back to the experimental planner in the YARP environment (as displayed in Fig. 5.7) to allow the planner to start a new trial (remembering that this planner was running with a 0.1*s* timeout for individual planners as in Section 5.3.1). The planning results of the proposed and OMPL-provided algorithms were stored for comparison. The results are in favour of the proposed algorithm, both in terms of total computation time and cost. It should be noted, however, that this comparison is not entirely fair, since the proposed algorithm solves a much easier problem in a heuristic fashion (using only a very simplified Cartesian space representation for the planning and collision checking) and offloads a number of expensive steps to the other modules (smoothing, inverse kinematics and control of robot in joint space). Conversely, the *C-space* planners deal with a complete representation of the robot geometry and provide a collision-free, smooth trajectory that is kinematically feasible. Nonetheless, the most important implication from these results is that with the 10*s* time-out, RRT*, PRM* and RRTConnect delivered a successful solution in only 11.93%, 9.17% and 8.26% of cases respectively—whereas the proposed algorithm did this in 100% of cases, with an average total planning time versus the other algorithms of 0.454*s*, 0.440*s* and 0.411*s*, respectively. Thus, the standard *C-space* planners cannot be deployed in my scenario where real-time response is required.

Method	mean time	cost EE	cost Elbow
RRT*	10.012	0.772 ± 0.178	0.548 ± 0.109
proposed	0.454	0.492 ± 0.039	0.463 ± 0.085

TABLE 5.1: Summary of planning results of RRT* in *C-space* (ROS-MoveIt!) and the proposed method

Method	mean time	cost EE	cost Elbow
RRTConnect	3.989	1.839 ± 2.621	1.261 ± 1.8378
proposed	0.411	0.494 ± 0.042	0.556 ± 0.119

TABLE 5.2: Summary of planning results of RRTConnect in *C-space* (ROS-MoveIt!) and the proposed method

Method	mean time	cost EE	cost Elbow
PRM*	10.059	0.837 ± 0.333	0.599 ± 0.210
proposed	0.440	0.491 ± 0.037	0.524 ± 0.086

TABLE 5.3: Summary of planning results of PRM* in *C-space* (ROS-MoveIt!) and the proposed method

5.3.4 Integration of the planner and the controller

In this experiment, the setup of the first experiment (Section 5.3.1) was again utilized, where an unknown number of obstacles were first generated randomly. After a feasible geometric path was generated in Cartesian spaces, the supervisor created a corresponding (Cartesian) trajectory and fed this into the controller (cf. Section 5.2.4). The problem thus became a trajectory tracking task.

In total, 20 trials of this experiment were conducted with the same initial pose for the robot's left arm and the goal region. Statistically, the tracking errors were $0.0132 \pm 0.0085m$ and $0.0174 \pm 0.0129m$ (on average) for the end-effector and the elbow, respectively. The tracking results of both the end-effector and the elbow of one of these trials are reported in Fig. 5.11. As seen from the figure, the proposed controller was able successfully to manipulate the motion of the robot's arm to follow the generated collision-free path, i.e.

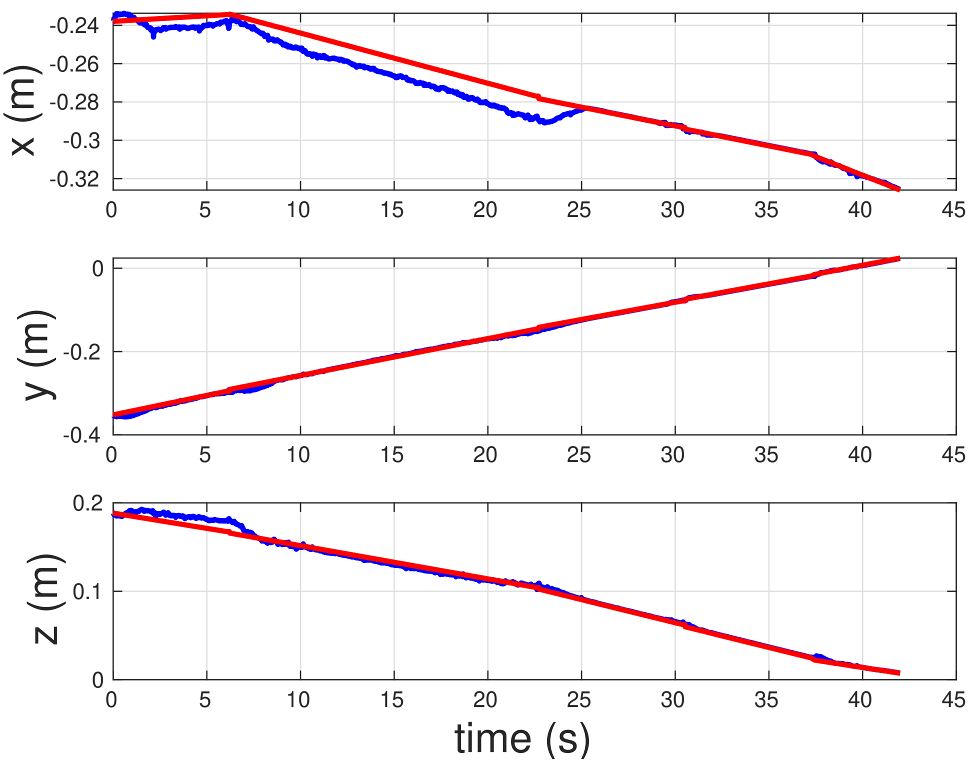
the trajectories of both end-effector and elbow (blue curves) followed the desired trajectories (red curves) from the planner.

5.4 Discussion

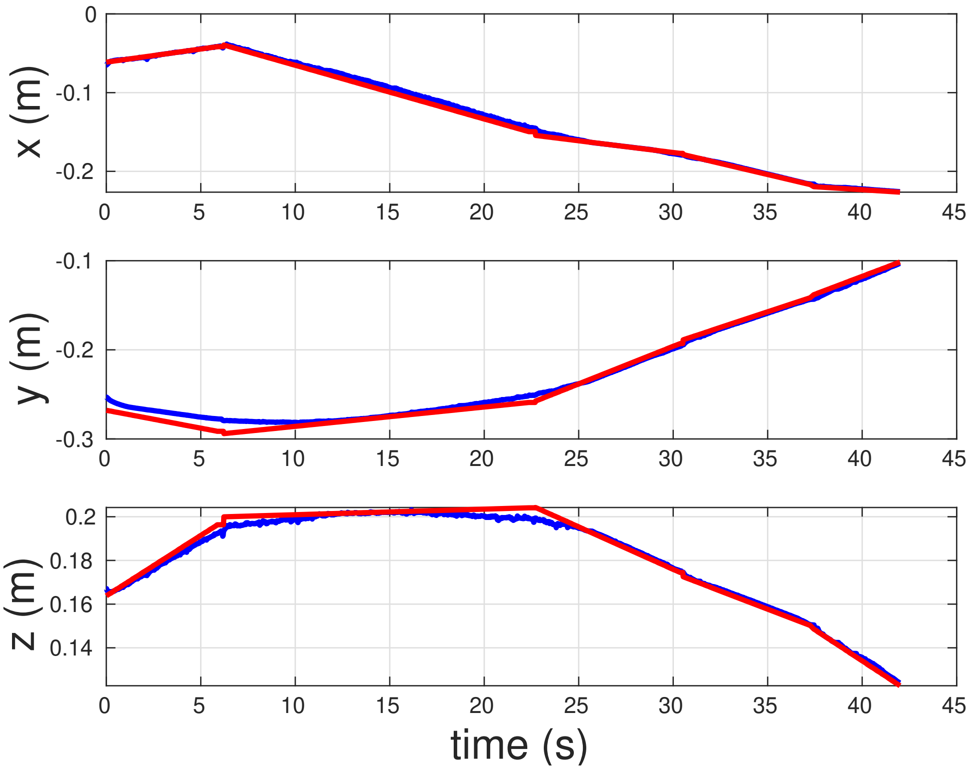
This chapter presents a motion planning component that is designed to work as part of a control framework for a humanoid robot that is reaching in a cluttered, dynamically changing environment. Moreover, the overall framework contains additional control layers to guarantee the robot's, as well as the environment's safety, by relying on an artificial pressure-sensitive skin as well as force/torque sensing (see Chapter 4). Thus, a perfect, collision-free, plan is not a requirement; instead, what is being sought is a module that delivers approximate collision-free paths very quickly. Given the high number of robot DoFs (seven arm joints and three torso joints) and the fact that the obstacles may not be static, it is evident that standard configuration space planning solutions cannot cope with this scenario. For example, [Wise and Bowyer, 2000] survey configuration space mapping techniques, but none of the manipulators reviewed feature more than six DoFs.

Here, therefore, a solution was devised that employs the standard RRT* sampling-based algorithm, but directly in the Cartesian space (workspace) in order to construct a collision-free path for the end-effector. In order also to consider the occupancy of the robot body, a collection of heuristics was utilized. Only the robot forearm was considered in addition to the end-point, and the planner was modified to operate in a hierarchical, modular fashion: local planners were launched to seek plans for the control points on the forearm, such that they matched with the via points obtained for the end-effector. Collision-checking was very simple and fast, through inequalities in the workspace, and employing a simple heuristics to dilate obstacles when considering the forearm midpoint path. The results demonstrate, first, that this solution delivers real-time performance (plans faster than 1*s* on a standard PC) in the vast majority of cases in a significantly cluttered environment. Second, the results are suggestive of the fact that asymptotic optimality of the plans is preserved even for the additional control points. Third, a test of state-of-the-art algorithms shows that solutions cannot be found in a reasonable time for the same scenarios (<10*s*).

With respect to the last result (comparison with *C-space* algorithms), it has to be clearly stated that my method is solving a much easier problem. In a way, it is transforming planning for a many-DoF manipulator with kinematic



(A) The end-effector



(B) The elbow

FIGURE 5.11: Tracking results of the end-effector and elbow of the robot's left arm against the trajectory (red curves) generated by the planner

constraints to the case of a mobile robot moving in 3D space. The fact that I am dealing with a manipulator is re-introduced through additional heuristics adding the forearm occupancy. These approximations means a collision-free path cannot be guaranteed in all cases. Furthermore, the inverse kinematics problem and the generation of smooth, kinematically feasible trajectories is also not dealt with—it is offloaded to Cartesian solvers and controllers that are available on my platform and are also proven to cope with real-time requirements (see Chapter 3 and Chapter 4).

A number of improvements can be considered for future work. First, the algorithm has been empirically found to deliver the desired real-time performance, but the hierarchical, recursive-like, structure does not provide any guarantees in respect to the maximum planning time (frequent replanning may be triggered in highly cluttered environments). The complexity of the plain algorithm should scale roughly linearly with the number of control points in moderately cluttered spaces. The efficiency of the proposed solution could therefore be improved in the future: the tree that has already been constructed for the end-effector could be profitably recycled instead of being started anew (as in the current algorithm) when it is not possible to find a collision-free path for the forearm and the elbow. Second, the asymptotic optimality of my solution is suggested only empirically–this needs to be addressed theoretically in the future. Finally, the overall planned multiple-target control framework–of which the planning module is only one component–needs to be tested on the real robot.