Cooperative Assembly with Autonomous Mobile Manipulators in an Underwater Scenario

Davide Torielli

DIBRIS - Department of Computer Science, Bioengineering, Robotics and System Engineering

University of Genova

In partial fulfillment of the requirements for the degree of

Robotics Engineering

30th August, 2019

“Choose a job you love,
and you will never have to work a day in your life”
Confucius (maybe)

Acknowledgements

I would like to thank my mother, my father, and my whole family, without who I would not be here. I would like to thank my sweetheart and my love, Sara (aka Cucurucho) which has always supported me and she always will. She was also really important for reviewing my English.
I would like to thank the Spanish lab, IRSLab, especially Javier, who really helped me a lot despite every morning at 10:12 he distracted me with Spanish almuerzos. I would like also to thank professors from my home country, professor Giuseppe Casalino, and professor Enrico Simetti, my two thesis supervisors. I want also to remember my Spanish supervisors, Professor Pedro J. Sanz and Professor Raul Marin Prades.
I think that I have to thanks also my colleague and friend, Fabio. He distracted me, too, with a lot of very long telephone calls. Anyway they were useful for him to learn something from me =). *Maybe*, *sometimes*, they were also useful for me.
Last but not least, I have to thank myself, because I'm great. Bravo Tori.

Abstract

Robotics is spreading in all the relevant sectors of the human life. The importance of studying this field is confirmed by all the various applications where robots are used: exploration of space and sea, industry, healthcare, transportation and so on. This thesis aims to improve the current state of the art in a particular field: Underwater Robotics. Currently, the research in this area focuses on improving robots capabilities to make them more and more efficient in performing missions autonomously. A particular advancement is towards the cooperation between multiple agents. With cooperation the robotics systems can perform more and more difficult tasks, such as carrying a long and heavy object in an unstructured environment.
Specifically, the problem addressed is an *assembly* one known as the *peg-in-hole* task. In this case, two autonomous manipulators must carry *cooperatively* (at kinematic level) a *peg* and must insert it into an *hole* fixed in the environment. Even if the *peg-in-hole* is a well-known problem, there are no specific studies related to the use of two different autonomous manipulators, especially in underwater scenarios. Among all the possible investigations towards the problem, this work focuses mainly on the kinematic control of the robots. The methods used are part of the Task Priority Inverse Kinematics (TPIK) approach, with a cooperation scheme that permits to exchange as less information as possible between the agents (that is really important being water a big impediment for communication). A force-torque sensor is exploited at kinematic level to help the insertion phase. The results show how the TPIK and the chosen cooperation scheme can be used for the stated problem. The simulated experiments done consider little errors in the hole's pose, that still permit to insert the *peg* but with a lot of frictions and possible stucks. It is shown how can be possible to improve (thanks to the data provided by the force-torque sensor) the insertion phase performed by the two manipulators in presence of these errors.
Another part of the thesis deals with computer vision algorithms: a third robot exploits particular methods to estimate the *hole*'s pose. Different techniques are compared to *detect* and to *track* the *hole*, considering the errors they provide in the pose's estimation.
Even if the problem is simplified (due to its complexity), this thesis could help further works. The focus is on the particular problem stated, but the methods and tools exploited can be useful also for other applications, not only underwater-related.

Contents

1 Introduction **1**
- 1.1 State of the Art . . . 2
  - 1.1.1 Previous Works in Underwater Missions . . . 2
  - 1.1.2 The Control Framework . . . 4
  - 1.1.3 The Peg-in-Hole Assembly Problem . . . 7
- 1.2 Motivation and Rationale . . . 8

2 Control Framework **10**
- 2.1 Definitions . . . 10
- 2.2 Control Objectives . . . 11
  - 2.2.1 Control Objectives classification . . . 12
  - 2.2.2 Equality and Inequality Control Objectives . . . 12
  - 2.2.3 Reactive and non Reactive Control Task . . . 13
  - 2.2.4 Inequality Control Objectives Activation and Deactivation . . . 14
- 2.3 Task Priority Inverse Kinematics . . . 15
  - 2.3.1 Notes on Conflicting Objectives . . . 16
- 2.4 Arm-Vehicle Coordination Scheme . . . 17
- 2.5 Cooperation Scheme . . . 18

3 Control Architecture: Methods **22**
- 3.1 Force-Torque Objective . . . 22
- 3.2 Objectives Prioritized List . . . 24
- 3.3 Change Goal Frame Routine . . . 26

4 Control Architecture: Simulation Results **28**
- 4.1 Choosing the Simulator . . . 29
- 4.2 Simulating the Firm Grasp Constraint . . . 31
- 4.3 Simulating the Collision Propagation . . . 32
- 4.4 Experiment's Assumptions . . . 33
- 4.5 Control Loop . . . 35
- 4.6 Results . . . 38

4.6.1 Perfectly known Hole's Pose . . . 40
4.6.2 Error on the Hole's Pose . . . 42
4.6.2.1 Change Goal Routine Results . . . 42
4.6.2.2 Force-Torque Objective Results . . . 42
4.7 Results with the Hole's Pose Estimation by Vision . . . 48
4.8 Results Discussion . . . 52
4.8.1 Perfectly Known Hole's Pose Discussion . . . 52
4.8.2 Change Goal Routine Discussion . . . 53
4.8.3 Force-Torque Objective Discussion . . . 54
4.8.4 Discussion About the Experiment with the Vision Part . . . 56
4.8.5 Standoff Discussion . . . 57

5 Vision: Methods & Results **59**
5.1 Vision Assumptions . . . 61
5.2 Tools . . . 62
5.3 Detection . . . 62
5.3.1 Already Known Coordinates Method . . . 63
5.3.2 Find Square Method . . . 64
5.3.3 Template Matching Method . . . 64
5.3.4 Detection Results . . . 65
5.4 Tracking . . . 67
5.4.1 Mono Camera Tracking . . . 68
5.4.2 Stereo Camera Tracking . . . 68
5.4.3 Stereo Depth Camera Tracking . . . 68
5.4.4 Tracking Results . . . 69

6 Conclusions **75**
6.1 Future Works . . . 77

A C++ Code Scheme **79**
A.1 Tools . . . 79
A.2 The Single Robot Code Scheme . . . 80
A.3 The Whole Code Scheme . . . 82

B Other Algorithms for Object Detection **84**
B.1 Corner Detection with the Shi-Harris Method . . . 84
B.2 Canny Edge and Hough Transform . . . 86
B.3 Bounding Box Detection . . . 87
B.4 2D Feature Matching & Homography . . . 88

References **99**

List of Figures

2.1 Arm-Vehicle Coordination Scheme in the TPIK 17
2.2 Relevant frames for the cooperation scheme 18
2.3 Cooperation Scheme in the TPIK 20

4.1 The Scenario with the two robot carrying the peg and the Vision robot watching the hole 28
4.2 Table of simulators comparison 31
4.3 Flow Scheme of the Control Loop 35
4.4 Scenario for tests without the vision part 38
4.5 Main frames related to the insertion phase 39
4.6 Plots with perfectly known Hole's pose 41
4.7 Plots of Tool and Goal frames with and without Change Goal 44
4.8 Plots of comparisons with and without Change Goal and Force objective 45
4.9 Plots with reference and activation of the Force-Torque tobjective . . 46
4.10 Plots of the velocity command generated by the Force-Torque objective 47
4.11 Scenario for the final test with Vision 48
4.12 Screenshots from the final experiment 49
4.13 Plots of results with Hole's pose estimation by Vision 50
4.14 Plots of cooperative robots velocities 51
4.15 Plots of cooperative tool velocity 52

5.1 The Vision robot watching the hole 59
5.2 Main frames related to the Vision phase 60
5.3 Hole Detection results with the three different methods 66
5.4 Tracking results with the three different detection initializations . . . 71
5.5 Tracking error plots with ideal detection initialization 72
5.6 Tracking error plots with Find Square detection initialization 73
5.7 Tracking error plots with Template Matching detection initialization . 74

A.1 C++ code scheme for the single robot 80
A.2 C++ code scheme for the whole architecture 82

B.1 Result of *goodFeaturesToTrack()* 84
B.2 Results of the Standard Hough Transform and the Probabilistic one . 86
B.3 Result of *findContours()* 87
B.4 Result of Bounding Box detection 87
B.5 Result of 2D Feature Matching 88

Chapter 1

Introduction

Nowadays, Robotics is spreading in any considerable area of human life. Fields such as exploration of deep space and sea, healthcare, industrial application, and transportation, show more and more usage of robotics systems. The research has the responsibility to tackle the increasing needs that these many applications have.

The underwater environment is one of the fields where Robotics is in a fast escalation, being the sea so important, from industry to environmental issues. Nowadays, different kinds of robot are largely used for underwater missions. A particular type of submarine robot is the Underwater Vehicle Manipulator System (UVMS): an autonomous underwater vehicle (AUV) capable of accomplishing tasks that require a certain level of dexterity, thanks to single or multiple arms.
A really innovative field for underwater missions is the analysis of cooperation between multiple agents, which permits to extend their flexibility. Cooperative robots are two or more robots, identical or different, that coordinate themselves to accomplish various objectives, from mapping an area to assembling an object.
This thesis focuses on a totally unexplored environment: cooperative *peg-in-hole* assembly with underwater manipulators. It is part of the TWINBOT project [TWINBOT (2019)], which is devoted to make a step forward for missions in complex scenarios, by extending the robots' capabilities.

Part of this thesis is developed at IRSLab at Universitad Jaume I, Castellòn de la Plana, Spain, during the time I spent as an Erasmus+ 2018/19 student, under the supervision of Professor Pedro J. Sanz and Professor Raúl Marín Prades. The IRSLab is the coordinator of the cited TWINBOT project.
Another part of the work is done at GRAAL at Università degli Studi di Genova, Italy, under the supervision of Professor Giuseppe Casalino and Professor Enrico Simetti.
The architecture is implemented in C++ and the code is avaible on GitHub at the following link: `https://github.com/torydebra/AUV-Coop-Assembly`.

A video of the final experiment is visible at the following link: https://streamable.com/kvoxq (online; accessed 10-08-2019).

This thesis is structured as follows. This Chapter 1 introduces the problem, recaps the previous works in this field, and states the objectives of the whole thesis. Chapter 2 defines the theory behind the work, focusing on the Task Priority Inverse Kinematics approach. Chapter 3 applies the explained theory to the actual problem, introducing new methods for the stated problem. In Chapter 4 the simulation set-up is described and results are discussed. Chapter 5 focuses on the vision part, explaining methods and examining results. In Chapter 6, conclusions and possible future works are given. In Appendix A further explanations about the code are given, and in Appendix B discarded methods (but maybe useful for other purposes) related to computer vision are briefly presented.

1.1 State of the Art

Robots have been massively introduced in various fields to help humans in different tasks. Nowadays, there are various strategies to use them in underwater environments. A manipulator (i.e. a robot's arm) is considered to be the most suitable tool for executing sub-sea intervention operations. Hence, unmanned underwater vehicles (UUVs), such as remotely operated vehicles (ROVs) and autonomous underwater vehicles (AUVs), are equipped with one or more underwater manipulators.

During the last 20 years, underwater manipulators have been used for many different sub-sea tasks in various fields, for example, underwater archaeology [Bingham *et al.* (2010); Drap (2012)], marine geology [Wynn *et al.* (2014); Urabe *et al.* (2015)], and military applications [Capocci *et al.* (2017)].

There are many specific tasks where the underwater manipulators are important, from salvage of sunken objects [Cheng Chang *et al.* (2004)] to mine disposal [Fletcher (2000); Djapic *et al.* (2013)]. One particular scenario is towards the oil and gas industry, where they are used for pipe inspection, opening and closing valves, drilling, rope cutting [Christ & Wernli (2013)], and, in general, to reduced field maintenance and development costs [Gilmour *et al.* (2012)]. A recent survey has explored the market related to this technology for oil and gas industry [Diaz Ledezma *et al.* (2015)].

1.1.1 Previous Works in Underwater Missions

In the early '90s, research in marine robotics started focusing on the development of underwater vehicle manipulator systems (UVMS). ROVs have been largely used, but they have high operational costs. This is due to the need for expensive support

vessels and highly qualified man power effort. In addition, the pilot which operates the vehicle and the arm experiences heavy fatigue in order to carry out the manipulation task. For the above reasons, the research started to increase the effort toward augmenting the autonomy level in underwater manipulation. For example, to reduce the operator's fatigue, some autonomous control features were implemented in work class ROVs [Schempf & Yoerger (1992)]. Another solution is to completely replace ROVs with autonomous underwater vehicles (AUVs).

Some pioneering projects in this field have been carried out in the '90s. The AMADEUS project [Lane *et al.* (1997)] developed grippers for underwater manipulation [Casalino *et al.* (2002)] and studied the problem of dual arm manipulation [Casalino *et al.* (2001)].

The UNION project [Rigaud *et al.* (1998)] was the first to perform a mechatronic assembly of an autonomous UVMS.

Early 2000s showed many field demonstrations. The SWIMMER project [Evans *et al.* (2001)] developed a prototype autonomous vehicle to deploy a ROV mounted on an AUV. This permitted to remove the need of long umbilical cables and continuous support by vessels on sea surface.

This work was followed by the ALIVE project [Evans *et al.* (2003); Marty (2004)], that achieved autonomous docking of an Intervention-AUV (I-AUV) in a sub-sea structure not specifically created for the AUV use.

The SAUVIM project [Yuh *et al.* (1998); Marani *et al.* (2009)] carried out the first autonomous underwater intervention mission. It focused on the searching and recovering of an object whose position was roughly known a priori. Here, the AUV weighted 6 tons, and the arm only 65 kg, so the dynamics of the two subsystems were practically decoupled and the two controllers were separated. The final mission consisted in the AUV performing station keeping while the arm was recovering the object.

After SAUVIM, a project called RAUVI [Prats *et al.* (2012)] took a step further. Here, the AUV performed a hook-based recovery in a water tank. Again, the control of the vehicle and of the arm were separated, even if the Girona 500 light AUV and the small 4-degrees-of-freedom (DOF) arm used had more similar masses than the ones of SAUVIM.

A milestone was the TRIDENT project [Sanz *et al.* (2012)]. For the first time, the vehicle and the arm were controlled in a coordinated manner [Casalino *et al.* (2012)] to recovery a black-box mockup [Simetti *et al.* (2014a)]. The used task priority solution dealt with both equality and inequality control objectives, although the inequality ones were only-scalar, except for the joint limits. This permitted to perform some manipulation tasks *while* considering also other objectives, for example,

keeping the object centred in the camera frame. In this project, only the kinematic control layer (and not also a suitable dynamic one) was implemented.

The PANDORA project [Lane *et al.* (2012)] focused on increasing the autonomy of the robot, by recognizing failures and responding to them. The work combined machine learning techniques [Carrera *et al.* (2014)] and a task priority kinematic control approach [Cieslak *et al.* (2015)]. However it dealt with only equality control objectives, with a specific ad-hoc solution to manage the joint limits inequality task.

The TRIDENT concepts were enhanced within the MARIS project [Casalino *et al.* (2014)]. The used task priority framework [Simetti & Casalino (2016)] permitted to *activate* and *deactivate* equality/inequality control objectives of any dimension (not only scalar ones). This project also extended the problem to cooperative agents [Simetti & Casalino (2017)].

TRIDENT and MARIS concentrated on using the control framework to perform not only grasping actions. A recent work [Simetti *et al.* (2018)] analyses how the method can be used in different scenarios, like pipeline inspection and deep sea mining exploration.

The DexROV project [Di Lillo *et al.* (2016)] is studying latencies problems which arise de-localizing on the shore the manned support to ROV operations.

The PROMETEO project [PROMETEO (2016)] plans to improve the use of underwater robotics in archaeological sites. It investigates the manipulation capacity when occlusions of objects can occur and with a wireless communication system to use the robot without umbilical cable.

The ROBUST project [ROBUST (2016)] aims to explore and to map deep water mining sites, through the fusion of two technologies: laser-based in-situ element-analysing, and AUV techniques for sea bed 3D mapping.

1.1.2 The Control Framework

In the '90s, industrial robotics researches focused on how to specify control objectives of a robotics system. This was done especially for redundant systems, i.e. systems with more degree of freedoms (DOFs) than necessary. This surplus is useful to perform multiple, parallel tasks; for example, avoiding an obstacle with the whole arm while the end-effector is reaching a goal. Given that such systems need to complete different goals, it has become important to have a simple and effective way to specify the control objectives.

The task-based control [Nakamura & Hanafusa (1986)], also known as operational space control [Khatib (1987)], defined the control objectives in a coordinate system that is directly linked to the tasks that need to be performed. This idea was followed by the concept of task priority [Nakamura (1990)]. In this theory, a more

important task is executed together with a less important task. To accomplish the whole action, the secondary task is attempted only in the null space of the primary one. This means that the secondary task is executed *only if* it does not go against the accomplishment of the first.

This concept was later generalized to multiple task priority levels [Siciliano & Slotine (1991)]. These works putted the position control of the end-effector as the highest prioritized one, while safety tasks (like joint limits) were only *attempted* at lower priority level.

First studies in control of redundant manipulators [Yoshikawa (1984); Maciejewski & Klein (1985)] managed the free residual DOF in such a way to solve the problem of singularity and obstacle avoidance for an industrial manipulator. Another work [Khatib (1986)] introduced the use of potential functions in industrial manipulators and mobile robots.

A different solution [Chiaverini (1997)] proposed a suboptimal approach. The secondary task was solved as if it was alone, but after it was projected in the null space of the higher priority one. To deal with singularities, a variable damping factor was used [Nakamura & Hanafusa (1986)]. This solution was later enhanced and called *null-space-based behavioural control* [Antonelli *et al.* (2008)]. The approach does not deal with the problem of algorithmic singularities that can occur due to rank loss caused by the projection matrix. Further works [Marani *et al.* (2003); Flacco *et al.* (2012)] focused on this problem.

Since those times, the task priority framework has been applied to numerous robotic systems, other than redundant industrial manipulators. Some examples includes mobile manipulators [Antonelli & Chiaverini (1998); Antonelli & Chiaverini (2003); Zereik *et al.* (2011)], multiple coordinated manipulators [Padir (2005); Simetti *et al.* (2009)], modular robots [Casalino *et al.* (2009)], and humanoid robots [Sentis & Khatib (2005); Sugiura *et al.* (2007)]. Furthermore, a stability analysis for several prioritized inverse kinematics algorithms can be found in [Antonelli (2009)].

The problem of the classical task priority framework, evident in all the previous mentioned works, is that inequality control objectives (e.g. avoiding joint limits) were never treated as such. In fact, the corresponding tasks were always active, like the equality ones. So, for example, also when the joints are sufficiently far from their limits, the fact that the task is active uselessly adds constraints and "consumes" DOFs. Thus, without a transition, the safety control objectives like joint limits could be only considered as secondary. Otherwise, other mission tasks, like reaching a position with the end-effector, can never be accomplished. This led to an undesired situation where safety tasks have a lower priority with respect to non-safety ones.

The challenge in activating (inserting) or deactivating (deleting) a task is that these transitions would imply a discontinuity in the null space projector, which leads to a discontinuity in the control law [Lee *et al.* (2012)]. Thus, in the last decade, researches focused on integrate safety inequality control objectives in a more efficient way.

A new inversion operator was introduced [Mansard *et al.* (2009b)] for the computation of a smooth inverse with the ability of enabling and disabling tasks in the context of visual servoing. But the work only dealt with the activation and deactivation of the rows of a single multidimensional task (so, not including the concept of different levels of priority). The extension to the case of a hierarchy of tasks with different priorities was provided successively [Mansard *et al.* (2009a)]. However, the algorithm requires the computation of all the combinations of possible active and inactive tasks, which grows exponentially as the number of tasks increases.

Another work [Lee *et al.* (2012)] modified the reference of each task that was being inserted or being removed, in order to comply with the already present ones, and in such a way to smooth out any discontinuity. However, the algorithm requires $m!$ pseudo-inverses with m number of tasks. For this reason, the authors provided approximate solutions, which are suboptimal whenever more than one task is being activated or deactivated.

Another approach [Faverjon & Tournassoud (1987)] directly incorporated the inequality control objectives as inequality constraints in a Quadratic Program (QP). According to this, the idea was generalized to any number of priority levels [Kanoun *et al.* (2011)]. At each priority level, the algorithm solves a QP problem, finding the optimal solution (in a least-squares sense). Slack variables are used to incorporate inequality constraints in the minimization process. If the solution contains a slack variable different from zero, it will mean that the corresponding inequality constraint is not satisfied. Otherwise, the inequality constraints are propagated to the next level and transformed into an equality ones (to prevent lower priority tasks from changing the best least-square trade-off found). A similar process is done for the equality constraints. A drawbacks of this approach is that the cascade of QP problems can grow in dimension rapidly. Another issue is that the activation and deactivation of tasks are not considered. This last point is important when temporal sequences of tasks are used, for example when assembling objects [Nenchev & Sotirov (1994); Baerlocher & Boulic (2004)].

Instead of a cascade of QP problems, another research [Escande *et al.* (2014)] proposed to solve a single problem finding the active set of all the constraints at the same time. Due to its iterative nature, the authors proposed to limit the number of iterations to achieve a boundary on the computation time, to be more suitable for a real-time implementation. But this solution is not optimal, and, again, activa-

tion/deactivation of equality tasks is not considered.

Improvements are made in the already cited TRIDENT project [Sanz *et al.* (2012); Simetti *et al.* (2014a)], where field trials proved how to consider activation and deactivation of scalar tasks. But the solution still lacks the ability to deal with activation/deactivation of multidimensional tasks, i.e. multiple scalar tasks at the same priority level.

The goals reached by TRIDENT are improved in the MARIS project [Casalino *et al.* (2014); Simetti *et al.* (2014b)], where, among the other accomplishments, task transitions were successfully implemented in the framework. In particular [Simetti & Casalino (2016)], possible discontinuities that could arise are eliminated by a task-oriented regularization and a singular value oriented regularization. Plus, the original simplicity of the task priority framework is retained thanks to pseudo-inverses.

1.1.3 The Peg-in-Hole Assembly Problem

The peg-in-hole is an essential task in assembly processes in various fields, such as manufacturing lines.

This task can be performed following the classical position control method. But this is possible only if precise position of the hole is provided, and the position control error of the robot is zero. In practice, these conditions can only be obtained in specialized scenario. In the case of more versatile robots, such as underwater manipulators, imprecisions and errors are unavoidable.

To deal with these problems, classical works exploit two kind of instruments: cameras and sensors. With camera(s), the robot can roughly recognize the objective (i.e. the hole) and inspect the overall process. Past researches [Miura & Ikeuchi (1998); Pauli *et al.* (2001)] use this idea to extract boundaries of the object. Another one [Chang *et al.* (2011)] uses visual feedback for a micro-peg-in-hole task (hole of $100\mu m$).

Other approaches perform precise assembly of the parts thanks to force/torque sensors installed on the wrist. A study [Shirinzadeh *et al.* (2011)] successfully accomplishes the assembly detecting the force of contact to compensate the positional uncertainty. Newman *et al.* study [Newman *et al.* (2001)] shows how sensors can be used to build map of force and torque values of each contact point. In another works [Dietrich *et al.* (2010); Abdullah *et al.* (2015)], the location of the hole was estimated using the measured reaction moment occurred by the contact. Another good aspect of the sensors is that they can guarantee stable contact through real-time contact force feedback [Oh & Oh (2015); Song *et al.* (2016)].

Other proposals [Chhatpar & Branicky (2001); Lee & Park (2014)] try to estimate the state of the contact using joint position sensor. This permits to not use the force/torque sensors on the wrist, which would need high control frequency,

and would increase overall cost and operation time. Some researchers [Park *et al.* (2013)] show that assembly task can be accomplished without contact force information and with inaccurate vision data. The proposed strategy mimics the human behaviour: the peg was rubbed in a point close to the object until the relevant objects mated using compliant characteristics. The compliance allows the robot to softly adapt to the hole [Lozano-Pérez *et al.* (1984); Xu (2015)]. A similar, unexpensive, approach is tested experimentally [Park *et al.* (2017)], without the use of force/torque sensors (i.e. no force feedback), nor Remote-center-compliance devices, and with inaccurate hole information.

1.2 Motivation and Rationale

Sea plays an important role in our societies. Many examples are given in the previous section 1.1. When such a kind of environment is so important, exploitation of robotic systems is necessary at different levels.
This thesis aims to improve the current state of the art in underwater intervention missions. Efforts in this direction can help the robots to accomplish more and more complicated tasks, substituting gradually their remotely operated versions (ROVs), and, in some cases, humans. This would help to reduce the costs, to increase the safety, to boost the performances, and, in general, to accomplish missions that before were unthinkable.

The peg-in-hole is one example of these complicated tasks. In general, robotic assembly problems have been addressed and explored widely, but, to the best of this author's knowledge, no works have been done for cooperatively assembly in underwater scenarios, except for the TWINBOT project [TWINBOT (2019)], which this thesis is part of. Productions related to this problem can help to fill this current lack and can make the technology to advance.

Cooperative agents augment the capability of the single, for example to carry an heavy object. It is important to notice that cooperation here is intended at *kinematic level*. So, for example, we are not speaking about robotic swarms, where more "planning" cooperation is explored with Artificial Intelligence techniques. Instead, here cooperation means two (or, in general, more than two) robots that share (in some way) their commanded system velocities (the usual output of the kinematic layer) to move together, without, in this case, make the common tool fall or break. So, there is not an "high level" reasoning with a planner, but there are mathematical formulas with vectors and matrices to make the robots *cooperate*.
Such improvements at kinematic level are important because they reduce the work-effort at higher levels (i.e. the planner), and they make the overall system faster.

At the time this thesis was being developed, the TWINBOT project was in an early

stage. So, this works evolves autonomously, always keeping an eye on the main objective of the project: improving capabilities of cooperative underwater intervention robots. The aim is to developed a kinematic control framework suitable for the cooperative underwater *peg-in-hole* problem stated, considering methods that can be used also for other robotics missions.
The Task Priority Inverse Kinematics (TPIK) approach is exploited for the single agent kinematic control (section 2.3), for the arm-vehicle coordination (section 2.4), and for the cooperation scheme (section 2.5).
A force-torque sensor is used to have data from the collisions that happen between the *peg* and the *hole* during the insertion phase. This information is used by two different methods which help to achieve the final goal and to reduce frictional forces that can damage the objects or the robots.
The first one is a new objective called Force-Torque objective (section 3.1). Its duty is to reduce the forces and the torques that act on the peg, moving it into the opposite direction. The objective is inserted into the TPIK list, among the other objectives. This is noticeable because we exploit a "dynamic" information at kinematic level.
The second method is called Change Goal routine (section 3.3). Its job is to change the origin of the goal frame (that is inside the hole) where the control drives the peg. This is done to compensate possible errors in the hole's pose. For example, if the goal is slightly on the left respect to the exact centre of the hole, lot of collisions happen with the left inner side, so lot of forces directed on the right are detected. This routine simply moves the origin of the goal frame according to the detected forces, on the right in the cited example. However, it must be considered that the *peg-in-hole* is simplified: the problem of having too big errors, which makes the peg collide with the external face of the hole, is not considered.

Another part of the thesis focuses on computer vision, specifically on the pose estimation of the hole (Chapter 5). In particular, stereo-vision methods are used by a third robot, which acts exclusively for the Vision purpose.
The Vision routine is divided into two phases.
The first is the *detection* (section 5.3) where the hole's structure is found in the images that the camera captures. Among all the algorithms tried, one based on shape detection and another based on template matching are discussed in the final results (section 5.3.4).
The second phase, the *tracking* (section 5.4), uses the first one as initialization for a *markerless model-based* method. In this case, trials with a mono-camera, a stereo-camera, and depth stereo-camera are compared and discussed.

Chapter 2

Control Framework

Introduction

In this section, the implemented control framework is presented, along with the necessary mathematical details. The architecture is constituted by two parts:

- The Mission Manager, which job is to supervise the execution of the overall mission. In this specific case, it is only a support for the subsequent part. It creates the *action*, that are a lists of control objectives that the Kinematic Control Layer must satisfy during the mission phases.
- The Kinematic Control Layer (KCL) focus on provide the system velocities (i.e. vehicle and joint velocities), given the list of control objectives from the Mission Manager.

This thesis focuses mostly on the Kinematic Control Layer. Discussions about how to implement a more complete Mission Manager (which acts not only as a support for the KCL, but, for example, it deals also with the *planning* of the mission phases) go out the scope of this work.

The framework is build from the one used in Simetti & Casalino (2017), Wanderlingh (2018), and Simetti *et al.* (2018). The sections 2.1, 2.2, 2.3, 2.4, and 2.5 derive from these works and in the following pages their main aspects are recalled.

2.1 Definitions

In this section, the necessary definitions are presented. We will consider the general case of a floating manipulator, made up of an l-DOF arm attached to a 6-DOF vehicle.

Let us define:

- The system configuration vector $\boldsymbol{c} \in \mathbb{R}^n$ of the robot: $\boldsymbol{c} \triangleq \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{\eta} \end{bmatrix}$,
  where $\boldsymbol{q} \in \mathbb{R}^l$ is the l-DOF arm configuration vector and $\boldsymbol{\eta} \in \mathbb{R}^6$ is the vehicle *generalized coordinate position vector*.
  The first three components of $\boldsymbol{\eta}$ form the position vector $\boldsymbol{\eta}_1 \triangleq \begin{bmatrix} x \\ y \\ z \end{bmatrix}$, with components in the inertial frame $\langle w \rangle$.
  The last three components of $\boldsymbol{\eta}$ forms the orientation vector $\boldsymbol{\eta}_2 \triangleq \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix}$ expressed in terms of the three angles roll, pitch, yaw (applied in the yaw-pitch-roll sequence Perez & Fossen (2007)). The singularity given by this Euler sequence that arise when $\theta = \pi/2$ is handled by a specific control objective, the *horizontal attitude* (section 3.2). As the name suggests, this objective assures the vehicle to stay away from the cited singularity.
  From the given definition, it results that $n = l + 6$

- The system velocity vector $\dot{\boldsymbol{y}} \in \mathbb{R}^n$ of the robot: $\dot{\boldsymbol{y}} \triangleq \begin{bmatrix} \dot{\boldsymbol{q}} \\ \boldsymbol{\nu} \end{bmatrix}$,
  where $\dot{\boldsymbol{q}} \in \mathbb{R}^l$ are the arm joint velocities and $\boldsymbol{\nu} \in \mathbb{R}^6$ is the vehicle velocity vector.
  The first three components of $\boldsymbol{\nu}$ form the linear velocities $\boldsymbol{\nu}_1 \triangleq \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix}$ and the last three form the angular velocities $\boldsymbol{\nu}_2 \triangleq \begin{bmatrix} p \\ q \\ r \end{bmatrix}$, both with components in the vehicle frame $\langle v \rangle$.
  The vehicle is considered fully actuated, so the vector $\dot{\boldsymbol{y}}$ coincides with the control vector that is the output of the kinematic layer.

2.2 Control Objectives

Let us consider what the robot has to achieve. An *objective* is a job that the robot must accomplish during the mission. Different objectives can be requested at the same time, for example we want the joints to stay away from their physical limits and the robot to maintain an horizontal attitude, while the end-effector is reaching

a desired pose.
In the following, specific discussions about these objectives are made.

2.2.1 Control Objectives classification

Control objectives can be classified depending on their importance (i.e. *priority*) in the context of the mission. In fact, some of them can be more important than others. For example, usually we want to avoid an obstacle if this one is in the trajectory given by the reaching goal objective. So, we want *before* to avoid the obstacle and *then* to continue towards the goal. In general, the idea is that the more important objectives have to been satisfied first, and then, *if possible*, also the less important ones. A general classification based on the *priority* is given here (from the more important class to the less important one):

- *Physical Constraints* objectives. This class includes the interactions with the environment (e.g. not pushing against a rigid surface, imposing a cooperative tool velocity when two robots are carrying together the tool).
- *System Safety* objectives. Usually (but not always), the safety is more important than the accomplishment of the Mission. As the name suggest, in this class there are objectives which aim to not damage anything (robot or objects), anyone (humans) and, in general, to not make the whole mission fail. Examples can be staying away from joint mechanical limits, or avoiding an obstacle.
- *Prerequisite* objectives. This class is for objectives needed to accomplish the actual mission. An example for a grasping Mission could be focusing the camera on the object to be grasped.
- *Mission* objectives, the actual objective that define the mission, like reaching an end-effector position.
- *Optimization* objectives. This class is to choose, among all the possible solutions (if multiple ones exist) the best one. For example, this category can be used to improve the energy consumption.

The classes of *priority* of these objectives are only a general classification. Depending on the application, objectives can have a different ordering.

2.2.2 Equality and Inequality Control Objectives

Difference between equality and inequality control objectives is another important classification. We consider a variable $\boldsymbol{x}(\boldsymbol{c}) \in \mathbb{R}^m$, dependent on the robot configuration vector $\boldsymbol{c}$, with *m dimension* of the control objective.
The control objective can be of two types:

- *Equality control objective*, the requirement, for $t \to \infty$, that $\boldsymbol{x}(\boldsymbol{c}) = \boldsymbol{x}_0$.
- *Inequality control objective*, the requirement, for $t \to \infty$, that $\boldsymbol{x}(\boldsymbol{c}) < \boldsymbol{x}_{max}$ or $\boldsymbol{x}(\boldsymbol{c}) > \boldsymbol{x}_{min}$ or $\boldsymbol{x}_{min} < \boldsymbol{x}(\boldsymbol{c}) < \boldsymbol{x}_{max}$.

Please note that here symbols $=, <, >$ refers to element-by-element comparison of the vectors.
To make the notations lighter, from now on the dependency of the variable $\boldsymbol{x}(\boldsymbol{c})$ on the configuration vector $\boldsymbol{c}$ will be omitted.
The importance of the subdivision between *equality* and *inequality* will be clarified later.

2.2.3 Reactive and non Reactive Control Task

Mathematically speaking, the aim of a control objective is to drive the variable $\boldsymbol{x}(\boldsymbol{c})$ toward a point $\boldsymbol{x}^*$ where the requisite (introduced in the previous section 2.2.2) is satisfied. For this scope, each control objective has always associated a *feedback reference rate* (in this thesis also denotes as *reference velocity*).
An example of *feedback reference rate* is:

$$\dot{\boldsymbol{x}}(\boldsymbol{x}) \triangleq \gamma(\boldsymbol{x}^* - \boldsymbol{x}), \quad \gamma > 0 \tag{2.1}$$

That is a simple proportional law, where γ is a positive gain proportional to the desired convergence rate for the considered variable. From now on, the *feedback reference rate* will have always this formulation.

To link the considered variable $\boldsymbol{x}(\boldsymbol{c})$ to the system velocity vector $\dot{\boldsymbol{y}}$, the following relationship is used:

$$\dot{\boldsymbol{x}} = \boldsymbol{J}\dot{\boldsymbol{y}} \tag{2.2}$$

that expresses how the system velocity vector $\dot{\boldsymbol{y}}$ influences the rate of change of the variable $\boldsymbol{x}$. $\boldsymbol{J} \in \mathbb{R}^{m \times n}$ is the so-called *task-induced Jacobian*.
The aim of having the actual $\dot{\boldsymbol{x}}$ as much as possible equal to the desired reference $\dot{\hat{\boldsymbol{x}}}$ is called a *reactive control task*.

There are situations where a task has not an associated control objective. It happens when an external agent (e.g. an human operator with a console) provides directly the reference rate. So, there is no *feedback reference rate* to calculate, because it is generated by the external agent. In the case of the human operator, it is this one that, seeing (in some way) how the system is behaving, adjust the command, making in its brain a sort of law like the one in formula (2.1). When no control objective is present, we speak about *non-reactive control task*.

2.2.4 Inequality Control Objectives Activation and Deactivation

During the execution of a mission, not always each inequality control objective is relevant. As an example, maintaining joints away from their mechanical limits is a safety objective which is needed only when the joints are actually near their limits. When a joint is sufficiently far away, there is no necessity to overconstrain the system imposing an additional velocity.
To deal with this, we speak about *activation* and *deactivation* of control objectives and of their relative control tasks. Let us define the following *activation function*:

$$a(x) \in [0, 1] \tag{2.3}$$

as a continuous, sigmoidal, function, which assumes 0 values within the validity region of the control objective. The validity region is intended as an interval where the objective is satisfied and it is far from not being satisfied any more (with satisfying we intend to accomplish the requirement explained in section 2.2.2). At the margin of this region, a smooth transition from 0 to 1 is present, to gently activate/deactivate the control objective when necessary.
For example, considering a scalar ($m = 1$) inequality control objective with the requirement $x(\boldsymbol{c}) > x_{min}$ the *activation function* can be defined as:

$$a(x) \triangleq \begin{cases} 1, & x(\boldsymbol{c}) < x_{min} \\ s(x), & x_{min} \leq x(\boldsymbol{c}) \leq x_{min} + \Delta \\ 0, & x(\boldsymbol{c}) > x_{min} + \Delta \end{cases} \tag{2.4}$$

where $s(x)$ is a smooth decreasing function joining the two extreme value 1 and 0, and Δ a value to create a zone where the inequality is satisfied but we want the objective to be activated a bit because we are near the margins. This is also important to prevent chattering problems that would occur if the objective was activated instantaneously.
Similar definitions of the activation function can be done for the other two kinds of inequality control objectives.

In general, for multidimensional control objectives ($m > 1$), the activation takes the form of a diagonal matrix:

$$\boldsymbol{A} \triangleq \begin{bmatrix} a_1 & & \\ & \ddots & \\ & & a_m \end{bmatrix} \tag{2.5}$$

where the diagonal elements a_i are activation functions defined similarly to the one in formula (2.4).

Obviously, for the equality control tasks the activation function is a constant equal to 1, because these tasks are always "active". In the multidimensional case, the activation is an identity matrix.
Same reasoning can be done for the *non-reactive* control tasks, being absent the variable $x(\boldsymbol{c})$, and being the reference velocities directly given.

2.3 Task Priority Inverse Kinematics

In this section, the core of the control architecture is explained.
We describe an *Action* $\mathscr{A}$ as a list of prioritized control objectives (with their associated tasks), each one positioned at a defined priority level k (where lower k means higher priority). With this notation, the following symbols are defined:

- $\dot{\tilde{\boldsymbol{x}}}_k \triangleq \begin{bmatrix} \dot{\tilde{x}}_{1,k} & \cdots & \dot{\tilde{x}}_{k_m,k} \end{bmatrix}^T$ is the vector of the reference velocities for the control task k, with a task dimension k_m.
- $\dot{\boldsymbol{x}}_k \triangleq \begin{bmatrix} \dot{x}_{1,k} & \cdots & \dot{x}_{k_m,k} \end{bmatrix}^T$ is the current rate of change of the task k.
- $\boldsymbol{J}_k$ is the Jacobian relationship which relates the current rate-of-change $\dot{\boldsymbol{x}}_k$ with the system velocity vector $\dot{\boldsymbol{y}}$ as in equation (2.2).
- $\boldsymbol{A}_k \triangleq \text{diag}(a_{1,k}, \cdots, a_{k_m,k})$ is the diagonal matrix composed by the activation functions described in section 2.2.4.

It is important to notice that, in the practice, different objectives can have the same priority. In this case, it is possible to simply stack the vectors and matrices to obtain a objective and a related task k that includes both objectives. For simplicity, but without loss of generality, different objectives will be considered always at different priority levels.

The aim of the kinematic layer is to find a system velocity vector $\dot{\tilde{\boldsymbol{y}}}$ that satisfies *as much as possible* the requirements of each objective of the action $\mathscr{A}$. Given the presence of different objectives with different priorities, we have to satisfy first the higher priority ones, and then, *if possible*, the lower priority ones. To do this, a sequence of nested minimization problems must be solved:

$$S_k \triangleq \left\{ \arg \text{R-} \min_{\dot{\boldsymbol{y}} \in S_{k-1}} \left\| \boldsymbol{A}_k (\dot{\tilde{\boldsymbol{x}}}_k - \boldsymbol{J}_k \dot{\boldsymbol{y}}) \right\|^2 \right\}, \; k = 1, 2, \ldots, N, \tag{2.6}$$

where $S_0 \triangleq \mathbb{R}^n$, S_{k-1} is the manifold of solutions of all the previous tasks in the hierarchy, and N is the total number of priority levels. The notation R-min is introduced

in Simetti & Casalino (2016), and it indicates a series of regularization to avoid singularities.
This problem is the so called *Task Priority Inverse Kinematics* (TPIK). To solve the formula (2.6), the so called *iCAT* (inequality Constraints And Task transitions) framework uses the following algorithm 1:

Algorithm 1 iCAT

$\boldsymbol{\rho}_0 = \mathbf{0}$
$\boldsymbol{Q}_0 = \boldsymbol{I}$
for k=1 **to** N **do**
  $\boldsymbol{W}_k = \boldsymbol{J}_k \boldsymbol{Q}_{k-1} (\boldsymbol{J}_k \boldsymbol{Q}_{k-1})^{\#,\boldsymbol{A}_k,\boldsymbol{Q}_{k-1}}$
  $\boldsymbol{Q}_k = \boldsymbol{Q}_{k-1} (\boldsymbol{I} - (\boldsymbol{J}_k \boldsymbol{Q}_{k-1})^{\#,\boldsymbol{A}_k,\boldsymbol{I}} \boldsymbol{J}_k \boldsymbol{Q}_{k-1})$
  $\boldsymbol{\rho}_k = \boldsymbol{\rho}_{k-1} + \mathrm{Sat}\left(\boldsymbol{Q}_{k-1} (\boldsymbol{J}_k \boldsymbol{Q}_{k-1})^{\#,\boldsymbol{A}_k,\boldsymbol{I}} \boldsymbol{W}_k \left(\dot{\boldsymbol{x}}_k - \boldsymbol{J}_k \boldsymbol{\rho}_{k-1}\right)\right)$
end for
$\dot{\boldsymbol{y}} = \boldsymbol{\rho}_N$

The special pseudo inverse operator $(\cdot)^{\#,\boldsymbol{A},\boldsymbol{Q}}$ [Simetti & Casalino (2016)] manages some invariance problems of (2.6); the function Sat(·) [Antonelli *et al.* (2009)] controls the variable saturations. Details of the procedure can be found again in [Simetti & Casalino (2016)].

2.3.1 Notes on Conflicting Objectives

From section 2.2.1, it should be understood that lower priority tasks are not always satisfied. The problem with this arises when the main mission objective (e.g. reaching a point with the end effector), that is not at the higher priority, can't be never accomplished, thus failing the general mission. This can be the case when an obstacle is met: the robot may stuck in a point of *local minima* (that is however better than crash into it). This means that the robot can't find a trajectory toward the goal and, *at the same time*, avoids the obstacle. So, it remains still because avoiding the obstacle is an objective with more priority. This is a general problem of all reactive control methods. The solution must be found at the mission manager level, which should plan another trajectory (e.g. with intermediate way-points) and/or another sequence of Actions. This problem is not considered in this work.

2.4 Arm-Vehicle Coordination Scheme

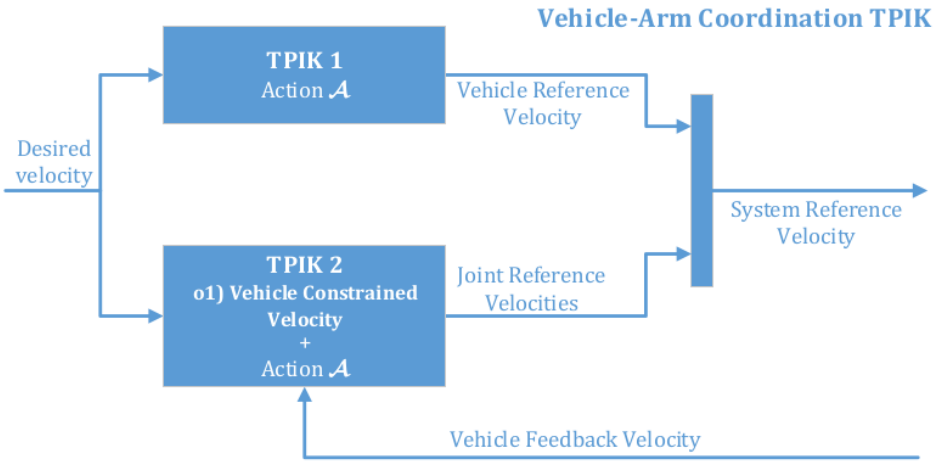


Figure 2.1: A scheme showing the two Task Priority Inverse Kinematics blocks for the arm-vehicle coordination

Inaccuracies in velocity tracking of the vehicle can have effects on the arm. A relevant problem arises when disturbances of the floating base, caused by thrusters and/or its large inertia, propagate and affect the end effector motions [Simetti & Casalino (2017)]. To solve this, a kinematic decoupling of the arm and the base is done, implementing it within the task priority approach.
As in the previous sections, we consider an Action $\mathscr{A}$, that is a list of prioritized objectives to be satisfied. The idea is to have two TPIK running in parallel as shown in figure 2.1:

- **TPIK 1**. It considers the vehicle together with the arm as a whole full controllable system. From its output $\dot{\tilde{\boldsymbol{y}}}$, only the vehicle velocity component are taken (discarding the arm ones).
- **TPIK 2**. It considers the vehicle as totally non controllable. So, a *non-reactive* task (2.2.3) is added at the top of the priority list $\mathscr{A}$ to *constrain* the output vehicle velocity to the real one (measured in some way). The other objectives of $\mathscr{A}$ remain unchanged. From its output $\dot{\tilde{\boldsymbol{y}}}$, only the arm part is taken.

At the end of the procedure, the two parts of $\dot{\tilde{\boldsymbol{y}}}$ are put togheter to compose the final system reference velocity vector.

Thanks to the TPIK 2, the joint velocities are *optimized*, in the sense that they follow *at best* the objectives of the action $\mathscr{A}$ considering also the *measured* vehicle velocity and its influence on the objectives.

For real mobile manipulators, in general, a multi-rate control of arm and vehicle is used, which means that velocities for the arm and for the vehicle are given at different frequency. This is common because usually the arm can be controlled more precisely and its performance are better than the base. The coordination schema proposed here is suitable for such an implementation: the TPIK 2 can run at higher frequency, updating the commanded arm velocities more frequently than the vehicle ones.

2.5 Cooperation Scheme

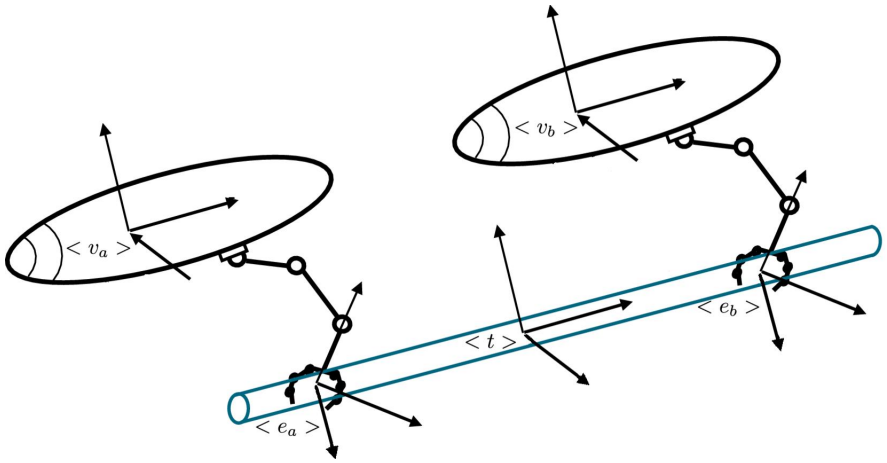


Figure 2.2: The frames of the two cooperative vehicles carrying a common object

In this section, the discussion about cooperation is explained. We limit the explanations to only two cooperative robotic systems, but, in general, more agents can be considered.

The *cooperation* is used to carry a common tool with the two manipulators, without making it fall or break. This is done at kinematic level: the scheme provides suitable system velocities $\dot{\tilde{\boldsymbol{y}}}_a$ and $\dot{\tilde{\boldsymbol{y}}}_b$ for the two robots considering the constraint given by the carried common object. So, both system velocities $\dot{\tilde{\boldsymbol{y}}}_a$ and $\dot{\tilde{\boldsymbol{y}}}_b$ must cause same Cartesian velocity $\dot{\boldsymbol{x}}_t$ to the tool.
The coordination policy that will be presented takes care of the bandwidth restriction typical of underwater scenarios. Thus, it deals with the cooperation in a decentralized manner, limiting as much as possible the amount of data exchanged.

Furthermore, this scheme is different from the classical *leader-follower* ones, because, as we will see, the "leadership" changes based on the difficulties in tracking the ideal tool velocity that one robot can meet.

It is assumed that the object is held firmly by both agents, so no sliding happens during the missions. The two robots agree on a shared fixed frame, so, their respective tool frames $\langle t_a \rangle$ and $\langle t_b \rangle$ and the object frame $\langle o \rangle$ are coincident:

$$\langle t \rangle \triangleq \langle t_a \rangle = \langle t_b \rangle = \langle o \rangle \tag{2.7}$$

In figure 2.2 the main frames related to the cooperation are shown.

The firm grasp assumption imposes that:

$$\dot{\boldsymbol{x}}_t = \boldsymbol{J}_{t,a}\dot{\boldsymbol{y}}_a = \boldsymbol{J}_{t,b}\dot{\boldsymbol{y}}_b \tag{2.8}$$

with $\dot{\boldsymbol{x}}_t$ the object velocity with component on $\langle t \rangle$; $\dot{\boldsymbol{y}}_a$, $\dot{\boldsymbol{y}}_b$ the system velocity vectors of agents a and b (introduced in section 2.1); and $\boldsymbol{J}_{t,a}$, $\boldsymbol{J}_{t,b}$ the system Jacobians of agents a and b with respect to $\langle t \rangle$. These Jacobians tell how the tool velocity $\dot{\boldsymbol{x}}_t$ is affected by the system velocities $\dot{\boldsymbol{y}}_a$ and $\dot{\boldsymbol{y}}_b$. Due to the firm grasp assumption, the tool velocities caused by $\dot{\boldsymbol{y}}_a$ and $\dot{\boldsymbol{y}}_b$ must be equal.

Let us rewrite the second part of equation (2.8) as:

$$\begin{bmatrix} \boldsymbol{J}_{t,a} & -\boldsymbol{J}_{t,b} \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{y}}_a \\ \dot{\boldsymbol{y}}_b \end{bmatrix} \triangleq \boldsymbol{G}\dot{\boldsymbol{y}}_{ab} = 0 \quad \Longleftrightarrow \quad \dot{\boldsymbol{y}}_{ab} \in ker(\boldsymbol{G}) \tag{2.9}$$

$ker(\boldsymbol{G})$ represents the subspace where $\dot{\boldsymbol{y}}_{ab}$ is constrained to lay for the firm grasp assumption.

We could consider the two manipulators as a unique system simply stacking correctly vectors and matrices. To transport cooperatively the tool, an additional *physical constraint* objective would be added to the TPIK list, to ensure the control outputs a command $\dot{\tilde{\boldsymbol{y}}}$ which satisfies the constraint (2.9). In practice, with this new objective, in the minimization problems of (2.6) we would have $S_1 = ker(\boldsymbol{G})$.
The problem with following this way is that we are not considering that the two vehicles are separate entities. This idea would be feasible when the agent is a single robot with two arms. Instead, in this case, exchanging all the vectors and matrices between the robots during the TPIK procedure would not be possible, especially in an underwater situation. So, another method must be considered.

The equation (2.8) can be expressed in the Cartesian space as:

$$\dot{\boldsymbol{x}}_t = \boldsymbol{J}_{t,a}\boldsymbol{J}^{\#}_{t,a}\dot{\boldsymbol{x}}_t = \boldsymbol{J}_{t,b}\boldsymbol{J}^{\#}_{t,b}\dot{\boldsymbol{x}}_t \tag{2.10}$$

$$(J_{t,a}J_{t,a}^{\#} - J_{t,b}J_{t,b}^{\#})\dot{x}_t \triangleq C\dot{x}_t = 0 \tag{2.11}$$

$$\dot{x}_t \in ker(C) = Span(I - C^{\#}C) \tag{2.12}$$

C is a particular matrix called *Cartesian Constraint Matrix*. The kernel of C expresses the space of achievable object velocities at the current configurations of the two robots.

The idea of the scheme is to put a non-reactive task at the top of the hierarchy, to constrain the desired object velocity $\dot{\tilde{x}}$ in this subspace. After this constraint, we are sure, at kinematic level, that both agents can follow this desired object velocity despite the possible different situations caused by the other objectives and by the different robots configurations.

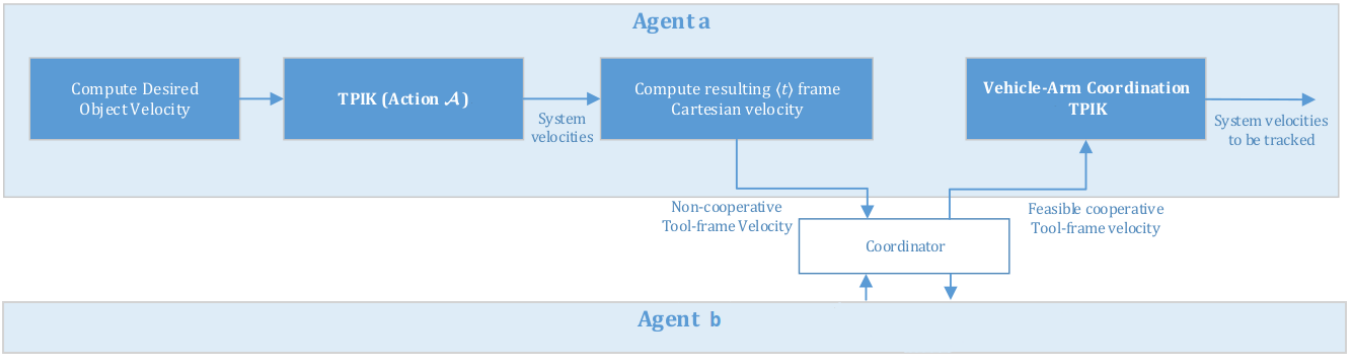


Figure 2.3: The cooperation scheme with its different steps. The Agent b block is equal to the Agent a one.

The scheme, sketched in figure 2.3 proceeds as follows:

- In the first step, the two agents calculate system velocities (using the TPIK explained in section 2.3) as if they were alone. So, we have:

  $$\dot{x}_{t,a} = J_{t,a}\dot{y}_a, \qquad \dot{x}_{t,b} = J_{t,a}\dot{y}_b \tag{2.13}$$

  where, in general, the two *non cooperative* tool velocities are different: $\dot{x}_{t,a} \neq \dot{x}_{t,b}$.

- The tool velocities are exchanged (i.e. they are sent to the coordinator) and a *cooperative* tool velocity $\dot{\hat{x}}_t$ is computed as:

  $$\dot{\hat{x}}_t = \frac{1}{\mu_a + \mu_b}(\mu_a\dot{x}_{t,a} + \mu_b\dot{x}_{t,b}), \qquad \mu_a, \mu_b > 0 \tag{2.14}$$

$$\begin{aligned} \mu_a &= \mu_0 + \|\dot{\tilde{\boldsymbol{x}}}_t - \dot{\boldsymbol{x}}_{t,a}\| \triangleq \mu_0 + \|\boldsymbol{e}_a\|, \\ \mu_b &= \mu_0 + \|\dot{\tilde{\boldsymbol{x}}}_t - \dot{\boldsymbol{x}}_{t,b}\| \triangleq \mu_0 + \|\boldsymbol{e}_b\|, \\ \mu_0 &> 0 \end{aligned} \tag{2.15}$$

where $\dot{\tilde{\boldsymbol{x}}}_t$ is the ideal velocity that, if applied, would asymptotically take the tool to the desired goal.
The *cooperative* tool velocity $\dot{\hat{\boldsymbol{x}}}_t$ is a *weighted* compromise between the two *non cooperative* ones. The *weights* μ_a, μ_b give more freedom to the robot which meet the highest error $\boldsymbol{e}$. This error is a way to understand how much one robot is in difficult in tracking the *ideal* tool velocity $\dot{\tilde{\boldsymbol{x}}}_t$.

- The new *cooperative* tool velocity $\dot{\hat{\boldsymbol{x}}}_t$ is not, in general, a *feasible* velocity that both vehicle can provide to the tool. So, an additional passage is required:

$$\dot{\tilde{\boldsymbol{x}}}_t \triangleq \left(\boldsymbol{I} - \boldsymbol{C}^{\#}\boldsymbol{C}\right)\dot{\hat{\boldsymbol{x}}}_t \tag{2.16}$$

  with $\boldsymbol{C}$ defined in (2.11).

- Each agent runs a new TPIK procedure, with a objective list identical to the first one, but with a *non-reactive* control objective at the top to track the *feasible cooperative* velocity $\dot{\tilde{\boldsymbol{x}}}_t$. The outputs of this procedure, $\dot{\hat{\boldsymbol{y}}}_a$ and $\dot{\hat{\boldsymbol{y}}}_b$ will be the final velocities which the kinematic layer provides.
  Moving the equality control objective to make the end effector reach the goal at the top of the hierarchy does not influence the safety tasks. This property is proven in Wanderlingh (2018).

The method assumes that the Coordinator can calculate the ideal tool velocity $\dot{\tilde{\boldsymbol{x}}}_t$, so it must know the transformation matrix between the goal frame $\langle g \rangle$ and the tool frame $\langle t \rangle$.

It can be noticed how the only information that the agents must exchange are the *non-cooperative* velocities $\dot{\boldsymbol{x}}_{t,a}$ and $\dot{\boldsymbol{x}}_{t,b}$, the matrices $\boldsymbol{J}_{t,a}\boldsymbol{J}_{t,a}^{\#}$ and $\boldsymbol{J}_{t,b}\boldsymbol{J}_{t,b}^{\#}$ (to build the Cartesian Constraint Matrix $\boldsymbol{C}$), and the feasible velocity $\dot{\tilde{\boldsymbol{x}}}_t$. Less data can be exchanged if we have Jacobians expressed analytically. In this case, instead of sharing the two 6×6 matrices, we can share only two configuration vectors $\boldsymbol{c}$ $(n \times 1)$, and make the coordinator calculate the Jacobians from their analytical expressions. Even less data can be shared if the *coordinator* is a procedure that runs on a robot, and it is not on an external node. In this case, practically only half of the amount of data must be shared through water.

Chapter 3

Control Architecture: Methods

In this chapter, the theory explained in the previous Chapter 2 is exploited to deal with the scenario stated for this thesis.
In Section 3.1, a new control objective, called *Force-Torque* objective, is added to the Action list of the TPIK approach. The aim of this new objective is to drive the peg "away" from collisions that may happen during the insertion phase. Thanks to force-torque data given by a sensor, this will help the mission, reducing the amount of collisions between the peg and the hole. It is important to notice that we are exploiting force-torque information at kinematic level.
In section 3.2, a list of objectives, including the new Force-Torque one, is presented. This composes the Action $\mathscr{A}$ that must be accomplished for the stated mission.
Section 3.3 describes another (additional) method which exploits the data given by the force-torque sensor. In brief, this new *routine*, called Change Goal, shifts the origin of the goal frame (which is inside the hole) according to the forces detected on the peg. The aim is to compensate the error given by a not perfectly estimation of the hole's pose.

3.1 Force-Torque Objective

Information from a force torque sensor can be exploited at kinematic level, inserting an additional control objective into the TPIK procedure. The aim of this objective is to zeroing the forces and torques acting on the peg. This is done by generating properly joints and vehicle velocities to drive the peg in such a way the forces and torques decrease. This objective is similar to the one used for pipeline weld inspection in Simetti *et al.* (2018).
If we visualize the resultant of the forces on the peg caused by the collisions as a vector, moving *linearly* the peg along this vector will cause the forces to decrease. The same idea can be use with torques, *rotating*, along the resultant vector instead

of moving linearly.

The *feedback reference rate* for this objective will be:

$$\dot{\tilde{\boldsymbol{x}}}_{ft} \triangleq \begin{bmatrix} \dot{\tilde{x}}_f \\ \dot{\tilde{x}}_m \end{bmatrix} \triangleq \begin{bmatrix} \gamma_f \\ \gamma_m \end{bmatrix} \begin{bmatrix} 0 - \|\boldsymbol{f}\| \\ 0 - \|\boldsymbol{m}\| \end{bmatrix} \qquad 0 < \gamma_f < 1, \quad 0 < \gamma_m < 1 \tag{3.1}$$

where $\|\boldsymbol{f}\|$ and $\|\boldsymbol{m}\|$ are the norms of the forces and torques vectors $\boldsymbol{f}$ and $\boldsymbol{m}$. Gains smaller that 1 are necessary to reduce the amount of speed requested. In fact, for example, if a force with a norm of 1 N (which is relatively small, so common to be detected) was present, a gain equal to 1 would mean to request a tool speed of 1 m/s, that is an exaggeration for this case.
It can be noticed that, instead of the full 3-dimensional vectors $\boldsymbol{f}$ and $\boldsymbol{m}$, the norms $\|\boldsymbol{f}\|$ and $\|\boldsymbol{m}\|$ are used. This is done to not overconstrain the system and to let more freedom to lower priority task. Even with norms, the collisions are reduced, because when we bring to zero the norms, also each component of the vector tends to zero.

The *feedback reference rate* of equation (3.1) is a velocity that the tool must follow. So, the Jacobian must be built considering this fact. For the *task-induced* Jacobian (section 2.3) of this new task, we have to split the linear and the angular part of the tool Jacobian $\boldsymbol{J}_t$. Then, due to the fact that we are considering the norms, we have to pre-multiplying the two parts for the normal vector of $\boldsymbol{f}$ and $\boldsymbol{m}$ transposed:

$$\boldsymbol{J}_{ft} \triangleq \begin{bmatrix} \boldsymbol{J}_f \\ \boldsymbol{J}_m \end{bmatrix} \triangleq \begin{bmatrix} \left(-\dfrac{\boldsymbol{f}}{\|\boldsymbol{f}\|}\right)^T {}^{lin}\boldsymbol{J}_t \\ \left(-\dfrac{\boldsymbol{m}}{\|\boldsymbol{m}\|}\right)^T {}^{ang}\boldsymbol{J}_t \end{bmatrix} \tag{3.2}$$

where $\boldsymbol{J}_f, \boldsymbol{J}_m \in \mathbb{R}^{1\times l}$; $\boldsymbol{J}_t$ is the Jacobian which express how the Cartesian tool velocity $\dot{\boldsymbol{x}}_t$ is affected by the system velocity vector $\dot{\boldsymbol{y}}$; lin, ang superscripts refer to *linear* (top three rows) and *angular* (bottom three rows) parts of $\boldsymbol{J}_t$.

This objective can be considered as a *pre-requisite* one (section 2.2.1), because it is better to reduce the collisions *before* going on with the insertion, also to avoid stuck. In truth, this kind of objective could be also considered as a Physical Constraints one, like in Simetti *et al.* (2018). In this case, the first choice is made. In both cases, it is always put at higher priority than the *reaching goal objective*. This will cause the robot to, first, try to nullified the forces and torques (if collisions happened), and only after (i.e. *if possible*) to move the peg towards the goal.
Deactivating the task is necessary when the forces and/or torques are zero, to not generate system velocities for this task when they are not necessary. So a smooth

activation function $\boldsymbol{A} \in \mathbb{R}^{2\times 2}$ is used (similarly to the generic one of section 2.2.4):

$$\boldsymbol{A}_{ft} \triangleq \begin{bmatrix} a_f & 0 \\ 0 & a_t \end{bmatrix}$$
$$a_f(\|\boldsymbol{f}\|) \triangleq \begin{cases} 0, & \|\boldsymbol{f}\| = 0 \\ s(\|\boldsymbol{f}\|), & 0 < \|\boldsymbol{f}\| \leq 0 + \Delta \\ 1, & \|\boldsymbol{f}\| > 0 + \Delta \end{cases} \tag{3.3}$$
$$a_m(\|\boldsymbol{m}\|) \triangleq \begin{cases} 0, & \|\boldsymbol{m}\| = 0 \\ s(\|\boldsymbol{m}\|), & 0 < \|\boldsymbol{m}\| \leq 0 + \Delta \\ 1, & \|\boldsymbol{m}\| > 0 + \Delta \end{cases}$$

where $s(\cdot)$ is a smooth *increasing* function from 0 to 1, and Δ a constant to create the smooth zone.
In this case, the activation function is also important for a mathematical detail. In fact, we can see from the Jacobian formula (3.2), that the norms are in the denominator. When they are near to zero, numerical issue (such as too big values in the Jacobian) may happen. This is a common problem when a task is used with the norm. But an easy solution is to deactivate the task when the norm is below a little value ϵ. In this case, for the force part:

$$a_f(\|\boldsymbol{f}\|) \triangleq \begin{cases} 0, & \|\boldsymbol{f}\| \leq \epsilon \\ s(\|\boldsymbol{f}\|), & \epsilon < \|\boldsymbol{f}\| \leq 0 + \Delta \\ 1, & \|\boldsymbol{f}\| > 0 + \Delta \end{cases} \tag{3.4}$$

The activation for the torque is similar.

Considering the minimization problem presented with the formula (2.6), for this specific objective the equation will be:

$$S_k \triangleq \left\{ \arg\text{R-}\min_{\dot{\tilde{\boldsymbol{y}}} \in S_{k-1}} \left\| \boldsymbol{A}_{ft}(\dot{\boldsymbol{x}}_{ft} - \boldsymbol{J}_{ft}\dot{\tilde{\boldsymbol{y}}}) \right\|^2 \right\} \tag{3.5}$$

with k that depends on the order of priority chosen for the new objective.

3.2 Objectives Prioritized List

In this section, the objectives inserted into the TPIK procedure, to form the Action $\mathscr{A}$ suitable for the mission, are listed and briefly explained.

The first task prioritized list, the one where the two robot act independently to each other, is:

- **Joint Limits avoidance** (*reactive, inequality, safety*). This objective keeps joint away from their mechanical limits. It must be at an high priority because it is a *safety* objective. It is also an *inequality* one to not overconstrain the system when joints are away from their limits.
- **Horizontal Attitude** (*reactive, inequality, safety*). This objective is to maintain the vehicle horizontal respect to the water surface. Most of the underwater vehicle are passively stable (and also not controllable) in roll and pitch, so this objective would be useless. In this thesis, where buoyancy is not simulated and the vehicle has full DOF, this objective is necessary. It is also important to not occur in the singularity given by the Euler Sequence used to describe the orientation (section 2.1), which arises when the pitch is equal to $\pi/2$. The objective is consider a *safety* one because it is more important than the accomplishment of the mission; and it is an *inequality* for the same reason of the previous.
- **Force-Torque** (*reactive, inequality, pre-requisite*). This objective is to reduce the forces and torques acting on the peg during the insertion. This objective is detailed in section 3.1.
- **Tool position control** (*reactive, equality, mission*). This objective is the one that defines the real mission. It is used to bring the tool towards the defined goal (i.e. inside the hole), so it always must be active.
- **Preferred Arm Shape** (*reactive, inequality, optimization*). This is a low priority objective to maintain the arm in a predefined shape. This shape permits the arm to have good dexterity (if the shape is chosen wisely) but it is also useful to transport the peg in a natural way. Being only an *optimization* objective, it is put at low priority.

The categories (written in italic) are explained in section 2.2. Please note that in the code there is also an additional *last task* which is used to cancel out any practical discontinuities during task activations [Simetti & Casalino (2016)].

After the first TPIK procedure is run for the presented list, it is called other two more times. One is for the cooperation between the two robots (section 2.5), and the other for the vehicle-arm coordination (section 2.4). Respectively, two *non-reactive* objectives are put at the top of the hierarchy listed above, as explained in the cited sections.

Please note that some important objectives related to safe transportation (e.g. obstacle avoidance, minimum altitude from seafloor, minimum distance between robots), grasping (e.g. camera centring object) are not considered because they are not necessary in the particular experiment chosen, and also because they are explored in other works [Simetti & Casalino (2017); Wanderlingh (2018); Simetti *et al.* (2018)].

3.3 Change Goal Frame Routine

In general, the frame where the tool is driven to by the control architecture (i.e. the *goal* frame) is known with some errors. This is the case when, for example, we have some computer vision algorithm to estimate the hole pose. This error between ideal goal and estimated one, both in the linear and in the angular components, can cause the peg to collide a lot with the hole. If the peg clashes against the hole structure face (i.e. outside the proper hole), it is pushed back and a stuck may happen. The result is that the peg will go on bouncing back and forth forever. In the literature, various methods (cited in 1.1.3) have been explored to deal with the problem of *finding the hole*, moving the peg on the surface. In this work, this is not explored.

This thesis focuses only the final part of the *peg-in-hole*, i.e. when the peg is inside the hole, but bad alignment causes a lot of collisions with internal hole's walls. In this case, usually, the peg does not stuck in a intermediate position, because the forces and torques acting on the peg *naturally* drive it inside the cavity. But, in such a way, the peg continuously scrapes along the hole's walls, possibly damaging itself, the hole and also the robot which can suffer some stress. In practice there is a chattering problem: the peg continuously *bounces* because the control wants to drag it towards the erroneous pose, while the hole's walls cause forces in a different direction.

The method explained in this section try to solve this problem, modifying the goal accordingly to the forces acting on the peg.
Let us consider the Cartesian coordinate of the origin ${}^w\boldsymbol{g} \in \mathbb{R}^3$ of the goal frame, projected on the world frame. According to the force detected, we modify this origin, providing ${}^w\boldsymbol{g}' \in \mathbb{R}^3$ as:

$$\begin{aligned} {}^w\boldsymbol{g}' &= {}^w\boldsymbol{g} + {}^w\tilde{\boldsymbol{f}} \\ {}^w\tilde{\boldsymbol{f}} &= {}^w\boldsymbol{R}_t \, {}^t\tilde{\boldsymbol{f}} \\ {}^t\tilde{\boldsymbol{f}} &= k[0, f_y, f_z], \qquad 0 < k < 1 \end{aligned} \tag{3.6}$$

where $[0, f_y, f_z]$ is the vector which represent the resultant of the forces acting on the peg, projected on the *tool* frame, but with the component along x put to zero; ${}^{w}\boldsymbol{R}_t$ is the rotation matrix from world to tool; k a positive gain smaller than 1.
The component on x axis of the force is neglected. This is done because the x axis of the tool frame $\langle t \rangle$ is the one along the length of the peg. So, we do not want that this component modifies the goal because it would change the wanted depth of insertion.

To proper utilize the data given by the sensor, that is the force vector, we must use a gain k. This gain is less than 1 for a similar reason as the one explained for the Force-Torque objective (section 3.1). For example, when a little force of 1N is detected, obviously we don't want to move the goal frame of 1 m or more. Instead, we want very little modifications, done every time a force (not null) is detected by the sensor. So the formula (3.6) shifts the goal at the same frequency of the provided sensor data. Obviously we can also update the goal less often, maybe with a bigger gain.

To understand better the method, we can take as an example the situation where the estimated goal is a bit on the left respect to the centre of the hole, but not so much to make the peg miss the hole. In such a case, the control architecture drives the peg on the left of the centre, causing a lot of collisions with the left side of the inner hole. So, the peg suffers forces with an important component in the right direction. Thus, this method shifts the goal to the right.
From this example should be noticed that this approach could cause the same problem with the opposite side of the cavity, if the goal is shifted too much on the right. For this reason, setting a suitable k is important.

Chapter 4

Control Architecture: Simulation Results

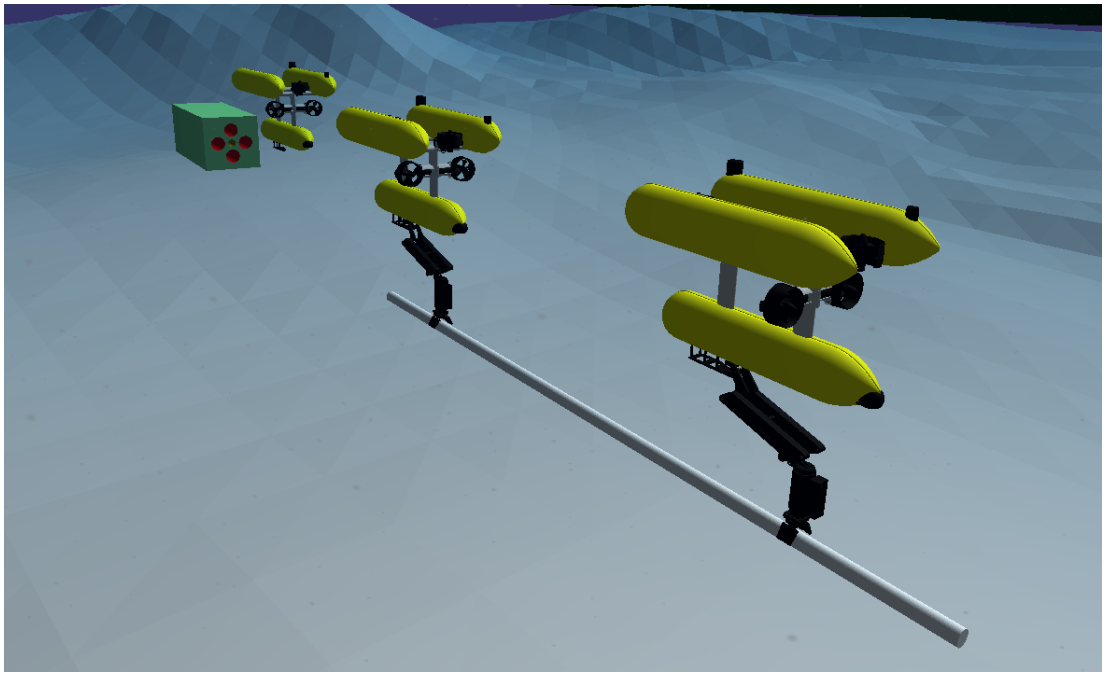


Figure 4.1: The Scenario of the experiment. The two twin robots are carrying the peg, while the third robot is watching the hole to estimate its pose.

In this Chapter, experimental set-up is described, and results are given and discussed. The code for the whole architecture is available at the following link: https://github.com/torydebra/AUV-Coop-Assembly; some details about it are discussed in section 4.5 and in appendix A. A video of the final experiment is visible at the fol-

lowing link: https://streamable.com/kvoxq (online; accessed 10-08-2019).

The scenario is made up of two Girona 500 I-AUV's, each one equipped with a CSIP Robot arm5E (a 4 DOF arm with a parallel yaw gripper). The final goal is to successfully coordinate the robots in such a way that the peg, hold by both manipulators, is inserted correctly in the hole, fixed in the environment. In the literature, this problem is known as *peg-in-hole*.
One robot is equipped with a force-torque sensor that permits to understand forces applied on the peg, caused by collisions during the insertion phase. This information is provided to both robots. A third robot is equipped with two cameras to estimates the hole's pose. The figure 4.1 shows what has just been described.
The chosen strategy divides the problem in two phases: Hole Detection and Insertion. In the first, preliminary steps are done to detect the hole. The third robot, not used for manipulation, is in charge of exploiting computer vision algorithms to estimate the pose of the hole. Details about this are given in section 5. The second phase explores the problems inherent to transportation of the tool, the interaction between the peg and the hole, and the communication between the carrying agents. This is described in this Chapter.

4.1 Choosing the Simulator

Some effort has been spent to choice a suitable simulator for the case. At the end, UWSim [Prats *et al.* (2012)] was chosen. It is a simulator largely used for this kind of scenarios, where underwater robots must accomplish some particular tasks. It provides a different variety of useful sensors (e.g. the used force-torque sensor and the cameras), and also personalized ones can be added. It uses ROS as the simulator interface, which makes it really easy to use. Through ROS messages, we can send commands to the robots and we can receive information from the going-on test. Contact physics is implemented using OSGBullet to integrate the physics engine Bullet with the 3D graphics toolkit OSG. For what concerns the collisions, these are calculated taking into account the compenetration between 3D models. The more the models are compenetrated, the more the forces and torques have big magnitudes. To know further details about how the simulator is built, especially for the contact physics part, please refer to the documentation of the cited software.

The cons of UWSim is that the simulation is fully kinematic, so no dynamic interactions are present (expect for contact physics). This means, for example, that velocities sent to the robot are immediately accomplished, that buoyancy is not present, and that it is not simulated the physic related to the object grasping. For the scope of this thesis, this lack is not important because dynamic is not considered. Further-

more, how the collisions between the tool and the hole affect the whole manipulator chain, can be simulated thanks to the information provided by the force-torque sensor, as explained in section 4.3.

To fill the UWSIM lack of dynamics, a good alternative can be FreeFloatingGazebo [Kermorgant (2014)]. In truth, this simulator is a plug-in for Gazebo and UWSim; it integrates them in order to achieve both dynamics (thanks to Gazebo) and visually realistic I-AUV simulation (thanks to UWSim). It is easy to use as UWSim, being ROS always the adopted interface, but also because the same scene (described by an *.xml* file) can be used. With this plug-in, we can simulate features as buoyancy and coupling dynamics between arm and vehicle (i.e. how arm movements affect the base). FreeFloatingGazebo has been taken into consideration for dynamic tests, but at the end it is not used due to the lack of time. However, further works toward dynamic simulations can begin from here.

Another simulator is Gazebo, widely known in all robotics fields. It is the de-facto standard simulator for ROS. Due to its generic purpose, it is not a ready-to-use simulator for an underwater environment, so it can be only a starting point to develop a software specific for this particular scenario (as it is done by FreeFloatingGazebo). Also other similar simulators, V-REP [E. Rohmer (2013)] and Webots [Michel (2004)] have been taken into consideration, but then they have been discarded for the same "not ready-to-use" reason like Gazebo.

Another interesting simulator is USV [Paravisi *et al.* (2019)], a really recent and in-development project. It takes the best from UWSim, Gazebo and FreeFloatingGazebo to implement realistic simulations. However, it is focused more on surface vessels dynamics.

More details on these and other simulators are available in Cook *et al.* (2014) and Paravisi *et al.* (2019). From Paravisi *et al.* (2019), a schematic comparison is taken and shown in figure 4.2.

Simulator	Waves	Buoyancy	Water Currents	Wind Currents	Thruster Underwater	Thruster above Water	Foil
UWSim	√	√	×	×	√	×	×
Gazebo	×	×	×	×	√√	√√	×
Freefloating Gazebo	√	√	√	×	√√	√√	×
VREP	√	√	×	×	√	√√	×
RobotX Simulator	√	√√	×	√	√√	√√	×
USVSim	√√	√√	√√	√√	√√	√√	√√

Figure 4.2: Schematic recap of the simulation comparison taken from Paravisi et al. (2019). × stands for no implemented feature; √ for a feature that is a discrete representation of the real one; √√ for a good feature reproduction of the real one. More details on how each feature is evaluated are available in the original paper.

4.2 Simulating the Firm Grasp Constraint

Being UWSim (introduced in section 4.1) only a kinematic simulator, some additions have to be made.
Without dynamics, simulating correctly the peg grasping is impossible. The simulator permits to fake it with an *object picker* sensor: when an object is sufficiently near to the point where this sensor is, it becomes "grasped" and, from then on, it will rigidly move with the whole robot. The problem here is that we have two robots that must take the tool, so the object can't rigidly move with both, but only with the first which catches it.
Furthermore, external forces applied to an object (grasped or not) can't be detected with the force-torque sensor, because, in the way it is implemented, it only detects forces acting on a vehicle part.

To solve this issue, a peg is modelled as an additional fixed joint attached to the end-effector of each robot. In this way, each peg is rigidly attached to its own robot. Now, the problem is how to maintain the two pegs perfectly overlapped during the whole mission, because, obviously, in real situation the tool is unique. This is also needed because the control architecture assumes a *firm grasp* of the tool, without any slipping. This means that the end-effector does not move respect to the peg, and, consequently, the end-effectors of the robots do not move respect themselves. It is also important to report that the force-torque sensor does not detect collisions between the two tools, so we don't have problem for this point o view.

In the simulation, collisions between the "pegs" and the hole cause the tools to

drive apart. This happens because collisions are propagated to the robots with a formula which use Jacobian (detailed in section 4.3). Jacobian derives from approximation of non-linear relationship, so results are not perfect. Thus, during the transportation, but especially during the collision propagation in the insertion phase, the two pegs distance themselves a bit. This causes that the control point for one robot is in a different position of what it expects, increasing the errors.

In real scenario, a firm grasp acts like a "glue": if the end-effector tends to go away from the grasping point, friction acts to maintain it to the contact point. This is true for very small errors; if the cooperation's performance is not good, the common tool falls down or something breaks.

In the simulation, to fake the firm grasp, an additional routine is implemented. It simply calculates the distance between the two pegs, and it generates robot velocities to nullify this gap. It is important to notice that this is an aid that we would have also in a real scenario, as explained before. The only difference is that, in real scenario, if the errors are too big the end-effector begins to slip, and it will never return to its original grasping point. In this case, it returns always to the initial point.

The velocities generated by this routine are not so big to hide bad cooperation; so the tests are suitable to evaluate the proposed architecture, and to simulate real behaviours.

4.3 Simulating the Collision Propagation

When a robot interacts with the environment, each contact generates forces on it. Missions related to assembling any objects can't be studied in a properly manner without some considerations about these forces. In a *peg-in-hole* mission, collisions between the peg and the hole will be transferred through the whole kinematic chain until the floating base, causing disturbances to the whole robotic system. Thus, it is necessary to simulate these behaviours. Being UWSim a kinematic-only simulator, an additional feature is implemented to cope with these kind of collisions.

Let us define $\boldsymbol{f} \in \mathbb{R}^3$ and $\boldsymbol{m} \in \mathbb{R}^3$ as:

$$\boldsymbol{f} = \begin{bmatrix} f_x \\ f_y \\ f_z \end{bmatrix} \qquad \boldsymbol{m} = \begin{bmatrix} m_x \\ m_y \\ m_z \end{bmatrix} \tag{4.1}$$

being $\boldsymbol{f}$ and $\boldsymbol{m}$ the resultant force and the resultant torque (projected on the tool frame $\langle t \rangle$) of all the forces and torques acting on the tool.

These vectors generate a disturbance on the whole system as a velocity $\dot{\boldsymbol{y}}_\delta \in \mathbb{R}^n$. This velocity can be written as [Siciliano *et al.* (2009)]:

$$\dot{\boldsymbol{y}}_\delta \triangleq \begin{bmatrix} \dot{\boldsymbol{q}}_\delta \\ \boldsymbol{v}_{1\delta} \\ \boldsymbol{v}_{2\delta} \end{bmatrix} = \begin{bmatrix} k_q \\ k_{v1} \\ k_{v2} \end{bmatrix} \begin{bmatrix} ({}^{lin}\boldsymbol{J}_t)^T \boldsymbol{f} + ({}^{ang}\boldsymbol{J}_t)^T \boldsymbol{m} \end{bmatrix} \qquad 0 < k_q, k_{v1}, k_{v2} < 1 \qquad (4.2)$$

where $\boldsymbol{J}_t$ is the Jacobian which expresses how the Cartesian tool velocity $\dot{\boldsymbol{x}}_t$ is affected by the system velocity vector $\dot{\boldsymbol{y}}$; lin, ang superscripts refer to *linear* part (top three rows) and *angular* part (bottom three rows) of $\boldsymbol{J}_t$; $\dot{\boldsymbol{q}}_\delta \in \mathbb{R}^l$ are the joints velocities caused by the collisions; $\boldsymbol{v}_{1\delta} \in \mathbb{R}^3$ and $\boldsymbol{v}_{2\delta} \in \mathbb{R}^3$ are the linear and angular vehicle velocity caused by the collisions; k_q, k_{v1}, k_{v2} are positive gains smaller than 1, and in general different from each other because we are considering different types of velocities.
Similarly to the Force-Torque objective (section 3.1) and for the Change Goal routine (section 3.3), gains smaller than 1 are necessary when dealing with forces and torques to not generate too high velocities.

4.4 Experiment's Assumptions

It is important to detail the assumptions made during the simulation. In fact, some problems, that must be taken into account in a real environment, are not explored. This is necessary due to the difficulties of the particular mission analysed.
So, in this section, the main assumptions are summarized.

- Simulation is kinematic-only. This implies, for example, that the commanded velocity to the vehicle and the arm are accomplished *instantaneously* and *perfectly*. Another implication is that the movements of arm and of the vehicle don't influence each other at all. The only exceptions are the Firm Grasp constraint (section 4.2) and the Collision Propagation (section 4.3).

- The initial configuration shows the peg already grasped *correctly* by both robots. Also, the point where the end-effectors have grasped the tool and the peg's dimensions are known. This implies that the relative position between each robot and peg's tip is *perfectly* known.
  Such an initial configuration has been chosen because the grasping phase and problems arising during cooperative transportation have been explored in other projects like MARIS and ROBUST (e.g in the work Simetti & Casalino (2017)) and in the on-going TWINBOT.

- A common reference frame (denoted as $\langle w \rangle$ - *world* in the whole thesis) is used to know the relative poses among objects, carrying robots and the Vision robot.

This assumption is mostly needed to make the Vision robot share correctly the estimated hole's pose.
In real situation, the underwater location of something is always an issue and it is never really precise. Good precision can be provided, for example, after some preliminary works in mapping the seafloor. Another method can be the exploitation of helper support vessels, for example, as explored the WiMUST project [Abreu *et al.* (2016)]. This can provide a common reference point somewhere.
Please note that, for this work, it is not important that the common frame $\langle w \rangle$ is located above the sea surface. The important thing is only to have a common point, that can also be underwater.

- No real communication problems between the two cooperative robots are taken into account.
  The presence of water gives relevant issues in a real situation. A *full-duplex* communication (i.e. sharing data *at the same time*) is impossible. Also, in general, data exchange is much slower respect to the air. Some experiments in simulated environment with different methods of underwater communication are detailed in Simetti & Casalino (2017).
  However, these communication issues are considered by the cooperative scheme (as explained in section 2.5); in fact it permits to exchange very few information between the two carrying agents.

- The two robots firmly grasp the peg. There is no sliding caused by robot movements. This point is detailed in section 4.2.

- During the insertion phase, the control architecture tries to resolve alignment errors *only if* the peg is inside the hole. No methods are implemented to deal with the problem of *looking for* the hole on the surface. So, if the peg touches the external hole surface (due to a too big hole's pose estimation error), it will bounce back and forth forever.

- The force-torque sensor is positioned on the tip of the peg, and it provides the resultant force and the resultant torque of collisions on the whole peg.
  This would obviously not possible in real applications. In real scenario, the sensor is usually put on the arm's wrist and it provides forces and torques respect to this point.
  In this case, we could have projected the force-torque sensor information on the wrist frame, to make more realistic simulation, but for simplicity this is not implemented.
  Furthermore, both robots have access to the sensor data, at the same frequency,

and without uncertainties (except errors due to how almost all physics engines compute collisions, that it is done with approximations to improve the performance).

Others assumptions, more related to the vision part, are detailed in section 5.1.

4.5 Control Loop

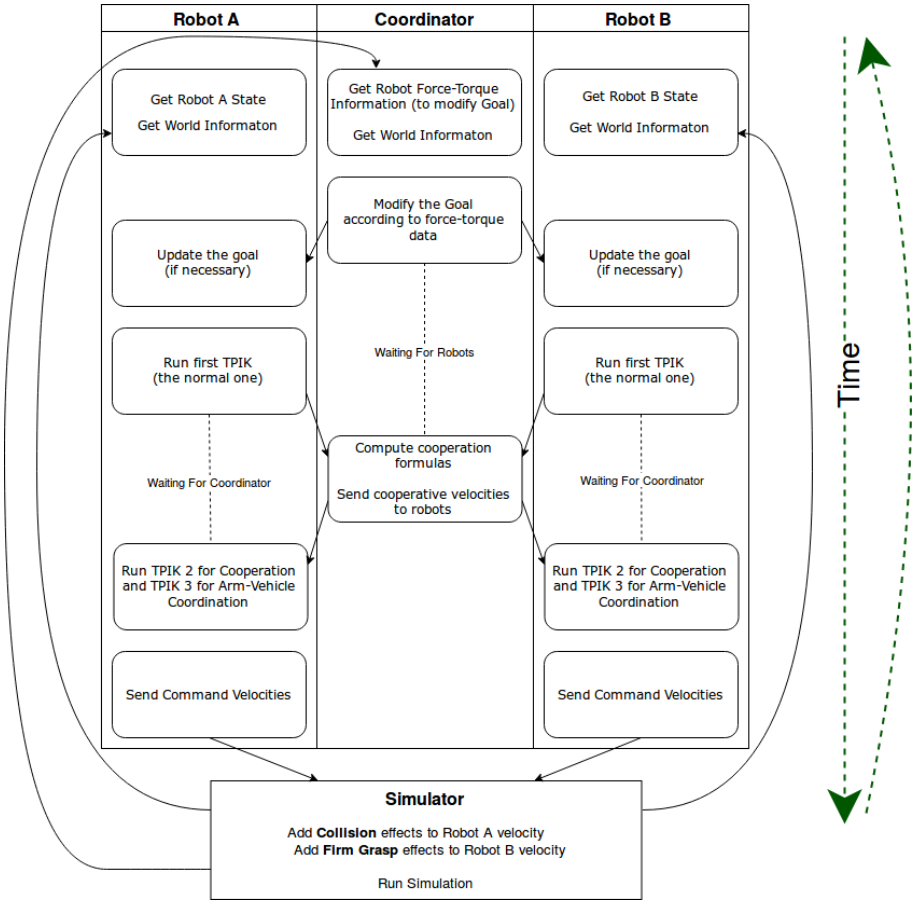


Figure 4.3: A flow scheme showing the main steps of a single control loop for the robots and for the coordinator. Blocks at the same horizontal level are executed at the same time. Arrows indicate sharing of data between blocks of different nodes. Note here that the Vision robot is not considered.

This section is written with the scope of giving a better idea on how the control architecture works.

The Vision robot, which job is to estimate the hole's pose, acts in a preliminary phase. It *tracks* the hole, thanks to stereo-cameras, and it sends the estimated pose to the Coordinator. Due to its nature, no complicated control is implemented for this agent: when the pose is sent, we simply move the robot away from the hole with keyboard (like a ROV) to not interfere with the insertion mission. This part is described in Chapter 5.
After the coordinator receive the hole's pose, it sends it together with a signal to the two carrying robots to make them begin the mission.

The two carrying robots are fully autonomous: as soon they get the hole pose, they proceed *without user intervention*. There are three nodes running at the same time: the two Robots (*A* and *B*) and the Coordinator. The latter is not a real *physical* agent: it is only a software routine. So, it can be physically inside a robot, from now on, the Robot A. In this way, communication issues (due to the underwater scenario) occur only between the two robots, and not among all the three nodes.

At the beginning of the mission, the Agent A, the Agent B and the Coordinator synchronize themselves, i.e. each one waits that the other two are ready. After this phase, the normal routine starts. In figure 4.3, the main instructions of the control loop are depicted.

- At the beginning of the control loop, each node gets the updated simulation state, e.g. pose of the robots, pose of the tool, information from force-torque sensor, and so on.
- The Coordinator, which (as said previously and without loss of generality) is a software routine inside the Robot A, modifies the goal's linear position (as explained in section 3.3), if some forces are detected. If the goal is updated, the two agents get this new information.
- In the third block's row, the Robots run the first TPIK procedure. Then, they send the necessary data to the Coordinator, which computes the cooperative velocities and sends them back to the robots. Finally, the two Agents run another two TPIK procedures, one for the cooperation (section 2.5) and the other for the vehicle-arm coordination (section 2.4).
- At the end, the two Agents send the system velocity vectors provided by TPIK to the simulation.
- Before sending the velocities to the "real" simulation, some disturbances must be added to the commanded system velocity vectors. For the Robot A, this means adding effects of collisions between the peg and the hole (section 4.3). Instead, for the Robot B, effects of the firm grasp constraint are added (section 4.2).

- After the simulator performs a step, the loop starts again.

It can be noticed that the two added physical interactions (collisions and firm grasping) are added only for one robot (A and B, respectively) and not for both.
This is done to not add simulation errors that could occur, and that would not happen in real scenario. For example, in real situation there are not two coincident pegs (as in this simulation) and so they can't really distinguish themselves. Putting the firm grasp constraint only on one robot helps to reduce disturbances that in real scenario are not present. Also, it is sufficient to fake a real firm grasping.
About the collisions, they affect, *directly*, only the first agent. In truth, they also affect the other one, *indirectly*, because the latter is *dragged* by the firm grasp constraint. So, practically, collisions affect the behaviours of both agents.
It is important to notice that these physical interactions do not hide control problems: if the control is setted badly, the whole mission fails (e.g. the two pegs diverge and/or compenetrate visibly with the hole).

Another thing to notice is that, when the Force-Torque objective (Section 3.1) is used, both Robots need the sensor data. Being the force-torque sensor only on Robot A, this means that additional communications between Robot A and Robot B are needed. As known, underwater data transmission is complicated and slow, and its amount should be kept as small as possible (from this problem it derives the used coordination policy).

An alternative to sharing force-torque data can be using another sensor on the Robot B. The problem following this direction could be that different sensors give not exactly same data. So, the two robots run each TPIK with different information. This alternative is not explored here.

Another solution can be simply to avoid using this objective: this would decrease the performance of the mission (as we will see later) but results are good anyway.

It must be noticed that also to update the goal additional information has to be exchanged between the two robots.

4.6 Results

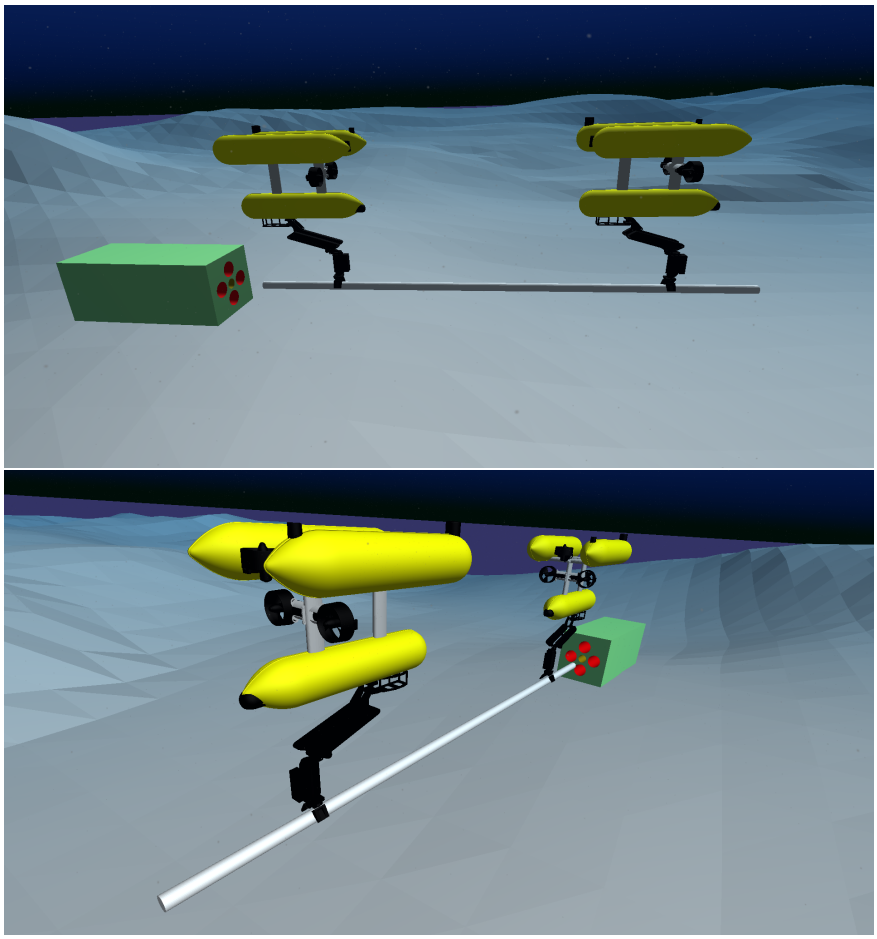


Figure 4.4: Two different points of view of the scenario for the results presented in this section. The pose estimation with vision is neglected here.

In this section, four different experiments are presented. An additional last experiment, comprehensive of the Vision part, is presented in section 4.7. Discussions and results analysis are given in section 4.8.

As said, in this section trials do not take into consideration the vision part. This is done to have a hole's pose error arbitrarily settable, that permits to discuss independently the performance of the control methods used.

The figure 4.4 shows the robots initial position. Below, some important details about the simulations are listed:

- The **Peg** is a six meters long cylinder, with a diameter of 0.10 meters.
- The **Hole** is a cylindrical cavity with a diameter of 0.14 meters. It is at the centre of a cuboid structure. In the figure 4.4, the hole is the yellow cavity between the four red circles (that are present only to aid the vision algorithms).
- The initial position of the agents is near the hole: the peg's tip is almost aligned perfectly to the hole, and it is at almost 0.44 meters from it.
  To be precise, the vector from peg's tip to the hole has components : $[0.441, -0.008, -0.018]$. The peg's tip frame has the x-component along the length of the peg; the y-component lies on the tip's surface and it points to the left; the z-component points downward (as in figure 4.5).
  The orientation from the peg's tip frame to the hole frame is described by these Euler angles: $[0, 0, 1.942]$ (*roll, pitch, yaw*, in degrees).
- The mission's aim is to drive the peg inside the hole with a depth of 0.2 meters.
- All the vectors displayed in the next plots are projected in the world frame (which orientation is visible in fig. 4.5).

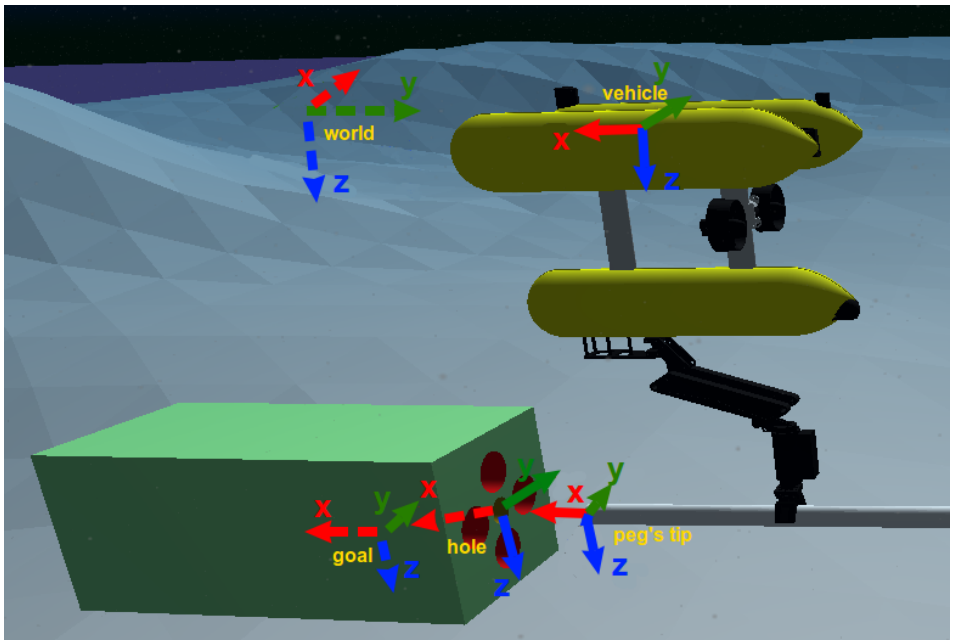


Figure 4.5: Detail of a screenshot where the main frames are drawn. The world frame is present only to clarify its orientation; its origin is not in that point. For the hole frame, the x-axis goes inside the cavity. The goal frame $\langle g \rangle$ has the same orientation of the hole frame, and it is shifted of 0.2 meters in the direction of hole's x-axis.

In the following results, only a few plots (fig. 4.14 and fig. 4.15) are related to the cooperation scheme of section 2.5, and no long discussions are made for them. This choice has been made because no robot has difficulty in tracking the ideal common tool velocity $\dot{\boldsymbol{x}}_t$. So, no very interesting plot occurs when *non-cooperative* and *cooperative* velocities are compared. Experimental results about this particular scheme used can be found in Simetti & Casalino (2017) and Wanderlingh (2018).

The focus of the results is on the implemented methods for helping the insertion phase: the Force-Torque objective (section 3.1) and the Change Goal routine (section 3.3). Further outcomes are presented about the added simulation procedures: the Firm Grasp constraint (section 4.2) and the Collision propagation (section 4.3). These last two extensions are important to improve the simulation in an otherwise pure-kinematic scenario.

4.6.1 Perfectly known Hole's Pose

In the first experiment, the hole's pose (and so the goal frame $\langle g \rangle$) are known without uncertainties. The plots of figure 4.6 show: how the forces and the torques act on the peg; the converging positional error from the goal to the peg's tip; the tool velocities generated by the collisions; the tool velocities caused by the firm grasp constraint.

Perfectly known Hole's pose

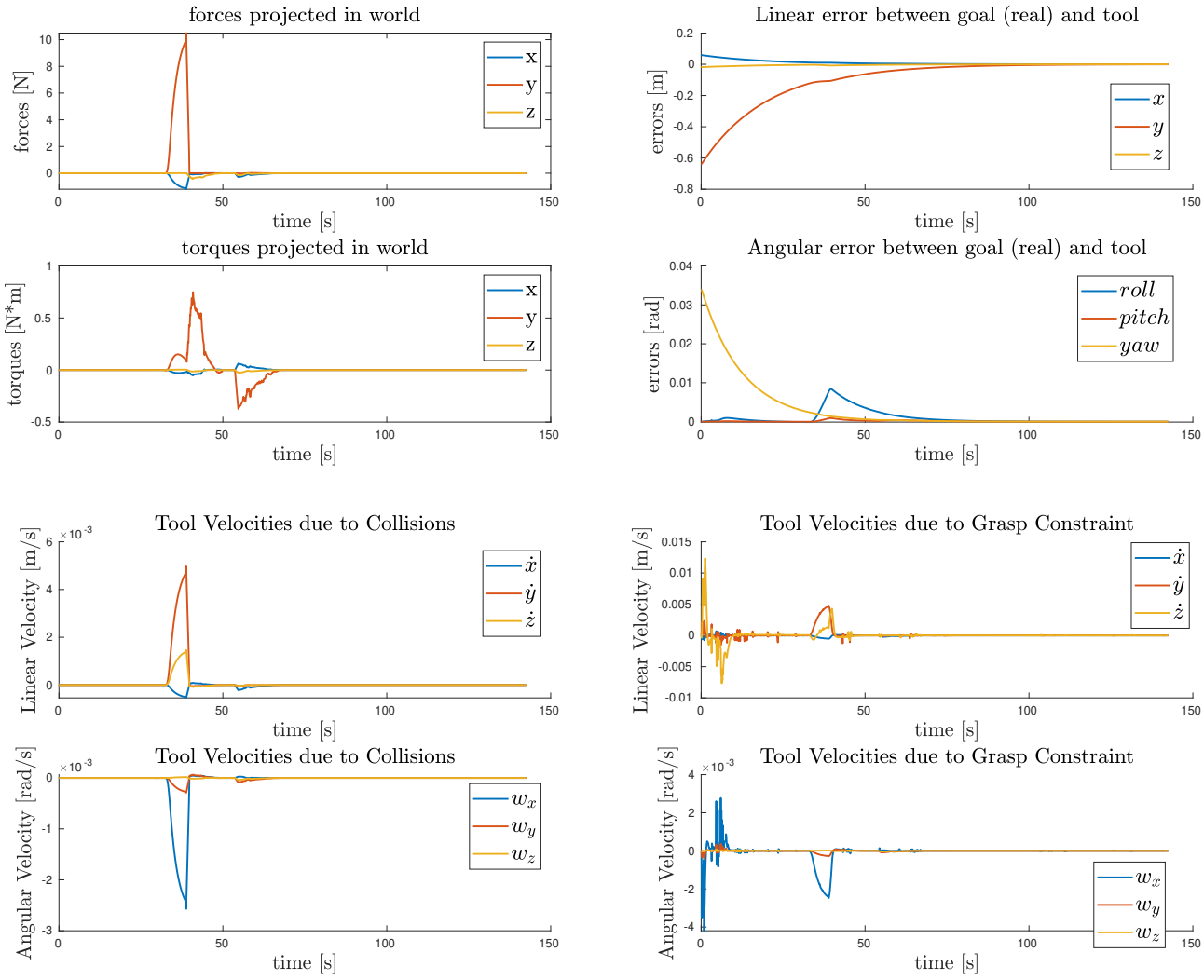


Figure 4.6: The results with the hole's pose known without errors. The upper left plot shows the forces and torques acting on the whole peg: they are the components of the total resultants projected on the world. The high peak in y is due to the first contact between the peg and the hole surface. The upper right plot shows the convergence of the error between goal frame and tip frame. The lower left plot displays the tool velocities generated by the system motions caused by the collisions propagation. The lower right plot shows the tool velocities generated by the Firm Grasp routine.

4.6.2 Error on the Hole's Pose

In general, a perfect pose estimation is never achievable, so the control should take into account that errors can be present. In this experiment, an error of 0.015 meter is added along the x-component of the goal (considering the goal projected in the world frame). So, the peg is driven a bit on the right respect to the centre of the hole, causing a lot of collisions with the right side of the cavity.

Three different experiments with the given error have been conducted. In the first, the control does not use any method to exploit the force-torque data (as in the previous experiment of section 4.6.1). In the second, the Change Goal routine (described in section 3.3) is used to try to correct the pose error. In the third, both the Change Goal routine and the Force-Torque objective (described in section 3.1) are included. The addition of the new objective in the TPIK list is done to try to reduce the amount of force and torques acting on the peg.

4.6.2.1 Change Goal Routine Results

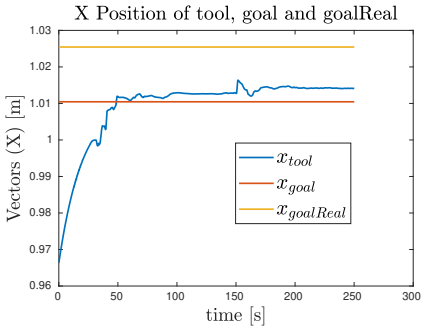
As explained in section 3.3, it has been implemented a routine to move the goal's origin according to the forces and torques detected by the sensor. A comparison of the results with and without this method is visible in figure 4.7. It can be seen that the goal is changing, and at the end of the experiment the added error is compensated.

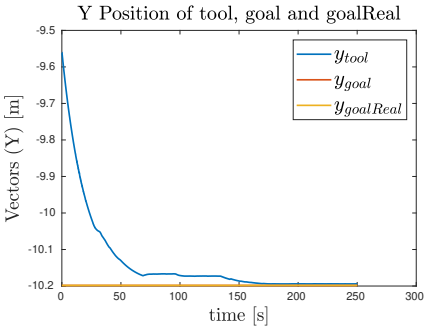
4.6.2.2 Force-Torque Objective Results

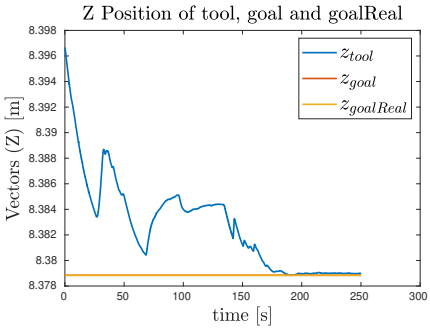
Besides changing the goal, it is useful to exploit the force-torque sensor also at kinematic level, using the provided information in the TPIK approach. The new added objective is described in section 3.1.
In figure 4.8 the results of the three different experiments are compared. It is visible that, when also the new objective is used, the forces and the torques have the smallest peaks. Meanwhile, the convergence of the error between the goal and the peg's tip is maintained as good as in the second experiment thanks to the presence of the same Change Goal routine.
In figures 4.9 and 4.10 details on how this new objective works are shown. When some collisions happen, the reference and the activation grow to make the tool move in a way to reduce the force and the torque magnitudes. The figure 4.10 shows the velocities generated by the objective *as if it was the only one* in the TPIK list, so they are not the real velocities given to the system.

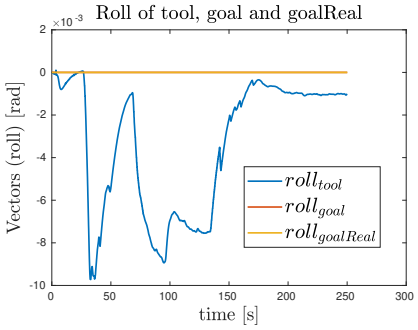
Error of 0.015 meter on x-axis of the goal
Linear and Angular component of the goal frame and the tool's tip frame

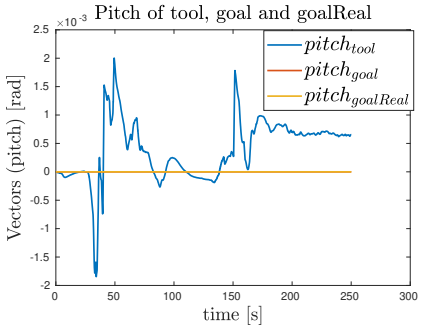
Without Change Goal routine

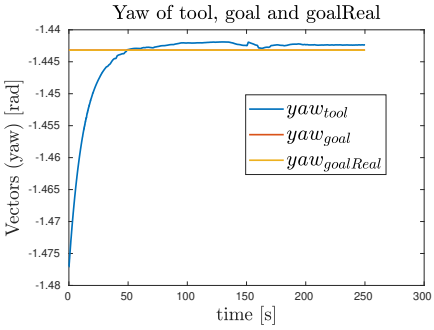




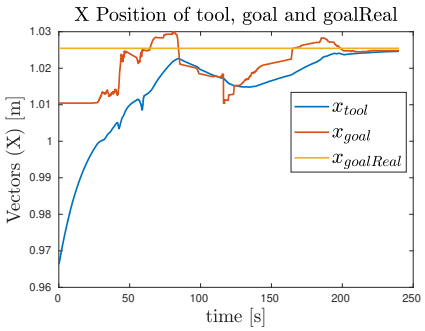


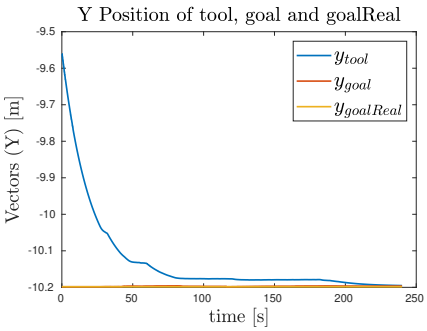


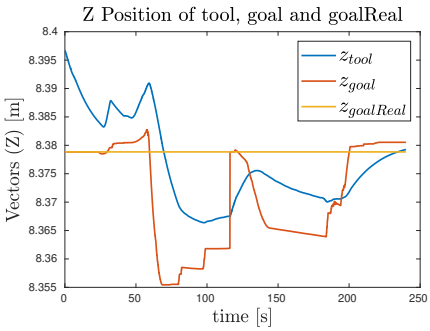


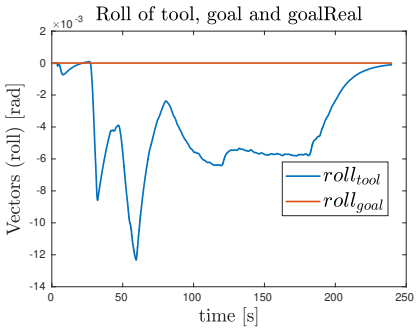


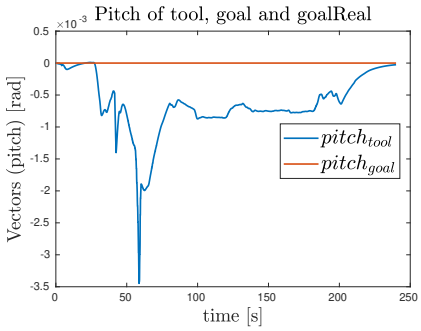
With Change Goal routine











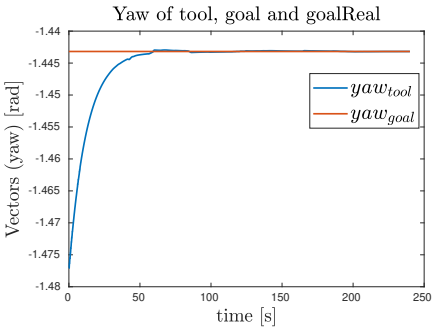
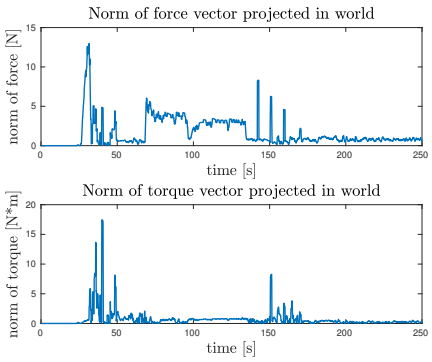


Figure 4.7: Results of the two experiments without and with the Change Goal routine (both without the Force-Torque objective). The plots show the pose of the goal frame $\langle t \rangle$ and of the tool's tip frame divided into their linear and angular components. The upper six plots show results without the Change Goal routine, the bottom ones show results with the routine. All the components are respect to the world frame. For the linear part, the yellow lines represent the position of the goal without errors. The red lines represent the position of the goal that the controller uses. Please note that in some plots the red and yellow lines are coincident because the component is know without errors and *it does not change. It is clearly visible that, when the goal is modified, the red lines go toward the yellow ones, correcting the initial hole's pose error.*

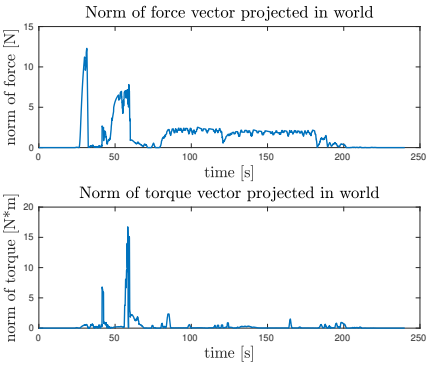
Error of 0.015 meter on x-axis of the goal

Norm of the forces and torques acting on the peg

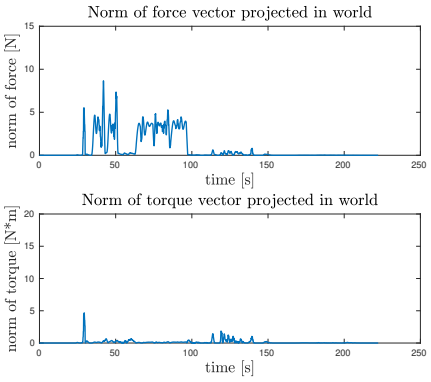
Without Change Goal routine



With Change Goal routine

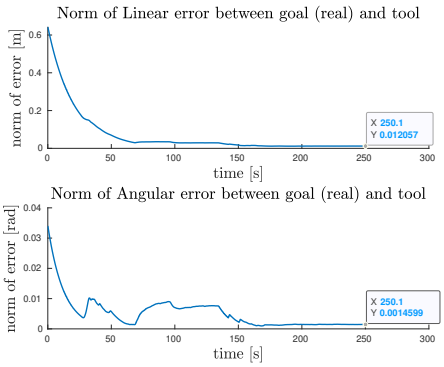


With Change Goal and Force Task

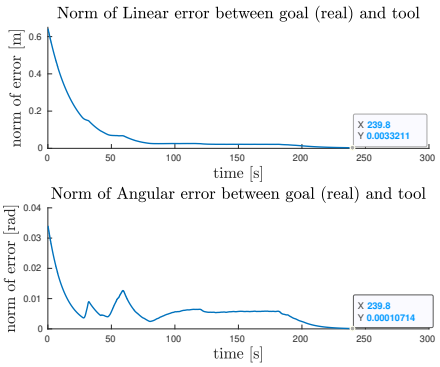


Norm of the error between ideal goal (without the added error) and tool's tip

Without Change Goal routine



With Change Goal routine



With Change Goal and Force Task

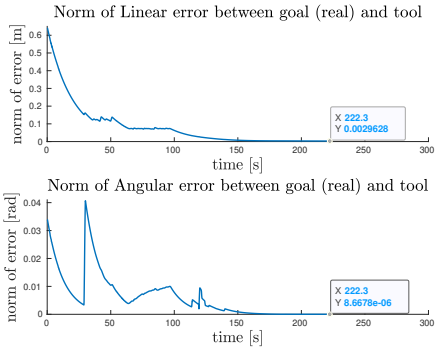


Figure 4.8: Comparison of results of the three methods: vanilla, Change Goal and Change Goal with Force-Torque objective. The three upper plots show the norm of the force and the norm of the torque acting on the peg. It can be noticed that, in the cases where the goal is modified, at the end their norms goes to zero. The three lower plots show the norms of the error between goal and tool's tip. In the case without the Change Goal routine, the norms converge anyway, but to a slightly bigger value than the one of the other two cases. In fact, when the Change Goal routine is used, the initial hole's pose error tends to be corrected.

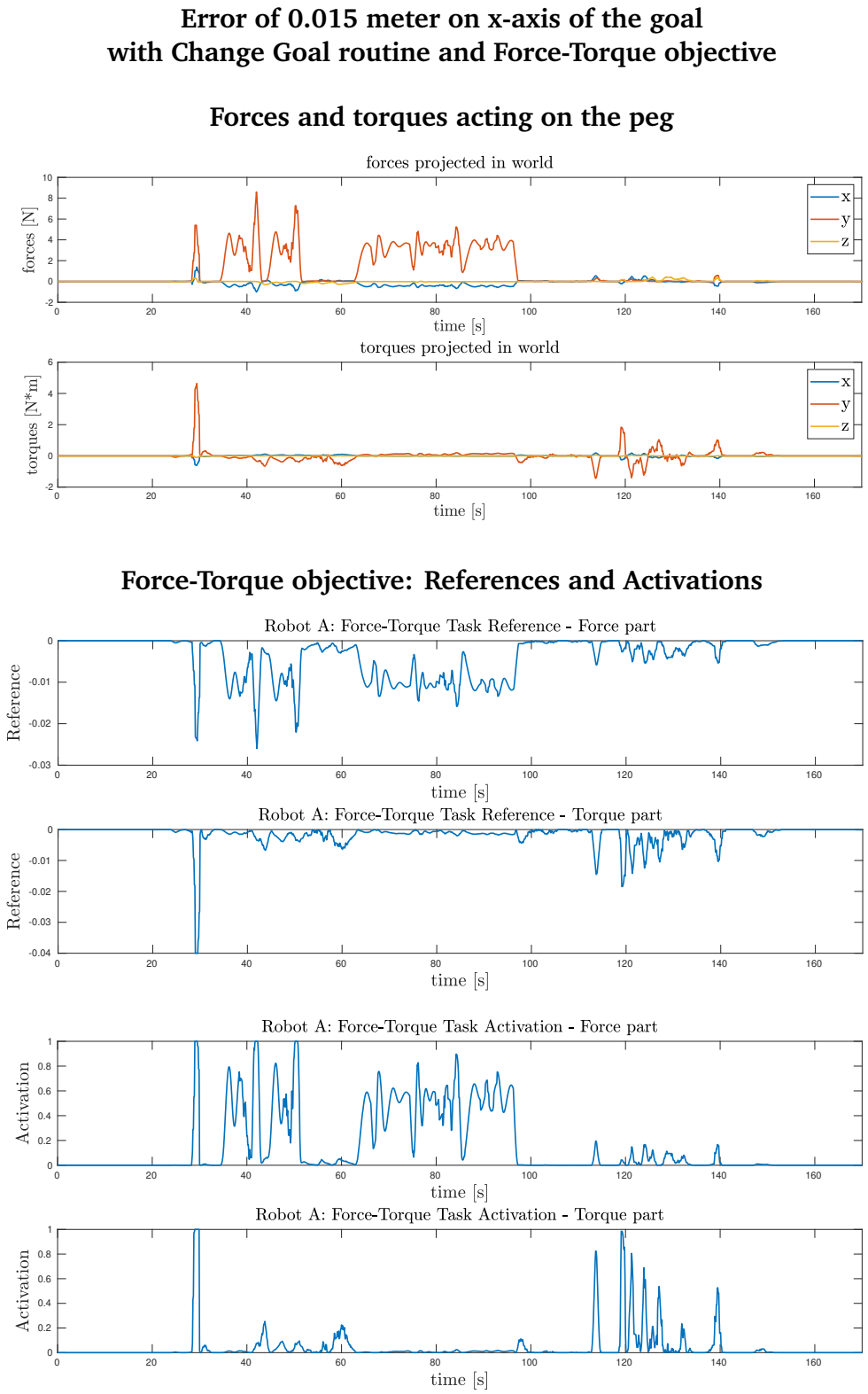


Figure 4.9: The forces and torques acting on the peg (above), and the corresponding generated references and activations of the Force-Torque objective (below); they are calculated by robot A, but for robot B they are the same.

Results with error of 0.015 meter on x-axis of the goal with Change Goal routine and Force-Torque objective

Velocities generated by the Force-Torque objective only

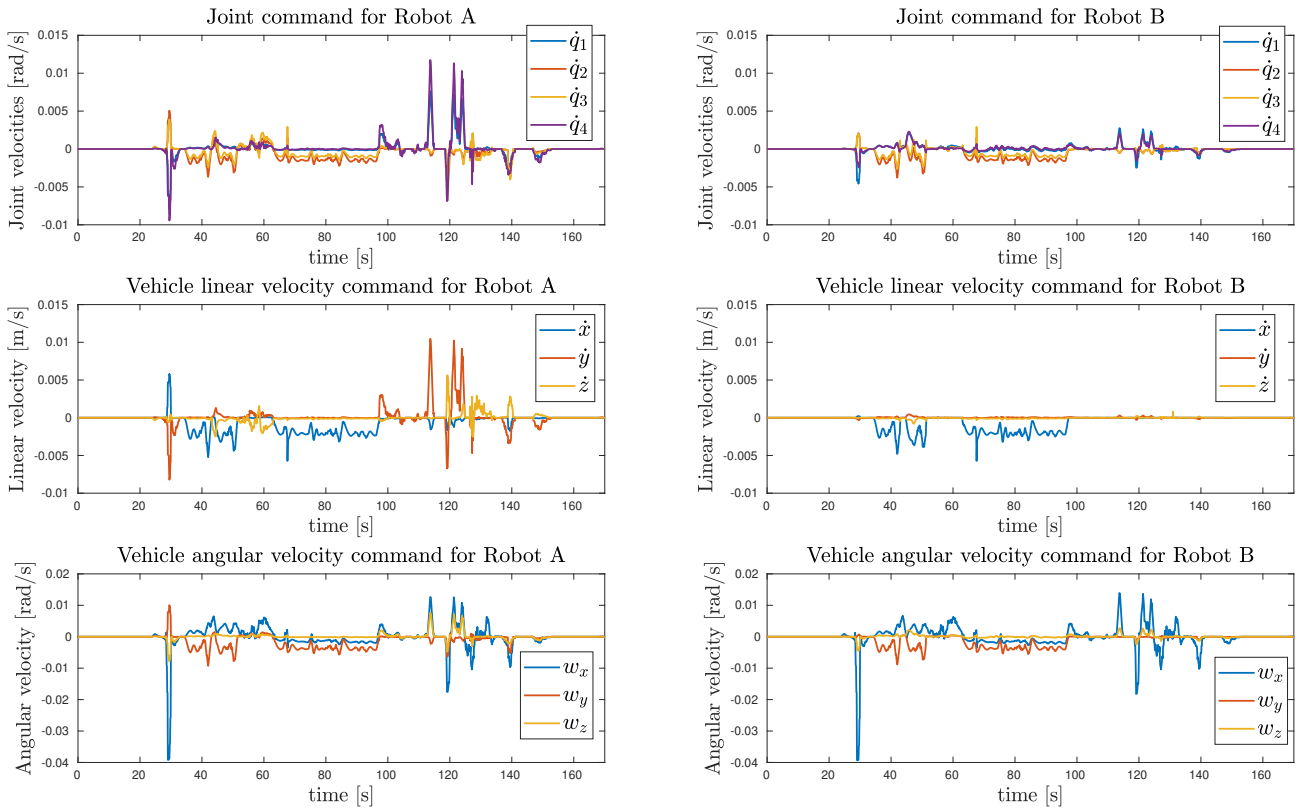


Figure 4.10: The velocity command generated by the Force-Torque objective, for the Robot A. These are the velocities that the task generates; the vectors depicted are simply the result of $\boldsymbol{J}^{\#}_{ft}\ \dot{\boldsymbol{x}}_{ft}$ to show how the objective works. So they are not the real one applied to the system because with this formula higher priority objectives are not taken into consideration. For the two robots, the reference $\dot{\boldsymbol{x}}_{ft}$ and the activation $\boldsymbol{A}_{ft}$ (visible in figure 4.9) are the same because they act with the same data; it is the Jacobian $\boldsymbol{J}^{\#}_{ft}$ which is obviously different and which makes the two plots dissimilar.

4.7 Results with the Hole's Pose Estimation by Vision

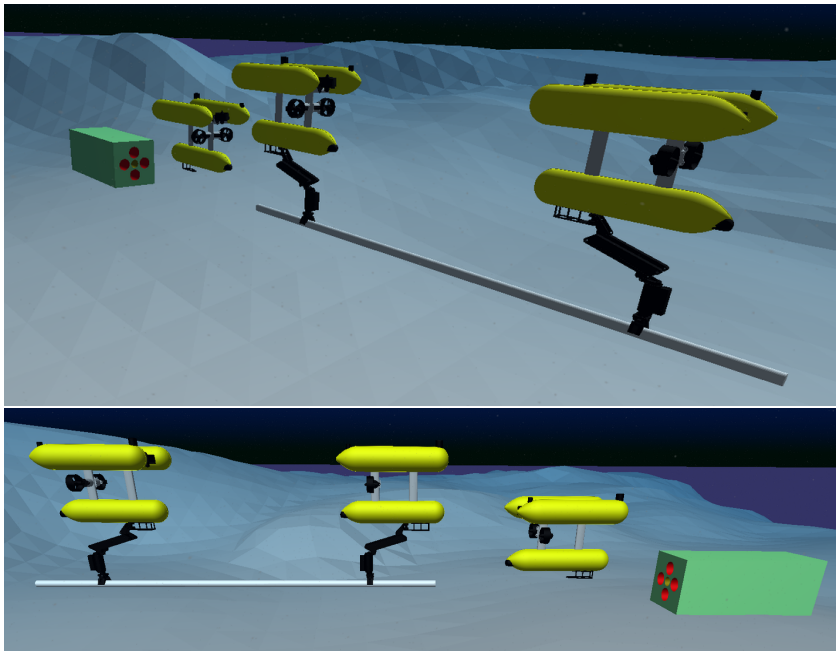


Figure 4.11: The starting position of the robots for the final experiment, where it is used the preliminary phase to estimate the hole's pose with vision.

In this section, results with the preliminary vision phase are presented.
Differently from the previous experiments of section 4.6.2, the error of the hole's pose when using the Vision robot is not so influential on the mission. In norm, the pose estimation error is less than 0.006 meters for the linear part and 0.01 radians for the angular part (with the best method, as shown in figure 5.6 of section 5.4.4). However, the experiment is interesting not only because it puts all the mission phases together, but also because it shows the outcome when the error is "spread" among all linear and, especially, angular component (which was not considered before). Even more, the carrying robots start farther from the hole than before. In this simulation, both the Change Goal routine and the Force-Torque objective are used.
The test's details described at the beginning of section 4.6 are still valid, except for the initial position of the two carrying robot. This time, the distance between hole and peg's tip has component $[3.590, 0.039, -0.041]$ for the linear part (and the same as previous for the angular part: $[0, 0, 1.942]$ *roll, pitch, yaw*, in degrees).
Some screenshots that show the main phases of the simulation are visible in figure 4.12. The interesting plots about the performances are shown in figure 4.13. To give an idea of the magnitude of the velocities involved, cooperative system velocities $\dot{\hat{y}}_a$

and $\dot{\hat{\boldsymbol{y}}}_b$, and cooperative tool velocity $\dot{\hat{\boldsymbol{x}}}_t$ are displayed in figure 4.14 and figure 4.15. Please note that these velocities do not include the collision propagation and the firm grasp constraint, they are only the output of the kinematic layer.

Screenshots from the final experiment

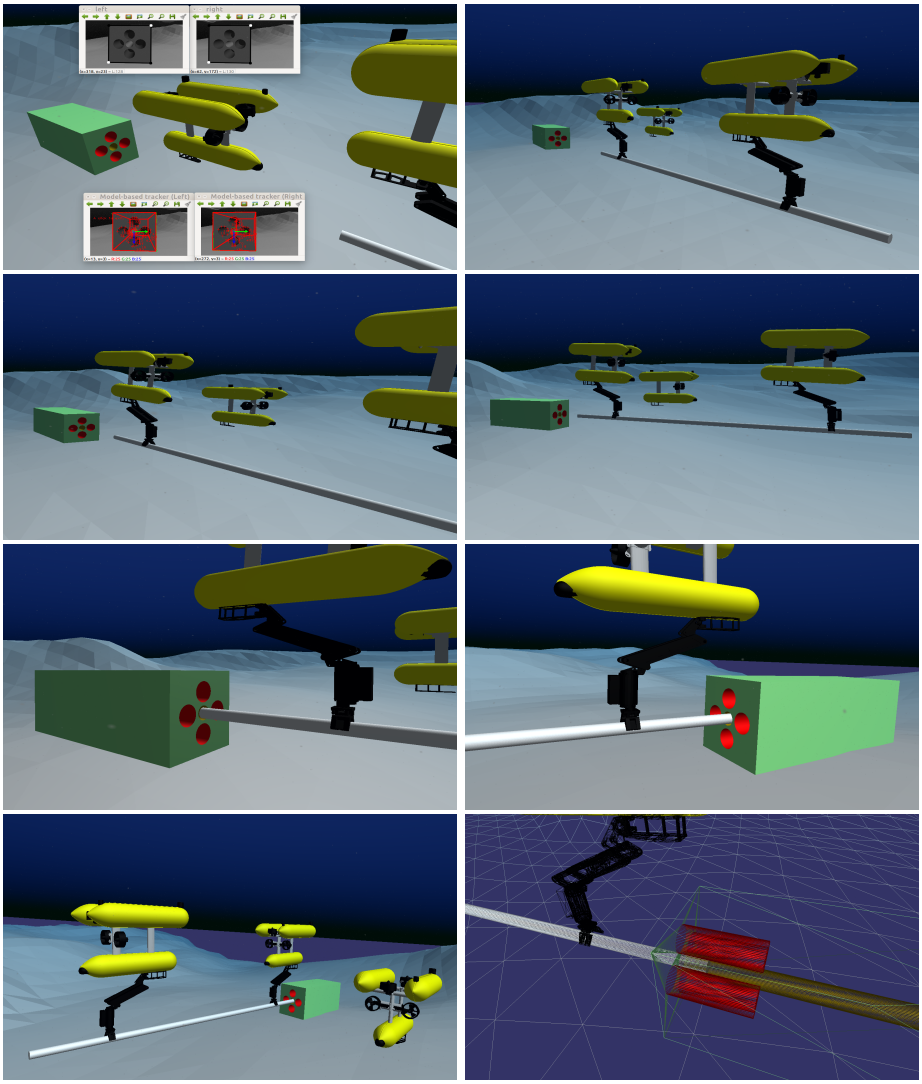


Figure 4.12: Screenshots from the final experiment with Vision. From the top to the bottom, from the left to the right: a) The Vision robot has detected the squares and it is tracking the hole's pose; b) The Vision robot is driven away and the carrying robots are ready to begin; c,d) the two robots are cooperatively transporting the peg to the hole; e) The first "contact" between the peg and the hole; f) The peg is being inserted; g) The robots stop because the tool has reached the desired depth (0.2m); h) A polygon wire-frame view mode of the simulator to see the peg inserted.

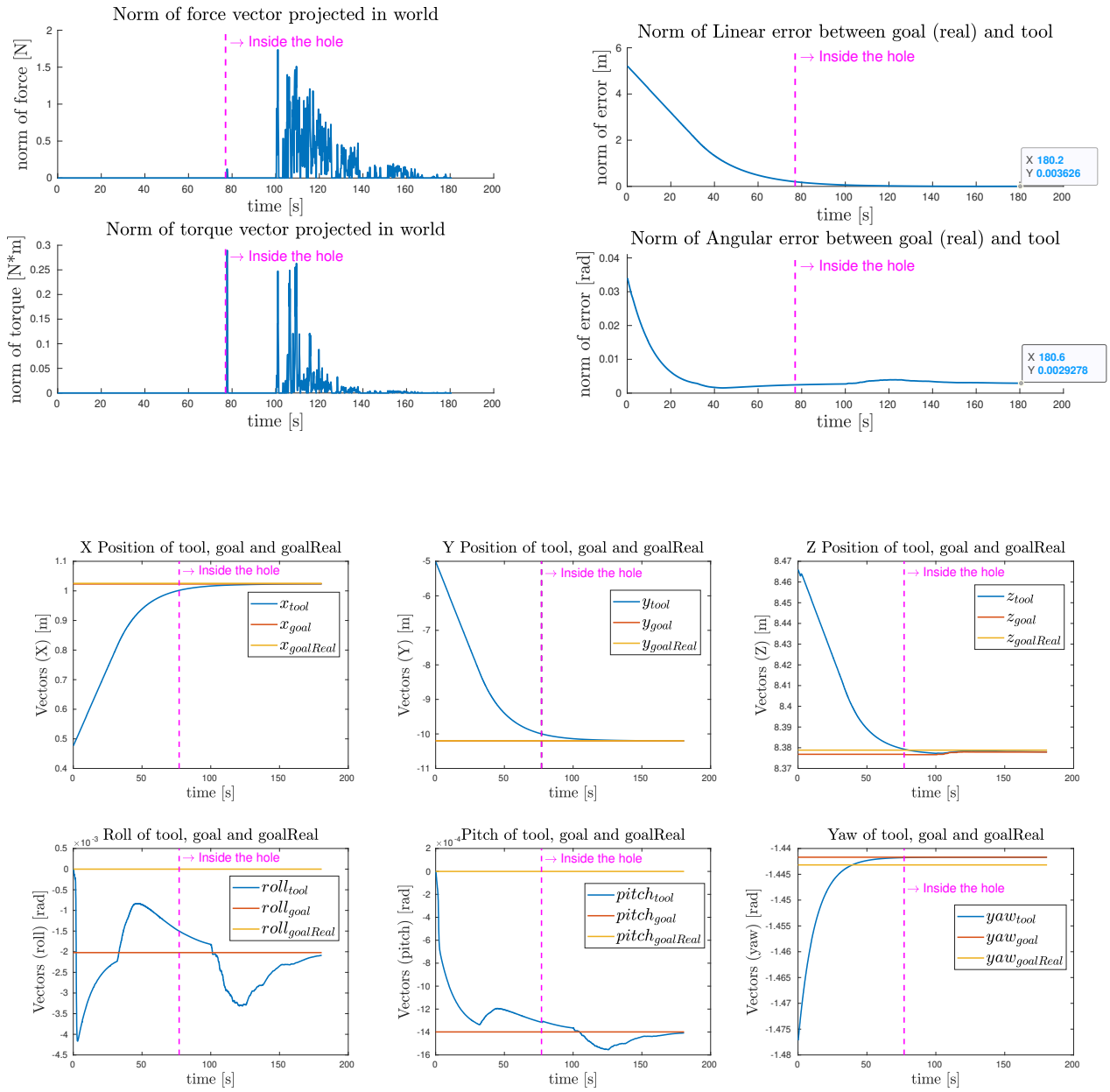


Figure 4.13: Results with the hole's pose estimated by the best one of the tested vision algorithms. At second 77 the peg's tip goes inside the hole (magenta vertical lines). Being the pose estimation really good, forces and torques are not so big as in the previous tests (upper left plot). Furthermore, modification of the goal are almost not noticeable (lower plot). The positional error from goal to peg's tip converges to a small value (upper right plot). The visible error for the angular part is due to the fact that the orientation of the goal frame has some imprecisions, due to not perfect hole's pose estimation.

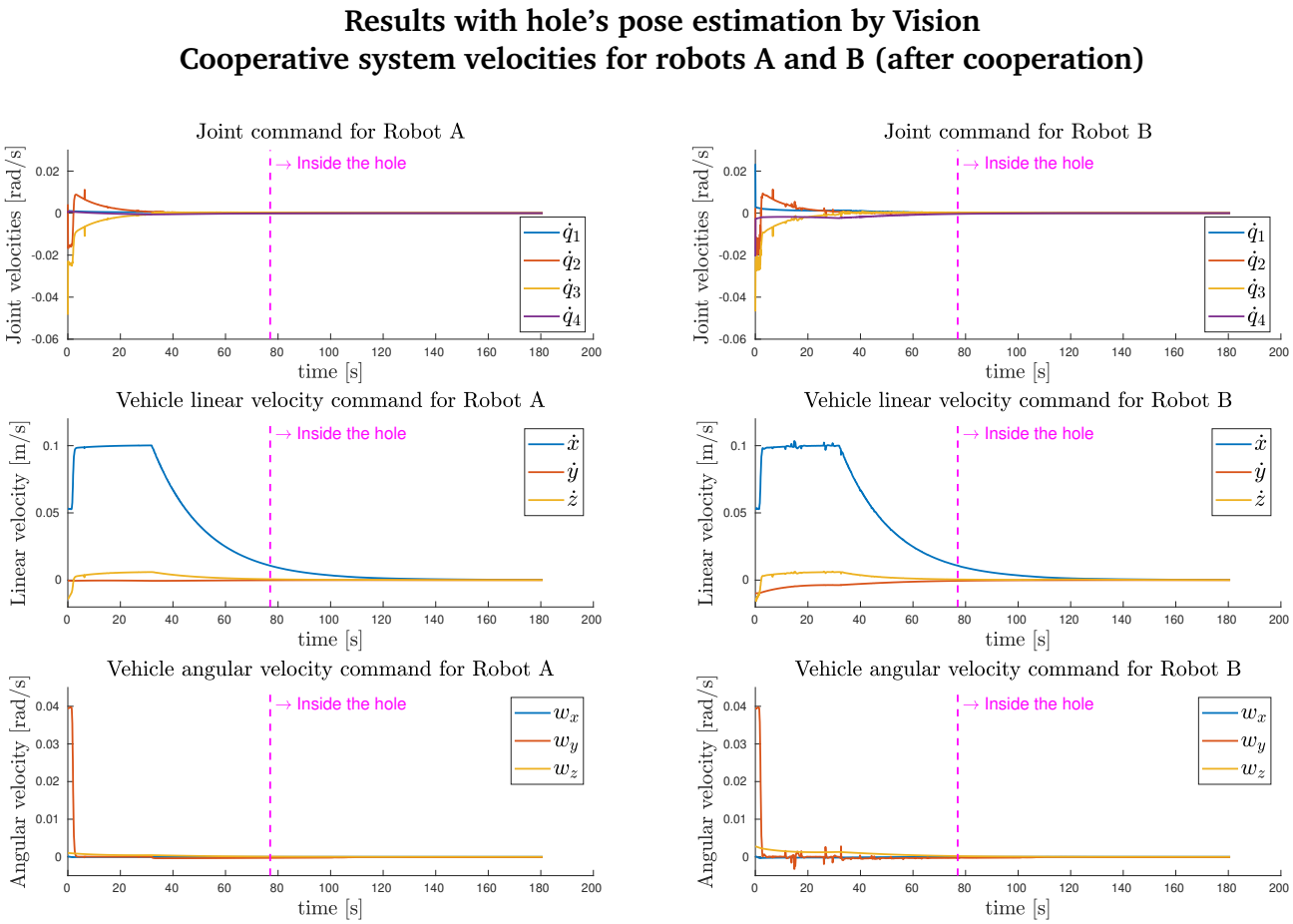


Figure 4.14: The system velocities $\dot{\hat{\mathbf{y}}}_a$ and $\dot{\hat{\mathbf{y}}}_b$, outputs of the kinematic layer after the cooperation policy (section 2.5) and the arm-vehicle coordination scheme (section 2.4). These are not always the real *robot velocities because they do not include velocities caused by collisions (for robot A) and by firm grasp constraint (for robot B).*

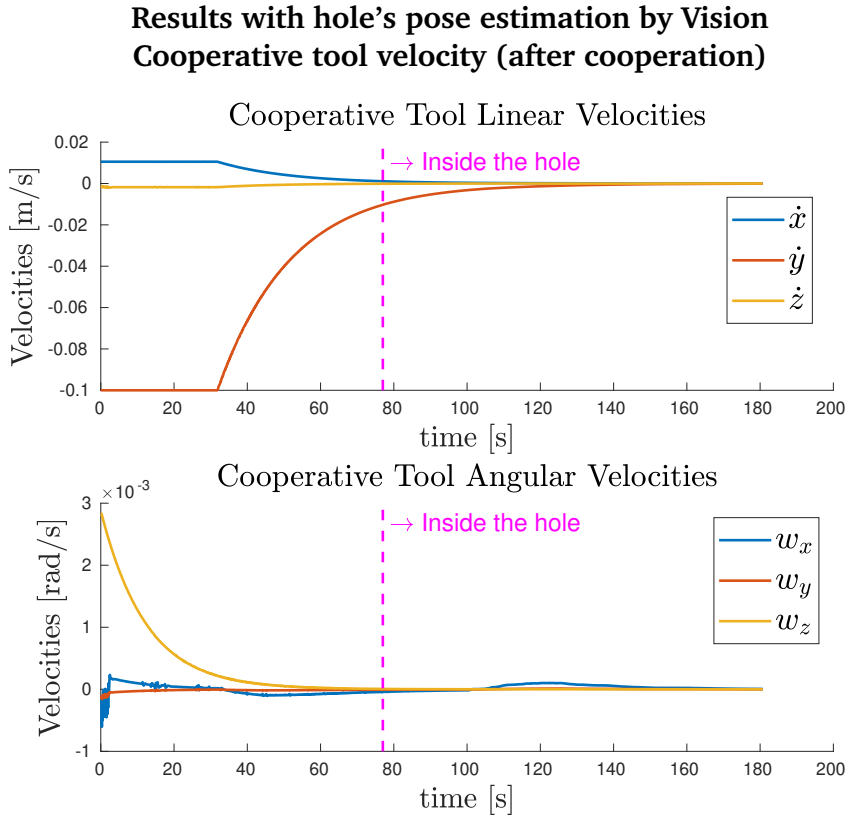


Figure 4.15: The tool velocity $\dot{\tilde{\boldsymbol{x}}}_t$, the one that the coordinator sends to both robots during the coordination policy. It is the feasible *one that* both *robots provide to the tool (section 2.5), and it is caused by the system velocities of figure 4.14. So, it is not always the* real *velocity that the tool has because collisions and firm grasp constraint are not included.*

4.8 Results Discussion

In this section, considerations about the results shown in the previous pages are made.

4.8.1 Perfectly Known Hole's Pose Discussion

When the goal frame is known without error, the results are good even without the Force-Torque objective and the Change Goal routine.

This is clearly visible in figure 4.6. Despite no presence of errors, some collisions happen anyway. This is due to the fact that the peg is driven directly toward the goal (inside the hole), without considering that there is the hole structure. Anyway, the forces and torques acting on the peg "help" the insertion phase because they drive *naturally* the tool inside the hole. The peaks of the magnitudes are due to the first contact with the hole: even if the pose is perfect, this does not mean that a perfect alignment is present *before* reaching the goal frame.

The lower left plot of figure 4.6 shows the tool velocities caused by the collisions, and in the lower right one we can see that the second tool "follow" the velocities of the first one, due to the firm grasp constraint (around second 40). In other words, it is as if the robot B would be dragged by motions causes by collisions on robot A. In the same plot, the chaotic velocities caused at the beginning are due to a not-perfectly coincident initial position of the two tools, that is impossible to achieve due to numerical errors of the simulation. Anyway, after a while they go to zero, so this initial error does not affect the rest of the experiment.

The upper right plot of figure 4.6 shows how the error converges along its components. The small peak in roll is caused by the initial collisions with the hole.

Even if a perfect goal pose is obviously impossible to have, this experiment is useful to understand how the peg is inserted if ideal conditions are present. This is important: from now we know that, in any case, the forces and torques acting on the peg may help *naturally* the insertion phase.

4.8.2 Change Goal Routine Discussion

When an error in the goal pose is present, the results obviously get worse. The first noticeable thing is in figure 4.8: due to the error the collisions are more present during the insertion (first plot). An interesting thing is that the peg is inserted at the wanted depth anyway (as can be seen in the right lower plot, where the norm converges to a small value). An issue is that the forces are not nullified at the end, which means that continuous pressure between the peg and the hole is present.

The first idea to reduce the number of the collisions is to introduce the routine which change the goal frame. From the plots in the middle of figure 4.8, we see that the overall behaviour is improved: the forces and torques have a littler norm in general (upper plot) and the final position of the tool is more precise (lower plot).

In fact, from figure 4.7, the six lower plots show us that the error in x is corrected by the routine. It can be noticed that also the z component is modified, even if it has no initial error. This is caused by the fact that the routine is active: so, all the components are modified in the direction of the detected forces.
Obviously, we can't assume to have an error only along a single axis, so we can't modify only the axis where we know there is the error (x-axis in this case).
In truth, also the y-component is shifted, even if the change is very little because the component of the force acting along the length of the peg is neglected to not modify the wanted depth of insertion (as explained in 3.3).

From the same figure 4.7, another thing to notice is that the routine does not correct perfectly the error in all the components. In fact the goal along the z-component is a bit erroneous at the end of the experiment. This happens because the peg has reached a point where the forces are zero, and so no more modifications can be done. This is caused by the fact that the peg has a smaller diameter than the hole, and so there is a tiny tolerance zone inside the cavity where collisions do not happen.

4.8.3 Force-Torque Objective Discussion

In general, the goal changes slowly: this means that, especially when the first contact happens, the forces and torques magnitudes are big as in the vanilla case.
The Force-Torque objective implemented helps to reduce these peaks: as soon as a force (or torque) is detected, the objective responds to it and generates a reference velocity that moves the peg away from the walls of the hole. The reduction of the magnitudes is visible in the upper plots of figure 4.8.

In the lower plots of the same figure 4.8, it is shown that the positional error between the goal and the tool is not affected so much by this objective. In fact this plot is similar to the one of the second experiment. In truth, the convergence of the norm to a small value is a bit slower than the one of the case where this objective is not used.
The first factor for this is that now the collisions are considered by the kinematic layer. So, being the new objective at an higher priority respect to the reaching goal objective, the kinematic control tends *first* to nullify the forces and torques and *then* to drive the tool toward the goal. This, sometimes, could increase the time to accomplish the mission (as in this case). Anyway, some other times it can *decrease* the mission time because less stalemates happen, thanks to the fact that this objective may help to drive the peg away from the collision. In the second plot of figure 4.8, we can see that the norm of the force stays at a nearly constant value around the

interval $80s - 180s$, as well as the norm of the error in the plot below. This shows a situation of stalemate that is then solved. Presence of standoffs like this one are discussed later.
A second factor is simply that, being the particularity of the problem, each experiment is different from the others, so sometimes the mission is "lucky" and it has less difficulties.

Figure 4.9 shows how the new introduced objective generates references and activations in correspondence of the forces and the torques detected. Thus, obviously, the references and activations shapes (lower four plots) are similar to the forces and torques shape (upper two plots). In this case they are even more similar because the norm (which is the quantity that the objective controls) is given by almost only one component, the y one.
The activation is a function which assumes only values from 0 to 1, so it is always positive. The reference instead, has opposite sign respect to the forces and the torques. This is because the objective wants to *nullify* the forces and the torques, so it provides a velocity in the opposite direction.
In the reported plots reference and activation are the ones calculated by the robot A. However, the agents receive the same data from the sensor (except small synchronization problems that can happen), so the reference $\dot{\boldsymbol{x}}_{ft}$ and the activation $\boldsymbol{A}_{ft}$ are the same for the robot B.

Figure 4.10 shows the results of $\boldsymbol{J}_{ft}^{\#}\,\dot{\boldsymbol{x}}_{ft}$, for both robots. These are the system velocities that we would give if the Force Torque objective was the only one in the TPIK list, without considering the activation, and a simple pseudoinverse for the Jacobian was used (without any regularization). So, neither the collision propagation, the firm grasp constraint, nor the cooperation are included.
As explained before, the reference $\dot{\boldsymbol{x}}_{ft}$ is the same vector for both agents. The thing that makes the two velocities so dissimilar, is the Jacobian $\boldsymbol{J}_{ft}$, which is obviously different for each agent because they are not in the same configuration. This difference, at this point, it is not a problem for the cooperation because we have still to deal with the coordination policy.

The plots of figure 4.10 are shown only to point out that the velocities are not so big to be not feasible. In truth, at least for the vehicle part, they could be even too small to be followed well. This can be solved making the objective to control only the arm, or by increasing the gains (but taking into account that bigger gains would mean greater risks of bad behaviours).
Another thing we can see is that they are not so smooth, and they have a lot of fast changes. However, firstly we have to consider that the activation, which the main

work is smoothing the behaviour, is not present in the $\boldsymbol{J}_{ft}^{\#}\,\dot{\boldsymbol{x}}_{ft}$ formula plotted.
Then, magnitudes are so little (in the $0.01 rad/s$ order for joints, and in the $0.01 m/s$ and $0.01 rad/s$ order for vehicle) that these fast changes are not so important.
Last, we can't do so much because these changes are given by external data (the forces and the torques) that we have to deal with.

4.8.4 Discussion About the Experiment with the Vision Part

The final test of section 4.7 (which includes the Vision part) is interesting for two main reasons. The first one is that the peg does not start so near the hole as in the previous trials (but it is almost aligned to it as before). So, we can see that the transportation phase is done in a good manner by the cooperative robots. This is noticeable even if no difficulties (e.g. difficult trajectories, an obstacle, a joint limit, and no vehicle and arm dynamics) are encountered.
The second reason is that the hole's pose has also some error on the angular components. This affect the orientation of the goal frame where the peg is driven to. Anyway, this error is small and it does not influence too much the insertion.

In fact, as we can see in the upper left plot of figure 4.13, the norm of the force and the norm of the torque are smaller than the ones of the previous experiment. Further, the error converge smoothly (upper right plot); the bigger error in the angular norm is obviously due to the fact that now there are also some imprecisions in the orientation of the hole. Also, little modifications are done to the frame goal, because its origin is almost with no error (lower plots).

Figure 4.14 shows some plots about the cooperation. No big difficulties are met by the robots during the transportation and the insertion. So, the system velocities, that are the outputs of the final TPIK procedure (after the cooperation policy and the arm-vehicle coordination), are similar between the two robots. It must be remembered that these velocities are not the applied ones because the disturbances caused by collisions and firm grasp are not included.
However, for the robot B (right plot) we can see that tiny peaks are present around the interval $20s - 40s$. These can be caused *indirectly* by the firm grasp constraint. When the tool of the robot B is driven away a bit, in the next control loop the kinematic layer must do a little more effort in recovering the trajectory, causing these tiny peaks. Being the cooperation policy included, if these peaks are high (as around second 8) the robot A helps the other one, in fact a little peak is present also for the latter.

The last figure 4.15 shows the *feasible cooperative* tool velocity $\dot{\tilde{\boldsymbol{x}}}_t$, that is the one

that the coordinator sends to the two agents, after assuring that is achievable by both (as explained in section 2.5).
The *non-cooperative* velocities $\dot{\boldsymbol{x}}_{t,a}$ and $\dot{\boldsymbol{x}}_{t,b}$ are not shown because they are very similar to the cooperative one. This is caused by the fact that no robot has difficulties in providing the *ideal* tool velocity.
Even if disturbances of collisions and of firm grasp constraint are not present, this plot gives an idea about the tool's speed during the mission. The object is driven slowly because the mission needs so: the insertion phase, made by two kinematic cooperative manipulators, puts big challenges and we can't afford to have too big gains.

4.8.5 Standoff Discussion

A last thing to report is that some experiments meet a standoff situation at some point during the insertion. This happens when the collisions continuously make the peg "bounce" on the inner hole walls, making it moving back and forth. The "bounce" is due to a bad peg alignment inside the hole, so the tip continuously touches the inner wall.

In these cases, sometimes, the methods used do not manage to solve the stalemate in a reasonable time. The problem can be due to different factors.
One could be that the simulation, being only kinematic, is not precise. So, even if the gains are really small, in any case the velocities are instantaneously provided to the system, which causes always a bit of chattering. This complicates the work of the simulator (to calculate the collisions) and of the whole mission (that could meet almost instantaneously a strong force or torque).
Another problem could be a not so good setting of the many gains that we have to put.
Other one can be given by how Collision Propagation work. Collisions propagate on the system through Jacobians, that are mappings derived from linear approximations of non-linear things.
Other one can be that, with the Change Goal routine, orientation's modifications are not made, so no corrections are doable for the angular part.
Additional problem to take into account is that some synchronization issues may happen. The coordination policy, in the way it is implemented, assures the synchronization (section 4.5), but we have also other kinds of exchanged information. For example, the force-torque sensor data is shared using ROS topics in a simple way, so with the actual software we can't know exactly when information arrives to each node. Considering also that on a single machine we run the simulation and the two

robot software (plus the coordinator) this could cause even more synchronization problems. This may be solved with further code improvement, but it would go out of the scope of this work and, also, it would not be so useful for the real application (where we don't run everything on a single machine).

The standoff problem is more influential when the hole's pose error is bigger, or, for example, when we have also orientation error such as in the last experiment. However, the methods show a good starting point to improve the current state of the art in this unexplored problem underwater.

Chapter 5

Vision: Methods & Results

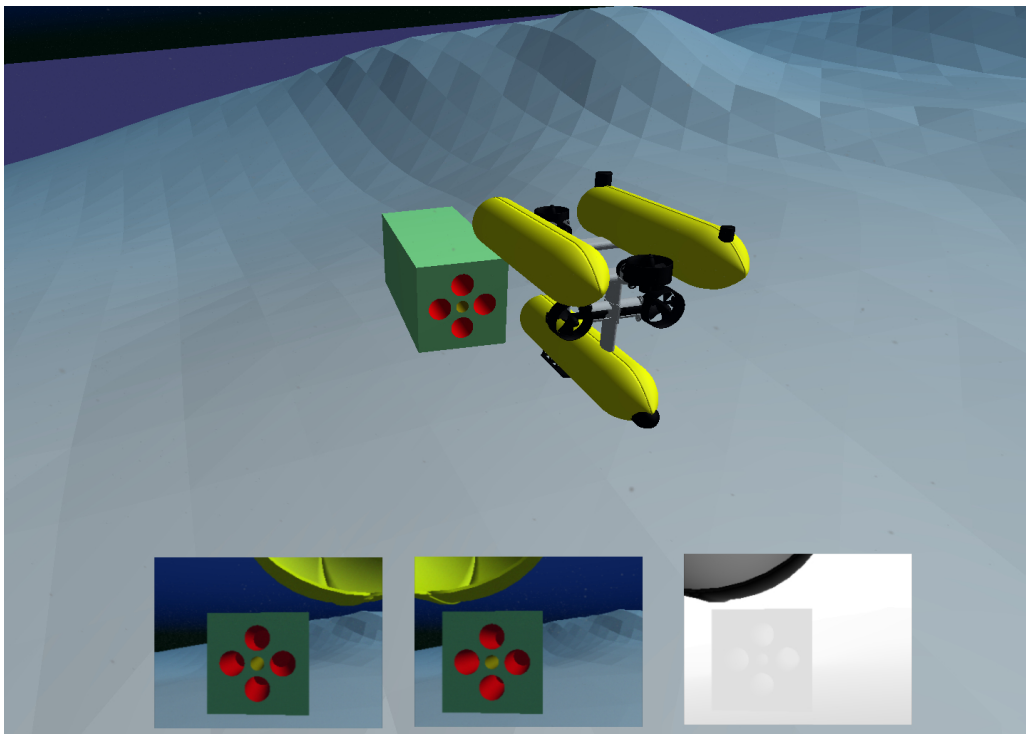


Figure 5.1: The vision Robot watching the hole. The hole (in yellow) is in the centre of the cuboid. The red holes around are there to help vision algorithms. Below, the right RGB, the left RGB and the left depth cameras are shown. In the methods, only one of the left cameras is used.

This Chapter covers exclusively the Vision part. So, here they are presented methods and tools to estimate the hole's position with computer vision algorithms. To not go outside the scope of this thesis, the methods are only introduced, briefly explained,

and compared; no theoretical background is given. So, no mathematical formulas are illustrated for the used vision functions.

Before the two carrying robots can approach the hole, obviously the hole's position must be know, at least roughly. In the considered scenario, a third robot is present, as depicted in figure 5.1. Its work is devoted exclusively *to detect* and *to track* the hole. In the simulation, another Girona 500 AUV is used for this job, without the arm. It is evident that, in a real scenario, a smaller and more efficient robot should be used for the vision, seeing that no manipulation capability are needed. In fact, in the original TWINBOT [TWINBOT (2019)] simulation, a smaller BlueROV is present. Anyway, for this thesis, another Girona 500 is used to not deal with an additional robot model.

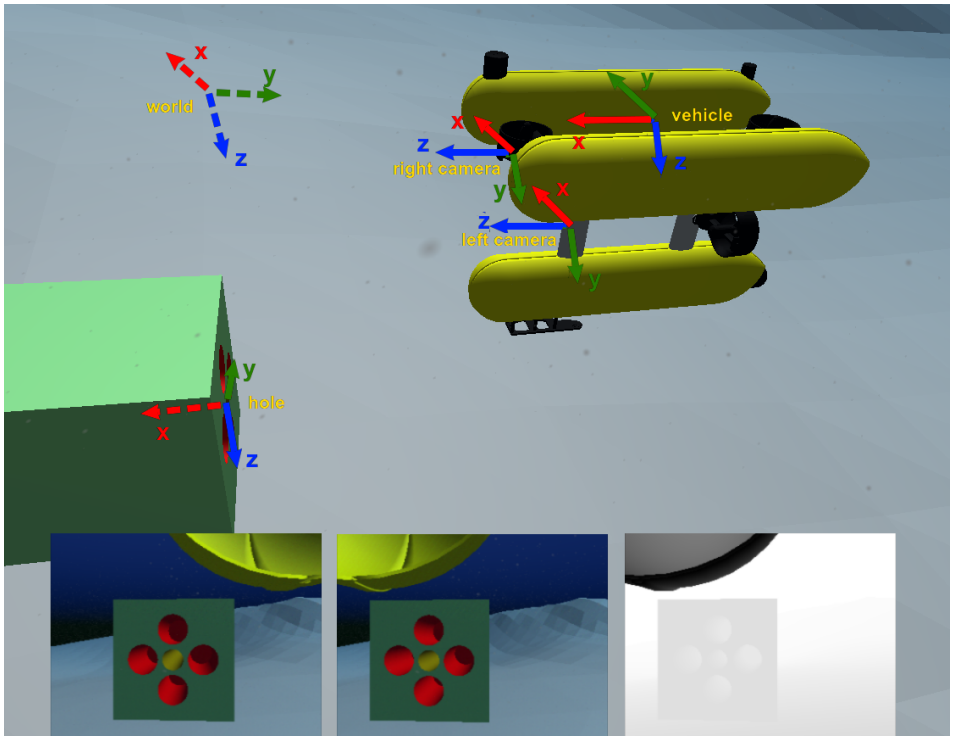


Figure 5.2: Screenshot where the main frames for the Vision are depicted. The cameras 3D models are not present. The left and the right cameras have same orientation, and the origin of the right one is shifted of 0.2 meters along left camera's x-axis. Vehicle and cameras axis are parallel. Hole's x-axis goes inside the cavity. World's frame is depicted only to understand its orientation, its origin is not in that point.

The Vision robot is equipped with two cameras which point in front of it, as

shown in figure 5.2. They are used for three different methods as:

- Two distinct cameras, independent of each other (mono camera case).
- As stereo cameras, thus exploiting stereo vision algorithms.
- As RGB-D camera, i.e., a stereo vision couple where the left one is a RGB camera and the right one is a Depth camera.

The job is divided in two phases: *Detection* of the hole (section 5.3) and *Tracking* of the hole's pose (section 5.4).

5.1 Vision Assumptions

For the sake of simplicity, some assumptions are made:

- Known *intrinsic* camera parameters. These parameters are used by algorithms to take into account how the single camera sees the scene. No image distortion is present.
- Known *extrinsic* camera parameters, i.e. the position and the orientation of cameras (respect to the vehicle), and thus it is also known the relative pose between the two cameras (needed for stereo vision algorithm).
- No external disturbances for Vision, such as underwater light reflections or bad visibility.
- Hole's model known. This means that dimensions of the cuboid which contains the hole are known. Further explanation about this are given successively in section 5.4.
- A "friendly" cuboid structure of the hole. The front face is coloured and additional holes are present, as can be seen in fig. 5.1. This helps both the *detection* phase and the *tracking* phase.

About the robot, other assumption are:

- The pose of the vehicle respect to the inertial frame (the *world*) is known. This is important to share the hole's pose with the carrying agents. Further details about this are given in section 4.4.
- The initial position of the robot is such that it is facing the front face of the hole. It must be noticed that methods explained in the next sections can be adapted to relax this hypothesis. For example, the vehicle could turn around z-axis until the hole is detected with one of the methods described later.

- Once the robot has tracked the hole and the pose is sent to the twin robots, it must go away to not interfere with the insertion phase. This is done through a keyboard (as a ROV) but it is not difficult to improve the code to let it go away autonomously. It also must be noticed that, thanks to the tracking, if the robot moves (because it is commanded to do so, or for water currents) the pose estimation keeps working.

5.2 Tools

To deal with the pose estimation, some external libraries are used. In this section they are listed.

- **OpenCV** (Open Source Computer Vision Library) [Bradski (2000)], an open-source BSD-licensed library that includes several hundreds of computer vision algorithms. In this thesis, it is used mostly for the detection part, even if some of its functionalities are used also by ViSP (for example for the keypoint tracking).
- **ViSP** (Visual Servoing Platform) [Marchand *et al.* (2005)], another open source library that helps to develop applications which exploit visual tracking and visual servoing techniques. It is interesting because it is more specific than OpenCV for robotic fields. In this work, it is used for the tracking phase.
- **PCL** (Point Cloud Library) [Rusu & Cousins (2011)] a library for 2D/3D image and point cloud processing. In this work, it is used by ViSP when depth images are used. However, further works can use it as another tool to deal with the vision part.

5.3 Detection

Object Detection means detecting a particular shape (i.e. the *object*) in the scene. This is important to initialize the common tracking algorithms known in the literature.

In fact, for the method exploited in this work, the detection step must provide a correspondence between some pixels in the 2D image and some points of the 3D object. It is important to notice that the needed 3D coordinates refer to the object frame (the *hole* in figure 5.2), and not to an "external" frame. Seen that the object model is assumed to be know, the knowledge of some 3D coordinates directly derives from this assumption.

Four is the minimum number of points accepted by the tracking algorithm. The more the points are, the more the tracking is good. Plus, points should lie on different surfaces of the object, to have better results.
Anyway, in this case, good tracking results are obtained also not considering these two aspects. The four points chosen are the corners of the front face of the cuboid which contains the hole.
The 3D coordinates chosen for the simulations are represented in the *.init* file 1.

File 1 *The* .init *file which describes the position on the 4 corners of the front face, respect to a frame positioned in the centre of the hole, with x-axis going inside the hole, y lying along the surface pointing on the right, z pointing down to the seafloor (this hole frame is depicted in figure 5.2).*

```
1: 4              # Number of points
                  # Coordinate order is x y z. The unit of measure is the meter
2: 0 -0.4 -0.4    # top right corner
3: 0 0.4 -0.4     # top left
4: 0 0.4 0.4      # bottom left
5: 0 -0.4 0.4     # bottom right
```

The work of the Detection step is to provide pixels' 2D coordinates that correspond to the 3D points of file 1. This must be done for each camera, except for the depth one (when used).
To provide these correspondences, two methods are evaluated: *Find Square* (section 5.3.2) and *Template Matching* (section 5.3.3). A third method (section 5.3.1), where the 2D coordinates are precise as much as possible (i.e. they are selected by hand clicking on the exact pixels), is used as a benchmark. This is also helpful to analyse the tracking results when the 2D coordinates are almost perfect.

Other methods and functions for the detection are briefly explained in Appendix B. Another, not explored, method can be using some code tags (like QR codes) on the cuboid surface. However, in underwater situations this can be difficult to be put in practice.

5.3.1 Already Known Coordinates Method

As explained, with this method the 2D coordinates are perfectly known. This is done by letting the user to click on the four pixels which contain the square's corners.

Given that the image is made by discrete pixels, it is impossible to have an ideal point which is exactly the corner, but the errors for this are not noticeable.

5.3.2 Find Square Method

This method is taken from an OpenCV tutorial (https://docs.opencv.org/3.4/db/d00/samples_2cpp_2squares_8cpp-example.html).
A rough explanation of how the method works is presented:

- This method looks in each image channel (that is only one if it is a black and white image, or they are three if it is a coloured image) to find squares.
- First, it pre-processes the image to reduce noise, using a pyramid scaling. Then, it exploits the Canny Edge Detector [Canny (1986)] to highlight the edges (results of Canny are visible in figure B.2 of Appendix's section B.2).
- The OpenCV function *findContours()* is called to extract contours of shapes with the algorithm described in Suzuki & Be (1985). The output after this passage is visible in figure B.3 of Appendix section B.3.
- Each shape's contour is approximated to be more like a regular polygon, i.e. with less vertices and edges.
- Finally, the algorithm looks if the shapes founded are similar to a square or to a rectangle. This is done checking if the internal angles of the contours are approximately 90 degrees. Furthermore, shapes with too little area are discarded to eliminate noise.
- The returned shapes are described by their four corners, that is what we were looking for.

An additional function is called to be sure that the order of the returned corners is the same order of the 3D points, otherwise correspondences are obviously erroneous.

5.3.3 Template Matching Method

Template Matching means to find a pattern (in this case, the face of the hole) inside a scene. This method is a well-known tool used in many applications.

It is important to notice that an additional image (the *template*) is necessary. So, we have to assume that an image of the hole's square face is provided.

The code implemented follows an OpenCV tutorial (https://docs.opencv.org/3.4.6/de/da9/tutorial_template_matching.html).
In brief, the *template matching* finds the point in the scene which has the best similarity (or the least dissimilarity) with the provided template. This is done considering intensity values of the pixels in the neighbourhood area of each point in the image. In practice, the template is shifted all over the scene and a formula is computed for each shifting. Various formulas to compute similarity (or dissimilarity) are provided by OpenCV and are detailed in the library documentation. The chosen one in the experiments is the so-called *squared difference*.

To have correct results, it is important to scale up and down the template and to compute multiple times the similarity. This because usually the template's size is not equal to the size of the object in the scene.
So, for each scaling, a best similarity point is detected. Then, all the similarity points are compared and the best one is taken. At the end, a rectangle with the template (scaled) dimensions is built considering the best point as the centre. The corners of this rectangle are the 4 points which we were looking for.

5.3.4 Detection Results

For this scenario, the Find Square method is better than the Template Matching one. As can be seen in figure 5.3, differences from the ideal method and the Find Square one are barely visible.

Looking at the way it works, it should be noticed that the Find Square method gives good results only if the camera faces the cuboid structure approximately at the front. If a side face is more visible, it will be the one where the rectangle is detected. This is not a problem because the tracking algorithm can be initialized also by a side face, but we have to give the 3D points which correspond to this side. So we must know which face the robot is looking at.

In addition, this method is suitable if no other squares of similar dimensions are present in the visible scene. If this would happen, some further works are needed to take the right one.

Another problem is that it is not suitable with other kind of shapes (a circle structure, for example). This is obvious because the algorithm only detects squares and rectangles.

It is also important to point out that, sometimes, the method fails to find any

shapes in the right image, when the initial robot position is the one chosen in the experiment. This happens approximately 30% of the time, and it may show a very bad robustness and a low predictability of the method. However, the fail is detectable (by a human operator but also easily by the software) and another trial can be repeated.

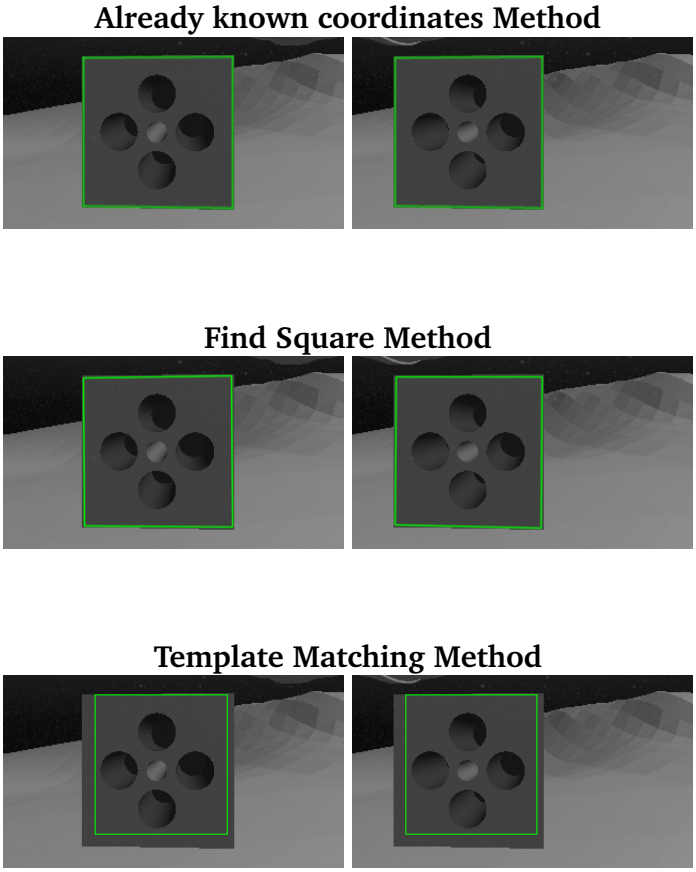


Figure 5.3: Results of the three different detection method. The green rectangle is the estimate position of the square. The output of the detection step are the four corners of the green rectangle.

As said, the Template Matching is less precise than the first method.
Apart this, it has different problems than previous. Firstly, if the face is viewed from a different angle, an other template image is needed, with an orientation similar to what the robot is seeing. In general, lot of template images at different angles are needed. Otherwise, some processing of the template image is necessary to orient it in a different way.

Secondly, the orientation can make the face to be a distorted square in the 2D image. With the first method, contours are followed well even if the quadrilateral is not a perfect square (until a certain point obviously). With the Template Matching, the only way to find the corners is by building a shape equal to the template around the similarity point. So, the real edges can be followed badly and the resultant corners can be imprecise (as can be seen in figure 5.3).

A better quality of the Template Matching method respect to the other is that it can be used for different shapes, not only rectangle ones. Plus, the object to be found can be also a complicated one, full of details that would put a "geometric shapes detector" (like the Find Square) in big difficulties.

5.4 Tracking

Object Tracking means to follow the motion of an object of interest during time. Both the object and the camera can be mobile, even if, in this case, only the cameras move (actually, the robot moves with the cameras rigidly attached to its body). Tracking an object is usually done to estimate its pose respect to the camera frame.

In this work, we speak about a *markerless model-based* tracking. Thus, no markers attached to the object are needed but the object structure must be known. In this scenario, it is sufficient to give to the algorithm the 3D dimensions of the cuboid structure of the hole (i.e., the position of the eight corners respect to the object frame), and the hole's dimension and position in the front face (i.e. position of three points which describe the circumference respect to the object frame).
The library chosen for the tracking phase, ViSP [Marchand *et al.* (2005)], uses an own format called *.cao*, which syntax is described in the library documentation.
As explained in section 5.3, the tracking algorithm must also know the 2D-3D correspondence of *at least* four points belonging to the object. In the experiments, the provided ones are the 4 corners of the front face.

Three different trackers have been implemented: a *Mono Cameras Tracker* (section 5.4.1), a *Stereo Camera Tracker* (section 5.4.2), and a *Stereo Depth Camera Tracker* (section 5.4.3). Results and comparison of them are discussed in section 5.4.4.

A tracker is linked to each camera. For RGB cameras, it can be of three types: *edge-based* [Comport *et al.* (2006)], *keypoint-based* [Pressigout & Marchand (2007)] or a mix of both. During the experiments, the hybrid method emerged as the most precise, so all the results in section 5.4.4 refer to this one.

For the depth camera used in the Stereo Depth Camera tracking, the tracker's type can be *normal* or *dense* [Trinh *et al.* (2018)]. Being the *dense* one more robust, it is the chosen one for the trials. Please note that it is also computationally heavier for larger matrix computations, but speed performance is not considered here.

5.4.1 Mono Camera Tracking

This method derives from the ViSP tutorial *markerless generic model-based tracking using a colour camera* (https://visp-doc.inria.fr/doxygen/visp-daily/tutorial-tracking-mb-generic.html).

The implementation is straightforward: after setting the trackers (i.e. giving edge detection and keypoint detection parameters, camera parameters, and 2D-3D correspondences of the four corners), at each loop the tracker estimates the transformation matrix between each camera and the object.

In this method, the left and the right cameras are independent. Thus, each one provides a different pose estimation. It is not so easy to understand when one camera provides better results that the other, without taking the real pose as benchmark. So, in the applications could be difficult to choose one pose or the other. A good compromise could be to do a mean of them.

5.4.2 Stereo Camera Tracking

This method derives from the ViSP tutorial *Markerless generic model-based tracking using a stereo camera* (https://visp-doc.inria.fr/doxygen/visp-daily/tutorial-tracking-mb-generic-stereo.html).

The code is analogous to the previous one, except that in this case also the relative pose between each camera must be provided. If this is unknown, some method for stereo calibration must be used. Due to the fact that now the cameras are not independent, a unique pose is provided, that, as we will see later, has better precision than the previous one.

5.4.3 Stereo Depth Camera Tracking

This method derived from the ViSP tutorial *Markerless generic model-based tracking using a RGB-D camera* (https://visp-doc.inria.fr/doxygen/visp-daily/tutorial-tracking-mb-generic-rgbd.html).

This method is similar to the previous one, except that now the right camera is a depth one, thus it provides range images.
The functions used for the depth images need the support of another library, PCL [Rusu & Cousins (2011)].
Another difference is that the depth camera does not need to initialize the 2D-3D correspondences, so the *Detection* step has to be done only for the left camera.

5.4.4 Tracking Results

In this section, performances of the three trackers are evaluated. For each one, the three different types of detection initialization (explained in section 5.3) are considered to evaluate the effects of detection's error on each kind of tracker.

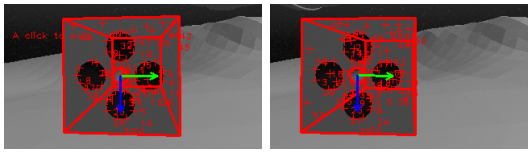
Experiments have been conducted with a lot of simplifications: no disturbances, no camera distortions, very good visibility, nice object shape. Results described here can give only an idea on how to proceed in a more realistic environment.

In the scenario, the robot is perfectly still while it is tracking the object, even if the tracking methods can be used with moving objects and/or moving cameras. The vehicle is positioned in front of the square face, slightly on the right. The original images taken from cameras are cut to delete a region where a part of the vehicle is visible. This is done to make this part to not cause useless disturbances to the vision algorithms. In the depth images, this is not necessary, because there is no interference.

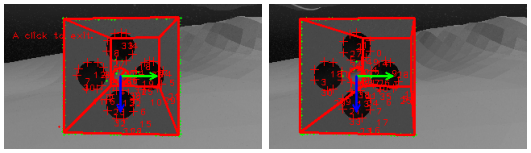
In figure 5.4 the detected shape and the estimated frame are drawn on the camera images. Differences are barely visible when comparing the first two initialization methods (the ideal one and the Find Square one) in all the three tracking methods.

Instead with the Template Matching detection initialization (last group of images of figure 5.4), the lower precision (visible in figure 5.3) is paid in tracking results, especially in the depth case. This is clearer in figure 5.7 where the error is plotted. With this initialization, the depth-stereo method is even worse than the monocular case.
This can be due to the fact that the depth image is not initialized with 2D-3D correspondence; thus we pay more the initialization error, being done only in the left image.
This demonstrates that it is not always better to have a stereo RGB-D camera instead of a normal stereo RGB. This is an interesting result and should be further explored with more realistic scenes.

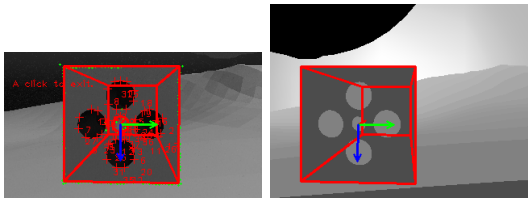
Already known coordinates Initialization



Mono Cameras Case

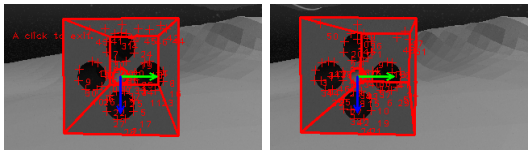


Stereo Cameras Case

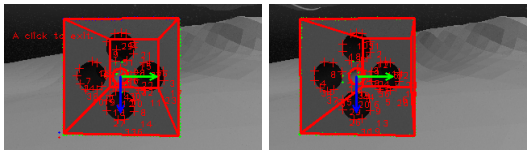


Stereo Depth Camera Case

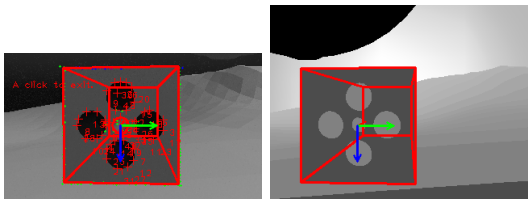
Find Square Initialization



Mono Cameras Case



Stereo Cameras Case



Stereo Depth Camera Case

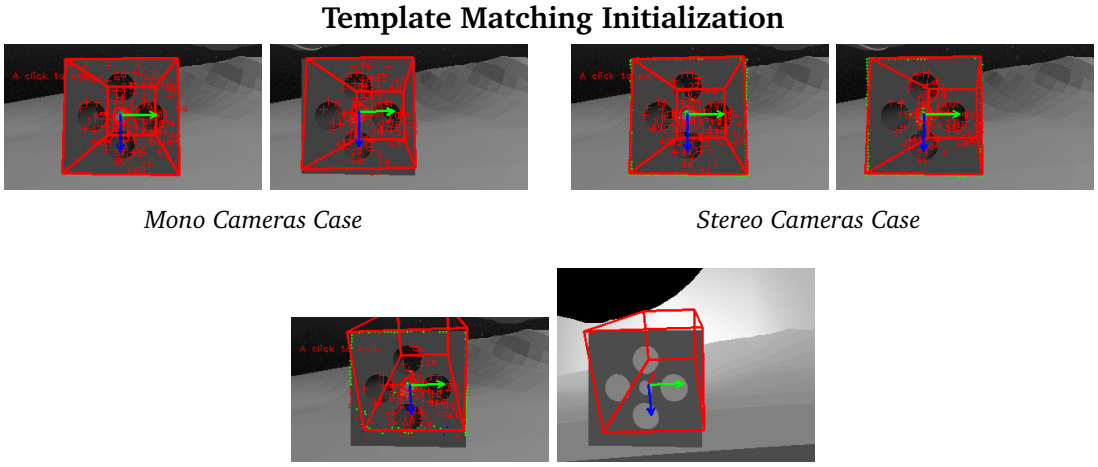


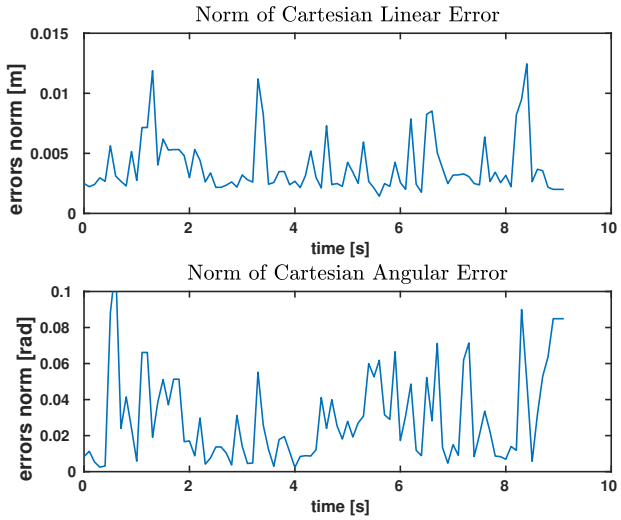
Figure 5.4: Results of the three tracking methods with the three detection initializations. Red lines are the contours of the model which represents where the object is estimated to be; little red + marks are the points detected by the keypoint algorithm. Green dots are the tracked correspondent points between the left and the right images, in fact they are present only in the stereo cases. The arrows represent the estimated object frame: green for y-axis, blue for z-axis, red for x-axis (which goes inside the hole and it is barely visible).

With the two good initializations (the ideal one and the Find Square one), the two stereo methods have similar results, overall better than the mono case. This is visible in the errors' plots of figure 5.5 and figure 5.6. Anyway, we can also see that the mono camera case has not so bad performance. Considering that a stereo camera is much more expensive, a tracking with a single mono camera can be an advisable deal.

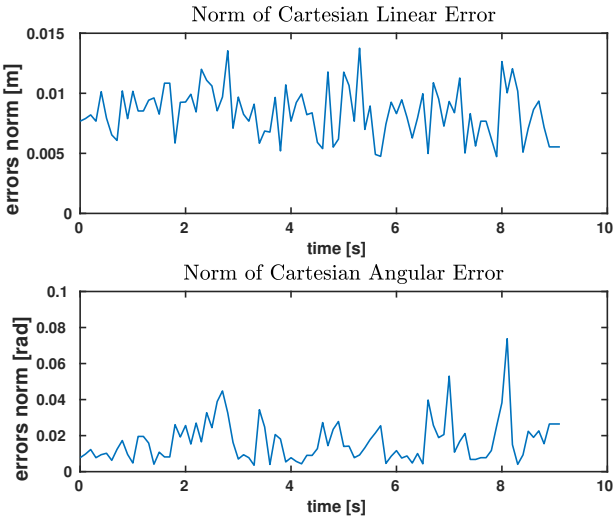
Another interesting outcome for the monocular case, is that the position of the camera influences the results. This is because different view angles provide different tracking performance.

In the plots of the errors (fig. 5.5, fig. 5.6, and fig. 5.7), a lot of variations can be seen while the time goes on, although the robot and the object are still. This is due to the nature of the tracking algorithm, which continuously updates the pose at each new image received by the camera. Anyway, it must be noticed that the variations are little for both the linear and the angular parts. So, taking the pose at a certain time instead of an another is not so influential.

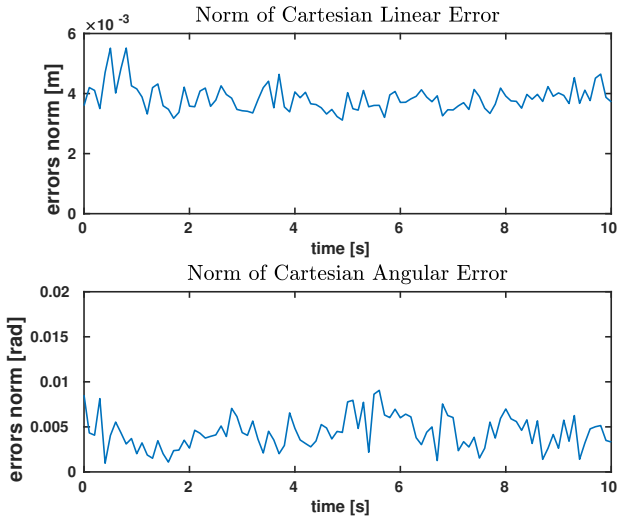
Already known coordinates Initialization



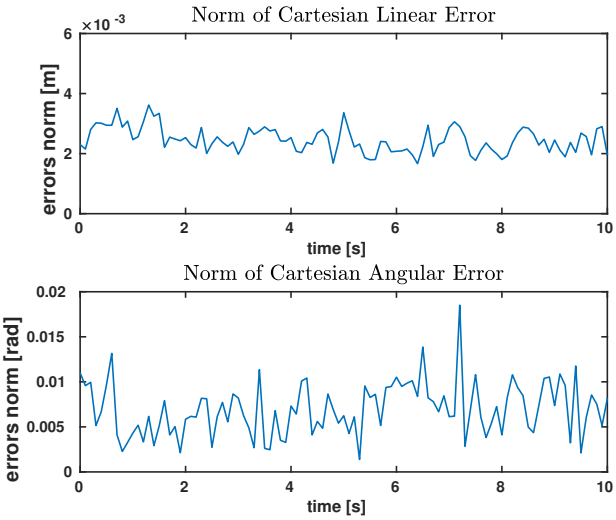
Mono (left Camera) Case



Mono (right Camera) Case



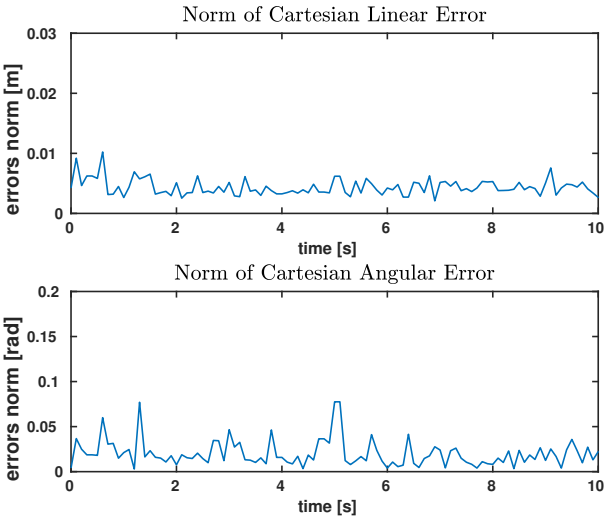
Stereo Camera Case



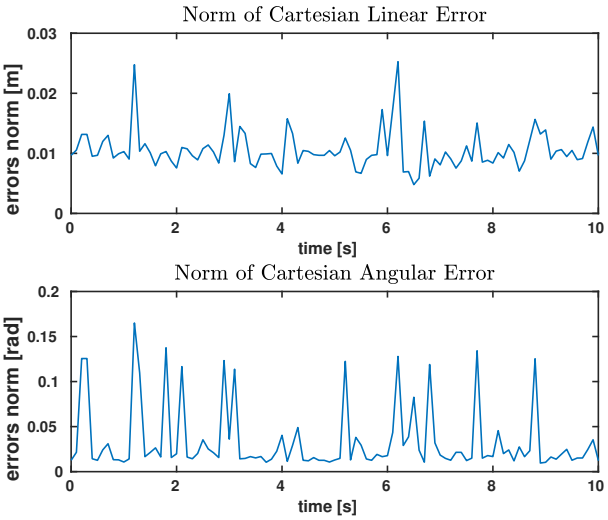
Stereo Depth Camera Case

Figure 5.5: Linear and angular error (in norm) between the true pose and the estimated one. The tracking is initialized with the ideal detection method of section 5.3.1.

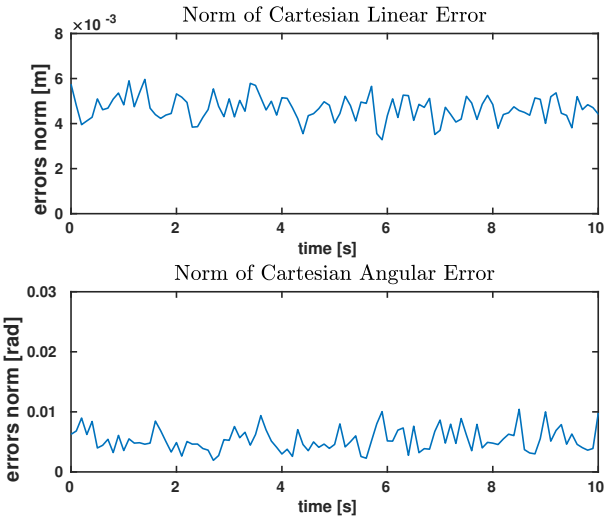
Find Square Initialization



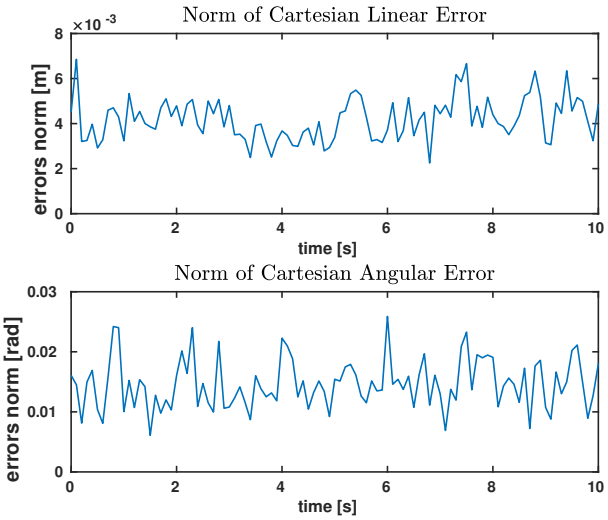
Mono (left Camera) Case



Mono (right Camera) Case



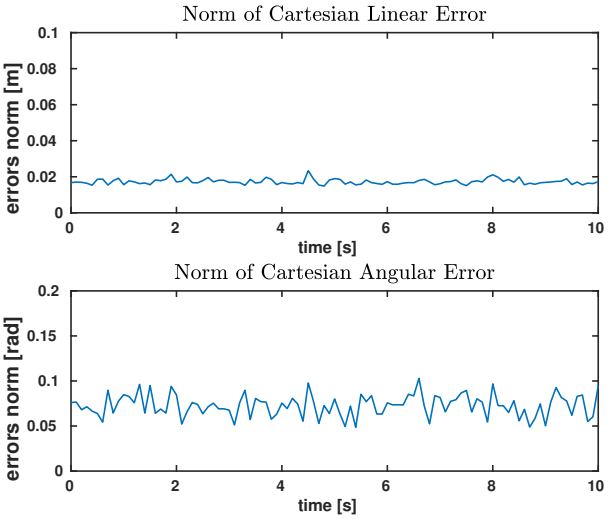
Stereo Camera Case



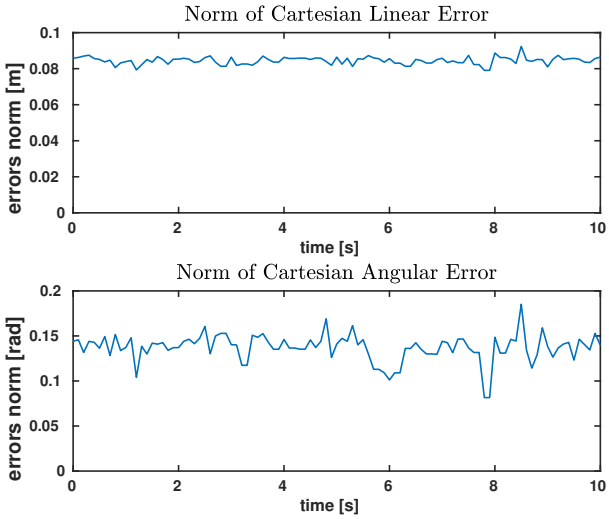
Stereo Depth Camera Case

Figure 5.6: Linear and angular error (in norm) between the true pose and the estimated one. The tracking is initialized with the Find Square detection method of section 5.3.2.

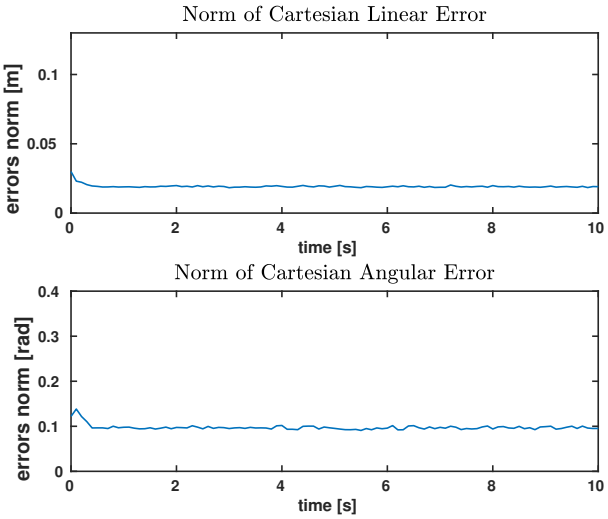
Template Matching Initialization



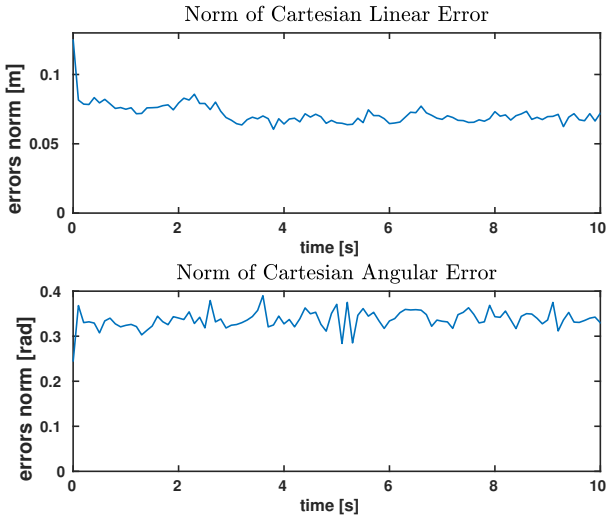
Mono (left Camera) Case



Mono (right Camera) Case



Stereo Camera Case



Stereo Depth Camera Case

Figure 5.7: Linear and angular error (in norm) between the true pose and the estimated one. The tracking is initialized with the Template Matching detection method of section 5.3.3.

Chapter 6

Conclusions

This thesis has presented a kinematic control architecture for two cooperative autonomous underwater manipulators.
The robot collaboration is done at kinematic level, exchanging vectors and matrices to produce a common tool Cartesian velocity. The cooperation scheme takes into account that underwater communication is difficult, and it keeps the amount of exchanged data as low as possible.
The experimental results showed how the Task Priority Inverse Kinematics approach used can deal with an assembly task: the *peg-in-hole*. Being an unexplored problem for cooperative underwater manipulator, the scenario is simulated with many simplifications and assumptions.
Part of the problem deals with computer vision techniques. The thesis shows how some detection and tracking algorithm can be exploited to estimate the hole's pose. For the insertion phase, a force-torque sensor is used to help the accomplishment of the mission, thanks to the data provided exploited by the control architecture.
Both the Control and the Vision part can help other works in various robotics fields, not only related to underwater intervention missions.

Code implementation is suited to easily manage objectives (e.g. to delete offline some objectives to try a different method). Big effort has been made in providing a modular and flexible code architecture. For example, the dependency from ROS (Robotics Operating System) is kept at minimum to make possible to easily adapt the code to a different kind of interface and/or simulator (which does not rely on ROS to communicate).

In the Chapter 1, an introduction about the context is given, and the previous works in the relative fields has been recalled.
In Chapter 2, the principal points of the theory behind the Control Architecture are summarized. Here, the mathematical foundations for the Task Priority Inverse Kine-

matic approach are recalled, considering also a coordination policy between multiple agents that fits in the chosen approach.
In Chapter 3, the theory explained before is exploited to deal with the scenario stated by this thesis. A Force-Torque objective is inserted in the TPIK list to reduce the magnitudes of forces and torques that act on the peg during the insertion phase. This is noticeable because it is used a "dynamic" information (the force and the torque) in a pure-kinematic method. A simple, but suitable for the scenario, list of objectives is then described. An additional routine, part not of the kinematic layer but more of the Mission Managing layer, is explained. Considering the goal frame where the peg is driven to by the kinematic control, this new routine shifts its origin according to the direction of the forces acting on the peg. This is an additional method to further exploit the information given by the force-torque sensor.
In Chapter 4, the simulated environment is detailed and the experimental results are discussed. The chosen simulator, UWSim, is introduced, along with some other underwater simulators that can be useful for the interested reader. To make the insertion phase realistic, collisions between the peg and the hole have been inserted in the simulation. These collisions propagates through the whole robotic system, thus affecting the arm and the vehicle. Another routine is implemented to fake a firm grasp of the tool by the two robots. In real environment, this constraint is assured by frictional forces, but in a pure-kinematic simulator like UWSim the frictions are not present. The tuning of the gains permits to not hide bad cooperation between the agents. In the same Chapter 4, assumptions to simplify the problem are explained, and an idea of how the Control Loop runs is given. Finally, results of the experiments done are presented and discussed. Three main experiments have been carried out: without hole's pose error, with a fixed error of 0.015m along one axis, and one final test which includes the Vision part with the hole's pose error given by the *Detection* and the *Tracking* algorithms used.
Chapter 5 covers exclusively methods and tools used by the Vision robot. No theoretical background is given because it would take out of the scope of this thesis. The Chapter gives an idea on how two computer vision libraries, **OpenCV** (Open Source Computer Vision Library) [Bradski (2000)] and **ViSP** (Visual Servoing Platform) [Marchand *et al.* (2005)], are used. The Vision part is divided into two phases: Detection and Tracking. For both, different algorithms have been tested, compared and discussed. In particular, for the Detection part, some methods have been discarded and they have not been used any more for further trials, but they are anyway presented in Appendix B. They are all OpenCV algorithms that can help in other applications.
This Chapter 6 concludes the thesis and it gives some starting points for possible future works.
The Appendix A gives some details on how the software is implemented, together

with a list of some useful libraries used that surely can help to develop a control architecture in the C++ programming language.

For some on-going progresses in this scenario, it can be useful for the reader to follow the TWINBOT project [TWINBOT (2019)]. This thesis' context derives from the scenario of this project, but it evolves independently because at the time this thesis was being developed, TWINBOT was in a very early stage.

6.1 Future Works

Since the novelty of the application, further works can be pursued in various directions.
For what concerns the experiments, a dynamic simulation, along with a dynamic controller, can be introduced to better analyse the methods adopted. This would mean to include effects that would increase the realism of the simulation, such as buoyancy, thrusters modelling, disturbances of the arm to the vehicle and vice-versa, real tool-grasping effects, even some water currents. Some work has been done in this direction but then it has not been pursued due to the lack of time. These efforts, even if they are not presented in this thesis, showed that the first step to introduce dynamics could be using the plugin FreeFloatingGazebo [Kermorgant (2014)], mentioned in section 4.1. This one is the most suitable tool to be used from the actual work because its scope is to solve the lack of dynamics of UWSim, expanding the functionalities of the simulator. So, it would be easy to adapt the code to the new simulations. For example, the scene (i.e. the file which describes the simulated scenario) would be the same, and ROS would be always used as interface.

Regarding the actual chosen architecture, the Force Torque objective idea can be improved. For example, we can consider a different task reference, calculated not only as a proportional error between the desired force (that is zero) and the actual detected one.
Another improvement can be for the Change Goal routine. We could let the forces and the torques modify also the orientation of the goal frame, to reduce/eliminate the angular error between the real goal frame and the one estimated.

Regarding the insertion phase, additional problems can be explored. This thesis focused only on collisions that may happen when the peg is already inside the hole. If some contact between the external surface of the hole's structure and the peg's tip happens, the mission fails (i.e., it occurs a stalemate where the peg bounces forever against the hole's surface, unable to find the hole). In the literature, various methods have been explored toward this point. For example, researchers have con-

sidered the cases when the peg meets the hole with a bad alignment that creates a two or three points contact (as briefly explained in 1.1.3). However, to the best of this author's knowledge, the *peg-in-hole* problem has never been studied when the protagonists are two autonomous mobile manipulator (in any scenarios, not specifically underwater ones). So, it can be interesting to adapt old tools to this particular (cooperative and underwater) field.

Towards a more realistic intervention missions, efforts can be spent to consider ways to localize the robot under the water's surface. In this thesis, it is assumed that all agents have a reference frame in common, but, usually, underwater localization is really an issue. Some cooperative methods (this time not at kinematic level) can be considered to make some surface vessels help the underwater agents to localize themselves (a problem explored in the WiMUST project [Abreu *et al.* (2016)]).

In this thesis, some assumptions have been made for the Vision part. Further works could consider to relax some of them, for example to increase the difficulty of the detection and tracking phases. In an underwater scenario, not always the water permits to watch from afar, and illumination and distortions can be other important issues.

There is always a lot of work towards increasing the capacity of intervention-AUVs. For example, we can consider more specifically communication issues and, so, new techniques to exchange data between agents; especially in an underwater scenario, we can't share too much information among robots and with too high frequency. Also, other kinds of assembly problems can be addressed: for instance, a *peg-in-hole* one where the *peg* is held by only one robot and the *hole* by another one.
Some objectives of the TWINBOT project [TWINBOT (2019)] aim to study these two just mentioned problems.

Appendix A

C++ Code Scheme

Some effort has been spent to implement a flexible and modular C++ code architecture. The main focus is directed to facilitate the use of the TPIK approach. With this scheme, adding and removing tasks from the code is easy and straightforward. Furthermore, even if ROS is used as interface, other communication methods (for different simulators, or even for real robots) can be used easily, changing a little part of the code. This is due to the fact that all the ROS parts (the *interface* classes) are written in separate files from the ones of the main blocks.
Only a rough idea about the code scheme is given here, without too much details. Further explanations can be found in the github page (https://github.com/torydebra/AUV-Coop-Assembly) and in the code documentation (https://torydebra.github.io/AUV-Coop-Assembly/).

A.1 Tools

The Control Architecture is implemented in C++ language. Some external support libraries have been used, and, in this section, the most relevant ones are described. Please note that a particular section (section 4.1) is dedicated to the choice of the simulator, and another lists the main tools used by Vision (section 5.2).

- **ROS** (Robot Operating System), the well-known robotic middleware. It is used to communicate with the simulator, which means sending commands to robots and receiving information by the on-going simulations (e.g. robots states, data from sensors, streaming images from cameras).
- **Eigen** [Guennebaud *et al.* (2010)], a C++ library for linear algebra. It is very useful to deal with matrix computation and management in any C++ software.

- **CMAT**, another C++ library, developed at GRAAL laboratory of University of Genova (`http://www.graal.dibris.unige.it/`). It implements the core functions for the TPIK method, detailed in Simetti & Casalino (2016).
- **Orocos KDL**, a package to deal with kinematic and dynamic chains. Here it is used to compute the Jacobian of the robots, given the arm and the vehicle configuration.

A.2 The Single Robot Code Scheme

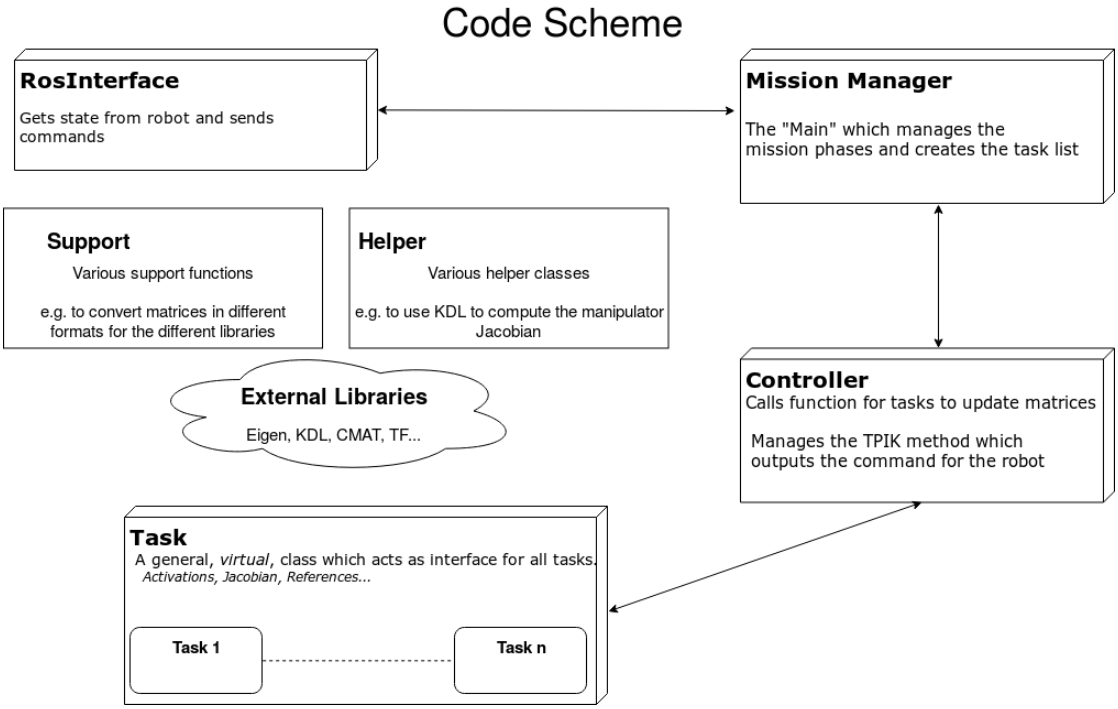


Figure A.1: The C++ code scheme that shows the relationship among the main blocks of the single robot. Parallelepipeds represent the principal blocks, while rectangles contain useful functions used by them. Arrows indicate the communication between the main blocks.

In this section it is presented the scheme for the single carrying robot. The two cooperative robots share the same code, and they are differentiated (when necessary) thanks to their names (*g500_A* and *g500_B*).

- **RosInterface**. This class is used to communicate with the simulator, e.g. sending commands to the robot, receiving state and sensor information, and so on. It is obviously ROS-dependent, and it should be replaced if another middleware is used.
- **Mission Manager**. This block is the "main". It initializes all the useful classes, it creates the tasks list, and it manages the whole mission.
- **Controller**. This class is the core of the control; in practice it is the kinematic layer. It generates commands for the vehicle according to the prioritized list of tasks. It is where the iCAT algorithm based on the TPIK approach (section 2.3) is used.
- **Task**. This is an *abstract* class. It acts as a base class for all the specific *concrete* classes of tasks. In this way, the controller and the mission manager simply handle a vector of pointer to Task. The Mission Manager creates a concrete class for each task and then it fills the vector. This vector is the prioritized list of the TPIK method. The controller can iterate the element of this vector and can call the abstract methods of Task without worrying of which real concrete task is actually inside the list.
- **Support**. It is a group of various *namespaces*, used for conversions, to print to file, and to use some mathematical formulas.
- **Helper**. It contains a group of classes and headers used to help the control scheme (e.g. to compute the Jacobian with KDL) and to log results.

A.3 The Whole Code Scheme

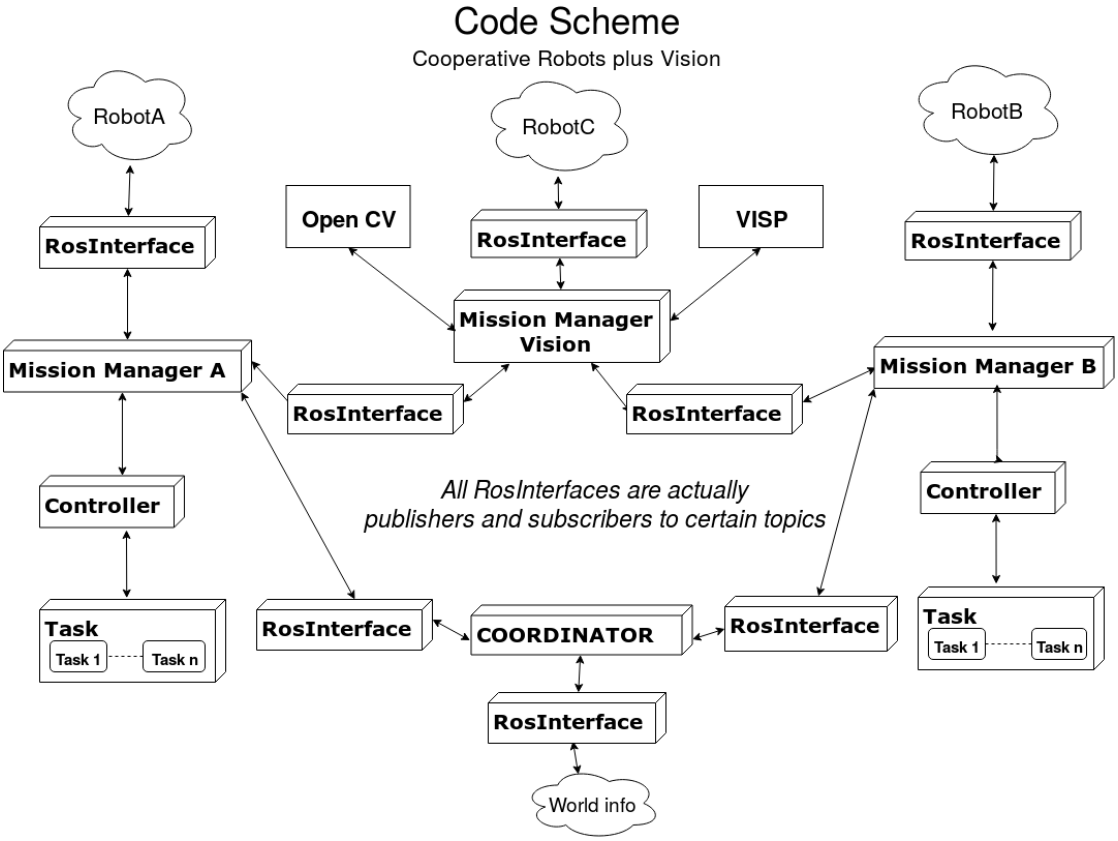


Figure A.2: The C++ code scheme that shows the relationship among the main blocks of the three robots. On the left and on the right side there is a zoomed out view of the previous figure A.1, corresponding to the scheme for the two carrying agents. In the centre, there are the blocks for the Vision robot and for the Coordinator.

The two robots must communicate between them and with the Vision robot. Like explained before, ROS is used to communicate with the simulator, but it is also used to make different nodes (e.g. Coordinator and Robots) to communicate.
Please note that the Coordinator is not a physical agent: it can be put as a software routine on one robot. This would help with the communication issues typical of underwater scenarios, because only data-exchange between the Coordinator and the other robot will pass through the water.
The scheme for the Vision robot is simple because it is driven as a ROV: it is not autonomous and so no TPIK is implemented for it. Anyway, it can be switched easily

into an autonomous robot, with or without TPIK (that it is not really necessary for this agent).
The Coordinator is a node in charge of dealing with the coordination policy explained in section 2.5. It also needs information from the world (i.e. the simulation) to compute the cooperative velocity.

Appendix B

Other Algorithms for Object Detection

During the simulations, several trials have been done to find a suitable algorithm for the detection of the hole structure. In Chapter 5 two methods have been discussed as the successful ones. In this appendix, others that have been discarded are briefly presented. Even if they are not used in the last versions of experiments, they can be useful for other purposes, such as detection of other kind of shapes.
Each algorithm is taken from OpenCV Detection tutorials (https://docs.opencv.org/3.4/d9/d97/tutorial_table_of_content_features2d.html), where also other interesting methods can be found.

B.1 Corner Detection with the Shi-Harris Method

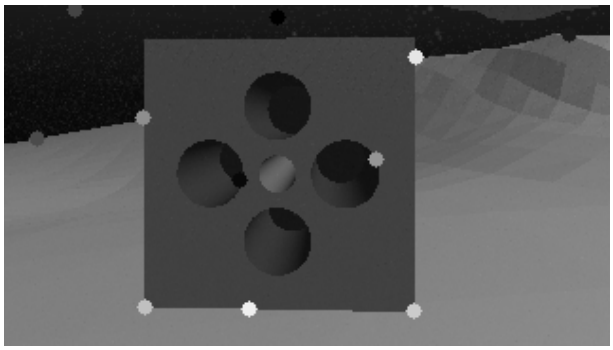


Figure B.1: goodFeaturesToTrack() result. Only two detected points are the real corners, and the upper ones are not detected.

Following the tutorial (https://docs.opencv.org/3.4.6/d8/dd8/tutorial_good_features_to_track.html), it has been implemented a corner detector with the Shi-Harris method [Shi & Tomasi (2000)] using the OpenCV function goodFeaturesToTrack().
This function, in our case, can be useful to find the corners of the hole structure, which is the necessary initialization for the Tracking method used after (section 5.3).

The original example lets change the number of maximum points to be found. This is useful to reduce the number of false positive corners. The main problem is that the real corners of the square are not the "best" ones. So we can't simply put this parameter equal to 4. On the other side, with a large number of points, would be then difficult to discriminate the right corners from the others.
Other interesting parameters are:

- **minDistance**. The minimum distance among the corners to be found.
- **qualityLevel**. A parameter which characterizes the minimum accepted quality of image corners.
- **blockSize**. Size of an average block for computing a derivative covariance matrix over each pixel's neighbourhood.
- **mask**. To specify a certain region of interest in the image. In such a way, corners are searched only in this region. The problem in our case is that without prior works we can't know where the interesting region is (i.e. the region which contains the hole).

The points detected are effectively good feature points (as can be seen in figure B.1). But, the best ones are not the ones that we want to detect (i.e., the corner of the square).
This method should be used as a low level algorithm, to then help higher level ones. For example, to construct some polygons and to check if these polygons are square/rectangles. However, to follow this direction should be better to start from the edges (as done in section 5.3.2).

B.2 Canny Edge and Hough Transform

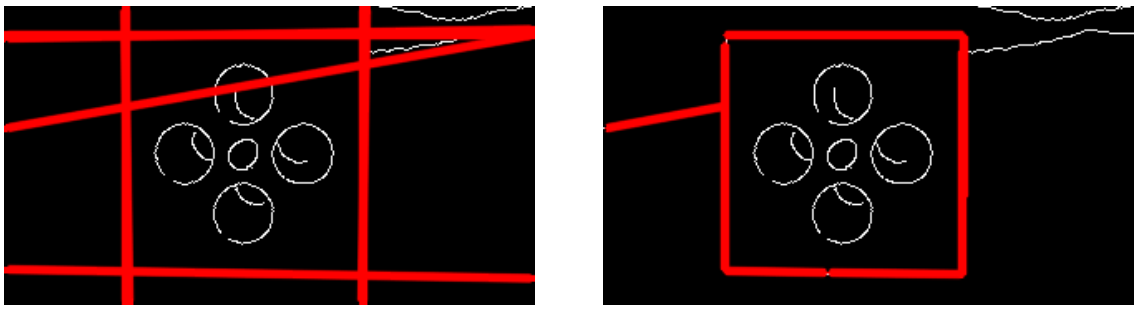


Figure B.2: Results of the Hough Standard Transform (left) and the Probabilistic one (right). In white they are depicted all the edges detected with Canny; the red lines are the detected straight lines, outputs of the method.

The Hough Transform [Duda & Hart (1972)] is a method to detect straight lines in an image. Usually, a preprocessing of the image with an edge detector is used to improve the results, for example with a Canny Edge Detector [Canny (1986)].

The OpenCV tutorial (`https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html`) makes use of two types of Hough Transform: the standard *HoughLines()* and the probabilistic *HoughLinesP()* [Matas *et al.* (2000)].
Results are visible in figure B.2. The results on the right shows that the probabilistic version is good to detect the square structure of the hole. Thus, this method can be used as a good preliminary step to then extract the corner from the detected square.

B.3 Bounding Box Detection

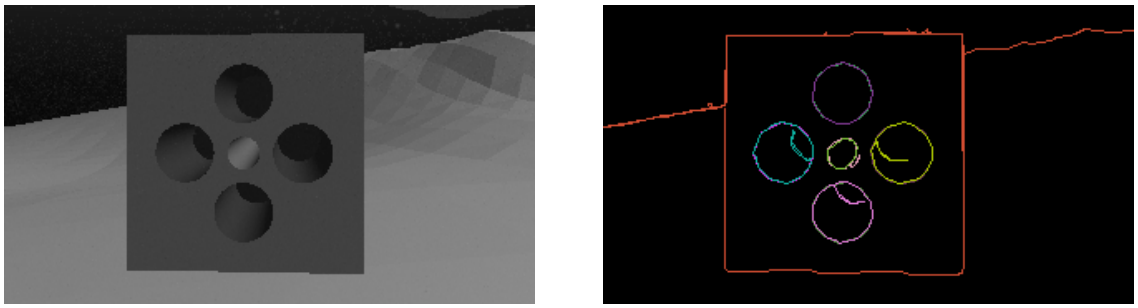


Figure B.3: Result of findContours()*: on the left the original image, on the right the contours detected.*

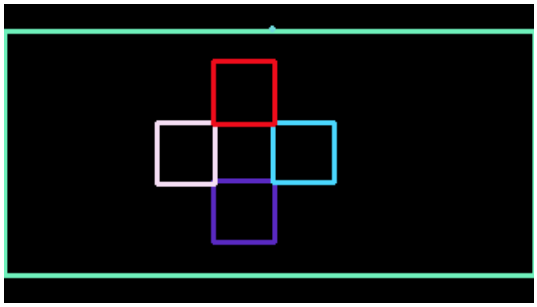


Figure B.4: Result of Bounding Box detection, where they are visible the drawn bounding boxes around the holes.

The code derived from an OpenCV tutorial (`https://docs.opencv.org/3.4.6/de/d62/tutorial_bounding_rotated_ellipses.html`).
First, a Canny edge detector is used to preprocess the image. Then, the function *findContours()* is called to retrieve contours with the algorithm described in Suzuki & Be (1985). As can be noticed, these initial steps are the same of the implemented method of section 5.3.2. The difference in this tutorial is that, after these passages, bounding boxes are drawn around some particular shapes detected.

The result after the first step is shown in figure B.3. We can see that this passage already reveals the square, that is important for the method described in section 5.3.2. Instead, the algorithm presented here continues in a different direction, which brings us to the final result of figure B.4.

This tutorial is recalled because it can be useful to find other kinds of shapes, for example an hole structure which is not a rectangle or a square.

B.4 2D Feature Matching & Homography

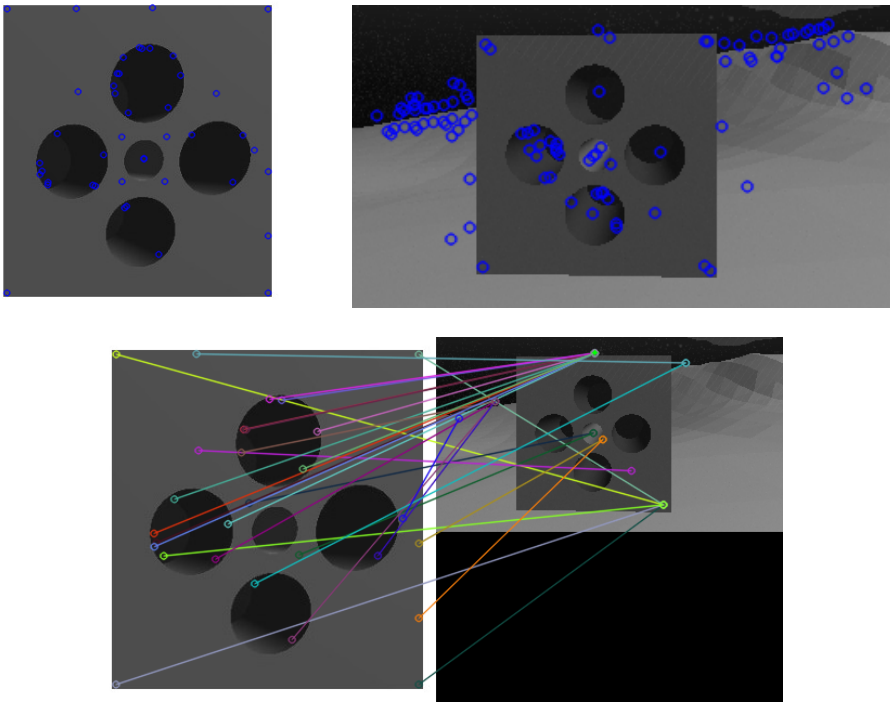


Figure B.5: Result of the 2D Feature Matching. Above, the blue circles are the detected features in the object image (on the left) and in the scene image (on the right). Below, the output of the matching passage, which shows clearly a bad outcome.

Image features are small patches that are useful to compute similarities between images. These are different from corner points; they indicates particular details that stand out in the image. Detecting these areas is useful to recognize objects of interest, as a sort of *template matching* (please note that this method is not a template matching as the one of section 5.3.3).
The *descriptors* of these features contain the visual description of the patches, which are used to recognize similarities between different images.

This method needs an *object image* and a *scene image*. The first is a sort of template which contains only the object to be found (in this case, the square face of the hole). The second is the image in which we want to detect this object (in this case, what the camera is seeing).
After good features are extracted from both images, the *descriptors* are used to *match* them, thus, detecting the object in the scene. Then, it is necessary to find the perspective transformation between the object image and the scene (i.e. find the ho-

mography). This is needed to take into account that usually the pose and the scaling of the object in the scene are not the same of the ones of the object image.
The OpenCV tutorial (https://docs.opencv.org/3.4/d7/dff/tutorial_feature_homography.html) uses different tools:

- **SURF** (Speeded Up Robust Features) Detector [Bay *et al.* (2006)] to extract the features, and to compute the descriptors.
- **FLANN** (Fast Library for Approximate Nearest Neighbors) matcher [Muja & Lowe (2012)] to match the features of the two images.
- **Lowe's ratio test** [Lowe (2004)] to take only the best matches.
- **RANSAC** (RANdom SAmple Consensus) [Fischler & Bolles (1981)] method to find the homography with the function *findHomography()*.

In this case, results are unsatisfactory, as can be seen in figure B.5. The main problem is that in this particular scene there are not nice distinct features. Also, the symmetry of the structure does not help, because there are a lot of particulars that are similar (like the square sides and the holes). As can be seen in the results of the previously cited tutorial, good outcomes are obtained for food boxes. In fact, this method is often exploited for scenes where a lot of details are present (e.g. graffiti painting, supermarket shelf). In underwater cases, realistic infrastructures have not so many details, and so also the simulated scenario of this thesis.

There are a lot of parameters to set for the three main tools (SURF, FLANN, and RANSAC). Various options have been tried but no-one was satisfactory. Also, different detectors (like SWIFT [Lowe (2004)]) and matchers (like Brute-Force), have been tried, again with poor results.
Anyway, the variety of tools and parameters makes this method suitable for a lot of applications, and it should be taken into consideration in other applications.

References

ABDULLAH, M., ROTH, H., WEYRICH, M. & WAHRBURG, J. (2015). An approach for peg-in-hole assembling using intuitive search algorithm based on human behavior and carried by sensors guided industrial robot. *IFAC-PapersOnLine*, **48**, 1476–1481. 7

ABREU, P., MORISHITA, H., PASCOAL, A., RIBEIRO, J. & SILVA, H. (2016). Marine Vehicles with Streamers for Geotechnical Surveys: Modeling, Positioning, and Control. *IFAC-PapersOnLine*. 34, 78

ANTONELLI, G. (2009). Stability analysis for prioritized closed-loop inverse kinematic algorithms for redundant robotic systems. *IEEE Transactions on Robotics*, **25**, 985–994. 5

ANTONELLI, G. & CHIAVERINI, S. (1998). Task-priority redundancy resolution for underwater vehicle-manipulator systems. In *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*, vol. 1, 768–773 vol.1. 5

ANTONELLI, G. & CHIAVERINI, S. (2003). Fuzzy redundancy resolution and motion coordination for underwater vehicle-manipulator systems. *IEEE Transactions on Fuzzy Systems*, **11**, 109–120. 5

ANTONELLI, G., ARRICHIELLO, F. & CHIAVERINI, S. (2008). The null-space-based behavioral control for autonomous robotic systems. *Intelligent Service Robotics*, **1**, 27–39. 5

ANTONELLI, G., INDIVERI, G. & CHIAVERINI, S. (2009). Prioritized closed-loop inverse kinematic algorithms for redundant robotic systems with velocity saturations. 5892–5897. 16

BAERLOCHER, P. & BOULIC, R. (2004). An inverse kinematics architecture enforcing an arbitrary number of strict priority levels. *The Visual Computer*, **20**, 402–417. 6

BAY, H., TUYTELAARS, T. & VAN GOOL, L. (2006). SURF: Speeded up robust features. In A. Leonardis, H. Bischof & A. Pinz, eds., *Computer Vision – ECCV 2006*, 404–417, Springer Berlin Heidelberg, Berlin, Heidelberg. 89

BINGHAM, B., FOLEY, B., SINGH, H., CAMILLI, R., DELAPORTA, K., EUSTICE, R., MALLIOS, A., MINDELL, D., ROMAN, C. & SAKELLARIOU, D. (2010). Robotic tools for deep water archaeology: Surveying an ancient shipwreck with an autonomous underwater vehicle. *Journal of Field Robotics*, **27**, 702–717. 2

BRADSKI, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*. 62, 76

CANNY, J. (1986). A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **PAMI-8**, 679–698. 64, 86

CAPOCCI, R., DOOLY, G., OMERDIC, E., COLEMAN, J., NEWE, T. & TOAL, D. (2017). Inspection-class remotely operated vehicles—a review. *Journal of Marine Science and Engineering*, **5**, 13. 2

CARRERA, A., PALOMERAS, N., HURTOS, N., KORMUSHEV, P. & CARRERAS, M. (2014). Learning by demonstration applied to underwater intervention. vol. 269. 4

CASALINO, G., ANGELETTI, D., BOZZO, T. & MARANI, G. (2001). Dexterous underwater object manipulation via multi-robot cooperating systems. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, vol. 4, 3220–3225 vol.4. 3

CASALINO, G., ANGELETTI, D., CANNATA, G. & MARANI, G. (2002). The functional and algorithmic design of AMADEUS multirobot workcell. In S.K. Choi & J. Yuh, eds., *Underwater Vehicle Technology*, vol. 12. 3

CASALINO, G., TURETTA, A., SORBARA, A. & SIMETTI, E. (2009). Self-organizing control of reconfigurable manipulators: A distributed dynamic programming based approach. In *2009 ASME/IFToMM International Conference on Reconfigurable Mechanisms and Robots*, 632–640. 5

CASALINO, G., ZEREIK, E., SIMETTI, E., TORELLI, S., SPERINDÉ, A. & TURETTA, A. (2012). Agility for underwater floating manipulation task and subsystem priority based control strategy. In *International Conference on Intelligent Robots and Systems (IROS 2012)*, 1772–1779. 3

CASALINO, G., CACCIA, M., CAITI, A., ANTONELLI, G., INDIVERI, G., MELCHIORRI, C. & CASELLI, S. (2014). MARIS: A national project on marine robotics for interventions. In *22nd Mediterranean Conference on Control and Automation*, 864–869. 4, 7

CHANG, R.J., Y. LIN, C. & S. LIN, P. (2011). Visual-based automation of peg-in-hole microassembly process. *Journal of Manufacturing Science and Engineering*, **133**, 041015. 7

CHENG CHANG, C., YUAN CHANG, C. & TING CHENG, Y. (2004). Distance measurement technology development at remotely teleoperated robotic manipulator system for underwater constructions. 333 – 338. 2

CHHATPAR, S.R. & BRANICKY, M.S. (2001). Search strategies for peg-in-hole assemblies with position uncertainty. In *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No.01CH37180)*, vol. 3, 1465–1470 vol.3. 7

CHIAVERINI, S. (1997). Singularity-robust task-priority redundancy resolution for real-time kinematic control of robot manipulators. *IEEE Transactions on Robotics and Automation*, **13**, 398–410. 5

CHRIST, R. & WERNLI, R. (2013). *The ROV Manual: A User Guide for Remotely Operated Vehicles*. Elsevier, 2nd edn. 2

CIESLAK, P., RIDAO, P. & GIERGIEL, M. (2015). Autonomous underwater panel operation by GIRONA500 UVMS: A practical approach to autonomous underwater manipulation. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 529–536. 4

COMPORT, A., MARCHAND, E., PRESSIGOUT, M. & CHAUMETTE, F. (2006). Real-time markerless tracking for augmented reality: the virtual visual servoing framework. *IEEE Transactions on Visualization and Computer Graphics*, **12**, 615–628. 67

COOK, D., VARDY, A. & LEWIS, R. (2014). A survey of AUV and robot simulators for multi-vehicle operations. In *2014 IEEE/OES Autonomous Underwater Vehicles (AUV)*, 1–8. 30

DI LILLO, P.A., SIMETTI, E., DE PALMA, D., CATALDI, E., INDIVERI, G., ANTONELLI, G. & CASALINO, G. (2016). Advanced ROV autonomy for efficient remote control in the DexROV project. *Marine Technology Society Journal*, **50**. 4

DIAZ LEDEZMA, F., AMER, A., ABDELLATIF, F., OUTA, A., TRIGUI, H., PATEL, S. & BINYAHIB, R. (2015). A market survey of offshore underwater robotic inspection technologies for the oil and gas industry. 2

DIETRICH, F., BUCHHOLZ, D., WOBBE, F., SOWINSKI, F., RAATZ, A., SCHUMACHER, W. & WAHL, F.M. (2010). On contact models for assembly tasks: Experimental investigation beyond the peg-in-hole problem on the example of force-torque maps. In

2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2313–2318. 7

DJAPIC, V., NAĐ, D., FERRI, G., OMERDIC, E., DOOLY, G., TOAL, D. & VUKIĆ, Z. (2013). Novel method for underwater navigation aiding using a companion underwater robot as a guiding platforms. In *2013 MTS/IEEE OCEANS - Bergen*, 1–10. 2

DRAP, P. (2012). Underwater photogrammetry for archaeology. *Special Applications of Photogrammetry*. 2

DUDA, R.O. & HART, P.E. (1972). Use of the hough transformation to detect lines and curves in pictures. *Commun. ACM*, **15**, 11–15. 86

E. ROHMER, M.F., S. P. N. SINGH (2013). V-REP: a versatile and scalable robot simulation framework. In *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*. 30

ESCANDE, A., MANSARD, N. & WIEBER, P.B. (2014). Hierarchical quadratic programming: Fast online humanoid-robot motion generation. *I. J. Robotics Res.*, **33**, 1006–1028. 6

EVANS, J., REDMOND, P., PLAKAS, C., HAMILTON, K. & LANE, D. (2003). Autonomous docking for Intervention-AUVs using sonar and video-based real-time 3D pose estimation. In *Oceans 2003. Celebrating the Past ... Teaming Toward the Future (IEEE Cat. No.03CH37492)*, vol. 4, 2201–2210 Vol.4. 3

EVANS, J.C., KELLER, K.M., SMITH, J.S., MARTY, P. & RIGAUD, O.V. (2001). Docking techniques and evaluation trials of the SWIMMER AUV: an autonomous deployment AUV for work-class ROVs. In *MTS/IEEE Oceans 2001. An Ocean Odyssey. Conference Proceedings (IEEE Cat. No.01CH37295)*, vol. 1, 520–528 vol.1. 3

FAVERJON, B. & TOURNASSOUD, P. (1987). A local based approach for path planning of manipulators with a high number of degrees of freedom. In *Proceedings. 1987 IEEE International Conference on Robotics and Automation*, vol. 4, 1152–1159. 6

FISCHLER, M.A. & BOLLES, R.C. (1981). Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, **24**, 381–395. 89

FLACCO, F., DE LUCA, A. & KHATIB, O. (2012). Prioritized multi-task motion control of redundant robots under hard joint constraints. 3970–3977. 5

FLETCHER, B. (2000). Worldwide undersea MCM vehicle technologies. 10. 2

GILMOUR, B., NICCUM, G. & O'DONNELL, T. (2012). Field resident AUV systems — chevron's long-term goal for AUV development. In *2012 IEEE/OES Autonomous Underwater Vehicles (AUV)*, 1–5. 2

GUENNEBAUD, G., JACOB, B. *et al.* (2010). Eigen v3 [online; accessed 29-june-2019]. 79

KANOUN, O., LAMIRAUX, F. & WIEBER, P. (2011). Kinematic control of redundant manipulators: Generalizing the task-priority framework to inequality task. *IEEE Transactions on Robotics*, **27**, 785–792. 6

KERMORGANT, O. (2014). A dynamic simulator for underwater vehicle-manipulators. In *International Conference on Simulation, Modeling, and Programming for Autonomous Robots Simpar*, Springer, Bergamo, Italy. 30, 77

KHATIB, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, **5**, 90–98. 5

KHATIB, O. (1987). A unified approach for motion and force control of robot manipulators: The operational space formulation. *IEEE Journal on Robotics and Automation*, **3**, 43–53. 4

LANE, D.M., DAVIES, J.B.C., CASALINO, G., BARTOLINI, G., CANNATA, G., VERUGGIO, G., CANALS, M., SMITH, C., O'BRIEN, D.J., PICKETT, M., ROBINSON, G., JONES, D., SCOTT, E., FERRARA, A., ANGELLETI, D., COCCOLI, M., BONO, R., VIRGILI, P., PALLAS, R. & GRACIA, E. (1997). AMADEUS: advanced manipulation for deep underwater sampling. *IEEE Robotics Automation Magazine*, **4**, 34–45. 3

LANE, D.M., MAURELLI, F., KORMUSHEV, P., CARRERAS, M., FOX, M. & KYRIAKOPOULOS, K. (2012). Persistent autonomy: the challenges of the PANDORA project. *IFAC Proceedings Volumes*, **45**, 268 – 273, 9th IFAC Conference on Manoeuvring and Control of Marine Craft. 4

LEE, H. & PARK, J. (2014). An active sensing strategy for contact location without tactile sensors using robot geometry and kinematics. *Autonomous Robots*, **36**, 109–121. 7

LEE, J., MANSARD, N. & PARK, J. (2012). Intermediate desired value approach for task transition of robots in kinematic control. *IEEE Transactions on Robotics*, **28**, 1260–1277. 6

LOWE, D.G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, **60**, 91–110. 89

LOZANO-PÉREZ, T., MASON, M.T. & TAYLOR, R.H. (1984). Automatic synthesis of fine-motion strategies for robots. *The International Journal of Robotics Research*, **3**, 3–24. 8

MACIEJEWSKI, A.A. & KLEIN, C.A. (1985). Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments. *The International Journal of Robotics Research*, **4**, 109–117. 5

MANSARD, N., KHATIB, O. & KHEDDAR, O. (2009a). A unified approach to integrate unilateral constraints in the stack of tasks. *IEEE Transactions on Robotics*, **25**, 670–685. 6

MANSARD, N., REMAZEILLES, A. & CHAUMETTE, F. (2009b). Continuity of varying-feature-set control laws. *IEEE Transactions on Automatic Control*, **54**, 2493–2505. 6

MARANI, G., KIM, J., YUH, J. & CHUNG, W. (2003). Algorithmic singularities avoidance in task-priority based controller for redundant manipulators. vol. 4, 3570 – 3574 vol.3. 5

MARANI, G., CHOI, S.K. & YUH, J. (2009). Underwater autonomous manipulation for intervention missions AUVs. *Ocean Engineering*, **36**, 15 – 23, autonomous Underwater Vehicles. 3

MARCHAND, E., SPINDLER, F. & CHAUMETTE, F. (2005). ViSP for visual servoing: a generic software platform with a wide class of robot control skills. *IEEE Robotics and Automation Magazine*, **12**, 40–52. 62, 67, 76

MARTY, P. (2004). ALIVE: An autonomous light intervention vehicle. *Scandinavian Oil-Gas Magazine*, **32**. 3

MATAS, J., GALAMBOS, C. & KITTLER, J. (2000). Robust detection of lines using the progressive probabilistic hough transform. *Computer Vision and Image Understanding*, **78**, 119–137. 86

MICHEL, O. (2004). Webots: Professional mobile robot simulation. *Journal of Advanced Robotics Systems*, **1**, 39–42. 30

MIURA, J. & IKEUCHI, K. (1998). Task-oriented generation of visual sensing strategies in assembly tasks. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, **20**, 126 – 138. 7

MUJA, M. & LOWE, D.G. (2012). Fast matching of binary features. In *Computer and Robot Vision (CRV)*, 404–410. 89

NAKAMURA, Y. (1990). *Advanced Robotics: Redundancy and Optimization*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edn. 4

NAKAMURA, Y. & HANAFUSA, H. (1986). Inverse kinematic solutions with singularity robustness for robot manipulator control. *Journal of Dynamic Systems, Measurement, and Control*. 4, 5

NENCHEV, D.N. & SOTIROV, Z.M. (1994). Dynamic task-priority allocation for kinematically redundant robotic mechanisms. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'94)*, vol. 1, 518–524 vol.1. 6

NEWMAN, W., ZHAO, Y. & PAO, Y.H. (2001). Interpretation of force and moment signals for compliant peg-in-hole assembly. vol. 1, 571 – 576 vol.1. 7

OH, J. & OH, J.H. (2015). A modified perturbation/correlation method for force-guided assembly. *Journal of Mechanical Science and Technology*, **29**, 5437–5446. 7

PADIR, T. (2005). Kinematic redundancy resolution for two cooperating underwater vehicles with on-board manipulators. vol. 4, 3137 – 3142 Vol. 4. 5

PARAVISI, M., H. SANTOS, D., JORGE, V., HECK, G., GONÇALVES, L.M. & AMORY, A. (2019). Unmanned surface vehicle simulator with realistic environmental disturbances. *Sensors*, **19**. 30, 31

PARK, H., BAE, J.H., PARK, J.H., BAEG, M.H. & PARK, J. (2013). Intuitive peg-in-hole assembly strategy with a compliant manipulator. In *IEEE ISR 2013*, 1–5. 8

PARK, H., PARK, J., LEE, D.H., PARK, J.H., BAEG, M.H. & BAE, J.H. (2017). Compliance-based robotic peg-in-hole assembly strategy without force feedback. *IEEE Transactions on Industrial Electronics*, **PP**, 1–1. 8

PAULI, J., SCHMIDT, A. & SOMMER, G. (2001). Vision-based integrated system for object inspection and handling. *Robotics and Autonomous Systems*, **37**, 297 – 309. 7

PEREZ, T. & FOSSEN, T.I. (2007). Kinematic models for manoeuvring and seakeeping of marine vessels. *Modeling, Identification and Control*, **28**, 19–30. 11

PRATS, M., PÉREZ, J., FERNÁNDEZ, J.J. & SANZ, P.J. (2012). An open source tool for simulation and supervision of underwater intervention missions. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2577–2582. 29

PRATS, M., ROMAGÓS, D., PALOMERAS, N., GARCÍA SÁNCHEZ, J.C., NANNEN, V., WIRTH, S., FERNANDEZ, J., P. BELTRÁN, J., CAMPOS, R., RIDAO, P., SANZ, P., OLIVER, G., CARRERAS, M., GRACIAS, N., MARÍN PRADES, R. & ORTIZ, A. (2012). Reconfigurable AUV for intervention missions: A case study on underwater object recovery. *Intelligent Service Robotics*, **5**, 19–31. 3

PRESSIGOUT, M. & MARCHAND, E. (2007). Real-time hybrid tracking using edge and texture information. *The International Journal of Robotics Research*, **26**, 689–713. 67

PROMETEO (2016). http://www.irs.uji.es/prometeo/, [online; accessed 25-october-2018]. 4

RIGAUD, V., COSTE-MANIERE, E., ALDON, M.J., PROBERT, P., PERRIER, M., RIVES, P., SIMON, D., LANG, D., KIENER, J., CASAL, A., AMAR, J., DAUCHEZ, P. & CHANTLER, M. (1998). UNION: underwater intelligent operation and navigation. *IEEE Robotics Automation Magazine*, **5**, 25–35. 3

ROBUST (2016). http://eu-robust.eu, [online; accessed 25-october-2018]. 4

RUSU, R.B. & COUSINS, S. (2011). 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China. 62, 69

SANZ, P., RIDAO, P., OLIVER, G., CASALINO, G., INSAURRALDE, C., SILVESTRE, C., MELCHIORRI, C. & TURETTA, A. (2012). TRIDENT: Recent improvements about autonomous underwater intervention missions. vol. 3, 1–10. 3, 7

SCHEMPF, H. & YOERGER, D.R. (1992). Coordinated vehicle/manipulation design and control issues for underwater telemanipulation. *IFAC Proceedings Volumes*, **25**, 259 – 267, iFAC Workshop on Artificial Intelligence Control and Advanced Technology in Marine Automation (CAMS '92), Genova, Italy, April 8-10. 3

SENTIS, L. & KHATIB, O. (2005). Control of free-floating humanoid robots through task prioritization. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, 1718–1723. 5

SHI, J. & TOMASI, C. (2000). Good features to track. *Proceedings / CVPR, IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, **600**, 593–600. 85

SHIRINZADEH, B., ZHONG, Y., TILAKARATNA, P.D.W., TIAN, Y. & DALVAND, M.M. (2011). A hybrid contact state analysis methodology for robotic-based adjustment of cylindrical pair. *The International Journal of Advanced Manufacturing Technology*, **52**, 329–342. 7

SICILIANO, B. & SLOTINE, J..E. (1991). A general framework for managing multiple tasks in highly redundant robotic systems. In *Fifth International Conference on Advanced Robotics 'Robots in Unstructured Environments*, 1211–1216 vol.2. 5

SICILIANO, B., SCIAVICCO, L., VILLANI, L. & ORIOLO, G. (2009). *Robotics: Modelling, Planning and Control*, 147–151. Springer-Verlag London. 33

SIMETTI, E. & CASALINO, G. (2016). A novel practical technique to integrate inequality control objectives and task transitions in priority based control. *Journal of Intelligent & Robotic Systems*, **84**. 4, 7, 16, 25, 80

SIMETTI, E. & CASALINO, G. (2017). Manipulation and transportation with cooperative underwater vehicle manipulator systems. *IEEE Journal of Oceanic Engineering*, **42**, 782–799. 4, 10, 17, 26, 33, 34, 40

SIMETTI, E., TURETTA, A. & CASALINO, G. (2009). *Distributed Control and Coordination of Cooperative Mobile Manipulator Systems*, 315–324. Springer Berlin Heidelberg, Berlin, Heidelberg. 5

SIMETTI, E., CASALINO, G., TORELLI, S., SPERINDÉ, A. & TURETTA, A. (2014a). Floating underwater manipulation: Developed control methodology and experimental validation within the TRIDENT project. *Journal of Field Robotics*, **31(3)**, 364–385. 3, 7

SIMETTI, E., CASALINO, G., TORELLI, S., SPERINDÉ, A. & TURETTA, A. (2014b). Underwater floating manipulation for robotic interventions. *IFAC Proceedings Volumes*, **47**, 3358 – 3363, 19th IFAC World Congress. 7

SIMETTI, E., CASALINO, G., WANDERLINGH, F. & AICARDI, M. (2018). Task priority control of underwater intervention systems: Theory and applications. *Ocean Engineering*, **164**, 40 – 54. 4, 10, 22, 23, 26

SONG, H., KIM, Y. & SONG, J.B. (2016). Guidance algorithm for complex-shape peg-in-hole strategy based on geometrical information and force control. *Advanced Robotics*, 1–12. 7

SUGIURA, H., GIENGER, M., JANSSEN, H. & GOERICK, C. (2007). Real-time collision avoidance with whole body motion control for humanoid robots. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2053–2058. 5

SUZUKI, S. & BE, K.A. (1985). Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, **30**, 32 – 46. 64, 87

TRINH, S., SPINDLER, F., MARCHAND, E. & CHAUMETTE, F. (2018). A modular framework for model-based visual tracking using edge, texture and depth features. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IROS'18*, Madrid, Spain. 68

TWINBOT (2019). http://www.irs.uji.es/twinbot/twinbot.html, [online; accessed 29-june-2019]. 1, 8, 60, 77, 78

URABE, T., URA, T., TSUJIMOTO, T. & HOTTA, H. (2015). Next-generation technology for ocean resources exploration (zipangu-in-the-ocean) project in japan. 1–5. 2

WANDERLINGH, F. (2018). Cooperative Robotic Manipulation for the Smart Factory. Ph.D. thesis, *Università degli Studi di Genova*. 10, 21, 26, 40

WYNN, R., HUVENNE, V., LE BAS, T., MURTON, B., CONNELLY, D., BETT, B., RUHL, H., MORRIS, K., PEAKALL, J., PARSONS, D., J. SUMNER, E., E. DARBY, S., DORRELL, R. & HUNT, J. (2014). Autonomous underwater vehicles (AUVs): Their past, present and future contributions to the advancement of marine geoscience. *Marine Geology*, **352**. 2

XU, Q. (2015). Robust impedance control of a compliant microgripper for high-speed position/force regulation. *IEEE Transactions on Industrial Electronics*, **62**, 1201–1209. 8

YOSHIKAWA, T. (1984). Analysis and Control of Robot Manipulators with Redundancy. In M. Brady & R. Paul, eds., *Robotics Research The First International Symposium*, 735–747, MIT Press. 5

YUH, J., CHOI, S.K., IKEHARA, C., KIM, G.H., MCMURTY, G., GHASEMI-NEJHAD, M., SARKAR, N. & SUGIHARA, K. (1998). Design of a semi-autonomous underwater vehicle for intervention missions (SAUVIM). In *Proceedings of 1998 International Symposium on Underwater Technology*, 63–68. 3

ZEREIK, E., SORBARA, A., MERLO, A., SIMETTI, E., CASALINO, G. & DIDOT, F. (2011). Space robotics supporting exploration missions: vision, force control and coordination strategy for crew assistants. *Intelligent Service Robotics*, **4**, 39–60. 5