Universitat Jaume I de Castellón
Departamento de Ingeniería y ciencia de los computadores

PhD Thesis

Visibility in underwater robotics: Benchmarking and single image dehazing

Author: Javier Pérez Soler

Supervisors: Pedro J. Sanz Valero
Raúl Marín Prades

Castellón de la Plana, May 2017

A mis padres y hermanos.

Acknowledgements

This thesis is the result of more than 5 years of work and it would have been impossible without the support of family, friends and colleagues. May the reader forgive me the discourtesy of following my wish to acknowledge their support in my mother tongue.

But before changing to my mother tongue, I want to thank professor Stefan Williams for hosting me at the Australian Centre for Field Robotics in Sydney for four months. Working with his group, and specially with Mitch Bryson and Dushyant Rao, was crucial for the development of the final chapters of this thesis. I also have to acknowledge the ACFR for providing and preparing the datasets used in the deep learning experiments.

En primer lugar, me gustaría agradecer a mis directores Raúl Marín y Pedro Sanz la oportunidad de poder trabajar en un campo tan motivante y desafiante como la robótica submarina a través de distintos proyectos en la frontera del conocimiento como Trident, Triton o Merbots. Además de por la ayuda y los consejos desde la experiencia a la hora de introducirme en la investigación.

Gracias a la Universitat Jaume I por haber financiado durante 4 años el desarrollo de esta tesis a través del plan de promoción de la investigación (referencia de la ayuda PREDOC/2012/47).

También tengo que dar las gracias a todos los miembros del IRSLab de los que he podido aprender, y con los que he colaborado y trabajado como un verdadero equipo durante estos años. A todos quiero dar gracias, Juan Carlos por su disponibilidad a la hora de ayudar, Javi por su espíritu de equipo, Toni por su decisión a la hora de afrontar problemas, David por sus ánimos y voluntad, Jorge por su insistencia en publicar, José Bernabé por sus consejos y Diego por su entrega.

Gracias también a Mario, pese al poco tiempo que coincidimos demostró ser un investigador sobresaliente y un compañero ejemplar que me ayudó en los primeros momentos de este trabajo. Además también tengo que agradecer al RobInLab, especialmente a Ángel, Carlos, Marco, Xavi, Gabriel, Majd y Gustavo, por su ayuda y discusiones en los descansos delante de una taza de café.

Durante mi estancia en Sydney tuve la suerte de coincidir con un pequeño grupo de españoles que hizo la estancia mucho más amena. Por ello también quiero darles las gracias, en particular a Guillem y a Raúl con los cuales compartí mucho tiempo dentro y fuera del laboratorio visitando y conociendo Australia, y en general a todas las personas con las que coincidí durante esos meses.

Gracias a mi familia y amigos de la Venta del Moro, siempre dispuestos a escuchar después de una larga semana de trabajo. Durante estos años han sido

una distracción necesaria a los problemas del día a día, haciendo el camino de esta tesis mucho más entretenido.

Quiero también agradecer a Loli por ser una más de la familia y preocuparse y cuidar de mí día a día desde bien pequeño, ayudándome a convertirme en una persona mejor.

Por último, quiero agradecer especialmente la ayuda y el apoyo de mi familia. A mis padres Luis y María Ángeles que me han dado una educación enseñándome a esforzarme para alcanzar cualquier meta. A Luis por estar siempre dispuesto a ayudar y ser un ejemplo de sacrificio y a Emilio que ha sido un director en la sombra aconsejándome en nuestros trayectos de ida y vuelta a la UJI.

Resumen

Una de las dificultades más grandes en la robótica autónoma submarina es lidiar con la falta de visibilidad en imágenes submarinas. La transmisión de la luz en el agua degrada las imágenes dificultando reconocer objetos, plantas o animales y consecuentemente poniendo en peligro el desarrollo de la intervención. Además, tecnologías comúnmente utilizadas en la robótica como los sensores infrarrojos y las cámaras que permiten recuperar información en 3D no son viables debido a la rápida atenuación de la luz infrarroja bajo el agua. Sin embargo, si se consiguiera superar estos problemas y el robot fuera capaz de percibir e interpretar correctamente el entorno supondría un gran avance en la robótica submarina que tendría un gran impacto en la sociedad facilitando la creación de estaciones de gas y petróleo *offshore*, investigación del océano, arqueología submarina o investigación de accidentes aéreos además de aplicaciones militares y recreacionales.

Las contribuciones de esta tesis son el análisis del impacto de la degradación de las imágenes submarinas usadas en algoritmos de visión, y propuestas para mitigarlo introduciendo un paso previo que permita recuperar los colores de la imagen en tiempo real.

En primer lugar, dada la ausencia de herramientas capaces de simular y analizar las intervenciones submarinas desde la perspectiva de la investigación, se ha desarrollado una arquitectura *software* para la investigación en robótica submarina. Esta arquitectura está formada por un simulador de robots submarinos denominado UWSim, y una herramienta de *benchmarking* capaz de evaluar objetivamente algoritmos independientemente de cómo estén implementados y de donde provengan. También se han añadido servicios en la nube a la arquitectura que permiten lanzar simulaciones y benchmarks de manera remota, ofreciendo un espacio para comparar resultados entre los usuarios de la herramienta.

El simulador UWSim es un proyecto de código abierto, disponible para la comunidad científica y que ya ha sido utilizado en múltiples proyectos financiados por la comunidad europea y por centros de investigación como herramienta de simulación de intervenciones submarinas. Por otra parte, la herramienta de *benchmarking* ha sido utilizada en experimentos de diferente índole para evaluar o comparar soluciones tan diversas como seguimiento de tuberías, interfaces de usuario, dragado o extraer estadísticas de intervención. Sin embargo, el principal uso en esta tesis es para analizar las consecuencias que produce la turbidez submarina en algoritmos de visión como seguimiento de objetos, mantener la posición o reconstrucción de objetos en 3 dimensiones. Para ello se han llevado a cabo experimentos en simulación, condiciones de laboratorio y mezclando ambas (*Hardware In the Loop*).

Teniendo en cuenta los resultados de estos experimentos, donde se puede ver que los resultados de visión son dependientes en gran medida de la turbidez del agua, también se ha realizado un estudio de diferentes alternativas para eliminar la turbidez en las imágenes. Para ello imágenes de diferentes lugares se han utilizado para evaluar algoritmos enfocados a recuperar los colores de las imágenes con una sola imagen como entrada.

Para evaluar estos algoritmos se utilizan mapas de profundidad obtenidos a partir de cámaras estéreo y reconstrucción 3D a partir del movimiento que se comparan con resultados intermedios de los mismos. Generalmente los algoritmos que eliminan la turbidez necesitan estimar la transmisión del medio, la cual está directamente relacionada con el mapa de profundidad. Por tanto, es posible establecer métricas objetivas para evaluarlos utilizando los mapas de profundidad como datos reales.

Por último, se han desarrollado dos soluciones basadas en *deep learning* que permiten corregir la degradación producida por la turbidez del agua a partir de una única imagen degradada. Estas soluciones explotan la capacidad de las redes neuronales convolucionales de procesar imágenes adaptando el problema de restablecer los colores para utilizarlo en una arquitectura convolucional. Las imágenes de entrada se utilizan para entrenar una red neural que será capaz de producir imágenes mejoradas y servirán de entrada a otros algoritmos de visión.

La primera solución propuesta utiliza pares de imágenes degradadas y restauradas a partir de un algoritmo que requiere una gran cantidad de imágenes y mapas de profundidad del mismo lugar para recuperar los colores sin degradar. Estos datos se utilizan para entrenar una red neural que es capaz de aprender la transformación y directamente producir imágenes mejoradas para procesar en algoritmos de visión a partir de una única imagen.

La segunda alternativa parte del punto de vista de la restauracion de imagenes donde se necesita un modelo de formación de imagen para recuperar los colores de la imagen. Uno de los parámetros fundamentales en este proceso es el mapa de profundidad, por ello se utiliza una red neuronal para obtenerlo. Los resultados muestran que no solo es posible hacer esto bajo el agua, sino que se consiguen mejores resultados que en el aire al contar con la atenuación de la luz como pista de la distancia. Utilizando este mapa de profundidad es posible estimar el resto de parámetros y restaurar la imagen utilizando el modelo de formación de la imagen.

Abstract

Dealing with underwater visibility is one of the most important challenges in autonomous underwater robotics. The light transmission in the water medium degrades images making the interpretation of the scene difficult and consequently compromising the whole intervention, that usually depends on a correct understanding of the environment. Furthermore, it is not possible to use commonly used technologies in robotics such as infrared projectors and associated cameras to detect 3D information due to the fast attenuation of infrared light underwater. However, being able to perceive and interpret the environment would represent an enormous advance in underwater robotics that would generate a great impact on society, facilitating oil and gas offshore industry, research of the ocean, underwater archaeology or air crash investigations besides the military and hobby applications.

This thesis contributes, by analysing the impact of the underwater image degradation in commonly used vision algorithms, and proposes to reduce these problems by introducing a single image dehazing step capable of running in real time.

Firstly, a benchmarking framework for underwater robotics research is proposed, motivated by the lack of suitable tools capable of simulating and analysing an autonomous underwater intervention. The developed framework is made of an underwater simulator (UWSim), and a benchmarking suite able to objectively evaluate software independently of its implementation or source. Furthermore, the framework has been extended to be used as a cloud service, making it possible to remotely launch simulations and benchmarks, automatically comparing the results with other researchers using the software.

The simulator this is presented here, is offered as an open source tool for the scientific community and has already been used in different EU funded projects and research institutions as their simulation software for underwater interventions. Regarding the benchmarking suite, several experiments have been conducted to evaluate the developed software and compare it with solutions that have already been proposed in the literature in different problems such as pipe following, natural interfaces, dredging or intervention statistics. However, the main effort has been made in analysing the effects of underwater visibility in vision algorithms like tracking, station keeping and 3D reconstruction including simulation, hardware in the loop and real experiments.

Secondly, motivated by the results of these benchmarking experiments that show a performance drop when the water is turbid, study of different alternatives to dehaze the image is presented. A group of images from different locations are used to evaluate algorithms capable of restoring the original colors of images with

a single image as input.

In order to evaluate the solutions, the depthmap retrieved with a stereo camera and 3D from motion is compared with side results of the dehazing algorithms. With the dehazing schemes it is usually necessary to estimate the transmission of the medium which is directly correlated to depth, thus making it possible to establish an objective metric for the comparison.

Finally, two approaches based on deep learning are proposed to correctly dehaze images using a still image as input. Exploiting the performance of convolutional neural networks in the image processing field, the dehazing problem is adapted to use a set of images for training and then is able to produce haze free images. The first approach makes use of restored images, obtained from an algorithm that requires several images and depthmaps from the same location to recover the original colors, in order to train a neural network to dehaze images.

A second alternative is proposed from the perspective of image restoration, using an image formation model to restore the degraded image. As the key parameter required to perform this restoration is the depthmap, a deep neural network is trained to estimate it. The trained neural network shows the underwater haze is a useful indicator for depth estimation as the attenuation is directly related to it. Using this estimation it is possible to restore degraded images and consequently enhance the performance of the autonomous underwater vehicle interventions.

Índice general

Capítulo 1

Introduction

The oceans cover more then 2/3 of the Earth's surface and the history of humanity has been critically conditioned by them throughout time as described in [Antonelli et al., 2008]. The oceans have provided a source of food through fishing and served as a highway for commerce and communication between nations. Nowadays, they are also an important source of food and other resources of utmost importance such as oil and gas or pearls used in jewelry. Commercially valuable minerals are also extracted from the sea such as salt, bromine, and magnesium.

Furthermore, the ocean is a key component of the ecosystem and specially of the climate system. According to [Bigg et al., 2003], it provides a temperature boundary for the atmosphere over 70 % of the globe. It also absorbs over 97 % of solar radiation incident on it at certain angles and provides 85 % of the water vapour in the atmosphere among other major ecosystem effects.

However, the scientific exploration of the sea is still far from complete. The knowledge of the ocean is mainly restricted to relatively shallow waters. The resources and understanding, chemical, geological, or even archaeological, of the deeper waters remain a mystery. Although, this information is physically closer to the humans the difficulties to explore it make it impossible in some cases.

The first explorations of the oceans were conducted through human occupied vehicles that highly restricted the depth limits that could be reached. These vehicles were substituted by Remotely Operated Vehicles (ROV's) that avoided risking human lives in the process.

Furthermore, ROV's proved to be a valuable tool reaching depths that greatly exceed the range of human divers. This new limits allowed to use them in the exploitation of offshore oil and deploying and maintaining underwater structures such as pipelines or underwater cables. Moreover, the ROV's are capable of reaching scientifically interesting zones, where they are able to recover information that is especially interesting for chemists archaeologists and geologists, making possible their study by researchers using cameras and bathymetic information.

However, these kinds of interventions are launched from support vessels, and remotely operated by expert pilots through an umbilical communications cable and complex control interfaces. ROV's are normally large and heavy vehicles that need significant logistics for their transportation and handling. Additionally, the complex user interfaces and control methods require skilled pilots for their use. These two

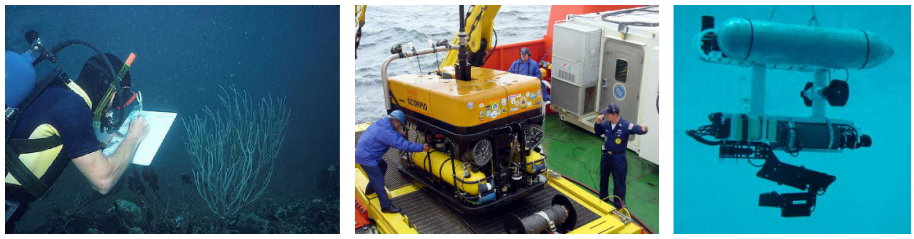

Figura 1.1: Ocean exploration evolution, from left to right diver, remote operated vehicle and autonomous underwater vehicle.

facts significantly increase the cost of this type of applications. Moreover, the need of an umbilical cable introduces additional problems of control, or range limitation. Finally, the fatigue and high stress that users of remotely operated systems normally suffer is another serious drawback.

Taking this into account, cheap and easy-to-use solutions for underwater interventions are needed. In order to solve this, a new concept that seeks for higher autonomy levels in underwater interventions named Intervention Autonomous Underwater Vehicles (I-AUV) arose [De Novi et al., 2009]. However, this technology is still at an early stage of development, so further research is required to reach higher levels of autonomy so that it may be used in a wider range of applications. The figure 1.1 shows this evolution of the ocean exploration.

Poor visibility is one of the principal problems for I-AUVs as it has to make sense of its environment through a camera. In this thesis underwater visibility degradation is studied from a benchmarking perspective and a solution for single image dehazing is proposed. For this purpose, different dehazing techniques are compared and finally a machine learning strategy is designed to achieve the desired features.

1.1. Previous research projects

The first pioneering works in the field of autonomous underwater robotics took place during the early 90s such as the OTTER AUV [Wang et al., 1995], ODIN vehicle [Choi et al., 1994] or the UNION project [Rigaud et al., 1998]. However, the first simple autonomous intervention arrived in the last decade when the systems were capable of demonstrating these features at sea.

The AMADEUS project, presented in [Lane et al., 1997], was the first attempt at developing a dexterous gripper for underwater applications. A 3-fingered hydraulic gripper was designed to mimic the motions of an artificial elephant trunk and manipulate objects. In a second phase the project added two coordinated arms with 7 degrees of freedom. The project started in 1993 and lasted until 1999.

During the 1996 to 1999 period, the Union project described in [Rigaud et al., 1998] focused on developing methods in order to increase the autonomy and intelligence of ROVs. The main effort was centred on the development of coordinated control and sensing strategies for manipulator and vehicle. However, only experi-

mental validation in simulated environment was achieved.

In [Evans et al., 2001] a hybrid ROV/AUV configuration for the SWIMMER project is described where an AUV transports the ROV near a deepsea facility which it can then operate. Additionally the ROV is connected to the umbilical cable in the subsea location thus it can be controlled from the surface. The system demonstrates an efficient way for inspection, maintenance and repair of oil production stations. The project took place between 1999 and 2001.

Similarly, the ALIVE project presented in [Evans et al., 2003] aims to develop an autonomous vehicle designed to dock in an unknown subsea structure similar to the oil industry. Once docked the 7DOF manipulator can then operate the underwater panel turning valves or connecting hot stabs. After 4 years, in 2004 the final demo was able to demonstrate the capability to navigate, dock and operate the panel.

The SAUVIM project, described in [Yuh et al., 1998], that took place between 1999 and 2009 by the Office of Naval Research and carried out at the Autonomous System Laboratory of the University of Hawaii focused on the design of an AUV to recover missiles from the seabed. The vehicle is designed to be a semi-autonomous vehicle with a fully functional manipulator controlled by a supervisor in a land based station. The supervisor controls the vehicle and the arm movements, but in an ideal case the vehicle could carry out the mission autonomously.

The expertise of the IRSLab in underwater robotics starts with the RAUVI (Reconfigurable Autonomous Underwater Vehicle for Intervention Missions) project in 2009. The main goal of the project was to develop the AUV technology necessary for autonomously perform intervention missions in underwater environments. The case of study proposed is the recovery of an object in the seabed as described in [Prats et al., 2012c].

In order to do so, the approach is divided in two steps: survey and intervention. The survey phase explores the zone looking for objects of interest using visual and acoustic data. After that, the vehicle surfaces and the gathered information is retrieved in order to prepare the intervention mission. The intervention mission is specified using a human robot interface that describes the object to be recovered and the vehicle autonomously recovers it from the seafloor.

The project demonstrated autonomous recovery of a flight recorder black box using an autonomous vehicle and a 4 degrees of freedom robotic arm with a hook in a pool and shallow water conditions. The system was capable of autonomously surveying the zone searching for the black box mock-up in the survey phase and autonomously grasp it with a hook in the intervention.

1.2. Context

The research for this thesis has been conducted in the Interactive and Robotics laboratory (IRSLab) at the University Jaume I of Castellón. The group has been doing research in underwater robotics since 2009 with the RAUVI project, although the members of the group have been working in robotic manipulation for a long time as part of the robotic intelligence laboratory (RobInLab). The group expertise focuses on underwater robotic manipulation as in [Prats et al., 2012c] or [Peñalver et al., 2015] using the 4 degrees of freedom lightweight ARM5E described in [Fernández et al., 2013].

Besides this, the efforts of the IRSLab have also been oriented towards the development of other fields related to the development and control of autonomous underwater vehicles. An example of this is the work related to human machine interfaces in [Sanchez et al., 2015] or [Garcia et al., 2010] leading to innovative ways of supervising or controlling an AUV. In the field of computer vision the group also works in detecting graspable shapes from 3D point clouds that come from stereo cameras as in [Fornas et al., 2016]. In [Fernández et al., 2015] a free floating control of the vehicle and arm in a dredge intervention is showed. A new research line in the group is the underwater wireless communications as preliminary results in [Centelles et al., 2015] show.

The group is also the developer and maintainer of one of the most used underwater simulators: UWSim. The simulator development is part of this thesis and it is described in detail in Chapter 2. The software was designed as a tool for testing and integrating perception and control algorithms before running them on the real robots.

Although the study of low visibility in the underwater environment may seem to fall outside the scope of the group expertise, the problem was already studied in the GUARDIANS project. The test bed of this project was the navigation and aid to fire and rescue service in an industrial warehouse in smoke. Consequently, the laboratory had already studied algorithms to deal with poor visibility such as [Sales et al., 2010] or [Marti et al., 2012].

Furthermore, the work presented in this thesis falls within the scope of the different research projects in which the IRSLab has participated. An image of the experimental validation of the projects can be seen in figure 1.2. Thus, the research in it contributed to the following projects:

- FP6 EU Project **GUARDIANS**: Group of unmanned assistant robots deployed in aggregative navigation supported by scent detection (FP6-IST-045269) funded by the european community. The project proposes the use of a swarm of autonomous robots developed to navigate and search an urban ground. The main example is an industrial warehouse in smoke, a very dangerous situation where human senses can be severely impaired. Robots warn of toxical chemicals, provide and maintain communication links assisting in the intervention.

- FP7 EU Project **TRIDENT**: Marine robots and dexterous manipulation for enabling autonomous underwater multipurpose manipulation (FP7-ICT-2009-248497) funded by the european community. The project proposes a new methodology for underwater interventions combining autonomous tasks with supervision. Dexterous autonomous manipulation was demonstrated recovering a flight recorder mock-up in a pool and shallow water conditions[1].

- MINECO project **TRITON**: Multisensory Based Underwater Intervention through Cooperative Marine Robots (DPI2011-27977-C03) funded by the Spanish ministry. The project aims to develop technologies close to the real needs of the final user facilitating the technological transfer of its results. The test bed for this project was an experimental validation in an intervention

[1] Video of the TRIDENT final experiment https://www.youtube.com/watch?v=2qf7ukrUcCc

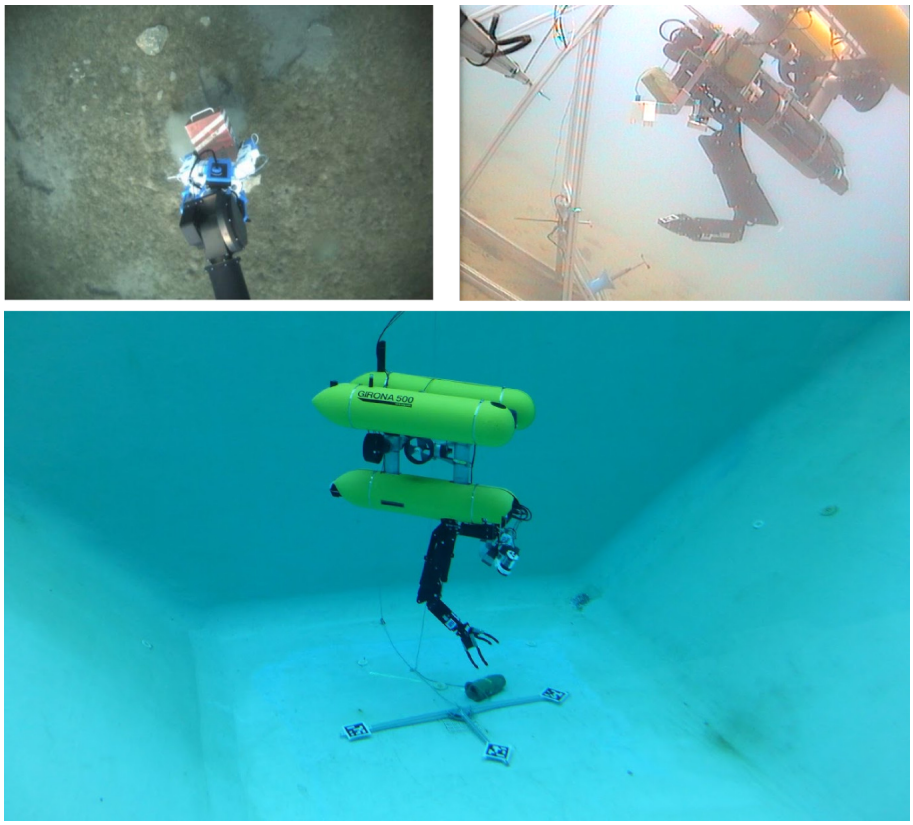

Figura 1.2: Images of the trials in the Trident (top left), Triton (top right) and Merbots (bottom) projects.

panel where the vehicle autonomously docked and manipulated a valve and hot stab in pool and sea conditions[2].

- UJI Project **MASUMIA**: Towards a predictive interface in the context of underwater robotics (P1 1B2011-17). The main goal of the project was to further develop the UWSim simulator and create a distributed and predictive interface for it. The results of the project show the development of abstraction layers and tools that make it possible to easily interact with the simulator.

- MINECO project **MERBOTS**: Multifunctional cooperative marine robots for intervention domains (DPI2014-57746-C3) funded by the Spanish ministry. In this ongoing project up to three heterogeneous vehicles cooperate to achieve different goals related to underwater archaeology. The case of study for this project is the autonomous cooperative localization, unearthing and grasping of an amphora located in the sea floor. The project goals have been achieved in pool conditions and will be tested in the sea soon.

[2]Video of the TRITON final experiment https://www.youtube.com/watch?v=xA2SGLi5TYg

Additionally, the thesis work also benefited from a 4 months research stay in the marine Australian Centre for Field Robotics (ACFR) of the University of Sydney (Australia). During the research stay the datasets and dehazing restoration algorithms used in this thesis were studied and prepared.

1.3. Aims and scope

As mentioned in the introduction, the goal of this thesis is to study the problems related with visibility degradation in the water medium and propose solutions that make it possible to enhance the performance of autonomous underwater vehicles. Taking into account the context of the thesis it is of utmost importance that the proposed approach is able to run in real time so it can be directly used in an intervention vehicle.

Furthermore, although most of the previous work in this field focuses on dehazing approaches that require several inputs or specific hardware, being able to obtain a restored image from a single image would offer several advantages. For instance, it will decrease the cost and payload of the vehicle making it possible to use it in smaller AUVs and to use it on smaller cameras placed in robotic hands.

It is possible to divide this general goal into more detailed objectives as follows:

- **Develop a suitable underwater simulation environment:** An specific tool designed for simulating autonomous underwater vehicles is necessary for faster development of algorithms. For this reason, this objective focuses on the development of a visually and physically realistic simulator covering the specific needs of underwater robotics. Furthermore, the simulator should be able to work as a supervision tool when the real robot is running the intervention and, it is not possible to view the system directly.

- **Design a generic benchmarking tool:** In order to compare different solutions it is necessary to have a tool capable of objectively measuring and evaluating the performance. Consequently, a generic suite that makes it possible to measure well defined metrics using groundtruth information is the main contribution of this goal. The tool needs to be abstractly defined so that it is possible to use it in any context or intervention.

- **Study the effect of underwater visibility:** Before proposing a dehazing algorithm, it is interesting to analyse how the water turbidity affects a vision algorithm so the need of it is demonstrated. Making use of the previously described benchmarking tool, experimenting with different algorithms under decreasing visibility conditions is proposed in order to understand the importance of image dehazing in the underwater context.

- **Propose a single image dehazing algorithm:** Once the need of a dehazing algorithm has been established, the development of this software is proposed. In order to do so, state of the art methods will be analysed in the context of the needs of autonomous underwater vehicles: real time performance and single image as input. The results must be tested with state of the art alternatives showing the feasability and limitations of the approach.

1.4. Outline

The different topics introduced in this thesis are presented in 8 chapters structured as follows.

In **Chapter 2** a framework suitable for underwater robotics research is presented. The need and development for a capable of benchmarking simulator is explaining showing the different capabilities and features.

Chapter 3 describes new trends in cloud simulation and presents an approach of the previously described framework in this context. Furthermore, two different cases are introduced demonstrating the possibilities of these kind of tools.

A summary of the most relevant applications of the presented framework for underwater research, besides the visibility study, are shown in **Chapter 4**, where three different cases are presented: the characterization of real experiments, dredging intervention evaluation and comparison of natural user interfaces.

The presented benchmarking suite is also used in **Chapter 5** to study the effect of water turbidity on vision algorithms. Three experiments are presented to compare trackers and 3D reconstruction techniques under different visibility conditions. The experiments include simulation, hardware in the loop and real benchmarking scenarios.

In **Chapter 6** single image dehazing algorithms are benchmarked in order to determine the best option to enhance vision algorithms in the underwater context. Alternatives capable of producing restored images in real time from single images are specially analysed and studied.

Chapter 7 proposes the use of machine learning solutions to generalize dehazing solutions from samples of images. Two solutions are proposed, one from the point of view of image enhancement and one from image restoration using an image formation model.

Finally, conclusions and future work are described in **Chapter 8** summarizing the work developed in the thesis.

Additionally, the datasets used for image dehazing are described in detail in the **Appendix A**.

Capítulo 2

A framework for underwater robotics research

This chapter presents the software framework developed through the thesis. The proposed framework makes it possible to study the behaviour of algorithms under different conditions, allowing objective comparison and evaluation. This has been used to study how different algorithms respond to water turbidity, motivating the work of this thesis. Firstly, the need for a framework with the described capabilities is discussed. Secondly, the state of the art in underwater robotic simulators and benchmarking suites is detailed. After that, the developed framework formed by UWSim, an underwater simulator, and a benchmarking module is described. Finally, a brief discussion and some conclusions extracted from the use of this software are outlined in order to show its validity.

The contents of this chapter have been published in different international peer reviewed conferences and a book chapter. In [Prats et al., 2012b] the UWSim simulator is initially presented. In the book chapter [Pérez et al., 2015] the benchmarking framework and work methodology are described using various examples of benchmarking. Finally, [Sanz et al., 2013] shows an application of UWSim simulating an underwater panel manipulation.

2.1. Motivation

As discussed in the introduction, underwater robotics is a challenging field due to the extensive resources required and the difficulty for researchers to observe the robot working underwater. For these reasons, a simulator as realistic as possible that allows to test and validate the system, is considered an extremely important tool. Moreover, it is also important to have supervision capabilities, so a visual reconstruction of what is happening to the robot in the experiment is available in real time.

Another difficulty when running field experiments in open sea or lakes, is the ever changing environment. Water turbidity, marine life, currents and tides are difficult to predict and completely unavoidable. In this thesis, a benchmarking step is proposed as a previous stage to field experiments. This software performs

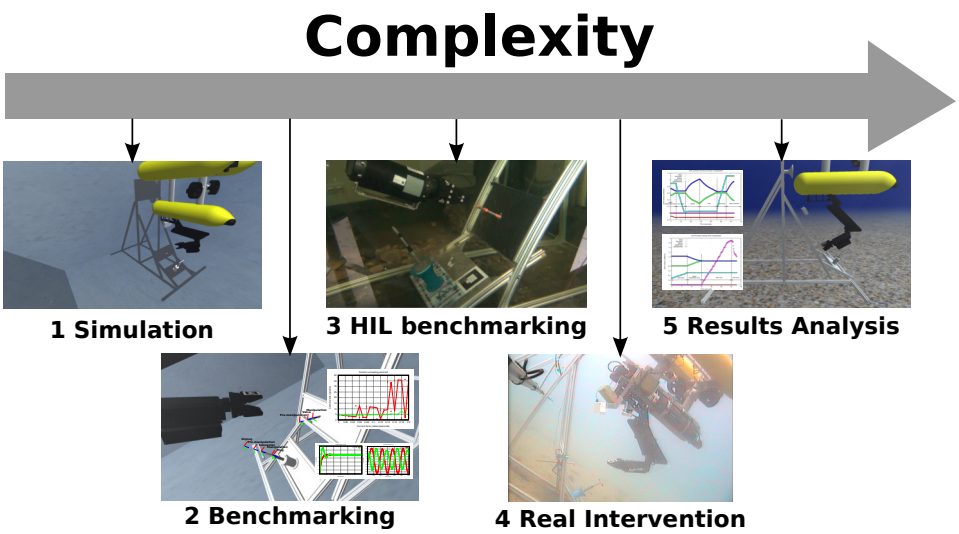

Figura 2.1: Methodology of experimental validation with increasing complexity.

repeated tests on the proposed algorithm under different conditions, evaluating its performance to detect weaknesses so that the researcher is able to fix them before real experiments. This step helps to test the system in a wide range of conditions, thus ensuring it will work under different environment conditions.

Besides this, a simulation software with hardware in the loop capabilities is also a crucial tool for coordinating complex projects such as the ones described in Chapter 1. It makes it possible to test different parts of the system without the need of the complete hardware or software facilitating the researchers work.

Finally, after the real intervention is performed, analysis and conclusions of the results achieved must be outlined. But, even if experiments are filmed by a professional diver, it is difficult to analyse the system performance from the recorded video. In order to deal with this, UWSim is capable of reconstructing the scene from the captured logs and provide valuable feedback in a reproduced simulation of the intervention. Furthermore, this reconstruction can be used to test new algorithms for the next real intervention.

All of these features are summed up in the development methodology in figure 2.1. As can be seen, the first step involves simulating the intervention. Secondly, the proposed solution is benchmarked under different conditions. After that, a Hardware In the Loop (HIL) benchmarking experiment takes place in a water tank or pool, adding simulated information when needed. At this moment, the system is ready to perform the real intervention. Finally, all the results from the intervention are gathered and analysed using the simulation tool.

Taking into account this methodology, the simulation and benchmarking phases are crucial for the final results. These phases allow the researcher to detect errors and possible weaknesses before actually performing the intervention, saving resources and time.

2.2. State of the art

The field of software tools for robotics has produced a vast amount of research. There are well established robot simulators for almost every use case: industry, humanoids, service, etc. However, the case of underwater robotics requires specific features and tools that are not usually available or easy to add to existing software. For instance, simulating the visual characteristics of water such as light back propagation, wave dependant light attenuation or underwater particles requires advanced graphic features (shaders, multi-pass rendering, render to texture, etc.) usually not used in these kind of simulators. For dynamic simulation it is necessary to add buoyancy, drag or current forces. For these reasons, the use of specific tools is important to achieve the best possible result.

2.2.1. Underwater robot simulators

In recent years multiple simulators for autonomous underwater vehicles have been developed, see [Craighead et al., 2007], [Matsebe et al., 2008] or [Cook et al., 2014] for a survey. Unfortunately, most of the available simulators are not suitable for underwater robotics research for multiple reasons. Some of them are specific for a project and cannot be adapted to other purposes. Other cases are closed projects that do not allow modifications, and, as a consequence it is difficult to interface theses systems with existing control architectures. Furthermore, support for underwater manipulators is not common in existing underwater simulators. Finally, some of these simulators are obsolete lacking the latest sensors or visualization realism desirable for proper simulation.

At the same time, adapting an existing field robotic simulator for the underwater environment is not an easy task. Although some simulators show water in their demos, it is not always possible to correctly simulate an underwater vehicle. Creating water visualization effects from zero is a challenging and time consuming task, but, adapting this to an existing architecture is not always feasible or worthwile.

Taking into account these restrictions only a few simulators, shown in figure 2.2, are suitable for the purpose of this work. One of these is MORSE, presented in [Echeverria et al., 2011], a general purpose academic robot simulator developed and supported by *Laboratoire d'Analyse et d'Architecture des Systèmes* at the *University of Toulouse*, France. MORSE relies on Blender Game Engine and Bullet physics engine. Although it is not an underwater robotic simulator, it is possible to simulate underwater scenes through Blender but requires shader and Blender Game Engine programming knowledge. The main advantage is that it is based on largely supported and active projects assuring documentation and availability, but, as the software is not specifically for the purpose of underwater robots, the learning curve is steep.

Another popular simulator is Gazebo, described in [Koenig and Howard, 2004], originally developed by *University of Southern California* and currently maintained by the *Open Source Robotics Foundation* (OSRF). It is built upon the Ogre3D rendering engine and supports several physics engines including Open Dynamics Engine (ODE), Bullet, Simbody and DART. It is probably the most suitable simulator for robotics, however there is no built-in way to create water or underwater

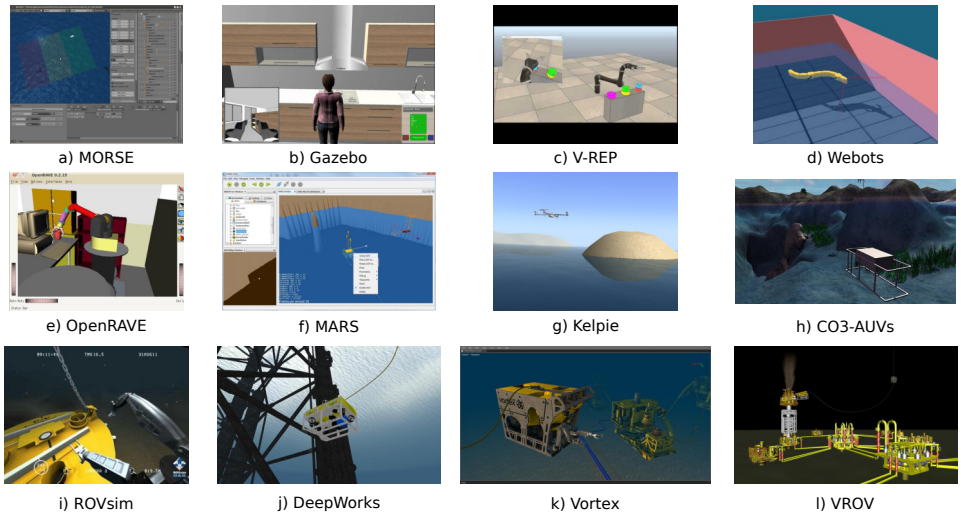

Figura 2.2: Screenshots of different reviewed robot simulators.

visual effects. It may be possible to create a plugin for this, but nowadays this plugin does not exist.

One of the most popular available robot simulators is Webots, shown in [Michel, 2004] initially developed by the *Swiss Federal Institute of Technology in Lausanne* and continuously maintained for 19 years. It uses ODE for dynamics simulation and a custom rendering engine. Webots is available for free in a limited version but the pro version can only be accessed after payment. It has support for underwater vehicles, but only a few works talk about this features and screenshots do not show realistic water rendering.

V-REP, shown in [Rohmer et al., 2013], is also a popular robotic simulator with dual license: commercial and free educational. V-rep uses a custom rendering engine with support for four physics engines: Bullet, ODE, Vortex and Newton. Although the simulator natively supports underwater robots, there has been no known application of it, and will possibly require to create and extend V-REP via plugins.

In the context of manipulation, OpenRAVE from *Carnegie Mellon University robotics institute*, detailed in [Diankov, 2010], is probably the most used tool. It is built in a plugin architecture allowing different physics engines to produce the desired results. The main drawback of this simulation tool is that it is focused on manipulation, giving the rendering capabilities a secondary role.

Another interesting open source project is MARS, presented in [Tosik and Maehle, 2014] and developed by *Institute of Computer Engineering* of *University of Lübeck*. This simulation environment based in jMonkeyEngine and Bullet Engine is specific for underwater and surface robotics. Unfortunately, MARS does not allow the use of manipulators, has a small community and it is developed in Java limiting the use of Robot Operating System (ROS) to the ROSJava library.

Kelpie, described in [Mendonça et al., 2013], is a specific simulator ROS-Based simulator for surface and aerial vehicles developed for the RIVERWATCH expe-

Cuadro 2.1: Parameters of different robot simulators suitable for underwater research.

Simulator	Availability	Underwater	Manipulators	Community	Previous use
MORSE	Open source	Possible	Yes	Medium	Yes
Gazebo	Open source	No	Yes	Large	No
V-REP	Dual: pro+edu	No	Yes	Large	No
Webots	Limited	Possible	Yes	Large	Yes
OpenRAVE	Open source	No	Yes	Medium	No
MARS	Open source	Native	No	Small	Yes
Kelpie	Open source	Native	No	Small	Yes
CO^3-AUVs	private	Native	No	Small	Yes
ROVsim	commercial	Native	Yes	Medium	Yes
DeepWorks	commercial	Native	Yes	Medium	No
Vortex	commercial	Native	Yes	Medium	No
VROV	commercial	Native	Yes	Medium	Yes
UWSim	Open source	Native	Yes	Medium	Yes*

riment in the ECHORD european FP7 project. This simulator core architecture is based on Gazebo and is built upon OpenSceneGraph (OSG), using osgOcean library to generate water rendering effects and Bullet as physics engine. However, this project is not available for downloading, and it seems to be a dead project.

The CO^3-AUVs simulator was developed during the project of the same name by *Jacobs university* of Bremen, detailed in [Rathnam and Birk, 2011]. This simulator uses OGRE as rendering engine and Bullet as physics engine. Although screenshots available show nicely rendered scenarios, there is not much information about it as it is not available for use and cannot be downloaded.

Besides generic robot simulators and underwater robot simulators, there is another group of suitable simulators: commercial simulators for Remote Operated Vehicle (ROV) pilots. This is an interesting family of simulators due to the quality and variety of available simulators. One of the most popular options is ROVSim [LLC, 2006] developed by *Marine Simulation*. It offers different products such as a vSHIP, ship simulator, ROVsim Pro, for near shore and inland marine operations, ROVsim O& G, specific for the offshore oil and gas industry and ROVsim Web delivered through web browser. All the products are focused on pilot training. But there are other alternatives such as *Fugro's* DeepWorks [Fugro,], Vortex by *CM LABS* [LABS,] and *GRi Simulations*'s Virtual Remotely Operated Vehicle VROV [simulations, 2009].

As can be seen there is no perfect simulator that covers all the needs of the project. In table 2.1 a summary with the most desirable features of the reviewed simulators can be seen. The most important parameters are availability, underwater capabilities and manipulators support. Availability describes if it is actually possible to use the software and if it is free. Underwater labels if it is possible to use it for underwater robots. Finally, manipulators indicates if the simulator allows to introduce robotic arms, also known as kinematic chains or only vehicles are allowed. Moreover, two more desirable features have been added: the users community size and if the simulator has been previously used for underwater research purposes.

For these reasons, at the time of beginning this thesis a basic simulator was being developed in the IRSlab. Through the thesis work this basic simulator was developed, and improved until reaching the UWSim, the underwater simulator

available nowadays. The main goals in mind while developing the simulator were:

- To be easily integrable with existing control architectures. Control algorithms are external to the simulator that in many cases only work as a visualization of the output computed by external programs. The software is mainly addressed to researchers and developers working in the field of underwater robotics, although special scenarios for ROV pilot training could also be implemented.
- To be general, modular and easily extendible. New robots can be easily included with eXtensible Markup Language (XML) description files. Support for widgets is also provided, making it possible to show useful information superimposed to the scene.
- To include support for underwater manipulators, thus making it possible to simulate underwater intervention missions. Kinematic chains can be created and controlled.
- To be visually realistic, and enable the user to configure important parameters such as water color, visibility, floating particles, etc.

A recent trend in robotic simulators are the web-based capabilities such as [Tellez, 2017], [Pavin et al., 2015] or ROVSim web. This feature makes it possible to concentrate the simulation in a dedicated server, or array of servers assuring the hardware is able to run the software. Furthermore, it is possible for more than one person to work with just one simulation server saving money and time. The potential applications include education, batch simulation or even distributed simulation for high precision requirements. In this research line, a web-based version and cloud features have been developed for UWSim described in chapter 3.

2.2.2. Benchmarking suite

Additional to an underwater robotic simulator, a tool for performance measurement of the developed solutions is also highly recommendable for underwater environments as discussed previously. These kind of programs are commonly referred to as a benchmarking suite.

Concerning benchmarking in robotics, a lot of effort has been made over the last few years. The robotics community has been very active in this context, and has identified as a key area the interaction of a robotic manipulation system with its environment. Indeed some recent European projects, like FP7-BRICS (Best Practice in Robotics), significantly contributed to this specific subject [Nowak et al., 2010], promoting the interoperability of hardware and software components and building a software repository of best practice robotics algorithms [Bischoff et al., 2010]. Moreover, following previous research in this field [DEXMART, 2009], it is clear that: “In the domain of robotics research, it is extremely difficult not only to compare results from different approaches, but also to assess the quality of the research. This is especially true if one wishes to evaluate the performance of intelligent robot systems interacting with the real world.” There are many definitions for the term “benchmark”, but a very simple one stated in the work mentioned above will be used, that is defined as “adds numerical evaluation of results (performance

metrics) as a key element. The main aspects are repeatability, independency, and unambiguity".

Comparative research in robotics has focused on competitions between systems such as [Holz et al., 2013] or [Amigoni et al., 2013]. Although it is required to share binaries or even source code, vital information such as configuration, build instructions and which system is required, is usually missing. This makes it extremely difficult to replicate the results after the competition. Furthermore, these competitions lead to very specific solutions for certain missions. In order to solve this, a general benchmarking suite is proposed.

Several benchmarking suites have been developed in the field of robotics. Many of them focus purely on a specific sub-field of robotic research but, to the best of the authors' knowledge, none of them is focused on autonomous underwater vehicles. In the grasping field, several suites have been presented such as the OpenGrasp Benchmarking suite [Ulbrich et al., 2011]. This suite is a software environment for comparative evaluation of grasping and dexterous manipulation using the OpenGrasp toolkit. It also provides a web-service that administers available benchmarks scenarios, models and benchmarking scores.

Another interesting benchmarking suite in the field of grasping is VisGrab [Kootstra et al., 2012], a benchmark for Vision-Based Grasping, which provides tools to evaluate vision-based grasp generation methods.

Motion planners, trajectory tracking and path planning have been very active research fields around benchmark metrics and benchmarking suites. In [Cohen et al., 2012], authors describe a generic infrastructure for benchmarking motion planners. This infrastructure makes it possible to compare different planners with a set of measures. The key point of the contribution is the easy to compare design due to ROS MoveIt! integration.

Rawseeds [Fontana et al., 2014], is a project focused precisely on benchmarking in robotics, although its global nature has been widely used for Simultaneous Localization And Mapping (SLAM). The aim of the Rawseeds project is to build benchmarking tools for robotic systems through the publication of a comprehensive, high-quality benchmarking toolkit composed of datasets with associated ground truth, benchmark problems based on datasets and benchmark solutions for the problems. Unfortunately this project lacks an automated comparison system.

The work presented in [Weisz et al., 2016], robobench, is a recent approach to a generic benchmarking platform. This approach uses software containers to avoid incompatibilities and be able to simulate and run the software to be evaluated. It also shows a set of benchmarks comparing software with different purposes. Unfortunately this suite is not available for any suitable underwater simulator.

In the SLAM problem some suites have been proposed such as [Nardi et al., 2015]. This software measures performance, accuracy and energy consumption of a dense RGB-D SLAM system. It provides synthetic sequences with trajectory and scene ground truth in order to compare different implementations and algorithms. But, once again, this is a problem specific benchmark not suitable for other uses.

Finally, there have been proposals of web-based benchmarking suites such as [Esteller-Curto et al., 2012] where authors propose an interesting test-bed internet-based architecture for benchmarking of visual servoing techniques allowing users to upload their algorithms.

As can be seen, none of the reviewed suites cover the underwater robotics issues.

For this reason, a generic configurable benchmarking suite has been implemented for the underwater simulator UWSim. This suite makes it possible to measure the performance of any software using the ground truth from the simulator and ROS as abstracting middleware.

2.3. UWSim, an open source simulator for underwater robotics

At the moment of starting this thesis work, UWSim was being developed by IRSLab in a very early stage. Only a simple non-configurable scene was available, as it was necessary to modify the source code in order to make any change in the environment or vehicle. Furthermore, the only available sensor was a non-configurable camera. Physics simulation has not yet been implemented. Finally, the only possible way to interact with UWSim was to send position messages to the vehicle and kinematic chain, not even speed requests were implemented at that moment. Due to the need for a capable simulation framework for the researchers in the TRITON, MERBOTS and TRIDENT projects in which this thesis is framed, UWSim has been improved until reaching the current state (version 1.4)[1].

The simulator has been implemented in C++ and makes use of the OpenSceneGraph (OSG) [Osfield et al., 2004], Bullet physics engine [Coumans, 2012] and osgOcean [Bale, 2012] libraries. UWSim is an active project available in http://www.irs.uji.es/uwsim/. OSG is an open source 3D graphics application programming interface used by application developers in fields such as visual simulation, computer games, virtual reality, scientific visualization and modeling. The toolkit is written in standard C++ using OpenGL and runs on a variety of operating systems including Microsoft Windows, Mac OS X, Linux, IRIX, Solaris, FreeBSD and recently also Android. Bullet is a free and open source physics engine which simulates collision detection and soft and rigid body dynamics. It has been used in video games and visual effects in movies and is the physics engine of many previously reviewed simulators such as MORSE, Gazebo or V-REP. On the other hand, osgOcean is another open source project that implements realistic underwater rendering using OSG. osgOcean was developed as part of an EU funded research initiative called the VENUS project [Chapman et al., 2006].

UWSim uses the above mentioned libraries and adds further functionality so that underwater robots can easily be added to the scene, simulate sensors, and do the interface with external control programs through the Robot Operating System (ROS). Figure 2.3 shows the main components and classes of UWSim in its current version. Basically, there is a *Core* module in charge of loading the main scene and its simulated robots; the *Sensors* module is in charge of loading the configured sensors; an *Intefaces* module that provides communication with external architectures; a *Dynamics* module that implements underwater vehicle dynamics; a *Physics* module that manages the contacts between objects in the scene; the *osgOcean*, in charge of rendering the ocean surface and special effects, and the*GUI* module, that provides support for visualization and windowing toolkits.

The main features of UWSim are described in the following sections.

[1] UWSim used in TRIDENT: https://www.youtube.com/watch?v=Hrj6wvTw3bc

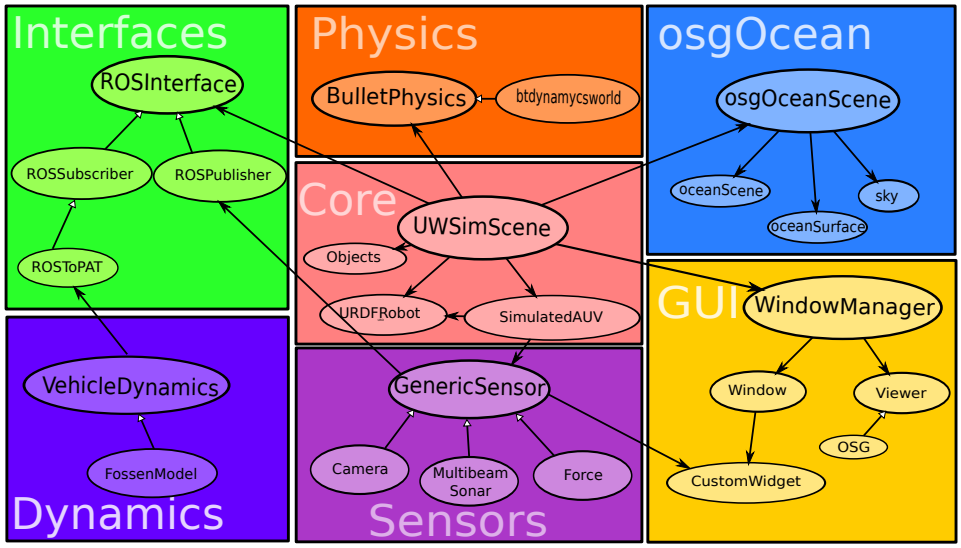

Figura 2.3: Architecture of UWSim, the underwater simulator.

2.3.1. Configurable environment

The 3D geometry to be loaded in UWSim can be easily configured with third-party modeling software as Blender, 3D Studio Max, etc. The basic scene can be freely modeled, including materials and textures. The resulting scene may be loaded in the simulator as long as it is exported to any of the formats that OSG can read (.ive, .3ds, .wrl, etc.).

In addition to the basic 3D structure, additional elements can be dynamically added, modified and removed from the main program. OSG represents the virtual scenario with a scene graph, where the nodes can be easily accessed and managed. This includes not only geometry nodes, but also cameras, light sources, etc. The complete scene can be described by the user with an XML file, or a Xacro file that generates an XML, where the main tags that can be used are:

- The **<oceanState>** block, that allows the configuration of ocean parameters such as wind direction and speed (controls the amount of waves), underwater color, visibility and attenuation factors.

- The **<simParams>** block, that lets the user disable visualization effects, set the window resolution, and set a world frame, given as an offset with respect to the default one. Some general configuration options are inside this block too, like physics solver, ambient light, gravity or physics solver. The augmented reality markers such as followed paths are configured in this block.

- The **<camera>** block sets the main camera parameters. The main camera is the viewer that observes the scene and renders to the main window. The user can set the camera motion mode (free camera vs. look at camera), and other parameters such as the field of view, aspect ratio and clipping planes.

It is also possible to set the camera parameters from the intrinsic calibration matrix.

- With the **<vehicle>** block, multiple underwater robots can be included. The user has to specify a robot description file in Unified Robot Description Format (URDF), default joint values in case the robot contains joints, the pose of the robot in the scene, and vehicle sensors if needed.
- The **<object>** block makes it possible to add other 3D models into the scene from existing file resources in any of the 3D formats supported by OSG.
- Finally, the **<rosInterfaces>** tags allows ROS interfaces to be attached to certain objects. These provide sensor information to external software such as pose of objects, images from virtual cameras, joint values, etc., and also receive external references used for updating the pose of objects and robots in the scene.

2.3.2. Simulated sensors

UWSim has twelve built in sensors for vehicles plus default position/velocity sensors available for vehicles and manipulator joints in the current 1.4 version. The default position and velocity sensors for vehicles provide 6DOF pose (x, y, z, r, p, y) in the scene, and in the case of manipulators, position and velocity for each joint is provided. Besides this, the following sensors are available:

- **Camera:** Provides virtual images of the 3D scene that can be used for developing vision algorithms. Can be initialized from intrinsic parameters, thus allowing the definition of simulated cameras with the same properties of their real counterparts. It is also possible to add gaussian white noise to add realism. This cameras can be declared in a stereo manner to simulate stereo cameras. The output can be seen in figure 2.4.
- **Range camera:** Produces the same output as a regular virtual camera, but only with range information. It can be initialized from intrinsic parameters to simulate the properties of a real device. Depending on the selected output interface, the result can be obtained in pointcloud or grayscale image formats.
- **Range sensor:** Single beam distance sensor. It measures distances to obstacles along pre-defined directions, thus simulating range sensors. Range limits can be specified to adjust to different simulated devices.
- **Object picker:** This sensor fakes a suction grasp when the object to be caught is close enough to it. It proved to be an interesting feature for the reproduction of mission logs when no physics simulation is performed.
- **Pressure:** Provides a pressure measure depending on the vehicle depth. It is possible to add gaussian noise to it.
- **DVL:** Estimates the speed at which the vehicle is travelling in the vehicle's reference frame. In order to add difficulty it is possible to add gaussian noise to the estimation.

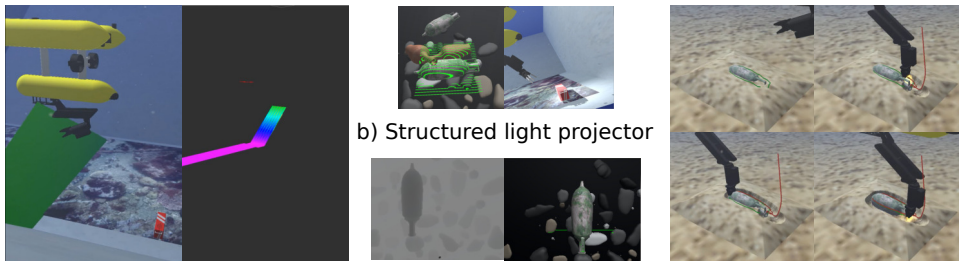

Figura 2.4: Screenshots of different sensor outputs.

- **IMU:** Estimates the vehicle orientation with respect to the world reference frame.

- **GPS:** Provides an accurate estimation of the vehicle's position relative to the world frame. However, it only works when the vehicle is close to the surface as happens with the real world GPS.

- **Multibeam:** Simulates an array of range sensors, providing distances to the nearest obstacles in a plane at constant angle increments[2]. It is created using a virtual camera and z-buffer implementation to obtain an efficient raytracing, thus being able to achieve a good performance. It is possible to configure the aperture of the sensor and the angle interval between rays, allowing a big number of hardware devices to be simulated. The point cloud generated can be seen in figure 2.4.

- **Force:** This sensor estimates the external force and torque applied to a vehicle or a part of it. In other words, it measures the collision force of a vehicle or a part of it, for instance, a robotic arm. In order to do so, every physics step it measures the position difference of the target vehicle with respect to another "ghost" vehicle, that does not collide with anything. This difference is analysed to obtain the force and torque applied to each part of the vehicle. Is specially interesting to have the information from this sensor so that the hidrodynamic simulation can pass through an external node, and afterwards add the collision forces to obtain a world response.

- **Structured light projector:** This device projects a laser or a light on the scene. It is possible to use any light pattern using an image texture to configure it. It also admits a *field of view* parameter to set the aperture of the light projector. The different possible configurations are shown in figure 2.4.

- **Dredge:** Finally, the dredge tool simulates a device designed to unearth buried objects in the seafloor. This device is designed for archaeology interventions such as the MERBOTS project final demonstration. A time lapse of a simulated dredging intervention can be seen in figure 2.4.

[2]Multibeam in UWSim: https://www.youtube.com/watch?v=zm2IE0dJo2E

2.3.3. Network interfaces

All the different robots sensors and actuators can be interfaced with external software through the network.

UWSim includes an interface for its integration with the Robot Operating System (ROS), that is a set of libraries and tools that assist software developers to create robotic applications that has become a de facto standard. ROS is a distributed system where different nodes can run on different computers and mainly communicate through *topics* via publishing/subscribing mechanisms. The ROS interface of UWSim makes it possible to run the simulator as another ROS node that can communicate with the rest of the architecture with the standard ROS communication facilities. This allows control methods developed in ROS to be seamlessly validated, either on UWSim, on the real robots, or even logged real interventions as long as they provide the same interface.

Through the ROS interfaces it is possible to access or update any vehicle position or velocity, to move arm joints, and to access the data generated by virtual sensors. Interfacing with Matlab and Simulink is also possible through the ROS Matlab support, added in the Robotic System Toolbox in Matlab R2015a[3]. Although, it was possible to do it with previous versions of Matlab through IPC bridge, but these methods are now outdated.

Furthermore, UWSim has support for *ROS interactive markers*, a message library for "regular" 3D markers and interactive markers that allow the user to interact with them changing their position, rotation, clicking or selecting context menu[4]. This feature allows new geometry to be dynamically moved, created or destroyed while the simulator is running. Thus, allowing a simple, configurable and neat user interaction with the simulator. Moreover any existing ROS node should work as long as they respect the messages protocol.

2.3.4. Physics simulation

Simulation of contacts is supported by the physics engine Bullet, wrapped in osgBullet for its use with OSG. This allows collisions and forces to be detected, and automatically update the scene accordingly. The different body collision shapes can be automatically generated from the 3D models. This physics engine is also in charge of estimating the force sensor values, providing collisions and reactions.

In order to perform dynamic simulation, an external simulation node, available in the simulator stack, is used[5]. This node provides dynamic simulation of underwater vehicles, interaction with simulated world which can be created using force sensors as external forces on the vehicle. It is in charge of simulating the hydrodynamics: thruster forces, buoyancy forces, drag forces, etc. As input, specific parameters of the vehicle that can not be inferred from urdf are needed, such as underwater masses, gravity center, buoyancy center or drag coefficients.

The decision to keep dynamic simulation in a separated node makes it possible to use different dynamic simulators until a generic complete solution is available. For instance, in some works simurv 4.0 [Antonelli, 2014] was used to simulate a

[3]Controlling UWSim with Matlab Simulink: https://www.youtube.com/watch?v=038rdcyZ0iQ
[4]UWSim interactive markers: https://www.youtube.com/watch?v=AfW24fLMVTY
[5]Dynamic simulation in UWSim: https://www.youtube.com/watch?v=74R75S8cqgQ

Figura 2.5: UWSim scene with a ship sailing in a clear ocean.

multibody dynamic robot in Matlab while visualizing and simulating cameras in UWSim. In [Kermorgant, 2014] the Gazebo simulator physics engine was used for underwater vehicle manipulator dynamic simulation while keeping UWSim as a visualization engine. Finally, it is also possible to not use a dynamic simulation at all, and monitor a real intervention.

2.3.5. Widgets

Support for customizable widgets can be added to the main window. Widgets are small windows that can be placed inside the main display in order to show specific data to the user. An abstract interface for the creation of custom widgets is provided. It allows specialized classes to be created for displaying useful data depending on the specific application. For instance, the virtual camera view is displayed on a widget, the position and size of which can be modified during execution by the user.

2.3.6. Visualization capabilities

UWSim exploits the potential of OpenSceneGraph using many advanced features such as shaders, particles or Level-of-Detail to provide a high quality visual simulation. osgOcean provides a visually realistic view of the ocean, as can be seen in figure 2.5, that can be configured to simulate different ocean states such as calm, stormy, windy, etc.

It is possible to seamlessly load complex meshes with multi-resolution textures generated externally from bathymetry and imagery. The built-in support for database paging that is used for loading different Level-of-Detail models from disk in real time makes it possible without a noticeable drop in performance. Thus, models

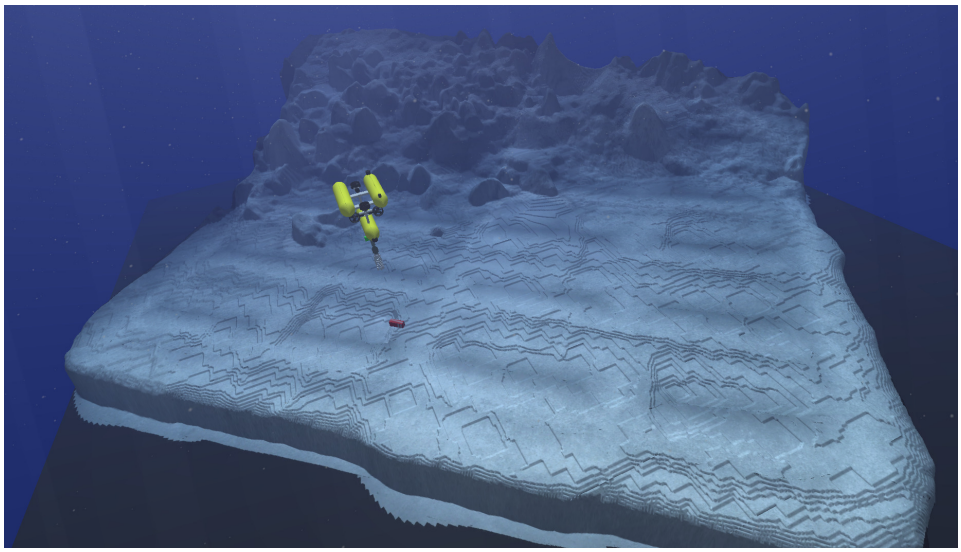

Figura 2.6: UWSim scene with multi-resolution terrain loaded.

generated in other projects like *VirtualPlanetBuilder*, that produce large terrains and textures common in underwater robotics, can be introduced. Figure 2.6 shows a multi-resolution terrain of a real reconstruction of a port in Soller, Mallorca, Spain from a raw source with 11 million points.

Shaders, small GPU programs that determine 3D surface rendering, provide a large number of visual effects such as godrays, underwater scattering, haze or glare. But, the most interesting thing about shaders is that due to its parallelizing possibilities there is almost no performance drop if it is properly coded. For instance, random gaussian noise in virtual cameras had to be implemented using shaders due to the huge performance drop it caused without them. Another interesting application of shaders is the structured light projector.

osgOcean uses particles to simulate small floating things that disturb vision. However, osg particles are used to simulate the dredging effect in the dredge tool. This feature increase realism when the device is close to the seabed and mimics the effect of sediments being stirred up on the seabed.

2.4. Benchmarking suite for UWSim

Together with UWSim, a generic module focused on evaluating and measuring the performance of any intervention algorithm has been developed. This suite, permits repeated tests to be performed in different environment conditions such as increasing water turbidity. As the simulator, benchmarking module has been implemented in C++ and uses ROS as middleware to interface with external software and UWSim as ground truth source. For the development of the module, two important objectives were taken into account:

- The first one was to be clear to the user, in other words, that it does not

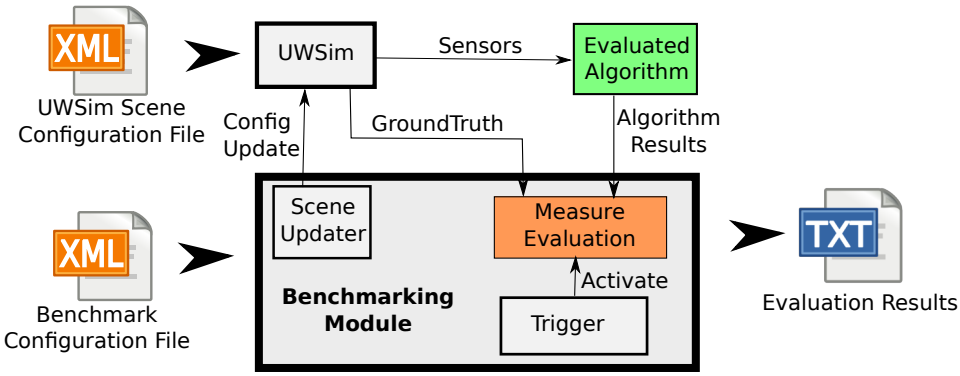

Figura 2.7: Benchmarking suite workflow diagram.

require major modifications to the algorithm that has to be evaluated.

- The second goal of the module development was that it must be adaptable to all kinds of tasks in the underwater robotics field.

Taking into account these goals, the designed benchmarking module workflow can be seen in figure 2.7. The evaluated algorithm takes the sensors input from UWSim and performs the actions through ROS interfaces. Thus, already integrated ROS software, a standard de facto nowadays, does not need any change, so the first objective is fulfilled. In order to accomplish the second goal, several configuration options are available in a XML file being able to activate or deactivate different measures depending on events and updating the scene accordingly. These files, abstract from specific scenes or implementations, describing how a task should be evaluated. Each of them can be assigned to one or more UWSim scene configuration files.

The benchmark configuration options are basically made up of three kinds of entities: measures, triggers and scene updaters. These entities have been created in a modular way, thus users can extend them and create new functionality easily.

- **Measures:** This entity is in charge of answering *how* to evaluate the algorithm. It is possible to choose a set of measures to evaluate the software from already implemented options such as time, path following integrated squared error, reconstruction3D error, distance, etc. If the desired evaluation measure is not available yet, it is possible to implement it filling a common interface acquiring the ground truth from UWSim and results from ROS interfaces. Finally, the last option is to introduce a static value, or dynamic through ROS, as ground truth that will be compared with the result of the evaluated algorithm obtained from an ROS interface.

- **Trigger:** Additionally to decide *how* to evaluate, it is possible to decide *when*. Triggers activate or deactivate measures depending on specific events. For instance, it is possible to activate a collisions measure while the vehicle is navigating but stop it when the vehicle has to grasp an object. Because at this moment, hand object contacts are not a bad result although they would

be detected as collisions. Another existing possibility, is measuring the time in different stages: for example measure the time it takes to search for the object and the time to grasp it.

- **Scene Updater:** This configurable option controls the environment. It helps to repeat the same experiment under different conditions without human intervention. There are some scene updaters already available that can be chosen, such as turbidity, illumination, current force or camera noise. But, it is also possible to add new updaters filling the corresponding abstract interface interacting with UWSim.

The features of the benchmarking suite are summed up in the following sections.

2.4.1. Automated results

Once the benchmarking has finished, caused by a stop trigger event, and all the scene configurations in the scene updater have been tested, results are written into output files. These output files are disaggregated for each scene configuration, containing results for each measure and global results that are a combination of multiple measure results. For instance, a navigation efficiency benchmark could use two measures: distance travelled and battery consumption and a global result of *distance/battery* under 3 different current forces. The results file would have a 3x3 matrix with results for each current force in lines, and *distance/battery*, distance and battery as columns.

Additionally, each measure can be configured to log its result at regular intervals in order to see its evolution over time, and not only the final result for each scene configuration. As a result, for each logged measure, the benchmark will generate a different output file, containing the variation among the measured results. Furthermore, each measure can contain details, for instance position error can be retrieved as a single absolute error in meters, or the error in each axis (x,y,z). Thus, it is possible to show the summarized version of the measure or the complete one depending on the situation.

The output file or files are written in a tabbed matrix form, making it possible to further process with any spreadsheet manager such as *Microsoft excel*, *LibreOffice calc* or *google spreadsheets*.

2.4.2. HIL benchmarking

In the case of visual algorithms, it is common to work with recorded images to improve the algorithms. This benchmarking suite goes a step further in this concept, allowing a real logged intervention to be used in the form of a rosbag. A rosbag is a file that contains a timed copy of every ROS message in the network of a real intervention. Using this information and filling the possible gaps with simulation it is possible to reproduce the intervention and measure the performance of different aspects of the real intervention, such as travelled distance, average velocity, etc.

Additionally, it is possible to mix simulation and real input using scene updaters to test different situations in a Hardware In the Loop setup. For instance, in Chapter 5 an application of this feature is showed. In the experiment a recorded video of a black box mockup is increasingly blurred with simulated water turbidity. This

makes the Hardware In the Loop experiment closer to the real world shortening the gap between the two worlds.

2.4.3. Real benchmarking

Finally, it is also possible to benchmark a real process just redirecting the ROS interfaces accordingly. As the ground truth is normally acquired from UWSim, a previous calibration step is needed to configure the virtual scene as similar to the real world as possible. Then the simulator will act as a ground truth source while the real robot is performing the intervention.

The only feature that may need to be developed is *scene updaters*. In this case, instead of modifying the simulated scene this entity should be able to interact with the real world. For this reason, the scene updater should be able to send messages to a real device. In Chapter 5 an application of this situation is presented where different 3D reconstruction algorithms are compared in varying illumination conditions. In order to control the illumination, different lamps were turned on or off.

2.5. Discussion and conclusions

The developed framework for underwater robotics research is based on two software packages: UWSim, an underwater robots simulator, and a generic benchmarking suite capable of evaluating robotic algorithms. In order to interface with these packages, ROS has been used as middleware, abstracting from implementation details and allowing any ROS based software to integrate with them.

UWSim, the underwater simulator developed, is an active open source project, ready for use by the underwater robotics community. Some European Community funded projects like MORPH [Kalwa et al., 2012] and PANDORA [Lane et al., 2012] have decided to use it after a comparison of the state of the art alternatives concluding: ÜWSIM is the most feature-complete simulator, and open to the ROS community".

Furthermore, at least 52 universities or institutes are known to use or have used UWSim, see figure 2.8 to see the distribution. This fact shows the need for a dedicated underwater robot simulator for research and that UWSim is actually filling that gap.

The AUV simulators review [Cook et al., 2014], compares different available open source softwares and concludes: .°verall, UWSim is an excellent fit for the simulation of underwater vehicles.". Although, it points some drawbacks too, for example: large simulations require to manually edit an XML and the lack of a convenient way to extend software. However, since the review, some work has already been done to solve these issues using Xacro macros to configure scenes, also a generic plugin interface to create new sensors has been developed.

About the benchmarking suite, it is noticeable to say that it has been extensively used by the IRSlab research group for different purposes: evaluating trackers under different turbidity conditions, position controllers depending on currents, 3D reconstruction algorithms, natural interfaces, dredging, path following and real intervention among others.

Figura 2.8: Known universities or institutes that use or have used UWSim.

This chapter presented the framework developed for underwater robotics research that is used in this thesis to study the effects of underwater turbidity in vision algorithms. Providing enough information to be able to choose the most convenient solution for each case and motivating further research in dehazing methodologies.

Capítulo 3

A benchmarking simulator on the cloud

As discussed in Chapter 2, simulation capabilities are necessary to reduce costs and development time in underwater robotics. Moreover, benchmarking features to compare and evaluate the system are also desirable. After a thorough state of the art review, a simulator with benchmarking capabilities was developed and presented on Chapter 2.

This chapter, inspired by the recent "on the cloud" trends, proposes an extension to the previous framework to provide online services for simulation, benchmarking and evaluation. These services allows anyone with a connected device to configure and launch experiments in a remote server being able to retrieve results and immediately compare with other users.

The proposed architecture and ideas described in this chapter have been published in international conferences and are under review for journal publication. Concretely, it has been published in [Pérez et al., 2014] and [Perez et al., 2014] showing different applications for the online execution simulation services.

3.1. Motivation

High fidelity simulation is a computationally expensive task. If the environment is to be properly simulated, it requires a high-end computer with a good graphics card. Furthermore, users that are using a different operative system or device might find installing and configuring all the necessary libraries discouraging.

When benchmarking is added to the formula, several configurations and/or algorithms may be tested, increasing the required time to simulate and compare all the possibilities. In order not to distort the results, the system can not be used while the simulation is running, and it is also convenient to use the same hardware for an objective experiment comparison. Thus, the simulation hardware needs to run for hours undisturbed to assure a fair comparison of the results.

Finally, nowadays users prefer to work with smaller portable devices such as netbooks, tablets or smartphones. These devices are not suitable for simulation, as

they rarely have a dedicated graphics card or enough computation capabilities. However, it is possible to connect almost every up-to-date device to Internet, reaching a virtually unlimited amount of services. Thus, it is possible to stay connected and work at home, while travelling, or at a different office: in fact almost everywhere.

Taking into account these facts, the next step for robotic simulators is to provide services through Internet. Allowing users to work with any device anywhere, adapting to new demands while reaching higher standards of realism. To achieve this, it is necessary to provide a simulation server that will carry the computation, and, also a way to access and interact with it. This capability is often referred to as "working on the cloud".

3.2. State of the art

With this goal in mind, some robotic simulators already provide capabilities on the cloud. This is the case of *the construct sim* [Tellez, 2017], capable of running Gazebo or Webots simulators in any device, anywhere, without installation. This is the most mature project where only a web browser is required to start simulating. It also provides a wide amount of robot and environment models to start with from their complete webinars and tutorials. However, it does not provide any tool for underwater robotics. Unfortunately, this amazing service is not free, the price depends on the number of simulation hours, and the CPU and GPU characteristics needed.

In [Pavin et al., 2015] an Autonomous Underwater Vehicles cloud simulator is proposed. In this case, the application is focused on navigation control problems and offers very few sensors and simulation capabilities. The system runs on any device through a reconfigurable web interface. However, there is not much information about it as it is not available for the community.

The *Open Source Robotics Foundation (OSRF)*, maintainers of Gazebo, developed its own tool to deploy and control simulation in the cloud: CloudSim [Foundation, 2015]. This software allows simulations to be launched in a cloud of machines over the Internet, allowing the users to access them and even teleoperate. Although it does not provide a service ready for use, it offers software to create your own service together with Gazebo.

UC Davis C-STEM Center has launched RoboBlockly in [Center, 2015]. This resource is focused on teaching robotics and maths to young students. It relies on Lego robots and the library Blockly from Google to generate executable C++ code that can be run in simulation and in the real robot. The Blockly library is a visual programming library in blocks format, that generates the code to be executed. The main drawback of this platform is it is very basic and uses a naive simulator.

The Robot Programming Network (RPN), presented in [Cervera et al., 2016], offers access to educational robot tools through a web interface. It allows the user to directly create and test code in an integrated dialogue. The tool is organized in moodle courses, that introduce different simulators or concepts to the students. Unfortunately, it lacks the tools or resources needed for underwater robotics research.

In the context of commercial ROV simulators, ROVSim, available at [LLC, 2006], also offers a web alternative. This software has the same graphics and physics

as the regular ROVSim, but it is delivered through a web browser under Windows and Mac. Although it still lacks compatibility in other platforms, such as android, it proves the commercial simulators are also working in simulation on the cloud alternatives.

Taking into account the advantages of these kind of suites, and that none of them supports underwater robots or benchmarking features, an online service for UWSim and the integrated benchmarking module has been developed.

3.3. Online execution

The online execution architecture for UWSim and the benchmarking module, allows simulations and benchmark experiments to be launched in a remote server using a web browser. It can be accessed in http://robotprogramming.uji.es/UWSim/config to configure the use cases presented below[1]. This system focuses on a simple solution that makes it possible to remotely execute simulations, configure the scene and benchmark, run an arbitrary algorithm to interact with the simulator and retrieve the results.

3.3.1. Configuration server

In order to interact with the system, only a web interface is needed, allowing it to be used in any connected device as long as it supports a JavaScript capable browser. This interface makes use of *rosbridge* through *roslibjs*, a library for interacting with ROS from the browser that uses WebSockets. In the case of video streaming, *mjpeg_server* is used. This library produces a compressed video streaming server via HTTP from ROS topics.

The web execution architecture can be seen in figure 3.1. The online server listens to web requests and serves the pages. Moreover, it also acts as an intermediary with the simulation server, which is inaccessible from the outside, as it pertains to the internal network servers in the domain uji.es and is protected by a firewall. Thus, it redirects the calls from *rosbridge* and *mjpeg_server* to itself. This way, the interaction with the simulation server is transparent to the client, which only makes calls through the JavaScript interface for *rosbridge*, and receives streaming video using *mjpeg_server* and results through *rosjslib*.

At the time of writing, it is possible to configure two types of benchmarks, visibility and station keeping, but adding new configuration options is a simple process. Just adding a few lines to the configuration interface, and making sure the simulation server is able to run it, is enough to add more benchmarking scenarios. Figure 3.2 shows a station keeping benchmark being configured under different devices.

Besides the type of benchmark, the user is also able to choose from a set of already implemented solutions for the problem. In both cases, different tracker configurations are available, and in the case of station keeping, the parameters of the controllers can be directly modified. Additionally, it is also possible to upload a custom code so that it can be evaluated.

[1] Online benchmarking: https://youtu.be/2OCmMnQUwDs

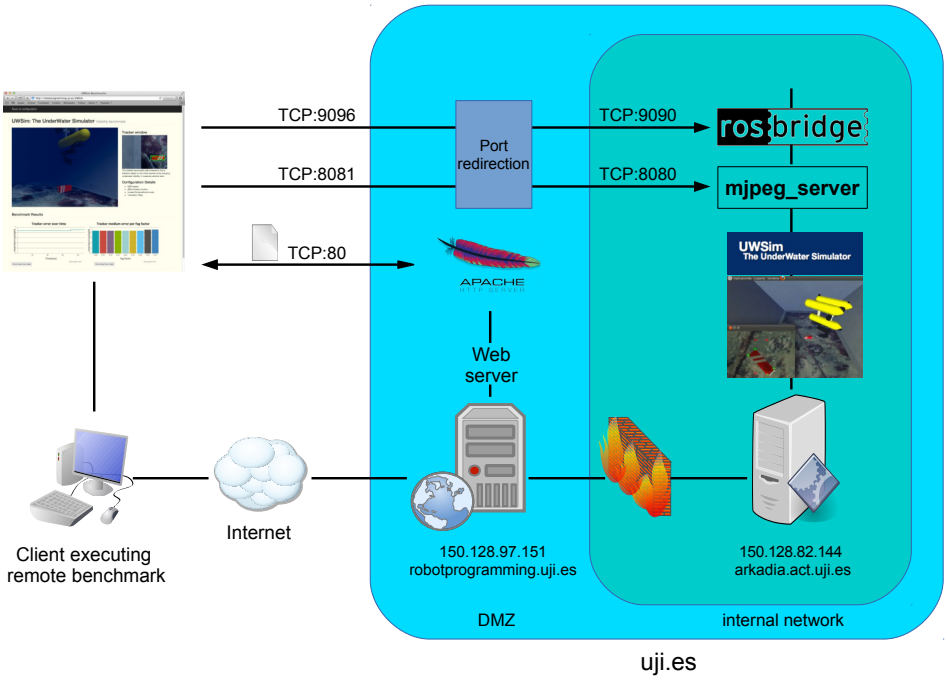

Figura 3.1: Architecture of the online UWSim benchmarking service.

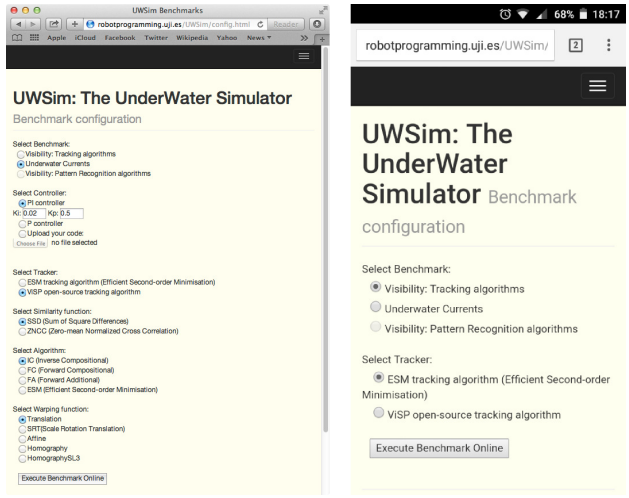

Figura 3.2: Online benchmark configuration interface in MAC OS and android.

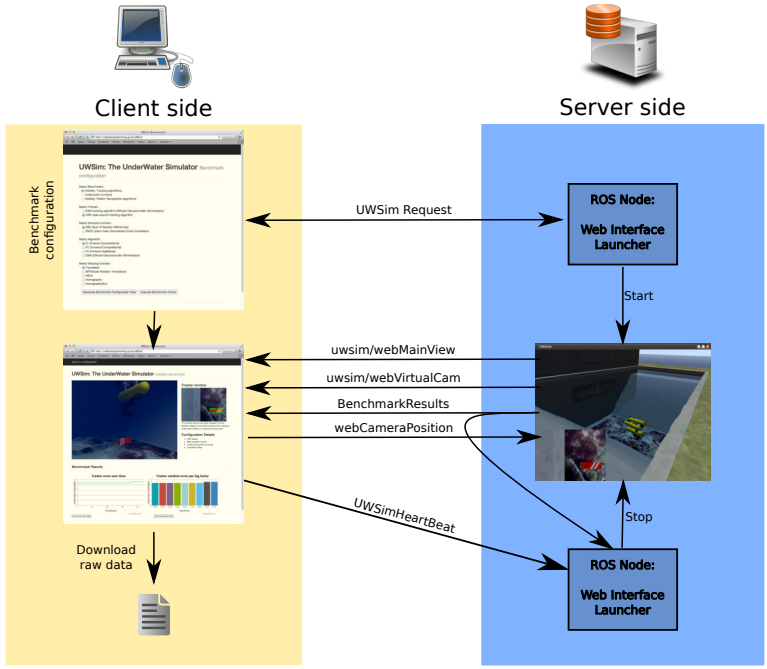

Figura 3.3: Web simulation service dataflow.

3.3.2. The simulation service

The functionality of the web execution service can be seen in figure 3.3. Firstly, the user selects the desired configuration options using the previously described configuration server. Once the user has configured the desired options, the benchmark is launched in the simulation server. In order to do that, a *uwsimRequest* service is called and sent to the simulation server. This is a type of message, that contains a dictionary of keys and values with the different configuration options in the form of two vectors. This request is received by the webInterfaceLauncher ROS node, located in the simulation server, that is responsible for executing the request. The node will answer the configuration server whether the configured options are valid or not.

Similarly, the users can upload their own code in the request, which will be stored on the server until the execution. In this case, the code provided by the user will replace the one already existing on the server by default. For simplicity and security reasons, it is only possible to execute code written in Python, but it is possible to allow execution of code written in C++, Matlab, Simulink or any ROS based code. If the request is successful a user ID will be sent to the client, and an instance of the simulator related to this ID is launched.

At the moment of writing, the simulation server is only capable of running a single instance of UWSim assuring good performance. For this reason, and in order to assure a fair comparison when running benchmarks, only one client can be using the service at the same time. This is fulfilled through the user ID, denying new connections while a user is running the service showing a server busy error. However,

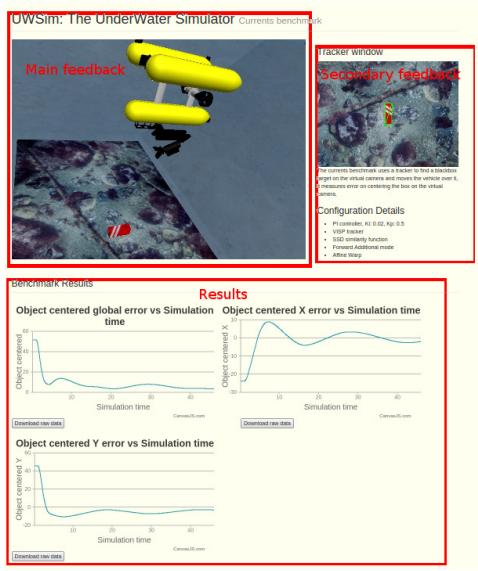

Figura 3.4: Online benchmark execution interface.

if the simulation server capabilities are extended this can be easily changed allowing clients to be launched in different machines.

Additionally to this, it is necessary to provice another mechanism which will indicate when a client leaves the simulation, and the service is ready for other clients. Although the ideal situation is the client sends an exit message, trusting the users good-will is not a reliable or robust method. Thus, two possible causes can cause a simulation to abort: the user closed the browser or the experiment that had been launched has finished.

In order to detect the first case, the JavaScript client is required to send a *webUWSimHeartBeat* each second with the user ID. If the simulation service does not receive a single heartbeat for 10 seconds, it considers the client has disconnected and exits the simulation allowing other clients to use it. This assures the detection of when the client left the web interface or lost connection, even if it was not on purpose. The only drawback is that it then not possible to launch a large simulation and consult the results in another session. This could be implemented requiring a user login or asking for an email and send the results to it at the end of the experiment.

The second abort case is the benchmark experiment ending. When a benchmark experiment is launched, the simulation server is continuously sending results to the client until the end. For this reason, the web interface launcher monitors the results communication, and if the simulator is not sending it, it means the experiment has already finished. When this happens, the user can still see the experiment results but needs to start a new simulation for further experiments.

While the simulation service is running, the client can interact with the simulator using the main feedback, secondary feedback and the results section, see figure 3.4. The main feedback displays the simulator view. Users can interact with

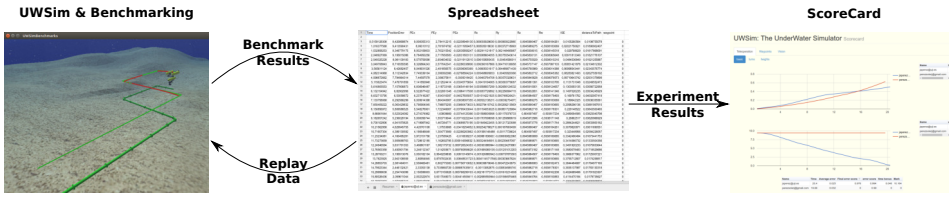

Figura 3.5: Benchmarking on the cloud workflow.

it by using the mouse, and can move through the virtual scene while the benchmark is running, as they would do in a local simulator. The secondary feedback, placed at the right part, shows information about the experiment that is being executed: virtual cameras, configuration options, etc. Finally, the results section shows the measures the benchmark module is computing using *canvasJS* library. *CanvasJS* is a powerful tool that allows data charting in real time using HTML5, assuring compatibility with different devices such as tablets, smartphones, etc. It is also possible to download the raw information so that it can be further processed by just pressing the corresponding button.

3.4. Benchmarking on the cloud

The previously described service allows remote simulations to be run and the results to be retrieved. However, it is not possible to directly compare with other simulation runs, it is required to download the data and run a different simulation so it can be compared. Furthermore, there is no way to compare with other researchers results, unless both researchers are already in contact.

In order to deal with this, a benchmarking on the cloud architecture has been designed. This architecture makes it possible to automatically upload the results to a comparison environment, and consult a scorecard interface with specific graphs and results for the experiment. Additionally, it is possible to reproduce any compared experiment from the stored results as a logged reconstruction, without any code information to preserve privacy.

This architecture is composed of two main pieces, an "on the cloud" storage space to hold all the results data and a web interface to show the results in the most convenient way for the experiment. The data workflow between these main parts can be seen in figure 3.5.

The "on the cloud" storage space uses *Google spreadsheets*. It is a common online shared spreadsheet, but it is possible to feed data through its API and can be consulted later. In order to feed the data, the UWSim benchmarking module presented in Chapter 2 has been extended in order to be able to upload experiment results to a previously configured spreadsheet. As the data to be uploaded is taken from the benchmarking module, it can be used in the online and local execution. From the user point of view only an authentication step is required with a valid email, thus it is possible to identify the experiment execution.

The spreadsheet is shared and visible for anyone with the proper link. For this reason some security measures are needed to avoid users deleting or modifying the results of a different user. Thus, the spreadsheet is divided in different sheets, one

for each user, that will be protected against modification by anyone but the user that created it or the proprietor of the spreadsheet. This assures the results were achieved by the user that owns the sheet and are not modified by anyone else.

The second piece, web interface scorecard, plays the role of a visualization tool. It is a regular HTML file with JavaScript that can be placed in any server or even executed locally. It uses *Google Charts* library to query the information from the spreadsheet previously described, and produce a visual result. The design and type of information displayed in this interface depends on the nature of the experiment, but a wide variety of possibilities is available such as linecharts, pie charts, barplots, tables, etc.

Besides this, it is also possible to visualize the performance of a target user. For instance, it is possible to see the intervention of any user that has published results and see how the vehicle moved in the scene. However, no code or details are stored to preserve privacy, only the log information that makes the visualization possible. This is a valuable tool for researchers, that can actually visualize the performance and compare it to their own solutions. In order to do so, it is only necessary to put the email of the user into the software and the system will automatically download the information and run the simulation.

3.5. Use cases

The previously described online capabilities of UWSim have been used in different applications. In the online execution service two experiments are available and can be configured: visibility and station keeping. Visibility experiments will be presented in Chapter 5 with other water turbidity experiments. The station keeping experiment aims to keep the vehicle in the desired position while underwater currents perturb the system using a tracker to acquire pose information. For the benchmarking on the cloud feature, an educational pipe following application is proposed, where students learn basic robotic algorithms using the presented architecture.

3.5.1. Station Keeping

The purpose of this experiment is to test the online simulator execution in a realistic simulation scenario. The use case focus on maintaining the simulated vehicle, Girona500, in a relative position from a target, in this case a black box, so it is able to start a manipulation intervention. However, the purpose of this experiment is to demonstrate the capabilities of the developed framework, more advanced and complex control techniques can be applied to properly solve the problem but they are out of the scope of this work. In this case the software architecture created to solve the problem, that can be seen in figure 3.6, is made of four software nodes:

- Tracker: The tracker is the software which starts the action, finding the target on a camera and publishing its position on the camera's image. In this experiment, two different trackers are used: ViSP's [Marchand, 1999] and ESM [Malis, 2004] with a choice of different configurable options available in the configuration step.

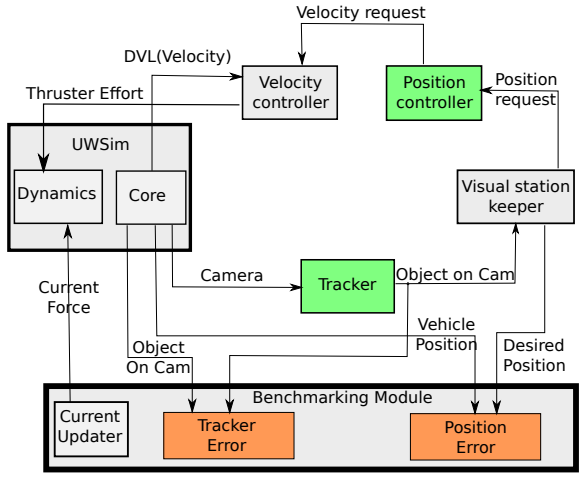

Figura 3.6: Software schema used in station keeping use case. Orange: benchmark outputs, Green: evaluated software available to configure online.

- VisualStationKeeper: This is the program that takes the tracker's position and converts it to the world's distances (meters instead of pixels). With this information, the software decides where the vehicle needs to move in order to be in the best position to start a manipulation action. In order to do so, it publishes the error between the vehicle position and goal position. In this case, it just tries to keep the vehicle on the top of the object to start an approach.

- Position controller: This controller is in charge of deciding the proper velocity to reach the goal assigned by the visual station keeper. The web based execution tool offers a P and PI controller, allowing their constants to be changed in the configuration step. It also provides an interface to upload Python code to substitute the default controllers.

- Velocity controller: This program reads the velocity reference from the position controller and the vehicle velocity measured from a DVL (Doppler Velocity Log) sensor. Then, it decides the effort needed on each thruster of the vehicle to achieve the required velocity. The software developed in this case is a proportional controller.

The simulation runs in "batch mode", using the benchmarking module to test different configurations sequentially. In this use case, the benchmark is restarted every 120 seconds to increase perturbations, simulating the velocity of underwater currents. Starting from no current at all, it tests from 0.0m/s to 0.4m/s in 0.1m/s steps. The current velocity is modelled using equation 3.1. As can be seen it is formed by constant current velocity, $module$, and a sinusoidal term, $variation * sin(time)$, multiplied by a random white noise factor, $noise$, in a pre configured direction. This produces a sinusoidal noisy force that simulates the underwater currents.

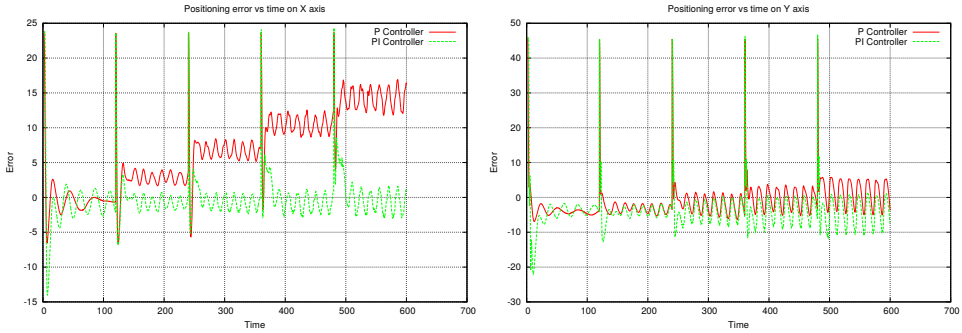

Figura 3.7: Controllers comparison error (left X axis, right Y axis) using Ki=0.02, Kp=0.5 on PI and Kp=0.5 on P controller.

$$CurrentSpeed = (module + variation * sin(time)) * noise \tag{3.1}$$

The system has been tested with two different controllers. As can be seen on figure 3.7, the P controller is not able to reach a 0 error when current force increases. It is possible to see this on the graph, as every 120 seconds the system is restarted using a higher current force. The minimum error it is able to achieve depends on the current force. On the other hand, the PI controller is able to reduce this error despite underwater currents, as was expected. Nevertheless, its sinusoidal behaviour is caused by the sinusoidal nature of currents, for which PI controller is not completely able to compensate.

Videos of the experiments can be seen in https://www.youtube.com/watch?v=hOrXNQdKJwE using the P controller and https://www.youtube.com/watch?v=EhjhJPNe2GE in the case of the PI controller. In the videos the current is displayed with an arrow indicating the current direction with size proportional to the current force.

As underwater currents have the sinusoidal component on the Y axis, the expected results are an increasing sinusoidal error on the Y axis. In the Y axis graph, both controllers are centered on error=-5 pixels instead of zero. This is caused by small errors on the tracker software which is guiding the vehicle to this position instead of zero. As can be seen, PI controller gets slightly better results, but it is not able to completely neutralize sinusoidal currents.

On the X axis, an increasing error is expected for P controller while PI controller should be able to drive the error to zero independently from underwater currents. Although in stronger underwater currents, PI controller needs around 30 seconds to reach a zero error, it is able to do it, while P controller stabilizes around an error of 14 pixels. Results when current force is low, or there is no current at all, are similar for P and PI controllers.

Besides comparing two different controllers, it is also possible to test different trackers and compare control results depending on the tracker used. Most trackers do not present big differences, but in general, the PI controller helps the tracking task due to its smoother movements. This means trackers achieve a slightly better performance with PI controller because movements are smoother.

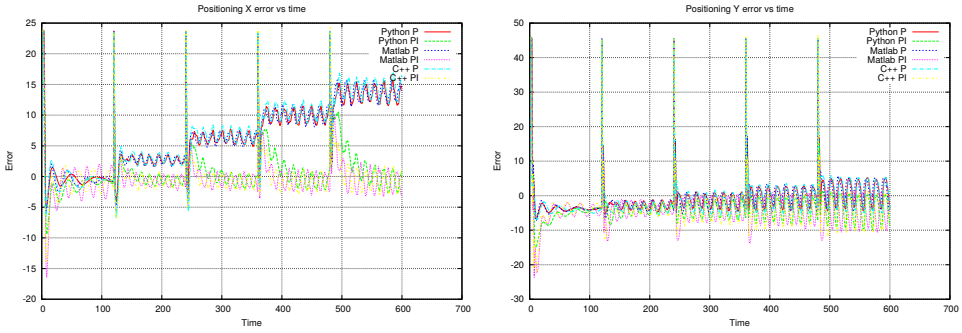

Figura 3.8: Controllers repeatability comparison error (left X axis, right Y axis) using Ki=0.02, Kp=0.5 on PI and Kp=0.5 on P controller.

To conclude, an underwater robotics problem has been proposed and solved using the developed framework. The experiment is available online, where it can be seamlessly configured and run. The proposed framework provided an easy way to objectively evaluate and compare different solutions under different underwater current conditions.

Finally, to prove the online simulation does not introduce any additional error or difference compared with a local execution, a repeatability experiment has been conducted. This experiment focus on testing the same controllers, as controllers can be easily developed in different languages ensuring the same behaviour, in different setups and languages. The hypothesis to prove is: as different implementations, setups and runs should achieve similar performance, the benchmarking module should be able to abstract from this implementation details. Three different cases are compared:

- **Online C++:** This is the previously described experiment. It was performed using the online simulation execution service that uses C++ implementation for the controllers. The experiment was performed in ROS hydro and UWSim version 1.2. The computer used for the experiment is a desktop *i5-650* at 3.20 Ghz with 12Gb DDR3 RAM and *nvidia Geforce 960GTX*.

- **Matlab Simulink:** This experiment was run locally in a laptop *i7-3630* at 2.4Ghz with 8Gb DDR3 RAM and *nvidia Geforce GTX 660M*. In order to test different languages, a Simulink implementation of the position controller was developed in Matlab 2015b using the Robotic Toolbox. The system was using ROS indigo and UWSim version 1.4.

- **Python:** The python implementation was tested in the online simulation server but executed locally instead of through the web interface. Thus, the computer specifications are the same as in the Online C++. However, a more recent version of the software was used in the experiment: ROS indigo and UWSim version 1.4.

The controllers tested, were chosen due to the simplicity to replicate the implementation in a different language. P and PI controllers are well-known mathemati-

cal functions that can be easily parametrized. Besides online and local comparison, Matlab Simulink was added in order to see if the ROS interface of Matlab added any perturbation to the system and to demonstrate a completely different implementation.

The figure 3.8 shows the results of the repeatability experiment. As can be seen, there is no significant difference between the three compared cases. The P controllers of each implementation produce a result that is not able to reduce the error to 0 as expected. On the other hand the PI controllers are able to control the constant perturbation. However, none of them is able to completely control the sinusoidal component as they were not designed for it.

This experiment demonstrates that the online simulation execution, as expected, does not introduce an additional error because the code is executed locally in each case. The interface acts as a configuration tool and visualizer, and does not interfere in the simulation. Furthermore different computers, languages and ROS versions have been tested showing no appreciable differences in the results. Thus, the benchmarking module is capable of objectively evaluating different software sources, providing crucial feedback for underwater robotics research.

3.5.2. Educational pipe following

The previous experiment used the web based execution, however it did not exploit the potential of the cloud results sharing. A potential application for these kinds of features is educational robotics. Nowadays, there is increasing interest in robotics on a global scale, commercial journals and TV programs talk about it. This interest can be used as motivation for teaching, even in topics not directly related to robotics because it stimulates social abilities, teamwork and creativity. There are numerous studies about this like [Johnson, 2003],[Goldman et al., 2004],[Alimisis, 2013],[Eguchi, 2013] and [Benitti, 2012].

As a consequence, there are many robotic competitions for students where they build and program a robot for a specific purpose. For instance FIRST Robotic Competition, presented in [Haynes and Edwards, 2015], is a competition for teams of 25 or more students to build industrial size robots to play a different game each year. RoboCupJunior, described in [Eguchi, 2016], is a student robotic sports competition with different disciplines such as football or dance.

Taking this motivating factor into account, an application for education has been designed and used with students. Although the absence of physical robots is a disadvantage, using a benchmarking on the cloud simulator like the one presented in this kind of application has some interesting advantages from the point of view of the educator like the following:

- **Faster development:** Abstracting from physical robot details helps to shorten programming and debugging times. Furthermore, results are repeatable, making it easier to properly identify weaknesses and understand the code. On the other hand, students do not face real problems that may appear when using the software in the real platform. In order to deal with this, some robots may be available as a final step in the work, combining the advantages of both worlds.

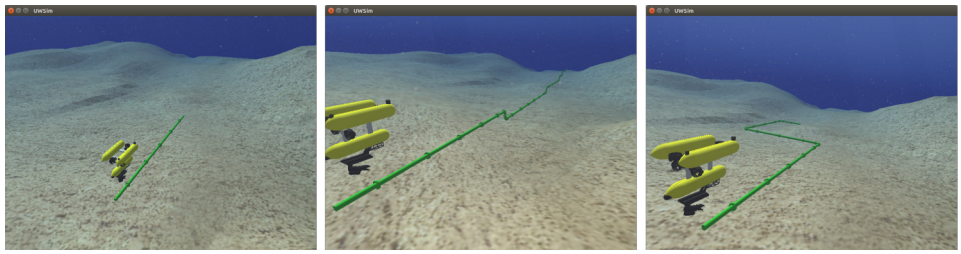

Figura 3.9: Different scenarios in the pipe following benchmarking use case. From left to right straight, height changes and turns.

- **Availability:** In these kind of applications a robot per student team is needed. However, robots are expensive and break down often, requiring frequent repairs through their lifetimes. Using a simulator assures every student will have a working framework to work with, and also allows expensive sensors to be dispensed with or robots to be used that may not be available otherwise.
- **Continuous learning:** It is easy to control the learning curve and be sure it is not too steep using information from the simulator when needed. For instance, it is possible to deactivate water dynamics in order to learn how to move the vehicle, and once the student feels confident enough activate it to increase difficulty.
- **Competitiveness:** Being able to instantly compare with other students or the teacher, motivates students to "win", and makes students work harder to obtain the best possible results. Furthermore, students are able to replay other solutions and study why other teams are obtaining better results.
- **Specific formation:** As the developed software runs in a simulated environment, it is possible to focus on a specific part. For example, it is possible to deactivate physics and focus on vision algorithms, or obtain ground truth object positions and learn robot manipulation techniques.

For this reasons, a pipe following application has been proposed to students of telerobotics, from the master in robotics EMARO+, so that they can progressively learn navigation, vision algorithms and basic control. The aim is to follow green underwater pipes to detect possible leaks in them[2] . Three different scenarios, see figure 3.9, are designed to face different difficulties that may appear in real scenarios such as straight lines, changes in heights and turns.

Moreover, an example implementation result is added to the comparison by the teacher, thus students can compare and analyse the differences with their solution. It also motivates students to achieve a better result, and try to "win" and achieve better results than the teacher's implementation.

The problem is divided into three steps to maintain a moderate the learning curve. Each of these steps is evaluated separately, so students can check their progress as they work. This means nine different evaluations are computed one per

[2]Dynamic pipe following: https://www.youtube.com/watch?v=2rjFQoixIRs

scenario and one per step. In the first step, students are asked to solve the problem in a teleoperated manner, thus sending commands from keyboard input. Secondly, the keyboard input is substituted by autonomous navigation with predefined waypoints. Finally, the vision algorithm is introduced, making it necessary to obtain waypoints from a camera sensor available in the vehicle.

In order to make the work easier for students, the ideal trajectory and the trajectory followed by the vehicle are shown in the rendered scenario with a green and red line respectively to provide visual feedback. Additionally, all the required software is provided in a virtual machine, so students can start working immediately.

Algorithm evaluation is carried by the benchmarking module. It uses the Integrated Squared Error(ISE), the sum of the shortest distance to the ideal trajectory in each time step. The performance is inversely related to this measure, as it increases with time and positioning error. The greater the speed and precision with which an algorithm is able to reach the end of the pipe, the better it will be evaluated. The final position error is also used to make sure the vehicle correctly reached the end of the pipe.

Taking into account these measures, the algorithm is evaluated depending on the average error $avg(error)$, final error $finalError$ and time $time$. Additionally, each scenario has a time reference $timeRef$, experimentally estimated. This reference describes the expected time for the operation, taking into account pipe distance, turns and height changes. The final mark is computed following equation 3.2.

$$mark = \left(1 - finalerror^2\right) * \left(0{,}1 - \frac{avg(error)^2}{0{,}1}\right) + \frac{time - timeRef}{100} \tag{3.2}$$

This evaluation equation proved to be an objective manner by which an evaluation for the pipe following problem could be provided. The first term evaluates the final position of the vehicle. If the vehicle didn't reach the end of the pipe it will be penalised. The second term is the average error through the intervention, measuring the precision during the intervention. Finally, the last one is a time bonus depending on the time reference that rewards faster executions. Furthermore, the results interface provides information of the evolution of these parameters through the operation as figure 3.10 shows.

As can be seen in this capture, the students chose different strategies for the pipe following predefined waypoints. The students "al01" and "al03" designed precise solutions that required more time to complete the path. Even though, the average and final error is really small they are heavily penalised for the time required and obtain a worse mark.

On the other hand, faster strategies such as "al02" and "al08" forget about the errors and try to move faster to the target position. This strategies are penalised in the average and final error obtaining a medium mark. These results prove the metric for evaluating the results is focusing on balanced strategies that try to accomplish the task precisely and as fast as possible as in the case of "al07" or "al06".

The complete results are available in http://www.irs.uji.es/uwsim/files/benchmarks/pipefollowing.html with the comparison of the student results. The students were asked to complete the dynamic version of the waypoints part and optionally the vision task, besides the kinematic solutions. As can be seen, all the students managed to complete the dynamic waypoints task reaching different

Figura 3.10: ScoreCard of the benchmarking on the cloud feature showing a pipe following results summary.

marks. Furthermore, three students out of eight proposed a solution for the dynamic version of the vision guided pipe following.

Although, only one solution was able to reach the end of the followed pipe in the most challenging environment, the students were motivated to work further than the required exercises proving the validity of the experiment. Furthermore, all the students developed a fully functional solution for the waypoints case. Additionally, complex concepts such as controllers were naturally learned using the example understanding the need of these tools.

3.6. Potential applications

Besides the previously described use cases, the presented simulator on the cloud can be used in very different applications. As discussed in the pipe following example, is an excellent tool for education and may be used in many disciplines related to robotics, and specially underwater robotics, such as vision, navigation, manipulation or control. But, it can be also used in research, proposing benchmark for specific problems allowing researchers to compare their results.

Furthermore, it is possible to use it as a tool to help with the integration problems derived from big projects as it helps to establish a common workplace where integration problems may arise. Additionally, it forces to standardize the communications and architectures used in an early stage saving time in critical moments: field experiments.

Regarding simulation capabilities, an online simulation service permits dividing the developing device from the machine that is actually running the simulation. This division makes possible to use cheaper and more comfortable devices such as tablets, or small laptops to design experiments and launch them in a more powerful, and less portable, machine without loosing precision. Moreover, it is possible to share a unique simulation machine and optimize resources in research groups.

However, in order to extend its use to other applications it is necessary to configure some parts, apart from the hardware needs. On account of simplicity most of the code developed has been designed in a generic way, permitting to reuse it with different purposes. For instance, the software to upload results to the cloud can be used with any benchmark as it automatically uploads the results, but requires a "format" to label it appropriately.

Taking into account this, the required steps to create a new benchmark on the cloud are the following:

- **Configure the scene:** Basically, create the necessary 3D models and robot description files for the simulation and create an XML file with the scene configuration, sensors and interfaces. It is the same requisites as configuring UWSim locally.

- **Configure the benchmark:** Similarly, an XML file describing the measures and scene updaters that will be used in the simulation is required. If the measure or scene updater is already implemented it is possible to directly use it, but if a new one is required it needs to be coded in the benchmarking module.

- **Configure the online execution:** Once the benchmark can be launched locally, it is necessary to edit the online execution service. For security reasons the launcher does not allow to run arbitrary scenes or benchmarks, but it just requires to modify a few lines to add a new configuration so it can be remotely called using the same service. Besides this, the results in the website need to be configured describing which output of the benchmark will be showed in a graph and the type, legend and title of each graph.

- **Configure the cloud comparison:** Finally, the cloud comparison requires some steps. A spreadsheet is required to place the results of the experiment. The ID of this spreadsheet will be required together with a user ID configured through the Google API for each participant in the cloud comparison. Additionally the scorecard needs to be adjusted for the experiment, showing the appropriate graphs from the specific benchmark results uploaded to the spreadsheet.

3.7. Conclusions

In this chapter, the online capabilities of UWSim have been presented. Built on the underwater simulator UWSim and the benchmarking modules previously described in Chapter 2, online execution and benchmarking on the cloud services have been implemented.

Due to the new trends of online services, that make it possible to work everywhere with any Internet connected device, robotic simulators need to adapt in order to be useful to researchers. Following this idea, the underwater simulator UWSim has been extended allowing researchers to access simulation capabilities everywhere with any device through a web interface. The advantages of these systems are availability, computation capabilities independent from device and offline simulation.

Furthermore, the benchmarking capabilities have even more possibilities in this environment, because it opens the door to objective comparison between researchers. Taking advantage of the online services, there is no need to install software and it is possible to create a shared results space where users can evaluate their implementations. It is also possible to store enough information to visualize an intervention and analyse possible weaknesses to improve the algorithm.

Both developed services have been used in a different experiment. The online simulation has been used in a station keeping experiment where two different position controllers were tested in increasing current conditions. The results showed the validity of the developed framework demonstrating PI controller performed better than P as expected. But, the real results are that the framework was able to do it using three different architectures: online, Matlab-Simulink and Python. These simulations achieved the same results independently of the implementation details. This proves the developed framework is able to objectively evaluate algorithms from any source, without introducing any noise or perturbation.

Due to the increasing interest for robotics in society, the use of robotic platforms has proved very motivating for students. For this reason, in the case of benchmarking on the cloud, an educational robotics application has been designed for validation. This is an interesting application as benchmarking can help to evaluate the students, and at the same time, motivate them to "win" in a competition like environment thanks to the automatically computed results.

The results of this application showed that students were able to easily learn and understand the basics of navigation, control and vision. The students compared different strategies and analysed the results using the proposed architecture to develop their pipe following approach. Furthermore, some students were motivated to work on additional tasks developing solutions for optional exercises.

Many interesting applications and features remain open for the future. Some of the most powerful robotic simulators have already started to develop their online features, meaning this market may be exploited in the future. The possibilities range from basic education for children to high end industrial robot simulations. All the key elements are already available, but the main difference will be in the communication interface, documentation and simulation capabilities. A powerful interface able to control all the simulation characteristics and visualize it when running is still needed, and an open research field. In this case, a simple application focused interfaces have been developed to demonstrate the possibilities of an online robot simulator.

Capítulo 4

An application perspective of benchmarking

Comparing solutions and establishing common metrics and benchmarks is one of the cornerstones of scientific research. Being able to objectively evaluate an application, method or procedure or at least decide if it is better than another option, is of utmost importance for enabling relevant progress. Furthermore, allowing suitable characterization of reproducible solutions is crucial to improve its robustness.

However, in the robotics field reproducing the results or comparing with other published results is difficult in some cases as pointed out in [Bonsignorio and Del Pobil, 2015]. Different hardware, software barebones and sometimes lack of details makes it difficult to reproduce the results of an alternative solution. Increasing the difficulty of working on the results of another researcher.

Moreover, robotics is not pure mathematics, thus a completely viable mathematical solution may not be valid in some cases. Sometimes experimental proofs of the performance of the proposed approaches are needed. As stated in the Group on Good Experimental Methodology and Benchmarking (GEM) guidelines of the European Robotics Network of Excellence (EURON) [Bonsignorio et al., 2008], it is necessary to validate the results by replicating them or comparing them in terms of the chosen performance criteria.

Even though robotics deals with a wide variety of problems, standard benchmarks, metrics or protocols can be established to validate the results. Establishing such standards will help to improve the quality of the research and contributions in the field.

In this thesis different software pieces are devoted to this specific issue. The framework presented in Chapter 2 formed by the UWSim simulator and the benchmarking suite assure an objective comparison of algorithms. Furthermore, this framework has been extended to work in the cloud as described in Chapter 3 adding interesting capabilities to the system. In Chapter 5 this software is used to study the influence of water turbidity in different aspects of autonomous underwater vehicle interventions.

Nevertheless, the proposed benchmarking framework has also been used in other works not related to water turbidity or image dehazing. In this chapter, some

of these experiments are described to show the capabilities and versatility of the presented benchmarking framework.

4.1. Characterizing real scenarios

During the TRIDENT project, two experimental validation tests were performed at sea. As a great quantity of data was gathered for further processing it is possible to reproduce the complete experiments in a simulated scenario to analyse them and be able to establish a benchmark to compare them with.

The presented framework was used to monitor the system, gather information and analyse it through benchmarking. The complete intervention was monitored using the simulation framework as a representation of what was happening under the water. The measures from sensors, positioning systems and cameras were mixed to provide a simulated model of the vehicle and environment showing a view of the real intervention. Furthermore, the complete intervention was logged in bagfiles, ROS log data files including the most relevant information, for further study.

The analysis consists of reproducing the experiments in a simulated setup with the information from the real intervention. Thus allowing a 3D view of the intervention to study the weaknesses and errors and improve the final system. In order to do so, a complete representation of the environment was reconstructed from the logged information. After that, a simulated model of the vehicle replicates the real movements in the simulated environment allowing to playback the real intervention in a 3D view.

The use of a free 3D view, being able to draw trajectory lines or simulated cameras can help to discover weaknesses or why the system did not achieve the expected results. Additionally, it is also possible to use the benchmarking suite to easily measure statistics such as travelled distances or positioning errors and evaluate the performance of the system after the intervention took place.

An example of this situation can be seen in figure 4.1, where a playback of the real intervention referenced to the reconstruction of the environment is showed. As can be seen, the trajectory of the end effector has been added to show how the hooking of the black box was performed. The image corresponds to the first experiments in Roses harbour, Girona (Spain)[1].

As can be seen, the replay of the intervention helps to understand what happened underwater, which may be difficult using only the video from the robot or a diver video. The image shows that the approach used by the end effector can be improved, even though it successfully hooked the target.

Besides the experimental analysis, the reconstruction of real environments increases the realism of the simulator allowing new situations to be explored and shortening the gap between real experiments. These new scenarios can be used to try new algorithms under development and test them in a simulated representation of the final location.

In the case of the final field experiments in Port de Soller (Mallorca, Spain), the reconstruction used a 3D point cloud besides the photomosaic also used in the first experiments. Using this information made it possible to create a 3D model of the

[1] Video TRIDENT roses: https://www.youtube.com/watch?v=ouCFOs0mnyA

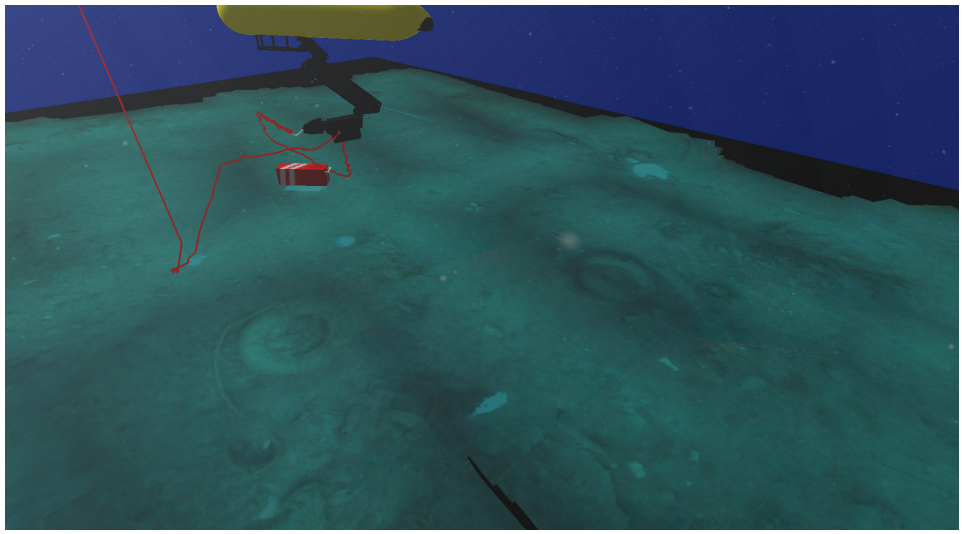

Figura 4.1: Playback of the real intervention in Roses field experiments using UWSim.

terrain that can be used for further analysis of the intervention or new simulations. The reconstructed terrain can be seen in figure 4.2.

As can be seen, the survey path, displayed in white lines, perfectly matches the reconstructed terrain changing depth. Furthermore, rocks, objects and a slope can be seen close to the harbour wall. This 3D representation proved to be useful to understand the vehicle movements during the experiments.

Finally using the benchmarking tool it was possible to establish the conditions of the experiments in order to compare them with other alternatives. For instance, survey altitude, distance, average velocity, time, maximum minimum and average seafloor depth or visibility were computed from the logged information.

The characterization of the Roses and Soller interventions of the TRIDENT project can be seen in table 4.1. Using these characteristics it is possible to establish a point of comparison between both experiments, and also with following experiments in order to compare the results. The parameters measured are the same but in the case of Soller the intervention is slightly different, as the grasping approach did not require a previous visual algorithm.

But in any case, it is possible to compare both interventions. For instance, the survey covers a much bigger terrain in the Soller experiments, thus the reconstructed terrain is bigger as can be seen in the figures. However, the required time for the survey is similar in both cases as the vehicle moved faster during the second intervention. For this reason, the second intervention is clearly better in the survey phase because it obtained information of a bigger zone in almost the same amount of time.

Regarding the altitude from the seafloor the Soller intervention required the vehicle to fly higher from the seafloor, 1.5 meters due to the rocky environment. This fact can make the resulting mosaic more imprecise as the cameras were further from the target. However, the visibility in Soller was slightly better making it easier

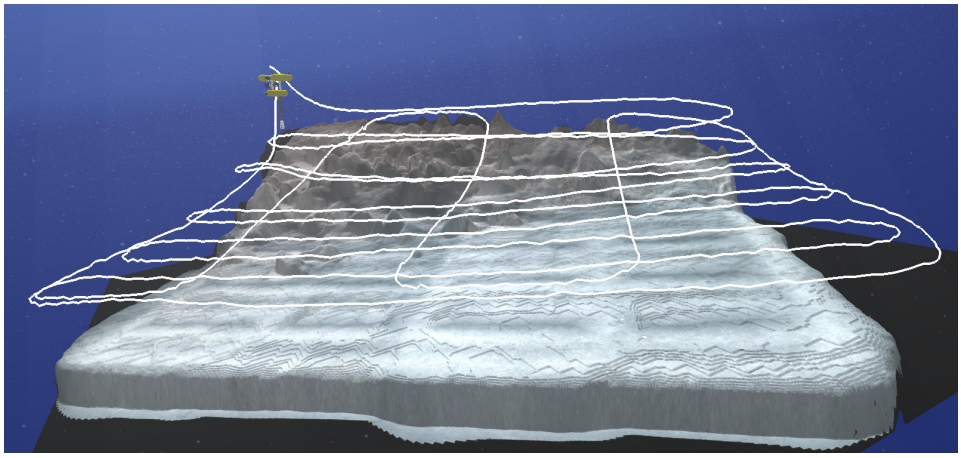

Figura 4.2: Playback of the real survey in Soller field experiments using UWSim.

Cuadro 4.1: TRIDENT field experiments intervention characterization.

	Survey	
	Roses	Soller
Surveyed area	8x10 m	25x20 m
Survey grid	3 rows x 6 cols	5 rows x 11 cols
Survey altitude from seafloor	1.3 m	1.5 m
Survey distance	82.9 m	357.67 m
Survey average velocity	0.1 m/s	0.37 m/s
Survey time	814 s	957 s
Maximum seafloor depth	3.19 m	8.18 m
Minimum seafloor depth	2.12 m	4.88 m
Target depth	3.07 m	8.08 m
Water	Sea water	Sea water
Visibility	1.5 - 2.5 m	2 - 3 m
Currents	Yes	Yes
	Intervention	
	Roses	Soller
Target	12 x 15 x 40 cm black box mockup with 3 cm handle	
Approach	Visual station	Visual free floating
Time of approach	87 s	-
Intervention	Hooking	Grasping
Time of intervention	67.5 s	143 s

to recognize objects from the distance thus in both cases the results are similar.

The roses environment is slightly less challenging due to the extremely shallow waters. The maximum depth for the intervention was 3.19 meters. This makes the work possible without additional illumination. Although in the Soller intervention this was also possible, processing the images was more challenging.

In the case of the manipulation, the most significant difference, besides the control strategy, is the grasping tool. The first intervention required that there was a handle on the object in order to hook on it and retrieve it, while the second case is capable of using a dexterous hand to grasp the object. Consequently, the Soller approach is a more flexible solution that can be applied to a wider range of objects. Regarding the control strategy, in the Roses field experiments a station keeping approach was used while in Soller a more robust visual free floating approach was taken.

In conclusion, benchmarking real interventions makes it possible to study in detail the results in detail and establish metrics to compare between different approaches. Furthermore, the correct parametrization of the interventions helps other researchers to compare and understand the results of their experiments.

4.2. Dredging benchmarking

Another problem studied using the presented framework is a dredging intervention to unearth an amphora or similar object placed in the seafloor[2]. One of the testbeds of the MERBOTS project is marine archaeology, this involves multiple challenges such as detecting possible shipwrecks, dredging the mud around the target and carefully manipulating the objects.

Furthermore, autonomous manipulation of these objects may not be the best approach due to the value of the manipulated objects. Thus, a semi autonomous solution where a user can monitor and decide high level commands is necessary. For this reason, a framework such as the one presented in this thesis is an interesting tool making it possible to monitor and easily specify high level actions so that the required action can be carried out.

In order to demonstrate the capabilities of the system, a simulated experiment involving the dredging intervention was designed. The main goal was to autonomously unearth a buried amphora placed in the seafloor using a dredging tool attached to a robotic arm. In order to do so, the framework presented in Chapter 2 was used together with Simurv 4.0. An example of the initial and final state of the target can be seen in figure 4.3.

Simurv, described in [Antonelli, 2014] is a kinematic and dynamic library for Underwater Vehicle-Manipulator Systems (UVMS) control algorithms available in Simulink - Matlab. Basically, it simulates the movements of vehicle with manipulators in an underwater environment. The default dynamic simulator of UWSim is substituted for simurv in order to increase the precision in dynamic multibody simulation.

However, simurv does not simulate the environment, cameras, dredging... required for the simulation. Thus UWSim is used to simulate the world and communicates with simurv in order to use the dynamic multibody simulation. Furthermore,

[2]Simulated dredging intervention: https://www.youtube.com/watch?v=sCoYF5WSI8Q

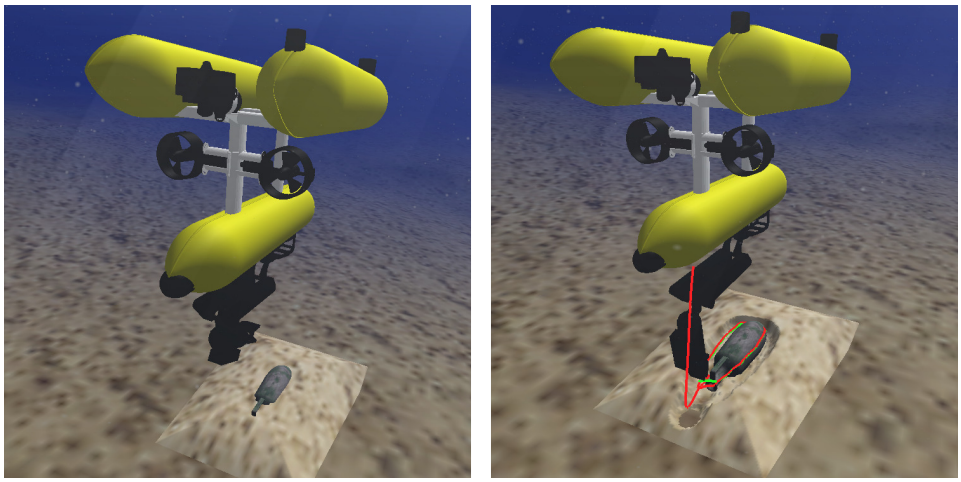

Figura 4.3: Initial and final state of the dredging intervention.

it is also possible to develop the kinematic control using simulink tools as simurv is already integrated in it.

The benchmarking module is in charge of measuring the results of the experiment. This could also be achieved using Matlab, but the UWSim benchmarking module can access groundtruth information from the simulation that is not available for Matlab. Thus, the evaluation can be made from a more objective perspective instead of relying on other sources of information.

The Intervention autonomous underwater vehicle is made up of the Girona 500 AUV [Ribas et al., 2012] with 6 degrees of freedom (DOF) and the light weight ARM5E [Fernández et al., 2013] with 4 DOF assembled on it. The vehicle will move using a free floating strategy that uses the DOF of the vehicle and the arm to control the movement of the dredge tool through a desired trajectory.

It is assumed that an archaeologist supervises the trajectory to be followed by the end effector using a 3D reconstruction of the area obtained in a previous intervention. Furthermore, the target is selected by the end user in that georeferenced 3D map making it easier to create a trajectory from it.

Using this previously gathered data, a 3D point cloud of the target of interest is the starting point for generating a path to be followed by the end effector of the arm where the dredge pump is located. RANSAC based shape detection methods [Schnabel et al., 2007] are used to segment the scene and extract the object points of the pointcloud, discarding the seafloor. Then a path is calculated where this points of the object collide with the seafloor, adjusting them to avoid colliding with the target or the seafloor.

As a consequence, the metric used to benchmark this experiment is the distance between the actual path and the optimal path to dredge the objects. The time elapsed could also be considered using an Integrated Squared Error (ISE), but as this is a precision task that should prevent collisions with the target, it was finally discarded. This assumes a user or archaeologist has previously checked the automatically created path to ensure that it is correct.

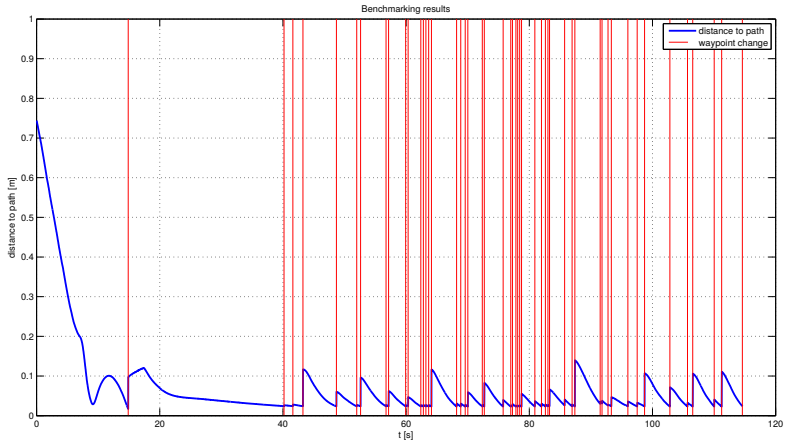

Figura 4.4: Dredging pump distance to the desired trajectory during the intervention.

The results of this benchmarking can be seen in figure 4.4 where the distance to the trajectory through time is depicted. The moment when the end effector arrives at a waypoint and decides to move to the next is marked with vertical red lines. As can be seen the initial distance is high and the controller reduces this quickly. In spite of using a simple controller, the end effector is able to correctly follow the path. The distance to the desired trajectory is limited, only small increments due to target waypoint changes increase the error.

As the results show, the metric correctly evaluates the dredging intervention without evaluating implementation details. Thus it is possible to use it for comparison with very different approaches in an objective manner. Furthermore, the logged intervention allows the intervention to be played back and new metrics or error that may be interesting to evaluate to be measured.

4.3. Natural user interface evaluation

The latest experiments demonstrate it is possible to evaluate devices and users instead of the real or simulated algorithms of previous experiments. This is possible by comparing the results when completing a task designed to be challenging while using this devices. In this experiment, different devices designed to pilot a virtual underwater vehicle are compared in terms of the advantages when moving around a virtual scene[3].

Controlling an underwater robot using a camera mounted on it is a difficult task. This is caused by the lack of specialized training in using robots, lack of time to interpret the information or missing information. As can be seen, the three causes can be avoided, or at least reduced, using a good human machine interface

[3]Natural interfaces used: https://www.youtube.com/watch?v=mUun7yM8238

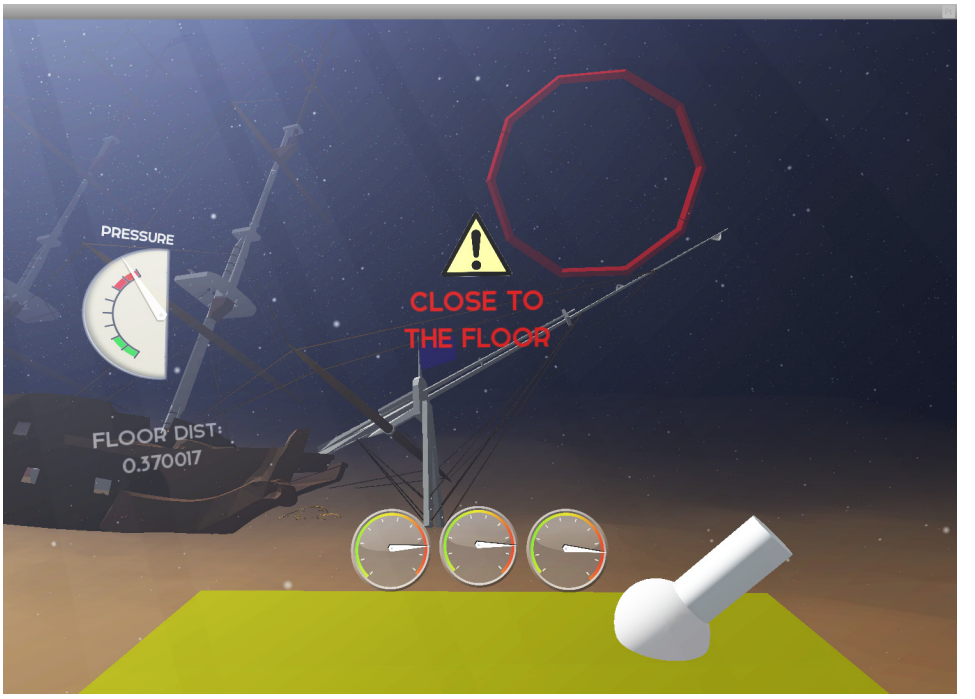

Figura 4.5: Simulated scenario for natural interface evaluation.

(HMI) that provides the correct information at each moment and is easy enough to use for non expert users.

Taking this into account a natural user interface has been designed giving users the sensation of being inside the robot thus piloting the robot becomes as natural as driving a car. In order to do this, the users should be allowed to control the camera orientation with head rotations and control the vehicle using the hands.

However, the advantages of this interface must be evaluated as new problems with the interfaces being used may arise. To evaluate this, a user experience experiment was conducted where 26 participants were invited to control a simulated underwater robot using slightly different setups. The participants had no previous experience with the interface, although 17 declared previous experience with video games. The performance of these participants was benchmarked measuring different parameters relevant to the experiment.

The proposed testbed is an underwater environment where users have to pilot the vehicle along a path through different rings. Furthermore, the simulator dynamics will detect collisions with the rings and any other obstacles in the scene penalising the users that collide with them. The scenario can be seen in figure 4.5.

The framework described in Chapter 2 was used for this purpose. However, this experiment required some modifications in order for the natural interface to be tested. First of all, before a head-mounted display (HMD) that tracks the user's head movements to move the camera accordingly, and offers an immersive virtual reality 3D perspective of the scene, could be used, it is necessary to set up stereo visualization.

Additionally, the use of two devices to move the vehicle were implemented, a classic joystick and Leap Motion. Leap motion is a device capable of detecting the hands of the user at a short range, thus transferring the hands position to robot movements allows the vehicle to be moved intuitively in the simulated scene.

However, the use of a head mounted display abstracts the user from the real environment making it difficult to use other devices. For this reason, different feedback techniques were implemented to facilitate the user task and to help him be able to achieve the desired goals.

A virtual cockpit was added to the UWSim visualization to simulate being inside a real vehicle and show relevant information such as depth, speed or proximity to obstacles. Furthermore, the virtual cockpit also has a virtual representation of the joystick showing its movements so the user knows if the command sent was correct.

Moreover, it is difficult for the user to use Leap Motion without seeing the device directly, because it is not possible to judge the relative position of the hand and the device. In order to solve this, two alternatives are proposed, the use of a small fan so the user can sense the airflow and perceive the device position, and the possibility of including the user hands in the virtual cockpit using an additional RGB-D sensor.

In summary, the evaluated users performed the experiments with five different setups. The five setups differs in the type of visualization and control in order to obtain the best possible result. The sequence of experiments is randomized to avoid learning effects through repetitions. The different configurations that were tested are listed below:

- **Setup 1:** Joystick teleoperation with no virtual cockpit.
- **Setup 2:** Virtual cockpit and head mounted display for visualization and camera orientation. The vehicle is controlled with a joystick.
- **Setup 3:** Virtual cockpit, head mounted display and leap motion device with no feedback.
- **Setup 4:** Same as setup 3, with virtual representation of hands in the virtual cockpit
- **Setup 5:** Virtual cockpit, head mounted display and leap motion with airflow haptic feedback for device localization.

In the figure 4.6 the evaluated setups are shown. As can be seen the interface is increasingly more natural and immersive, but at the same time more complex. The experiment tries to determine the best option for non expert users so that it can be made as easy as possible to use and produce the best possible results.

In order to do so four parameters have been automatically measured using the presented framework explained below. Additionally, a subjective evaluation from the users was made asking the participants to fill in a questionnaire related to the ease of use, fatigue, precision and frustration encountered while operating the system.

- **Time:** Navigation time for each segment of the route between rings. Rewards completing the path quickly.

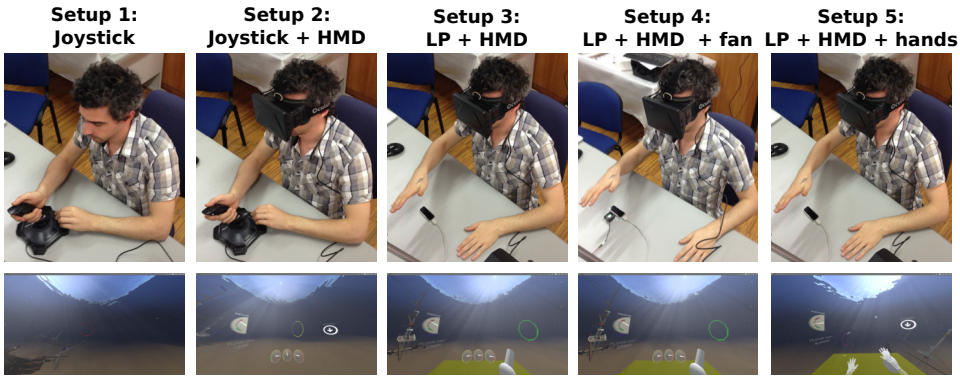

Figura 4.6: Screenshots and images of the evaluated natural interfaces setups.

- **Distance:** Measures the travelled distance for each segment between rings, thus penalising non linear trajectories.

- **Collisions:** Counts the number of times the vehicle collided with elements in the scene, including rings.

- **Steerings:** Measures the number of changes in direction or steering commands. A high number in this value will mean the user needed to correct the trajectory many times.

Using this experimental setup it was possible to identify the control set up that most favoured the non-expert user. The results showed that setup 2, head mounted display and joystick, obtained the best mean times for all the segments between rings. The users were capable of reaching the trajectory waypoints faster, with less collisions and travelling less distance.

Although the leap motion devices seem a more natural interface, the results showed it was difficult to manoeuvre the vehicle using it, leading to higher times and collisions. The use of airflow feedback helped in the case of rotations as the users could feel the position of the device and the hand visualization helped in altitude changes. But, the users still felt that the device was more imprecise, supporting the worse results with respect to joystick.

Regarding the first two setups, setup two with a head mounted display proved to be a better option as the user could naturally move the camera using the head, finding the targets faster and requiring fewer steering orders.

Finally, the use of a joystick to control the vehicle produced better results even without the head mounted display. Previous knowledge and the fact that the joystick device was easy to use proved to be a better option. The participants felt the leap motion device was imprecise and perceiving mechanical feedback was necessary for the experiment.

4.4. Conclusions

In this chapter three different cases where the presented framework was used have been described. The nature, objectives and evaluated software were completely different in each of them showing the versatility and usability of the designed benchmarking framework for underwater robotic research.

In the first case, real experiments are characterized and analysed using the UWSim simulator and benchmarking suite. Several parameters are measured using the logged interventions and that also allows the vehicle movements to be played back in order to have an additional view of the system for better understanding of the results.

In the dredging scenario a simulated unearthing experiment is evaluated using the described framework. Furthermore, the software uses an specific dynamic simulation engine in Simulink together with the controller to be evaluated. Even in this situation, the benchmarking suite is able to objectively measure the results and proves to be a valuable software for the development of new solutions analysing the weaknesses and results.

Finally, the natural user interface evaluation measures the user ability with different devices for controlling a robot. Comparing the results of these experiments it is possible to understand why a device is better for a specific task and the influence of a stereo device when using an underwater vehicle.

As can be seen, the benchmarks are completely different. In the first case a real intervention is analysed while in the second one a controller is benchmarked in simulation and in the third case the focus is on interface devices. This proves the proposed architecture offers a simple architecture to objectively evaluate and compare algorithms. The only essential requirements are the use of ROS as middleware and the configuration of objective metrics to evaluate the problem.

Furthermore, the software to be evaluated can be implemented in any language as long as it is ROS compatible thus it can communicate with the rest of the architecture. The use of UWSim is also interesting as the benchmarking suite can access internal data, thus obtaining groundtruth information that does not need to be transmitted.

The research leading to the results presented in this chapter is the result of collaboration with José Javier Fernández Fresneda in [Fernández et al., 2015] regarding dredging, and Juan Carlos García Sánchez in [Sanchez et al., 2015] regarding natural interfaces. The corresponding experiments, besides the benchmarking, will be part of their corresponding PhD. Thesis.

Capítulo 5

The problem of water turbidity in underwater robotics

One of the main drawbacks in underwater robotics is the difficulty of interpreting a captured image due to the water turbidity. Small particles reflect light producing a veil that superimposes in the image. The light is absorbed as it travels through the water column reducing the illumination of the environment. The light is randomly scattered blurring the image. Finally the light is attenuated depending on its wavelength producing bluish or greenish tones. Consequently, recognizing objects in these degraded acquired images is a challenging task.

In this chapter, the framework presented in Chapter 2 is used to measure the consequences of these degrading effects on image based algorithms. Being able to establish functional limits to the evaluated software, and know which solution to use depending conditions in the environment. For instance, to learn which is the best object tracker for non degraded images and for turbid waters, and then, in the real intervention, be able to choose the best option depending on the state of the water in which the operation is taking place.

Three different experiments are presented to show the impact of water turbidity on image based algorithms. Firstly, a visibility experiment, described in [Pérez et al., 2013], is performed where a set of visual trackers is evaluated in an object tracking situation as simulated turbidity increases. After that, the same experiment is repeated in a hardware in the loop environment using real images as input. Finally, a real experiment of 3D reconstruction that was published in [Perez et al., 2015] is shown, using illumination to simulate different conditions comparing a stereo camera and laser stripe reconstructions.

5.1. Introduction

The experiments which are presented here are focused on the TRIDENT project experimental scenario, that was a search and recovery problem. In it, a completely autonomous underwater vehicle had to look for a black box mock-up, previously placed on the sea floor, and recover it. Moreover, since it is rarely possible to predict the water state at sea, it is necessary to create a robust system.

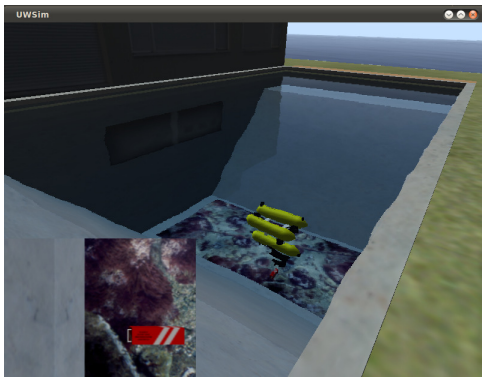

Figura 5.1: Visibility benchmark scenario. The Girona500 vehicle is placed above a target object to keep track of it while the visibility decreases.

Taking this into account, it is of the utmost importance to test the algorithms under a great variety of situations. One of these is when the water turbidity changes. Due to the underwater currents and changes in the composition of water, underwater visibility can suddenly change, making it difficult to see through the water. In the case of underwater robotics, these water turbidity affects any algorithm that uses a camera, which is one of the most important inputs to solve any problem.

In the TRIDENT project two tasks were highly dependant on cameras: localization and tracking of the target, and grasp planning. In the first case, it is necessary to be able to recognize the target, in this case a black box mock-up, at a certain distance allowing the vehicle to safely navigate in the area without colliding with anything while searching. The second problem, requires being able to extract 3D characteristics in order to compute the best grasping points to retrieve the object from the sea floor using a robotic arm.

In order to benchmark the first situation, an experiment is presented in section 5.2 recreating the environment in simulation. Then, in section 5.3, a hardware in the loop configuration is shown to test it in a more realistic environment. The second situation is studied in section 5.4, where different approaches are compared reconstructing in 3D the target black box mock-up.

5.2. Visibility benchmarking

In this experiment the purpose is to analyse how the change in visibility affects vision algorithms of detection and tracking. In the simulated setup shown in figure 5.1, the Girona500 vehicle with the ARM5E arm is situated above a black box target tracking it for further manipulation. The scene visibility is progressively decreased until the tracker is not able to find the target to test the limits of the algorithm to be compared. The experiment simulates the first phase of an intervention, like the one in the TRIDENT project, where the water turbidity changes due to an external cause like currents or mud from the bottom and the robot tries to keep track of the target.

5.2.1. Visual trackers compared

The trackers compared are the ones available in the ViSP (Visual Servoing Platform, INRIA Lagadic), presented in [Marchand, 1999], for template tracking. These trackers use image registration algorithms instead of the common feature based approaches, being able to find the target in a sequence of images. Two different similarity functions can be configured so that they can be used with different methods.

- Sum of Square Differences (SSD): That can be used in four ways, Inverse Compositional (IC), Forward Compositional (FC), Forward Additional (FA) and ESM.
- Zero-mean Normalized Cross Correlation (ZNCC): In this case only two of the previously mentioned methods are available, Inverse Compositional and Forward Additional.

Besides the registration criterion, ViSP allows different warps to be used. Warps describe the transformation of the tracked template between images with respect to the camera. Deciding which one to use, depends on the degrees of freedom of the tracked object in the scene. Usually, the most restrictive warp that can model the target movements in the camera is the best option. The ViSP library provides five warp alternatives to choose from:

- Translation: The most simple transformation available. It only considers translation in the two axis of the camera(x,y). It is only suitable for planar movements at a constant distance from the camera.
- Scale Rotation Translation(SRT): This warp considers a scale factor, rotation on camera axis(z) and a 2D translation as the previous warp.
- Affine: This warping function preserves points, straight lines and planes.
- Homography: Estimates the eight parameters of the homography matrix H.
- HomographySL3: Stores the same information as the homography warp but, the parameters are estimated in the SL3 reference frame.

Every possible combination of registration criterion and warp has been tested in the visibility change benchmark. As the tracked target has little freedom of movement with respect to the camera, the most restrictive warp should obtain the best results in each similarity function.

5.2.2. Experimental setup

The benchmark measures the tracker error as can be seen in the software architecture in figure 5.2. The error is measured in pixels as the sum of the error of each target's corner with respect to the tracker estimation nearest corner. Additionally, the centroid error between tracker estimation and simulator ground truth is also measured. The fog updater is in charge of updating the scene to progressively decrease the visibility. In order to do this, it modifies the fog parameter of osgOcean.

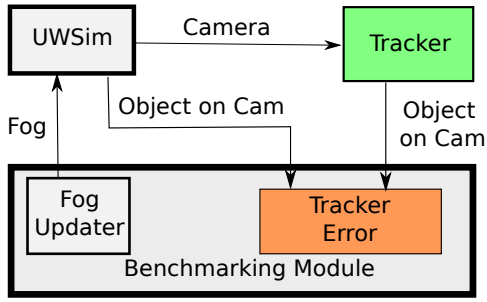

Figura 5.2: Software schema used in the visibility benchmarking. Orange: benchmark outputs, Green: evaluated software.

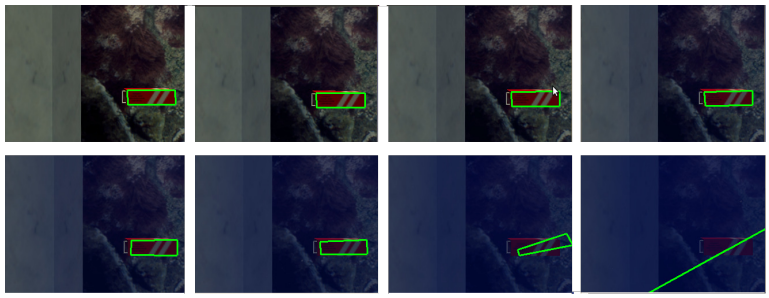

Figura 5.3: From left to right top to bottom, tracking algorithm in decreasing visibility scenario.

The fog level is a value ranging from 0 to infinity that defines the visibility in the water depending on the distance. Visibility will represent the intensity of objects' color that it is finally seen in the camera from 0 to 1. A 1.0 visibility is a perfect situation where water does not degrade the image while in 0 visibility it is not possible to see the object at all. The color of the object is mixed with the water color depending on this factor. The visibility depends therefore on the water fog level and the distance to the object as the formula 5.1 describes.

$$visibility = e^{-(fogfactor*distance)^2} \tag{5.1}$$

As can be seen the visibility exponentially decreases with distance and the fog factor, simulating a real underwater environment. As the distance is constant in this experiment, the visibility only depends on the fog factor. In the figure 5.3 different screenshots of the benchmark execution can be seen, where the fog factor controls the visibility of the target.

Finally, a small current has been added thus the vehicle moves slightly over the target rather than being perfectly stationary. It is a precalculated cyclic movement that guides the vehicle in a circular trajectory maintaining the target inside the camera at every moment. Thus, the tracker needs to keep track of the target as it moves inside the virtual camera.

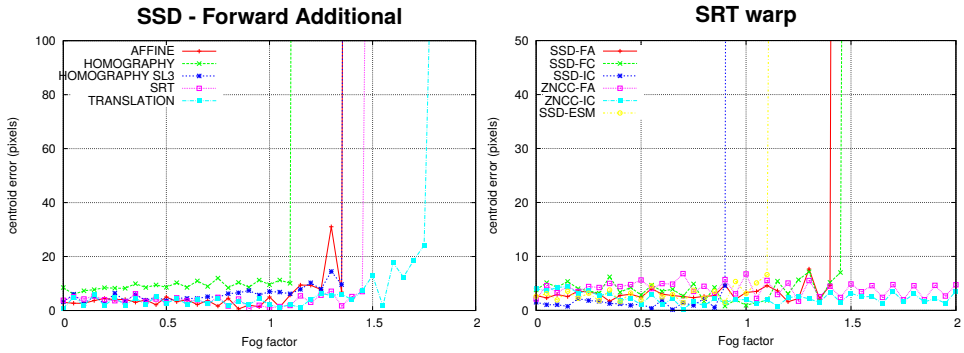

Figura 5.4: Results for the benchmarking visibility use case. Left: warp performance comparison, right: registration criterion comparison.

5.2.3. Results

The results are shown in figure 5.4. Additionally, in https://www.youtube.com/watch?v=QO2NZmFYuOI a video of the visibility experiment can be seen. In the graph on the left all the possible warps are compared using the same registration criterion, SSD forward additional. As can be seen, every warp is able to keep track of the object with slightly different performances ranging from 5 to 10 pixel error when the fog factor is under 1.0. As the visibility decreases and, as a consequence, difficulty increases the execution with less restrictive warps is not able to keep track of the target. As expected, translation warp is the one that obtains the best performance followed by SRT, then affine and homography SL3, and finally homography, which is the most generic warp.

About the registration criterion, big performance differences can be seen. A comparison using SRT warp can be seen in figure 5.4 right. In general terms, ZNCC methods seem to be much more resistant to visibility changes as both are able to succesfully track the target during the whole experiment. Although SSD registration criterion is able to resist small visibility changes, it starts to fail around 0.9 fog factor for forward additional and ESM methods and 1.4 in the compositional cases.

The complete results can be consulted in figure 5.5. This figure shows that the ZNCC inverse compositional is the best tracker being able to track the target with any warp during the whole experiment. In this experiment the translation warp is the most suitable as it is the most restrictive but still able to represent the possible movements between target and camera. However, in other cases where the vehicle needs to move in a different manner, translation warp may not be able to keep track of the target.

Compositional trackers show an interesting behaviour for this use case. They keep the error to a minimum of 5 pixels until the very last moment, when they start to loose the track. This is a desirable feature as the tracker does not obtain false positives in the tracking task.

The currents applied to the problem show a small cyclic error variance in some warps such as the homography ones. This is caused by different possible interpre-

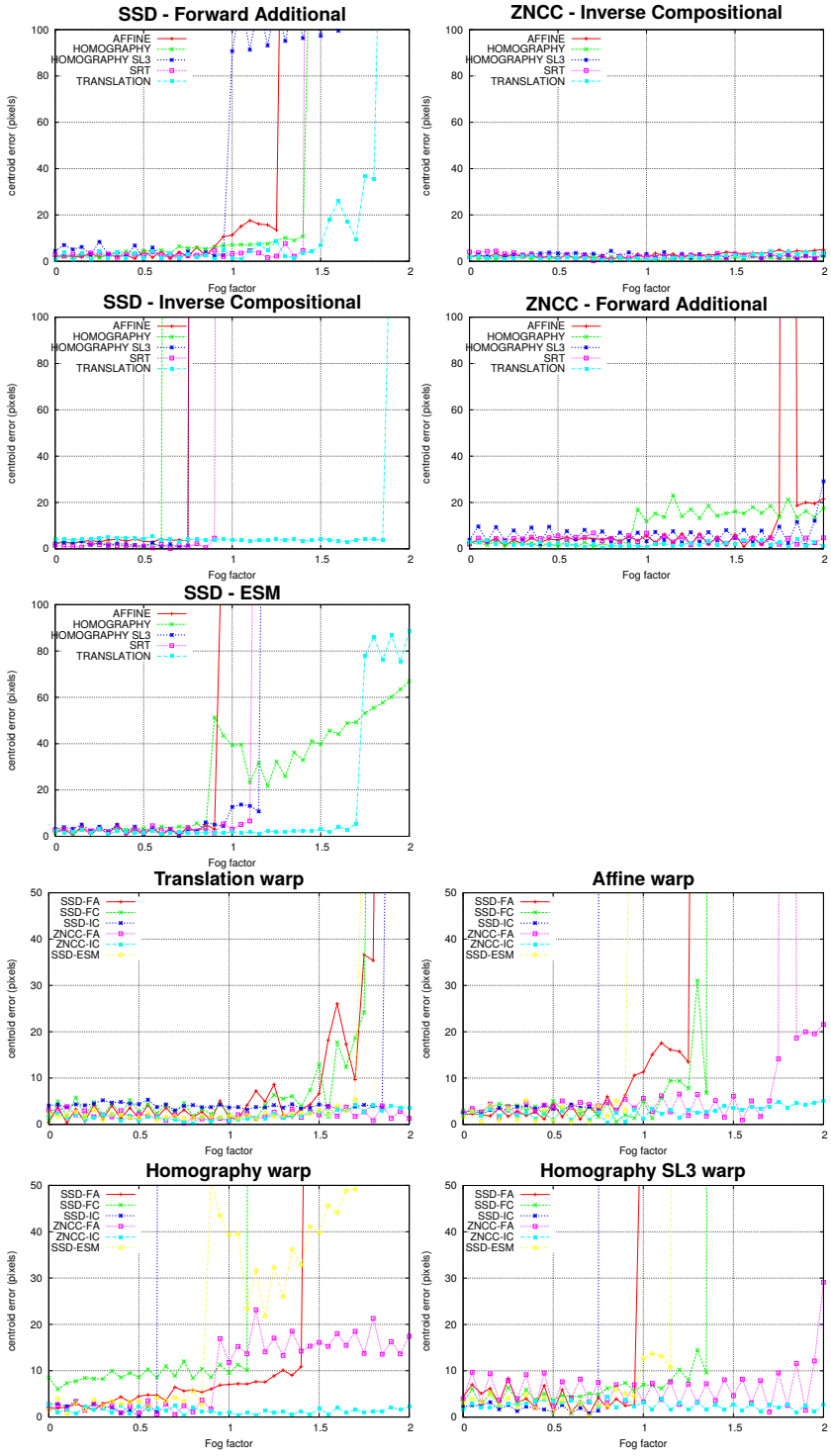

Figura 5.5: Results for the benchmarking visibility use case.

tations of the position, that in the case of more restrictive warps are not possible as not every move is allowed. Thus, small variations in the target position cause errors in the tracking estimation.

This experiment is available and can be executed in the online service described in Chapter 3. All the different combinations of registration criterions and warps are available for configuration and testing online. Finally, the results can be downloaded at any point of the experiment for further analysis.

5.3. Hardware In the Loop visibility benchmarking

The previously presented use case shows the tracker configuration is crucial to optimize the performance. However, all the presented results were achieved in a simulated environment, thus requiring additional experimentation in order to validate the results for real situations. For this reason, the previous case has been extended to benchmark the trackers being used in a more realistic scenario.

Nevertheless, a thorough experiment that allows the underwater visibility to be controlled is difficult to design. Taking this into account, a Hardware In the Loop (HIL) approach has been designed, allowing simulated sources to be mixed with real inputs to explore new situations. This solution offers a good balance between realism and the ease with which the experiment can be carried out.

5.3.1. Experimental setup

The goal of the experiment is the same as the previous use case: keep track of a target in movement seen by the camera while the visibility conditions get progressively worse. The main difference in this case is that real input images will be used to benchmark the performance of visual trackers. As the visibility conditions are difficult to control in a real environment, underwater turbidity will be simulated, thus it is possible to see how the system reacts to new challenging situations.

A schema of the experimental setup can be seen in figure 5.6. The first step is to acquire images of the target that has to be tracked, in this case a black box mock-up has been recorded in a water tank with almost perfect visibility conditions. Image acquisition was made in a water tank with a *uEye UI-1240ML-C-HQ* camera placed inside a cylinder moving on the water surface facing a black box mock-up as can be seen in figure 5.7,

This recorded experiment will be used for two purposes. It will be the input to UWSim in order to add simulated turbidity to it, and then be the input of the tracking software. On the other hand, the raw image from the recorded experiment is necessary for a ground truth of the tracking trajectory to be obtained.

In order to calculate the ground truth, a regular ESM tracker like [Malis, 2004] has been used under human supervision. As previous experiments proved, any of the evaluated trackers is able to achieve good performance with negligible error. Thus, the output of ESM tracker in the raw captured images in almost perfect visibility conditions is enough to obtain a ground truth that can be used for comparison.

Simultaneously, the raw image is also an input to UWSim which adds simulated fog to the image. Creating a new fogged image suitable for the proposed tracking

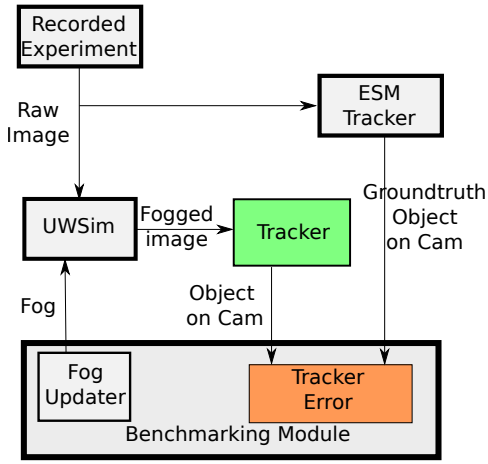

Figura 5.6: Software schema used in the HIL visibility benchmarking. Orange: benchmark outputs, Green: evaluated software.

Figura 5.7: Image acquisition setup. Left: camera in a cylinder moving in the water surface. Right: black box mock-up captured from the camera.

Figura 5.8: From left to right top to bottom, tracking algorithm scenario in decreasing HIL visibility scenario.

evaluation. The model used to reduce the visibility to the real image uses the previously described fog factor value in the equation 5.1.

As happened in the previous use case, the distance to the target is almost invariable due to the camera planar movements and rotations. This means that the only variable in the equation is the fog factor. Thus the simulator uses a fixed distance of 80 centimetres , which is the real measured distance, to the black box mock-up to calculate the amount of fog that needs to be added at every moment of the experiment depending on the fog factor. The experiment is configured to slowly increase the fog factor, reducing the visibility. The result of this fog addition can be seen in figure 5.8.

Finally, the benchmarking module is the one in charge of evaluating the tracking, as in the previous experiment. But, in this case the tracking ground truth comes from the ESM tracker instead of UWSim and sends the amount of fog to be tested to UWSim that will process it in a real image. Thanks to ROS middleware, this is transparent for the benchmarking module that sends and receives the same information through ROS topics.

5.3.2. Results

Using this setup, the experiment has been repeated for the ViSP tracker previously described combining registration criterion and warps. The results are similar to the simulation case, the algorithms that performed better in simulation still perform better in the HIL setup. However, there are substantial differences in the exact moment at which a tracker starts to loose the target due to the differences between real and simulated input.

Results can be seen in figure 5.9. In the graph on the left a warp comparison using SSD forward compositional as registration criterion is shown. As the results show, translation warp is still the best option due to the planar movements of the experiment. However it looses the target faster than in the simulation environment probably due to the rotation movements not present in the previous experiment.

The results also show a higher error value due to the difficulties that are created by real input, but are almost impossible to simulate such as unpredictable noise, light changes or transmission retards. These factors are also the cause of sudden changes in the performance. But, in general terms, the most restrictive warp is still

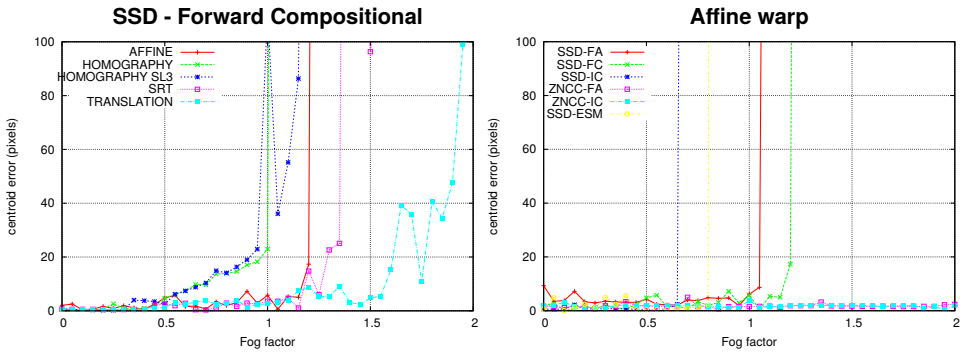

Figura 5.9: Results for the HIL benchmarking visibility use case. Left: warp performance comparison, right: registration criterion comparison.

the best option due to the limited movements of the camera in the experiment.

In the case of similarity functions, represented in the graph on the right of the figure, ZNCC registration criterion still offers the best performance. There are no noticeable differences between ZNCC methods, but SSD performance is highly influenced by the method used, FC and FA being much better than ESM and IC.

The other results, that can be seen in figure 5.10, follow the same trends as the previous experiment. The main difference is the noisier result caused by the real image input. But, ZNCC registration criterion and the simplest warp that is able to encode the transform between camera and target, is still the best option .

As a conclusion, the experiment confirms the simulation results proving the presented framework to be a valuable tool. It allows the best option to be chosen from a wide range of algorithms taking into account not only the goal of the intervention, but a range of environmental conditions that may affect the result.

In this case, underwater visibility is a key parameter to consider when choosing a template tracker. As can be seen, when the water turbidity is low, there is no noticeable difference between all the compared alternatives. However, as the turbidity increases performance drops for some of the reviewed algorithms letting the researcher choose the best option depending on the situation.

5.4. 3D reconstruction benchmarking

The two previously presented cases used simulation to reproduce the underwater turbidity and measure its effects on visual tracking algorithms. However, not only do visual trackers suffer the degraded images consequences, but any algorithm that uses a camera will be affected by low visibility conditions. Furthermore, simulation and hardware in the loop experiments help to introduce and study the problem, but real experiments must be conducted as the final step.

In this experiment a different use case is presented, comparing 3D reconstruction algorithms based on visual cameras. The objective is to be able to retrieve a 3D pointcloud, a set of 3D points, of an unknown underwater object in front of the vehicle. The experiment takes place firstly in simulation and then in a real

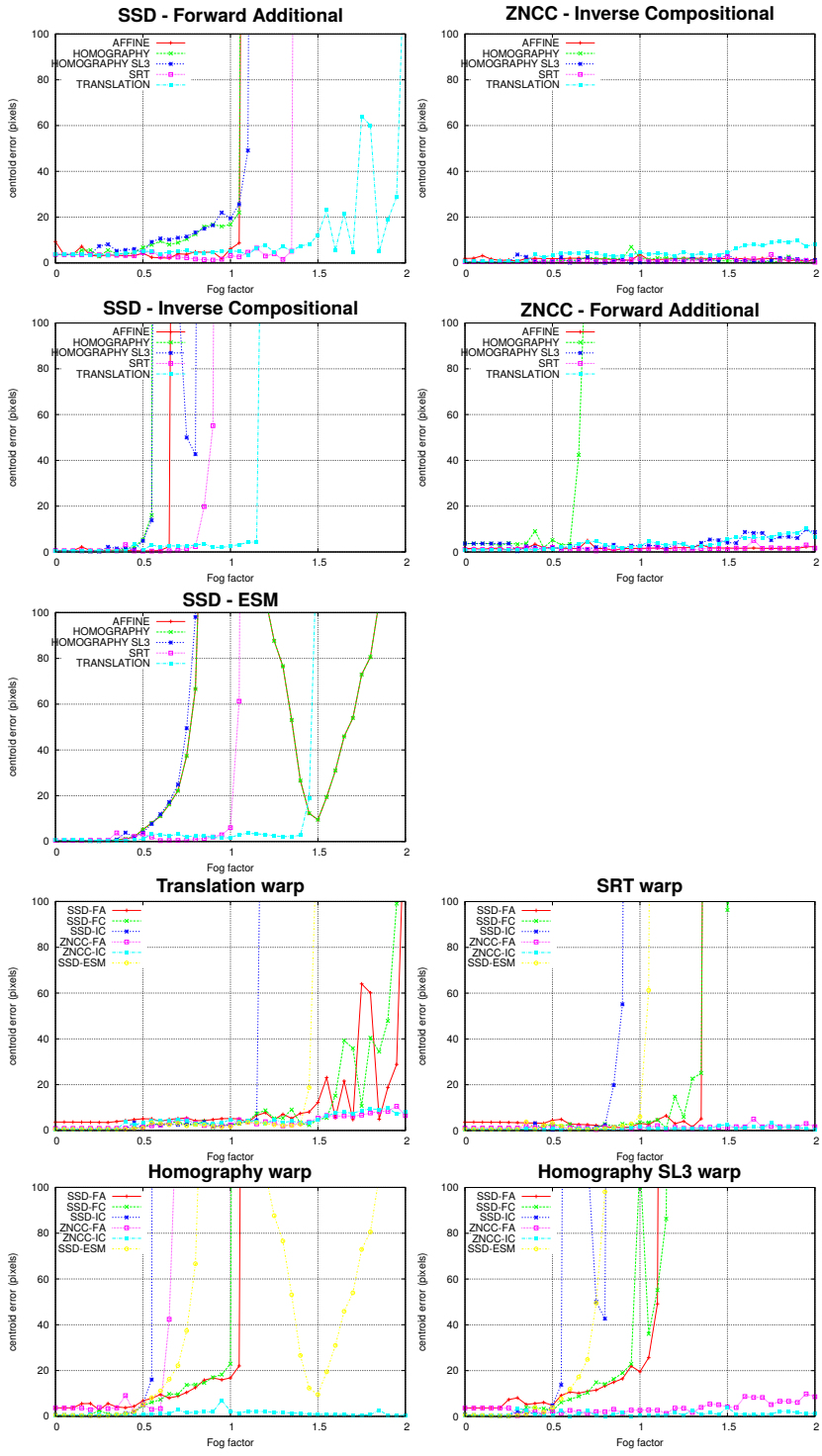

Figura 5.10: Results for the HIL benchmarking visibility use case.

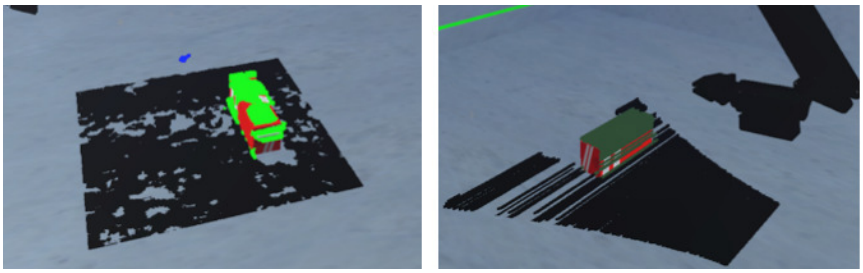

Figura 5.11: Stereo (left) and laser stripe (right) reconstructions comparison overlayed on ground truth simulated environment. Black points are filtered as ground, blue are considered outliers and green points represent reconstructed object.

laboratory environment using the framework presented in Chapter 2 as evaluation software. In order to do this, a calibration step must be introduced to the workflow so that the framework is able to model the real world.

5.4.1. 3D reconstruction algorithms

Two reconstruction algorithms have been compared in the benchmarking platform. A stereo reconstruction using two cameras, and laser stripe segmentation. These approaches are used to obtain a point cloud from the scene that will be evaluated in terms of quality. Both algorithms are using exactly the same code whether used in a simulation or in a real setup.

Figure 5.11 shows a simulation execution of both compared methods. The pointcloud output is drawn in the simulated environment, providing visual feedback of the algorithm using a color schema for floor(black), object(green) and outliers(blue).

The aim of the **stereo reconstruction** is to obtain 3D reconstruction in the form of a dense point cloud, where each pixel of the image is used in order to obtain a 3D point instead of computing them for certain features only: sparse reconstruction. A good reconstruction can be obtained only if the camera parameters are properly estimated. The parameters are computed with camera calibration tools and a calibration chequerboard.

In runtime, images from left and right side are undistorted and rectified using the aforementioned camera parameters, so that their scanlines align for fast stereo processing. Once the images are aligned, a local dense stereo correspondence algorithm can be applied. In this case, OpenCV block matching algorithm [Konolige, 1998] implemented in a ROS package is fast enough for most robotic applications, while only needing previous parameter tuning. With the chosen algorithm, both disparity images and dense point clouds can be obtained. This method estimates the corresponding pixel on the image on the right for every pixel on the image on the left, thus comparing each pixel to a block on the other image. The displacement between the two pixels is used to determine the 3D point coordinates based on the camera geometry computed in the calibration step.

In the case of **laser stripe reconstruction**, a laser light is used to project a line in the scene that will be detected in a texture camera. Knowing the positions of the camera and laser light, it is possible to retrieve a 3D point cloud of the projected line. As the laser light is placed in a robotic arm, it is possible to move it scanning the scene and in this way to obtain a scene reconstruction.

Before the system is able to perform a laser stripe reconstruction, it is necessary to calibrate it. So, with the aid of a marker placed in the gripper of the arm, the transformation between the camera and the end-effector ${}^{c}\mathbf{M}_{e}$, is calculated as explained in [Peñalver et al., 2014]. Then, using the Direct Kinematics of the arm, the relationship between the base of the arm and the end-effector is obtained ${}^{b}\mathbf{M}_{e}$. So, using these two matrices the transformation between the base of the arm and the camera is easy to calculate as equation 5.2 shows.

$$ {}^{c}\mathbf{M}_{b} = {}^{c}\mathbf{M}_{e} * ({}^{b}\mathbf{M}_{e})^{-1} \tag{5.2} $$

The next parameter that has to be obtained is the relationship between the laser and the end-effector ${}^{b}\mathbf{M}_{e}$. The camera installed in the vehicle is a stereo camera as a consequence, even though just one lens is used for the laser reconstruction, in the calibration step the two lenses are used to increase precision. Using the stereo camera, the 3D position of the pixels projected by the laser is obtained by triangulation. Using those 3D points, the RANSAC algorithm is used to determine the planar parameters of the laser plane [Inglis et al., 2012] ${}^{c}\mathbf{M}_{l}$. These parameters are referenced to the stereo camera using the previously obtained transformation between the camera and the end-effector ${}^{c}\mathbf{M}_{e}$. But, it is possible to reference the plane with respect to the laser using equation 5.3.

$$ {}^{l}\mathbf{M}_{e} = ({}^{c}\mathbf{M}_{l})^{-1} * {}^{c}\mathbf{M}_{e} \tag{5.3} $$

Concerning the reconstruction, the floor is scanned moving the elbow joint of the manipulator at a constant velocity between two predefined joint positions. At the same time, the camera captures images of the scene onto which the laser is being projected. For each image, a laser peak detector algorithm is used to segment the laser stripe from the rest of the image. This algorithm discards the pixels that are out of a predefined threshold of hue, saturation and value (HSV). Due to the laser stripe pattern and the fact that the camera is placed parallel to that line, there is only one point illuminated by the centroid of the laser at each column of the image. As a consequence, for each column of the image, the pixel with the highest intensity is selected. Using the center of masses algorithm, that is applied to this pixel and the five pixels above and below it, the position illuminated by the centroid of the laser is obtained with subpixel accuracy.

Finally, the segmented laser stripe is triangulated to obtain its 3D position [Prats et al., 2012a]. In order to triangulate each selected pixel, it is necessary to know the relationship between the camera and the laser ${}^{l}\mathbf{M}_{c}$ when the image is captured. So, at the moment of image acquisition, the values of the joints are stored. Using these values and the direct kinematics of the arm, the transformation between the end-effector and the base of the arm ${}^{b}\mathbf{M}_{e}$ is calculated. Finally, using this relation and the ones obtained in the calibration, the desired transformation can be easily calculated through equation 5.4.

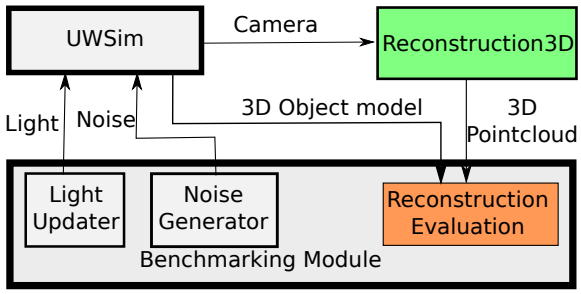

Figura 5.12: Software schema used in the 3D reconstruction benchmarking. Orange: benchmark outputs, Green: evaluated software.

$$ {}^{c}\mathbf{M}_{l} = {}^{c}\mathbf{M}_{b} * {}^{b}\mathbf{M}_{e} * ({}^{l}\mathbf{M}_{e})^{-1} \tag{5.4} $$

Although the compared algorithms are completely different and even use different hardware, it is possible to compare the results as both produce a 3D pointcloud. This 3D pointcloud is a representation of the scene with points referenced to the vehicle camera, allowing the robot to process it and decide the best approach to interact with the world.

5.4.2. Experimental setup

The basic setup for the experiments can be seen in figure 5.12. UWSim provides the camera input that will be used to reconstruct the scene and 3D object model that will act as ground truth. The benchmarking module evaluates the algorithm from the 3D object model from the simulator and the 3D pointcloud retrieved by it. As perturbations, the amount of light in the scene and the noise in the camera will be updated according to the benchmarking needs to test different environments.

In order to evaluate the 3D pointcloud produced by the 3D reconstruction system, a 3D object model is required. In the case of simulation, the model that is to be simulated can be used, but in the real benchmarking a high fidelity model of the object to be reconstructed is needed. The benchmarking module takes this object model as ground truth and a configuration file to get the position of the object with which the results can then be calculated.

Results are divided in four metrics that provide different information about the 3D reconstruction quality. These four metrics have been introduced in [Oude Elberink and Vosselman, 2011], a work about reconstruction metrics, as quality measures of 3D models, and are defined as follows:

- **Outliers:** The percentage of the reconstruction that it is further than a threshold from the target. These points are considered reconstruction errors. Every point that it is not an outlier is automatically defined as inlier.
- **Mean error:** Average distance from every 3D reconstruction inlier point to the nearest point on the object surface. This metric offers a precision measure of the reconstruction.

- **Standard deviation:** Standard deviation for the previous error. A high value in this deviation usually means misalignment in the reconstruction, due to bad calibration.

- **Coverage:** Surface percentage that is nearer than a precision threshold to a 3D reconstruction point. It measures the percentage of the target that is correctly reconstructed. The threshold should be chosen depending on the experimental setup. It is not an inlier measure, instead of measuring the percentage of points near the target, it measures the percentage of the target that has a reconstructed point nearer than a threshold. For instance, a perfect reconstruction of 3 faces of a box would return 50 % coverage instead of 100 % that an inlier metric would get.

However, these measures consider only reconstructed points of the object model while reconstruction techniques, even if the object is the only one present, in the scene do not distinguish between floor and object. For this reason, a previous filtering step is needed so ground points are automatically discarded and they do not interfere in the metrics.

In order to filter the ground points, an infinite plane at the base of the object model is considered. Every point closer to this plane than to the object surface, is labelled as floor. After this, each floor labelled point is then compared with the outlier threshold against the floor instead of the object. Points further than the outlier threshold are relabelled as outliers and the rest of the floor points are discarded for metric computation.

Besides mathematical results, in this case, the benchmarking module is able to overlay the labelled 3D reconstructed point cloud on the simulated 3D scene to get a visual result of the reconstruction using UWSim as visualization engine. Furthermore, 3D points are colored to show outliers, filtered ground points and object points. This is of great value not only for showing results, but for debugging reconstruction algorithms.

5.4.3. Simulation experiment

In the simulation experimentation, benchmarking capabilities have been exploited to test both reconstruction techniques under different conditions of illumination and camera noise. These conditions try to simulate the complex and adverse conditions in the underwater environment. Theoretically, the laser stripe reconstruction should not be affected by low illumination conditions as it produces its own light, but it may be more difficult to detect where the scene is brighter. The presence of noise should cause poorer performance on both methods.

The algorithms have been tested in conditions where the amount of light varies, ranging from 0 to 1.0 ratios where 0 means complete darkness while 1.0 is the correct illumination (default values in UWSim), as can be seen on Figure 5.13. This ratio affects all sources of light (ambient, directed and diffuse), but not that produced by the vehicle itself such as a laser projector.

Besides this, Gaussian noise has been added to the camera output from 0.00 % standard deviation to 0.10 %. This noise is added in an additive manner on RGB channels in the shaders, so it does not produce a performance drop. Basically, it

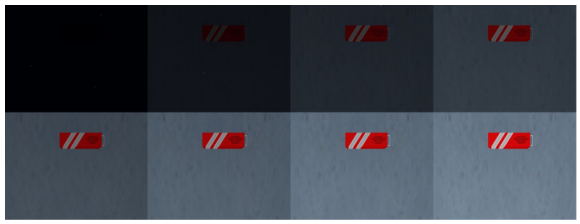

Figura 5.13: From left to right, top to bottom increasing light conditions on virtual camera.

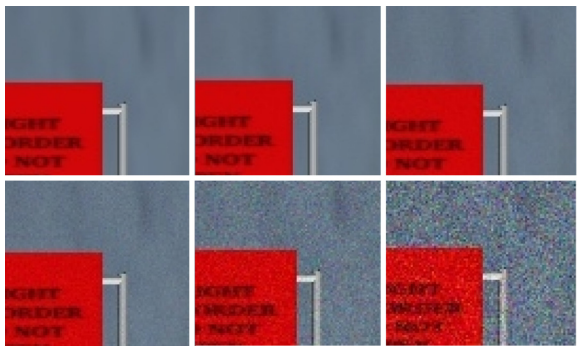

Figura 5.14: From left to right, top to bottom increasing noise conditions on virtual camera.

adds a Gaussian random value to each pixel, adding white noise to the virtual camera. Noise effect can be appreciated in figure 5.14.

Using these two scene updaters to test a wide variety of situations the 3D reconstruction algorithm has been tested in simulation. The object to be reconstructed is a black box mock-up as in the previous experiments, based on a search and recovery problem. At this moment, the object is correctly found and tracked but the main concern is to be able to reconstruct the target in order to compute good grasping points and finally recover it.

Coverage results for laser stripe reconstruction and stereo vision can be seen in figure 5.15. As expected, stereo vision reconstruction needs some light to achieve a good reconstruction while laser is nearly immune to light variation. In the case of stereo vision, the light ratio needed for good coverage depends on the added noise ranging from 0.05 to 0.6. Laser stripe reconstruction is able to achieve a good reconstruction in any case and slightly decreases its performance in conditions where the light is bright.

Regarding noise, stereo vision is again more sensitive to noise, especially in lower visibility conditions. On the other hand laser shows no noticeable differences between different noises on coverage. This was the expected result as laser produces its own light source, thus it is not affected by illumination changes, while stereo needs to match pixels from both images that get degraded with low visibility and noise.

In absolute terms, the laser is able to reconstruct 50 % of the object in almost

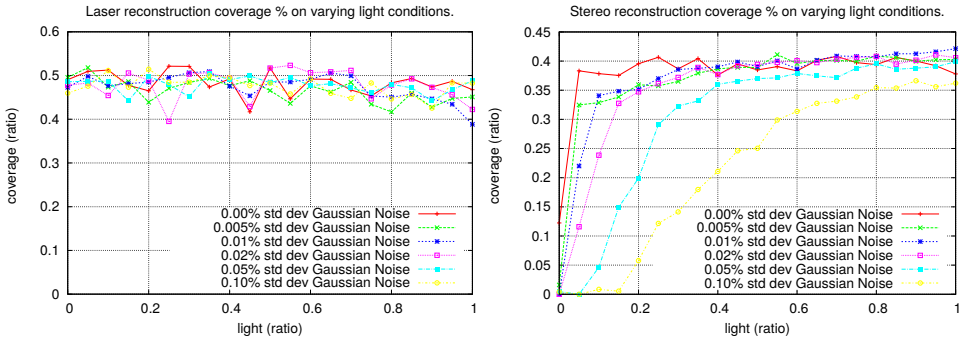

Figura 5.15: Coverage results on varying light conditions for different Gaussian noise on camera for Laser (left) and Stereo camera (right).

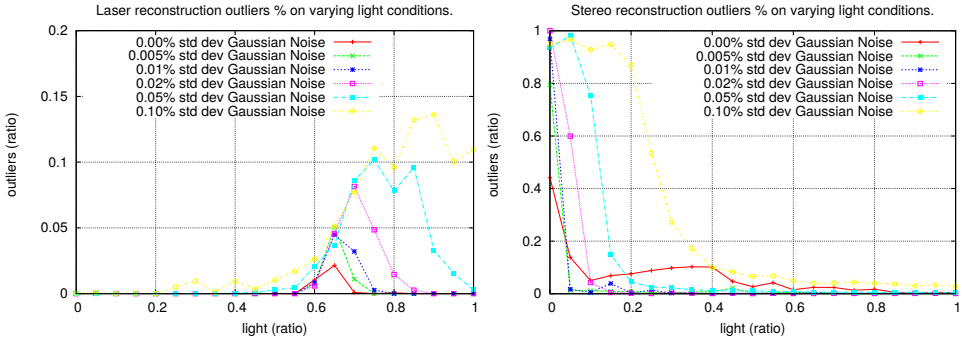

Figura 5.16: Outliers results on varying light conditions for different Gaussian noise on camera for Laser (left) and Stereo camera (right).

every situation and stereo vision reconstructs 40 % of the object in good light conditions. This is a good result taking into account only one view of the object is available, this means 3 out of 6 faces are not visible to the camera at any moment. Different approaches that move the vehicle around the target to reconstruct the object can be used to increase the performance, but this is out of the scope of this work. In conclusion, the laser is clearly better for the tested environment due to it's immunity to changes in illumination

About the mean error and standard deviation, both algorithms show similar results. In the case of mean error, due to the similar setup, both algorithms achieve 0.004 meters, which is a good value given the experimental setup. In order to increase the precision of both algorithms a higher resolution camera will be necessary, as the results are at the limit of pixel resolution, or the possibility to move the vehicle closer to the object. As the alignment of reconstruction and ground truth is perfect on simulation, because no noise nor perturbation were added in the calibration, the standard deviation is negligible.

Finally, the outliers results are depicted in figure 5.16 for both, the laser and stereo vision cases. As the image shows, stereo vision generates more outliers in

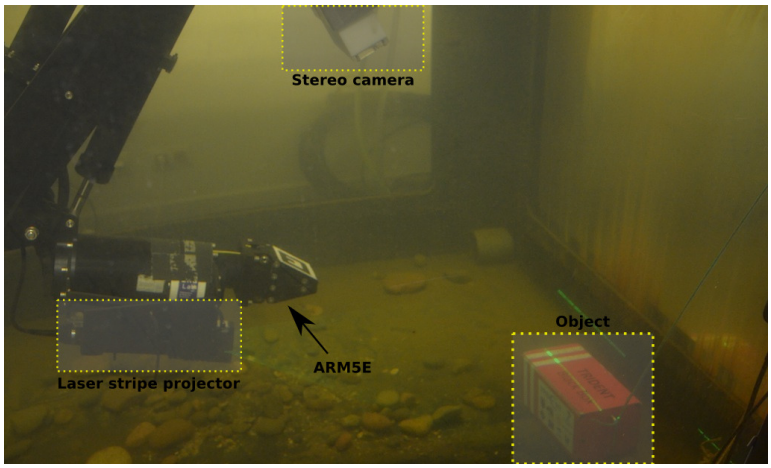

Figura 5.17: Physical benchmarking system: water tank, Lightweight ARM5E manipulator, stereo camera, laser stripe projector and black box mockup.

the absence of light, while the laser reconstruction produces a higher number of outliers in brighter environments. In both cases higher Gaussian noise exacerbates this behaviour increasing the percent of outliers.

In the case of stereo vision, noise and the absence of light make it more difficult to match the pixels on both cameras, and a large number of outliers appear reaching 100 % of outliers in extreme cases. On the other hand, in strong light conditions it is more difficult to find the laser light on the camera, and mistaken detections cause a high number of outliers.

It is also remarkable that there is a non-negligible percent of outliers in the stereo vision results for 0.00 % standard deviation noise. These outliers are caused by small floating particles simulated in UWSim, which are correctly detected by the system although they are not part of the object. The 3D stereo reconstruction is not able to find these outliers when the noise is higher and its impact decreases as more parts of the object are correctly reconstructed.

To conclude, laser stripe reconstruction seems to be a better option for the tested environment. The immunity to low illumination environment reduces the effect of noise and decreases the amount of outliers while maintaining precision. However, in the tested environment the color of the objects is different to the color of the laser laser and close to the camera, making the laser detection easier. When objects are far from the camera or similar to the laser color detecting the projected laser stripe will not be an easy task, thus reducing the performance.

In addition, real experiments need a camera laser projector calibration step that has not been modelled in the simulation. For these reasons real experiments are necessary to confirm the simulation results.

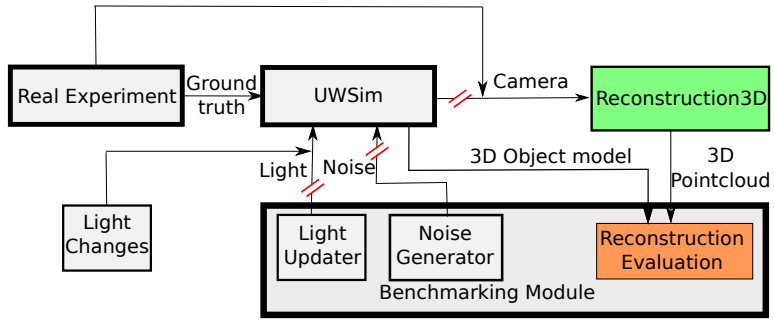

Figura 5.18: Software schema used in the real 3D reconstruction benchmarking. Orange: benchmark outputs, Green: evaluated software.

5.4.4. Physical benchmarking experiment

In order to be able to evaluate the compared algorithms and validate the simulated results in a real platform, a physical benchmarking platform has been used as can be seen in figure 5.17. It is formed by the following elements, matching the main elements in the virtual scene used in UWSim in order to reproduce the simulated experiment in laboratory conditions:

- **Water tank:** Dimensions 2.0 m (width) x 2.0 m (length) x 1.5 m (height).
- **Robotic arm:** Four degrees of freedom *ECA-CSIP Light-weight ARM 5E manipulator*, described in [Fernández et al., 2013].
- **Floating structure:** Simulates an underwater vehicle prototype to hold the arm. In this work the floating structure has been fixed to the water tank thus it does not introduce noise to the system.
- **Color camera:** Required to detect the laser, *Bowtech DIVECAM-550C-AL COLOUR camera is used.*
- **Laser stripe projector:** The *Tritech SeaStripe Laser Line Projector (MKIII)* is used to project a laser onto the object.
- **Stereo camera:** *Videre stereo camera* is used for stereo reconstruction.
- **Object:** Black-box mockup size 140 mm (width) x 300 mm (length) x 160 mm (height).

The previously used software architecture that can be seen in figure 5.12 needs some changes if it is to work in a real environment. The schema used is shown in figure 5.18. In this case the real experiment provides the ground truth models and positions to UWSim that will process it to generate the ground truth model required for evaluation. The evaluated algorithms no longer use the input from UWSim, now they use the real camera output. Finally the light changes are carried out in the laboratory instead of being generated by the benchmarking module.

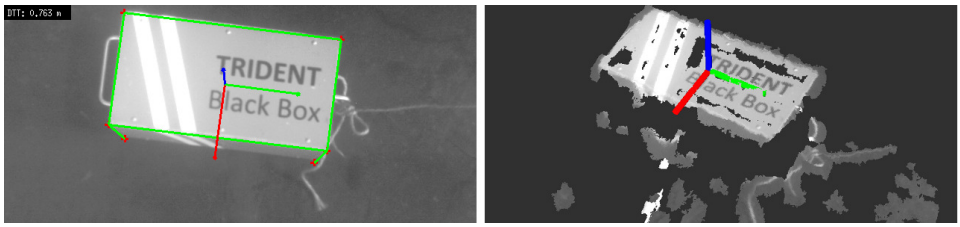

Figura 5.19: Camera image with manually initialized corners and estimated box pose (left). Ground truth box pose with the point cloud obtained with the stereo camera (right).

Besides a high fidelity virtual 3D model of the object to be reconstructed, relative positions between object and robot are required. In order to obtain that ground truth that UWSim needs to evaluate the reconstruction, a calibration step has been added. This step measures the position of the target to reconstruct with respect to the camera and using the robot kinematic model as the vehicle base reference.

In order to obtain the object pose ground truth, a pose estimation method has been used to compute the relative position of the target object, in this case a black box mock-up with respect to the camera. As dimensions of the object are well known, the box corners can be used to estimate its pose. In the case of using a different object, easily recognizable points could be used instead of corners. While it is possible to detect them automatically, it has been decided that the manual initialization by the user is less error prone, more general and best suited to get an accurate ground truth for the benchmarking system.

First of all, using an image of the target from the robot camera for calibration, the user clicks on the visible recognizable points, in this approach six corners were visible. These recognizable points must be matched in the 3D model and then, using the camera parameters, the object pose estimation is obtained using the ViSP library. In this case, the frame is placed in the center of the top face of the box, as can be seen in figure 5.19. As there are several methods in the ViSP library that can be used to obtain the estimation, all of them are used to estimate the pose and the one that minimizes the estimation error is selected.

This ground truth, however, is not perfect as small errors appear caused by the limited camera resolution (actual pixel size), user accuracy and camera calibration. Nevertheless, the resulting error is small enough to allow the object position to be considered as a suitable ground truth so that the metrics described in this experiment can be used. In fact, the camera calibration accuracy affects rectification and undistortion in these cameras, thus obviously introducing some shared error in this ground truth position and at the same time in the reconstruction processes.

In this experiment, conducted under real conditions, both systems have been tested under three different light conditions shown in figure 5.20, in order to replicate the simulation results. As the illumination of the environment is a key characteristic in this experiment, a lux meter was used to assure the replicability of the experiment. The lux meter was placed on a flat surface as near as possible to the black box. The values obtained for the testing scenarios were 12, 147 and

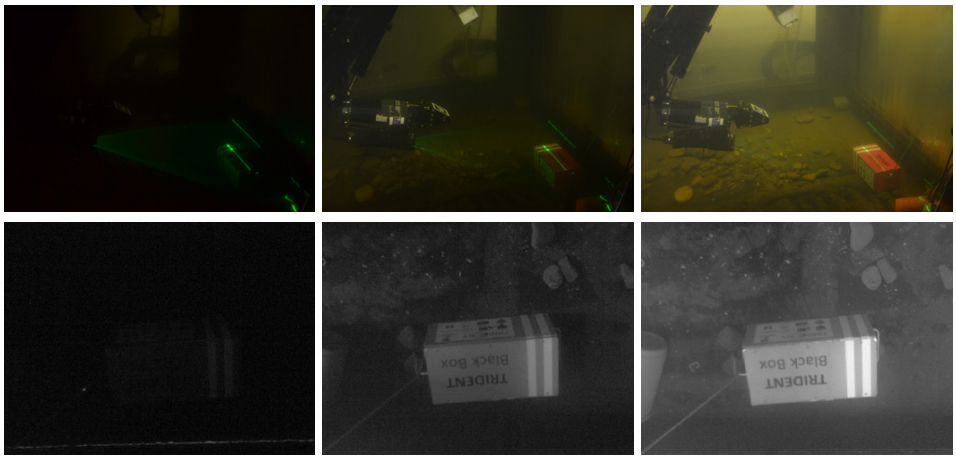

Figura 5.20: From left to right: low, medium and high light conditions. From top to bottom: external view and camera view.

207 lumens.

As can be seen in the first scenario, images on the left of the figure, there is almost no illumination. In the case of the laser stripe reconstruction, top images, the projector produces its own light thus it is the only visible clue, making it easier to detect. In the intermediate scenario both methods should perform well as the laser can be easily detected and there is enough light for the stereo reconstruction. However in the last scenario the stereo reconstruction has some advantage because the laser stripe is difficult to detect when projected onto a brightly lit surface.

In this case, the camera noise is not introduced as the real camera already has white noise, making it difficult to control the amount and characteristics of the addition of both signal noises (introduced and already present). Thus using only the three illumination conditions, both reconstruction algorithms have been compared using the presented benchmarking architecture.

Results for coverage are depicted in figure 5.21 on the left. Laser stripe reconstruction results are slightly worse than simulated ones. In this case, laser achieves around 32 %-38 % while in simulation it reached 50 %. Although laser works better in dark situations, it is highly resistant to light changes. On the other hand, stereo reconstruction is completely dependent on light conditions, achieving a 38 % of coverage in good light environments. These results support the ones obtained in simulation where both algorithms behave in a similar way.

Nevertheless, in both cases the results in the real environment are worse than simulated ones. This fact is caused by the unpredictable nature of the real world that introduces perturbations and noise that can not be easily simulated.

Regarding mean errors, laser and stereo have similar mean errors, around 0.008 meters. This result is noticeably higher than in simulation due to the added ground truth estimation error. The standard deviation, though, is greater than the one obtained in simulation, around 0.005 meters in stereo and 0.008 meters in the case of laser reconstructions. However, it is small enough to conclude that the tested

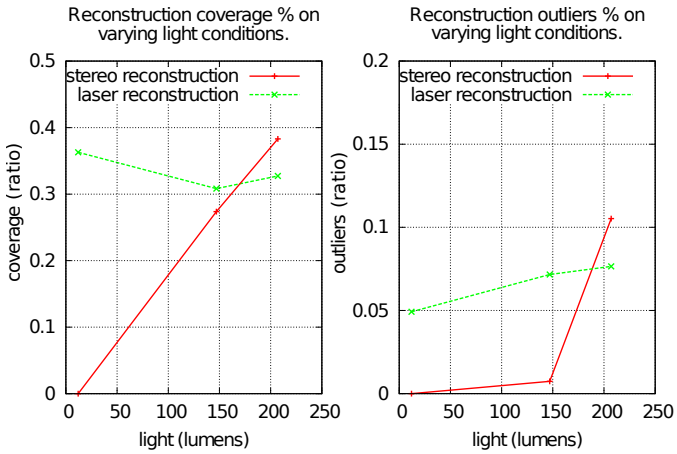

Figura 5.21: Reconstruction coverage (left) and outliers results (right) for Stereo camera and Laser on real scenario for different light conditions.

algorithms reached a good alignment and the ground truth estimation was fairly good. The higher deviation in laser stripe reconstruction is caused by the added difficulty to calibrate the relative positions of the camera and laser projector.

The outliers results depicted in the right figure 5.21 show that both algorithms increase the number of outliers as light increases. In this case, stereo reconstruction shows a 0 % on outliers in the absence of light because is not able to obtain any points. Once the stereo is able to retrieve some 3D points the percent of outliers suddenly increases to around a 10 % that is in line with the previously simulated experiment.

Besides the numerical results the presented approach is also able to provide the researcher with visual feedback of the reconstruction as can be seen in figure 5.22. In this visual feedback real point clouds are displayed onto UWSim simulation while being processed. In the images, black dots are filtered as ground points, green points are considered object points and blue points are labeled as outliers. As happened in simulation, laser reconstruction works better in low light environments while stereo needs some light to work properly.

A further analysis of the visual results shows that although the appearance of the laser reconstructions is better, there is a misalignment on the pointcloud, as the standard deviation of the results suggested. 3D laser reconstructions are slightly rotated with respect to the ground truth target due to small errors in camera to laser projector calibration. In the case of the stereo reconstruction, the ground reconstruction was very poor due to the absence of texture on it which meant that it was unable to match images from both cameras.

5.5. Conclusions

In this chapter three experiments have been presented to show the impact of underwater image degradation on robotics. The approach uses the previously des-

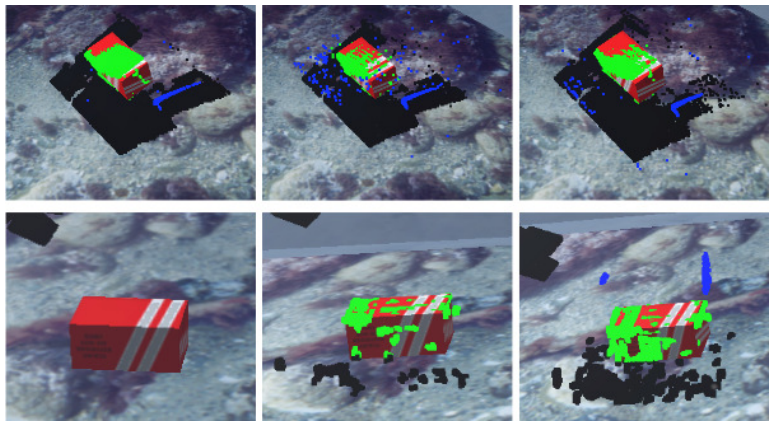

Figura 5.22: From left to right: low, medium and high illumination conditions. From top to bottom: real laser point cloud reconstruction and real stereo point cloud reconstruction overlayed on UWSim.

cribed framework for simulation and benchmarking to measure the results of different vision algorithms under decreasing visibility conditions. The experiments have been conducted in simulated, hardware in the loop and real conditions in order to validate the results.

The experimental scenario is focused on the TRIDENT project where an underwater vehicle searched for a black box mock-up to grasp it and recover the information in it. The main stages that may be affected by underwater turbidity are the detection and tracking of the target and the reconstruction to compute suitable grasp points.

The first experiment showed the results for the first detection and tracking stage in a decreasing visibility scenario. Different tracker configurations are tested following the target in a simulated camera to decide the best possible option for the final intervention. As can be seen, all the tested configurations are affected by water turbidity but ZNCC based functions showed a more resistant behaviour to visibility changes.

In the second experiment, the previous test was repeated in a hardware in the loop setup, using real camera input and degrading the images according to the needs of the experiment. This configuration allows the realism of the experiment to be increased, shortening the gap with real experiments, but still permits a large number of benchmarking tests to be carried out. The results confirmed the trends of the first experiment where ZNCC obtained better results keeping the track in decreasing visibility.

Finally, the last experiment introduced the 3D reconstruction scenario necessary to grasp unknown objects. In this case two alternatives were compared, a laser stripe reconstruction and a stereo camera. Both systems were tested in simulated an real environment in a variety of illumination and camera noise conditions. As expected, the laser alternative showed better performance in the absence of illumination due to the fact that it emits its own light while in brighter scenes it is more

difficult to detect the projected laser. Although this could be solved mounting strobes in the vehicle, this kind of illumination has some disadvantages: it exacerbates backscattering, high battery consumption and requires control to provide proper illumination at different distances.

In summary, every algorithm that makes use of an underwater camera will be greatly affected by the degradation that the images suffered caused by the physical properties of light transmission. Three different experiments have been presented to prove these effects in the particular case of a search and recovery problem. These experiments showed a thorough comparison of different alternatives helps to decide the best option for the final intervention.

Furthermore, a benchmarking framework has been used exploiting the advantages of these kinds of tools which can be used to conduct a set of experiments without unnecessary complications. These tests can be performed in simulation, real environments or even mixing both of them in a hardware in the loop setup.

Capítulo 6

Underwater image dehazing

As discussed in Chapter 5, underwater turbidity makes working with cameras a really challenging task. Classical vision algorithms to detect, track, reconstruct in 3D or even visually guided controllers are seriously affected by image degradation. Furthermore, this image degradation is difficult to predict since it depends on the underwater visibility conditions which are unpredictable, so it is not possible avoid it. As a consequence, autonomous underwater vehicles need to be prepared each time a robotic intervention mission is performed because water turbidity may appear decreasing the visibility for its cameras at any moment.

The solution to this problem is to use a dehazing approach able to partially restore the colors and characteristics of the acquired image. However, these techniques usually require a long processing time, invalidating them for real time application to increase robot performance in an underwater intervention.

This chapter discusses dehazing methodologies designed to decrease the effects of turbidity on vision algorithms, that can be used by underwater vehicles. Different approaches are considered for use online in a real mission. Suitable techniques for intervention are then objectively evaluated, using datasets acquired in real interventions, in order to see their performance in real underwater environments.

6.1. Introduction

In order to deal with underwater image processing, it is necessary to consider first the physics of light propagation in water medium. Physical properties of the medium cause degradation effects not present when light is travelling through air. Underwater images are essentially characterized by their poor visibility because light is exponentially attenuated and scattered as it travels through water, resulting in poor contrasted and hazy images. Attenuation and scattering of light limits the visibility distance to about twenty meters in clear water and five meters or less in turbid water making it difficult to detect objects that is necessary to manipulate.

The light attenuation process is mainly caused by two phenomena: absorption, which removes light energy, and scattering, which changes the direction of the path of lightwaves. The absorption and scattering processes of light in water influence the overall performance of underwater imaging systems. Absorption reduces the

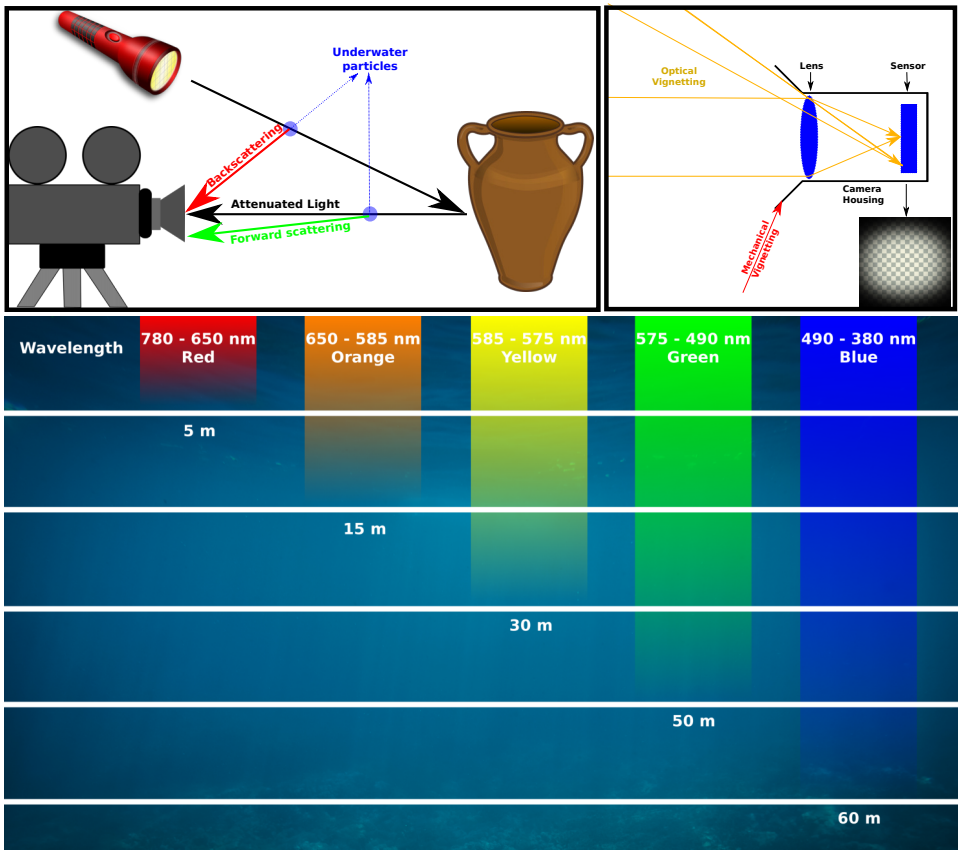

Figura 6.1: Underwater light propagation physic effects that degrade images. From top to bottom, left to right wavelength attenuation, scattering and vignetting.

amount of light as the robot goes deeper or further from the camera, colors drop off one by one depending on their wavelengths. As can be seen in bottom figure 6.1, the blue color travels the longest in the water due to the fact it has the shortest wavelength, with the result that the underwater images are essentially dominated by a blue color.

This is modelled in equation 6.1 as described in [Jaffe, 1990]. Where L_c is the light getting to the camera, $R(\lambda)$ is the reflectance of the surface depending on the light power, $b(\lambda)$ is the attenuation coefficient of the water for each wavelength λ and d is the distance from the camera to the reflected surface. Thus, the light is exponentially attenuated depending on the distance and the wavelength attenuation coefficient.

$$L_c = R\,(\lambda)\, e^{-b(\lambda)d} \tag{6.1}$$

The scattering effect, represented in top left figure 6.1, changes the direction of the light to the camera but, depending on the source of the deviated light two main types are distinguished. Forward scattering, randomly deviated light on its

way from an object to the camera, that generally leads to a blurring of the image features. On the other hand, backward scattering, the fraction of the light reflected by the water towards the camera before it actually reaches the objects in the scene, generally limits the contrast of the images, generating a characteristic veil that superimposes itself on the image and hides the scene, blurring the targets that have to be manipulated.

An approximation of the scattered light arriving at the camera $B(\lambda)$, can be computed assuming a uniform directional distribution of the scattered light $\gamma(\lambda)$ using equation 6.2 as described in [Jaffe, 1990]. Where $L(d)$ describes the light arriving at each point of the camera line of sight, where it can be refracted to the camera, in function of the distance along the line of sight d.

$$B\left(\lambda\right) = \int_0^d \gamma\left(\lambda\right) L\left(d\right) e^{-b(\lambda)d} dd \tag{6.2}$$

However, in practice it is possible to approximate $L(d)$ as a constant, and then compute the scattered light as in equation 6.3, as explained in [Bryson et al., 2016].

$$B\left(\lambda\right) = \frac{\gamma\left(\lambda\right) L}{b\left(\lambda\right)} \left[1 - e^{-b(\lambda)d}\right] \tag{6.3}$$

Absorption and scattering effects are caused not only by the water itself, but also by other components such as dissolved organic matter or small observable floating particles. The presence of floating particles known as “marine snow”, is highly variable in kind and concentration, and increases absorption and scattering effects. These particles, that may be visible in the camera, absorb or deviate light of a specific wavelength depending on their size and characteristics.

The visibility range can be increased with artificial lighting. But, in addition to scattering and absorption, it tends to illuminate the scene in a non uniform fashion, producing a bright spot in the center of the image with a poorly illuminated area surrounding it. Furthermore, if the camera is close to the lights it exacerbates backscattering effects, increasing the amount of deviated light.

Besides these physics issues of the light propagation in water medium, additional problems derived from the camera that captures the image may appear. Mainly, two effects can degrade the acquired image in the camera: vignetting and the sensor response function. Vignetting, described in top right figure 6.1 is a fading of the light at the corners of the image. This is caused by the geometry of the lens, and sometimes by the lens housing, that partially shades the light passing in from greater angles to the principle axis of the camera.

Equation 6.4 sums up the vignetting effect as described in [Kim and Pollefeys, 2008]. Where E is the irradiance that gets to the camera sensor, $V(r)$ is the vignetting effect for an image point at r distance from the image center.

$$E = V\left(r\right) L_c \tag{6.4}$$

Using a simple camera model made of a thin lens with R radius and an image plane at a distance l from it, the vignetting effect for a ray at θ angle with the optical axis can be computed as equation 6.5.

$$E = \frac{L_c \pi R^2 \cos^4\theta}{4l^2} \tag{6.5}$$

The last degradation effect occurs when the light gets to the sensor. A sensor response function is needed to transform this light into a value that represents the amount of light. This is typically controlled by the camera firmware and, although it is usually a linear and normalized function between zero and one, can take a great variety of forms including non linear response functions.

Equation 6.6 shows a generic sensor response function f to form the image I in the camera depending on the exposure constant k and the irradiance. In [Grossberg and Nayar, 2004] a review of sensor response functions can be found.

$$I = f(kE) \tag{6.6}$$

In summary, the images that are interesting for underwater robotics can suffer from one or more of the following problems: limited range visibility, low contrast, non uniform lighting, blurring, bright artifacts, faded color (bluish appearance), vignetting and noise. Therefore, the use of standard computer vision techniques in underwater imagery requires these added problems to be dealt with first. Once solved, object detection and grasp planning can be executed normally.

Taking into account these physical degradation effects from which underwater images suffer, the image processing can be addressed from two different points of view: as an image restoration technique or as an image enhancement method:

- The **image restoration** aims to recover a degraded image using a model of the degradation and of the original image formation; it is essentially an inverse problem. These methods are rigorous but they require many model parameters, such as attenuation and backscattering coefficients that characterize the water turbidity, which are difficult to estimate in practice. Another key parameter required is the depth estimation for every pixel in the target image. Although it is interesting to restore the original image from an object detection point of view, these techniques usually require a large amount of information and processing time. This processing time and required parameters makes it difficult to use them in a real intervention, where real time results are required from a single image.

- **Image enhancement** uses qualitative subjective criteria to produce a more visually pleasing image and they do not rely on any physical model for the image formation. These kinds of approaches are usually simpler and faster than image restoration methods, allowing them to be used in a mission to help vision algorithms. The main drawback, from an autonomous intervention perspective, is the subjective criteria. A more pleasing image for the human eye may not be the best for further image processing because object colors may be exacerbated in the enhancement step producing "false" colors in the results, complicating vision algorithms.

6.2. State of the Art

The proposed goal is to obtain a method to enhance images by restoring the original colors in real time so that autonomous robot performance will be enhanced. In order to do so, methods from both approaches, restoration and enhancement have

been considered, as long as they produce images suitable for object detection and grasping, since the possibility of working in real time with as few inputs as possible is the main requisite.

Restoring degraded underwater images requires modelling and estimating many parameters such as water absorption, scattering and depth map. These kinds of inputs are difficult to estimate from a single image. For this reason, a large set of images from the same location or a combination of different sensors are typically used. There is a large amount of work on restoring underwater images, the work in [Raimondo and Silvia, 2010] offers a detailed review.

The work in [Bryson et al., 2016] uses a whole dataset of images and depthmaps from the same survey to accurately estimate the water, light and camera parameters in order to restore the true colors of the image. As a result, the degraded input images are restored, obtaining the original colors of the image set. The main drawback of this technique is, it requires a medium sized dataset of overlapping images and depthmaps of the same area, which is not always available and makes it impossible to use it in real time applications. As a consequence, it is not applicable to the proposed scenario which is to improve the robot performance in an underwater intervention.

The authors in [Roser et al., 2014] propose a method using a stereo camera to obtain a depthmap, and use it together with a single image in order to estimate the remaining parameters to restore the image. This approach has several advantages, for instance it does not require additional hardware, and can be used in a real time application. However, this method depends on a dense depthmap that may not be available when the environment is not textured enough for the stereo camera to recover 3D information, such as mud or sand commonly present in underwater images. In order to deal with this, it uses some heuristic methods to fill the gaps where no 3D information is available. Finally, it is not feasible when only a monocular camera is available.

The work in [Vasilescu et al., 2011] dynamically mixes the illumination of an object in a distance dependent way by using a controllable multicolor light source in order to compensate color loss. Given a distance to the object, the system computes the known characteristics of water and adjusts the light to compensate the attenuation. This approach achieves a great color correction, but the main problem of these kinds of solutions is the need of specific hardware to solve the problem that increases costs and hardware complexity. Additionally, as attenuation exponentially increases with distance, the presented hardware works at a maximum of 5 meters distance and it is not suitable for images with substantial depth changes.

Similarly, in order to deal with scattering some methods use long image sequences using structured illumination like [Jaffe, 2010], polarizers like [Treibitz and Schechner, 2009] or changing the location of the light source as showed in [Treibitz and Schechner, 2012]. The key idea in these methods is that controlling the illumination between frames changes the backscatter. Thus it is possible to detect it and remove it from the images. But, as happened with the previously described method it requires specific hardware in the vehicle that adds complexity to the hardware and sometimes introduces new issues such as signal attenuation in the case of polarizers. Moreover, some of these methods require multiple images making it impossible to use them online in a real intervention.

Recent work has examined the use of Markov Random Fields (MRF) and a

training stage to learn how to assign the most probable color to each pixel [Torres-Méndez and Dudek, 2005]. The MRF is trained by using pairs of input and output images, learning transformations from a patch of degraded colors to restored colors. In order to acquire the desired output images, a light source is used. This method ignores the effect of the depth in the attenuation process, learning fixed transformations for objects independently of its depth making it imprecise for object detection.

In the context of single image dehazing, [Fattal, 2008] designed a theory based on image analysis to detect hazy parts of the image and estimate the airlight color. Using the image formation model it is possible to restore part of the original signal in hazy scenes given a single input image. This research line focuses on using a single image to estimate the required parameters to restore the image.

Continuing from this, [He et al., 2011] designs a new technique based on the Dark Channel Prior (DCP) assumption. Dark Prior techniques are based on the observation that in most of the non-background patches of outdoor haze-free images, at least one color channel has some pixels whose intensity is very low and close to zero. Using this knowledge, it is possible to obtain a dark channel of the image that is closely related to depth and haze. This makes it possible to estimate enough parameters of the image formation model to restore the original colors.

This has been proved to work in most outdoor air images, and has also been adapted to underwater environments in [Chiang and Chen, 2012] or [Drews et al., 2013]. The main disadvantage of this method is that it is based on a statistical observation that may not be valid for some cases, and many works rely on subjective visual results instead of objective numerical validation. However, it has the requirements for the image preprocessing, so that autonomous manipulation: single image input, image restoration and real time performance can be enhanced.

Although part of the DCP computation is highly time consuming, in some works such as [Liang et al., 2014] or [Zhang and Zhao, 2017] the parallel possibilities have been exploded, together with some simplifications, allowing to use it in a real time pipeline. This proves it is suitable for enhancing autonomous underwater vehicles performance.

Similarly, [Ancuti et al., 2011] relies on the same assumption as the DCP but does a different processing. A semiinverse transformation is used to distinguish haze free regions from sky or haze areas in a per pixel basis that makes it suitable for parallelization. The main drawback of this technique is it relies on the same observation as the DCP, that may not be valid for underwater images, and it has not been tested in underwater environment.

The work in [Ancuti and Ancuti, 2013] follows a slightly different approach to single image dehazing. It produces two images from the original input each one accounting for a different degrading effect, chromatic casts and lack of visibility. Then it mixes the result of both images to produce the best possible image using the previously generated inputs. The work is refined in [Ancuti et al., 2016b] and [Ancuti et al., 2016a] to produce three input images with the transmission, global backscattering and local backscattering estimations and using a multi scale fusion to obtain a single restored image. The transmission estimation steps also uses the DCP estimation.

Other alternatives such as [Tarel and Hautiere, 2009], rely on assumptions that are not fulfilled in underwater situations such as pure white color of fog regions in

Cuadro 6.1: Comparison of different method properties for image dehazing.

Method	single image	No additional hardware	real time
Bryson15 [Bryson et al., 2016]	X	√	X
Torres-Méndez05 [Torres-Méndez and Dudek, 2005]	√	X*	√
Roser14 [Roser et al., 2014]	X	√	√
Vasilescu11 [Vasilescu et al., 2011]	√	X	√
Descattering [Jaffe, 2010] [Treibitz and Schechner, 2009] [Treibitz and Schechner, 2012]	√	X	√
DCP Based, He11 [He et al., 2011]	√	√	√
Ancuti16 [Ancuti et al., 2016b]	√	√	√
Histogram equalization [Garg et al., 2011]	√	√	√

white balanced images. In the case of underwater imagery, the fog may have a bluish or greenish tone depending on the water characteristics. Using this assumption and a similar observation to the DCP the method is able to estimate the atmospheric veil and restore the image colors.

In the case of [Kratz and Nishino, 2009], the author proposes a Factorial Markov Random Field (FMRF) to jointly estimate the albedo and depthmap in order to dehaze the image. However, the FMRF requires scene specific priors as obtaining information about albedo and depth is an inherently ill-posed problem. The main drawback of this method is the priors proposed are based on the chromaticity of the image that is completely distorted in an underwater environment, thus making it impossible to use it in that context.

In terms of image enhancement, a histogram equalization is typically used as described in [Garg et al., 2011]. These techniques analyse the histogram and transform it to accomplish a determined distribution that produces visually pleasing images. The main drawback of this approach is it amplifies the noise in homogeneous regions and creates false colors. Some research lines work to palliate this problems like [Hitam et al., 2013], [Iqbal et al., 2010] and [Ghani and Isa, 2015] combining different histogram equalizations, target distributions and contrast stretching.

Another common strategy consists in maximizing the contrast of the image, as hazy images have usually small contrast. The work in [Tan, 2008] follows this strategy assuming a smooth layer of airlight. However, the results of this method suffer from halos near depth discontinuities and does not produce good results in saturated images.

As discussed before, an underwater image preprocessing technique that restores degraded image colors from water turbidity is required in order to increase vision algorithms performance. This technique needs to be executed in real time, thus it is possible to use it in an autonomous underwater vehicle, with limited hardware payload due to vehicle restrictions. For this reasons, in table 6.1 a comparison of the different state of the art methods is shown with the most important features of the system.

As can be seen only the histogram equalization and DCP, and other techniques that use it, fully accomplish all the required features for the proposed use case. The Markov random fields based solution in [Torres-Méndez and Dudek, 2005], does not directly need additional hardware but it means that groundtruth dehazed images have to be obtained, and that may be impossible to acquire without it. The main disadvantage of histogram equalization techniques is that they do not assure a realistic image restoration as they are usually not based in a formal model of the

image formation. In any case, they will be compared with the proposed dehazing solution in Chapter 7.

For this reason, in this chapter the performance of different dark channel prior transmission estimations is benchmarked. The main goal of this study is to analyse the possibilities of enhancing autonomous underwater vehicles performance preprocessing the image to obtain a more suitable input for vision algorithms.

6.2.1. Dark Channel Prior dehazing

The original Dark Channel Prior, presented in [He et al., 2011], was designed for hazy outdoor images instead of underwater imagery. However, it has also been used in underwater images due to the similar physical model of both phenomena. Its purpose is to recover the image without haze effects. In the process it estimates the medium transmission and atmospheric light from a single image, and then, using the image formation model, restores the original image.

The image formation model assumed by the DCP is expressed in equation 6.7 where E is the irradiance that reaches the camera sensor, R is the reflectance of the object, $\widetilde{t}$ is the transmission of the system and A is the global atmospheric light.

$$E = R\widetilde{t} + A\left(1 - \widetilde{t}\right) \tag{6.7}$$

The transmission of the system for a pixel x is defined in equation 6.8. Thus, the transmission $\widetilde{t}$ depends on a constant attenuation b and the distance to the object d.

$$\widetilde{t}(x) = e^{-bd(x)} \tag{6.8}$$

As can be seen, this model is similar to the underwater image formation model explained in section 6.1 combining attenuation as described in equation 6.1, and backscattering effects, expressed in equation 6.3. The global atmospheric light is now a constant describing the backscattering: $A = \frac{\gamma(\lambda)L}{b(\lambda)}$, and the transmission is not dependant on the wavelength because light attenuation does not depend on wavelength in air free images. For this reason, it is reasonable to use it in an underwater application.

As the image formation model used is widely accepted for this process, although more complex models may be used such as including vignetting or sensor response functions, the most important part is the parameter estimation. As a consequence, most of the DCP variants only change the process for estimating transmission and atmospheric light, keeping the rest of the process unchanged.

The transmission estimation is done taking into account the DCP assumption that non-background patches of the image have at least one color channel close to zero in non-hazy images, expressed in equation 6.9

$$\widetilde{t}(x) = 1 - \min_{y \epsilon \Omega(x)} \left(\min_{c} \frac{E^c(y)}{A^c} \right) \tag{6.9}$$

In it, $E^c(y)$ is the intensity of a pixel y for channel c and A^c the atmospheric light for channel c. The first minimization corresponds to the search in the local patch $\Omega(x)$ around a pixel x, and the second one around the RGB channels of the

Figura 6.2: DCP dehazed images in different steps showing halos, original source images from [He et al., 2011].

image. It looks for the minimum channel in a local patch around the target pixel, if this minimum is high it means the image is hazy at this point as it contradicts the previously declared assumption.

The estimation of the atmospheric light is made from the previous Dark Channel. As the Dark Channel is considered to be an estimation of the image haze, the atmospheric light will be one of the haziest pixels. The authors propose to use the brightest pixel of the top 10 percent of the haziest dark channel with the initial atmospheric light guess.

In summary, in order to estimate the haze and the atmospheric light two steps of the Dark Channel Prior will be made, one with an initial guess of the atmospheric light, typically 1, and then the final estimation of the atmospheric light will be calculated in order to refine the transmission estimation.

At this moment, the estimated transmission, usually called coarse estimation, suffers from noise and halos caused by the estimation procedure. The noise can be smoothed using any kind of denoising filter, but the main problem occurs at object borders where there are sudden changes of haze density caused by changes in depth.

Halos are artificial borders created in between two objects due to the nature of the minimization function in a neighbourhood patch. This can be seen in figure 6.2, in the coarse estimation the house is correctly estimated but there are halos in the tree branches and leaves. As can be seen, the refinement step smooths this effect and correctly removes the halos.

In order to solve this, a soft image matting step, as presented in [Levin et al., 2008], takes place to smooth the transmission estimation using the original image as a guide. The natural image matting is a process that extracts an object from the foreground of an image. This process together with a bilateral filter, described in [Tomasi and Manduchi, 1998], smooths the transmission respecting the original borders and avoiding artifacts. This is possible due to the similarities between the image formation model of the DCP, equation 6.7, and the image matting equation 6.10.

$$I = F\alpha + Bg(1 - \alpha) \tag{6.10}$$

In the above equation, F and Bg are the foreground and background colors respectively and α is the foreground opacity. As a consequence it is possible to

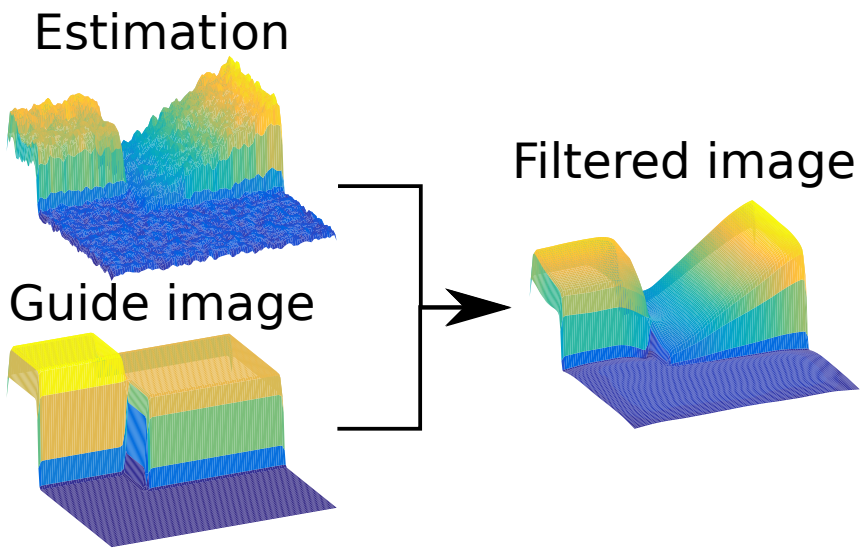

Figura 6.3: Guided filter example used in depthmaps.

apply a closed-form of matting to obtain the foreground and background. Then, the optimal refined transmission can be obtained solving a sparse linear system, and use it in a bilateral filter to smooth the image.

As this is an extremely slow process due to the non linear time complexity, other works propose the use of a guided filter, presented in [He et al., 2010], to make it faster. The guided filter performs an edge preserving smoothing on an image, using the content of a second guidance image. In this case, the coarse estimation is smoothed using the raw original image

This effect can be seen in the figure 6.3, the estimation is smoothed respecting the edges in the guide image. The black white images are represented as surfaces in order to see the filtering effect clearly. As can be seen, the random noise in the surface estimation has almost disappeared in the filter image, but it still preserves the edges already present in the guide image, this is called structure transference.

Finally, the original image is restored using the inverse formula of equation 6.11 derived from the assumed DCP image formation model.

$$R = \frac{E - A}{\widetilde{t}} + A \tag{6.11}$$

From this initial DCP dehazing, several variations and enhancements have been proposed to adapt it to the underwater environment. Most of them focus on small changes in the dark channel calculation, expressed in equation 6.9. On the other hand, other works change the smoothing step to accelerate the computation or to increase the performance of the technique.

The authors in [Carlevaris-Bianco et al., 2010] propose using the different attenuation of light for each color channel as a depth prior to estimate transmission instead of the dark channel. The red color attenuates much faster than green and

blue. Taking advantage of this fact, the so called **Bianco prior (BP)** compares the maximum intensity of the red color to the maximum of green and blue over a small image patch. Thus, the Bianco prior difference $D(x)$ can be seen in equation 6.12.

$$D(x) = \max_{y \epsilon \Omega(x), c \epsilon r} E^c(y) - \max_{y \epsilon \Omega(x), c \epsilon \{g,b\}} E^c(y) \tag{6.12}$$

In it, $E^c(y)$ is the intensity of a pixel x in color channel $c \epsilon \{r, g, b\}$ in the image E. The first maximum is computed for a patch in the red channel and the second one for the blue and green channels. The transmission $\widetilde{t}$ can be computed using the largest difference $D(x)$ that represents the closes point in the foreground using equation 6.13

$$\widetilde{t}(x) = D(x) + \left(1 - \max_x D(x)\right) \tag{6.13}$$

The **median dark channel prior (MDCP)**, presented in [Gibson et al., 2012], observes there is a loose relationship between the dark channel and the transmission when depth variation is not smooth and when the texture is high. This is partially solved by the image matting smoothing step. However this step is extremely slow and makes the whole process difficult to apply in real time applications. In order to solve this, the authors propose the dark channel computation should be modified, replacing a minimum with a median as equation 6.14 shows.

$$\widetilde{t}(x) = 1 - \underset{y \epsilon \Omega(x)}{median} \left(\min_c \frac{E^c(y)}{A^c} \right) \tag{6.14}$$

The variation proposed in [Drews et al., 2013], is based on the observation that the dark channel in underwater images corresponds to the red channel due to the extremely higher absorption of its wavelength. This red channel is independent from the scene depth and, as a consequence, from the transmission in most images. In this work an **underwater dark channel prior (UDCP)** is presented modifying the original so that only in the green and blue channels. Thus the estimation of the transmission is made through equation 6.15.

$$\widetilde{t}(x) = 1 - \min_{y \epsilon \Omega(x)} \left(\min_{c \epsilon \{g,b\}} \frac{E^c(y)}{A^c} \right) \tag{6.15}$$

The **median underwater dark channel prior (MUDCP)** is a combination of the two previously described priors. Described in [Lu et al., 2015], it shows the need of using a dual-channel, red and blue in this case, with a median operator as described in equation 6.16. Using this transmission estimation, the authors derive a depth map to further process the images.

$$\widetilde{t}(x) = 1 - \underset{y \epsilon \Omega(x)}{median} \left(\min_{c \epsilon \{r,b\}} \frac{E^c(y)}{A^c} \right) \tag{6.16}$$

Finally, authors in [Galdran et al., 2015] and [Codevilla et al., 2014] suggest the use of the inverse of the red channel to achieve a more precise result. This **red channel prior (RCP)** was suggested because of the fact that in underwater environments red intensity decays faster as distance increases. As a consequence, the transmission estimation step is modified accordingly as equation 6.17 expresses.

$$\widetilde{t}(x) = 1 - min\left(\min_{y\epsilon\Omega x}\frac{1-E^R(y)}{1-A^R}, \min_{y\epsilon\Omega x}\frac{E^G(y)}{A^G}, \min_{y\epsilon\Omega x}\frac{E^B(y)}{A^B}\right) \tag{6.17}$$

6.3. Dark Channel Prior Benchmarking

As discussed in previous sections the dark prior techniques are the best way to enhance the performance of an autonomous underwater vehicle in real time. It is possible to restore a degraded image with just a single image as input in reasonable time to use it in a running intervention. Moreover, it does not require additional hardware, and so it is possible to use it in any vehicle as long as it has a color camera.

However, there are many variations and different adaptations to the underwater environment. Furthermore, to the best of the authors knowledge, there is not an objective comparison of these techniques in the context of autonomous underwater vehicle imagery. For this reason, the existing dark prior adaptations are compared in an objective experiment to decide which is the best method for the target underwater vehicle images.

Bearing this in mind, a set of images have been prepared to test the DCP variations under different objective metrics. The images dataset include depth information retrieved with stereo cameras, stereo in motion and odometry that will serve as groundtruth of the transmission estimation. As the transmission estimation is linearly dependant on depth, some metrics are proposed to estimate the precision of the estimation using this groundtruth.

6.3.1. Image datasets

For this purpose, mainly two different datasets have been used to benchmark the DCP dehazing, although three additional datasets have been used for further tests and results. A full description of the datasets characteristics, acquisition details and samples can be consulted in the Appendix A. In this case rocks and rocks and sand have been extensively used.

These datasets were chosen due to the uniformity of the images and the availability of depth information. This information associated with each image allows the performance of the algorithms to be measured compared with the transmission estimation. The uniform texture across the images helps the DCP methodology being a best case for it.

Example images from each dataset are shown in figure 6.4. Images cover a different range of textures, vehicle depths, illumination conditions and distance from the camera to the seafloor.

6.3.2. Metrics

Typical approaches that evaluate the performance of the DCP are based on measuring different properties of the resulting image such as contrast, visibility, etc. Another option is to use simulated imagery, so that it is possible to obtain both a hazy and a clear image, to compare with a groundtruth image. But most of them,

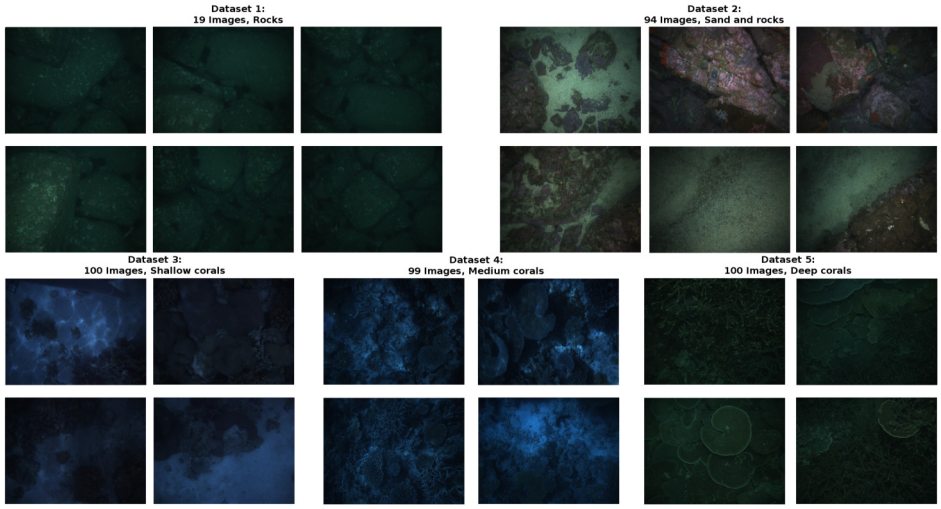

Figura 6.4: Images of the different datasets used in the thesis for image dehazing.

to the best of the authors knowledge, do not evaluate the estimated parameters of the image formation model that are the key to the technique.

In contrast, this work uses the dense depthmap obtained with a stereo camera as groundtruth of the transmission estimation. This is reasonable because the transmission estimated by the DCP is linearly proportional to the depthmap after exponentiating as equation 6.8 shows.

In this equation, d corresponds to the depthmap and $b(\lambda)$ is a function dependant on the wavelength of the light caused by the different attenuation of the light underwater. In the case of a single image it can be assumed $b(\lambda)$ is constant through all the image as the image covers a small region, so underwater turbidity should be almost constant. In the case of DCP estimation, the transmission is independent from the wavelength of the light so it will be assumed $b(\lambda)$ is constant or the transmission estimated is for a concrete wavelength, the consequences of this assumption in the results will be discussed later.

Taking into account this relation between transmission and depthmap, two metrics are proposed. The first is the sample Pearson correlation coefficient, expressed in equation 6.18, which measures the linear correlation between two variables X,Y giving a value between -1 and +1 where 1 is total positive correlation, 0 means the two variables are not linearly correlated and -1 is total negative correlation.

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y}, r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{6.18}$$

In order to calculate the correlation, the $log(\widetilde{t})$ and the stereo depthmap are used as variables X and Y, ignoring the pixels where the groundtruth depthmap is not available. If the DCP is correctly working, the correlation should be close to -1, but a positive correlation would be good too as it would mean the DCP is

the inverse of the transmission. However, this would only happen if DCP worked perfectly, any result above 0.5 is considered good in terms of correlation.

This metric is useful to know if both variables are similar in terms of appearance. If both variables are represented as a surface, the correlation describes the similarity of the shapes of both surfaces. Thus it can be considered as a local structure metric.

The second proposed metric is an Adjusted Root Minimum Squared Error (ARMSE). As it is not possible to directly apply the RMSE, because the compared variables are linearly proportional, multiplied by an unknown constant as described in equation 6.19. Actually, this constant is the transmission attenuation b, which is finally obtained from the groundtruth so both terms have the same mean.

$$\log \widetilde{t}(x) = -bd \tag{6.19}$$

In order to obtain the transmission estimation from the depthmap, the mean of the log estimated transmission and groundtruth depth is calculated, and an attenuation b is obtained to adjust the estimated transmission to estimated depth, minimizing the mean error between estimation and groundtruth. As a consequence, this is a best case RMSE that assumes the best possible value of b.

Furthermore, from the point of view of estimating a depthmap from a single image, the parameter b is the scale factor. This makes sense as it is impossible to obtain a 3D reconstruction from a still image due to the lack of scale information. However, it is possible to estimate an unscaled depthmap. This makes reference to the difficulty of distinguishing a photo of a realistic doll house from a photo of a real house, or the need to use a known object for the purpose of estimating the size of an unknown object in a photo.

Using this ARMSE metric also makes it possible to obtain an estimated depthmap from the DCP transmission which can be directly compared visually and numerically. Thus, 3D plots of the surfaces, error histograms, and RMSE can be obtained for this measure as the one shown in figure 6.5.

6.3.3. Compared algorithms

The previously described metrics have been used to test a wide variety of dark channel prior variations. The goal is to be able to decide which one works better in the proposed datasets and choose the best option for an autonomous underwater vehicle.

Besides the previously presented literature adaptations, 8 new versions have been added to the comparison in order to test different aspects of the process. Some of them are naive approaches to establish boundaries of a good performance and discard situations where the raw image is already related to the depthmap. This can be a common situation in underwater images due to the fast attenuation of light that makes objects that are further away appear darker.

The first proposed variation is a **blue channel prior** that uses only the blue channel of the image. It is a naive prior based on the observation that objects that are further from the camera have a bluish tone. As a consequence transmission is estimated using equation 6.20.

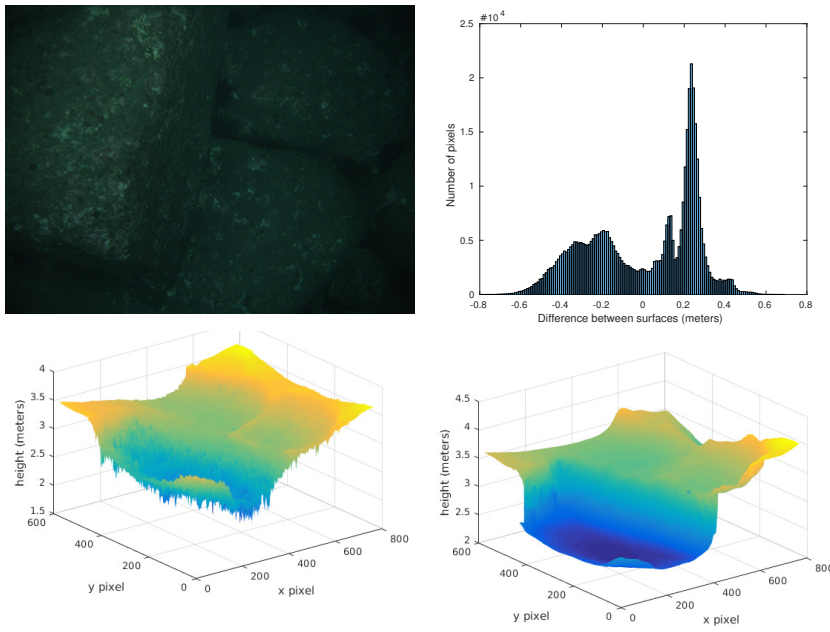

Figura 6.5: Visualization of the image (top left), error histogram (top right), DCP estimation surface (bottom left) and depthmap surface (bottom right).

$$\widetilde{t}(x) = 1 - \min_{y \epsilon \Omega(x)} \frac{E^B(y)}{A^B} \tag{6.20}$$

Similarly, the red channel attenuates faster than other channels. This fast attenuation can be used to derive a depth estimation, assuming a uniform texture as further objects will have a more attenuated red color than ones which are closer. It is the inverse situation to the previous case. Equation 6.21 describes the **only red channel prior**.

$$\widetilde{t}(x) = 1 - \min_{y \epsilon \Omega(x)} \frac{E^R(y)}{A^R} \tag{6.21}$$

Following this way of thinking AUV images usually fade in intensity as depth increases due to the illumination attenuation. However, in the RGB colorspace it is difficult to detect this intensity fade out. For this reason, the pixels are initially transformed to HSV (Hue, Saturation and Value) colorspace where the V axis corresponds to the luminosity of the pixel. This **HSV prior** uses just the luminosity information as equation 6.22 shows, where $HSV(X)$ is the transformation of a pixel X to HSV colorspace.

$$\widetilde{t}(x) = 1 - \min_{y \epsilon \Omega(x)} \frac{HSV(E(y))^V}{HSV(A)^V} \tag{6.22}$$

Another colorspace typically used in vision algorithms is the CIELAB. This colorspace also differentiates luminosity from color, in this case the l axis ranges from 0, total darkness to 100, full luminosity. The equation 6.23 describes the transmission estimation of the **CIELAB prior** casting the image to CIELAB colorspace to compute the dark prior using the luminosity in it. Like the previous case, $CIELAB(X)$ denotes the transformation of a RGB pixel X to CIELAB colorspace.

$$\widetilde{t}(x) = 1 - \min_{y \epsilon \Omega(x)} \frac{CIELAB(E(y))^l}{CIELAB(A)^l} \tag{6.23}$$

The following compared variation uses the ideas behind the median dark channel prior that says there is a loose relationship between the estimated transmission and the real transmission when depth variation is not smooth. But in this case, instead of using the median, the mean of the minimum channel in the patch is used as shown in equation 6.24. This **mean dark channel prior** should produce an image that does not require a smoothing step.

$$\widetilde{t}(x) = 1 - \underset{y \epsilon \Omega(x)}{mean} \left(\min_{c} \frac{E^c(y)}{A^c} \right) \tag{6.24}$$

The **inverse dark channel prior** has been introduced as a boundary measure. The tested hypothesis is if the DCP is a good estimation of the transmission the inverse should produce bad results. Thus, instead of computing the minimumm channel in the patch, it searches for the maximum. This can be computed using equation 6.25.

$$\widetilde{t}(x) = 1 - \max_{y \epsilon \Omega(x)} \left(\max_{c} \frac{E^c(y)}{A^c} \right) \tag{6.25}$$

Another boundary method added to the comparison is a **random prior**. In this case, a random channel of a random pixel in the patch is chosen to be representative of the transmission. This control version will distinguish the cases where the input image is already correlated with transmission and help to understand if the other priors are improving this initial correlation or not. The equation 6.26 shows the transmission estimation for this variant.

$$\widetilde{t}(x) = 1 - \underset{y \epsilon \Omega(x), c}{rand} \frac{E^c(y)}{A^c} \tag{6.26}$$

Finally, the last proposed algorithm is the **plane prior** that estimates a constant transmission for the whole image. This prior will act as a boundary for the ARMSE metric, anything above this error is worse than not predicting a surface at all. This estimation can be expressed as equation 6.27 shows.

$$\widetilde{t}(x) = 0{,}5 \tag{6.27}$$

To sum up, the compared DCP variations are the 6 presented in the dark channel section of the state of the art: standard DCP, Bianco prior, median DCP, underwater median DCP, underwater DCP and red channel prior. Additionally, the previously described 8 new implementations have also been included: only blue, only red, HSV prior, CIELAB prior, mean DCP, inverse DCP, random and plane.

6.4. Results

Four experiments have been conducted to test different aspects of the compared algorithms. These experiments try to understand the steps of the dark channel prior in order to benchmark all the possible combinations, and decide the best option for the case under study. Each experiment focuses on a part, maintaining the rest of the algorithm, thus at the end it is possible to deduce the importance of the changes made. The experiment goals are described below.

- **Original methods comparison:** In this case, the original methods are compared as described in the bibliography. The first dataset is used for the comparison as it is the least challenging environment, a single texture and colors.
- **Atmospheric light estimation:** The second important step in the dark channel prior dehazing methodology is the atmospheric light estimation. However, almost none of the reviewed works adapt this computation for the underwater environment. This experiment, omits this step in order to see its impact on the final result. As in the previous case, the easiest dataset is used to obtain a best case scenario.
- **Refinement benchmarking:** This experiment compares the two principal refinement steps, image matting and guided filter as well as no refining at all. The main focus is if the guided filter is able to perform similarly to image matting as it is faster to comput, thus allowing a real time application.
- **Different textures:** The final test compares the most promising configuration options in different datasets. Furthermore, it compares the performance in cases where only one texture is present in the image with multi textured images.

Although an experiment with the simulated software shown in Chapter 2 would be interesting, using this dehazing algorithms in it was discarded because of the differences between the simulated haze model and the real underwater image formation. Some degradation effects such as scattering or vignetting are not properly modelled in the osgOcean visual simulation, thus the results of the dehazing strategies would not be concluding. Moreover, simple processes can be designed to restore the degraded images from the simulation that outperform the current state of the art but would not work in real experiments, making it difficult to extrapolate results to real environments. However, this is included as future work in Chapter 8.

6.4.1. Original methods

In the first experiment, the different implemented solutions to be compared have been benchmarked without modifications, as described in the bibliography, in order to see the performance of each algorithm. The choices between image matting, guided filter or the method to estimate the atmospheric light have been respected as described in the corresponding papers. About the new variations that have been proposed, the settings from the paper that inspired them are used.

The rocks dataset, have been chosen due to the uniformity of the images. These images show a rocky seafloor with uniform colors that may help to retrieve depth

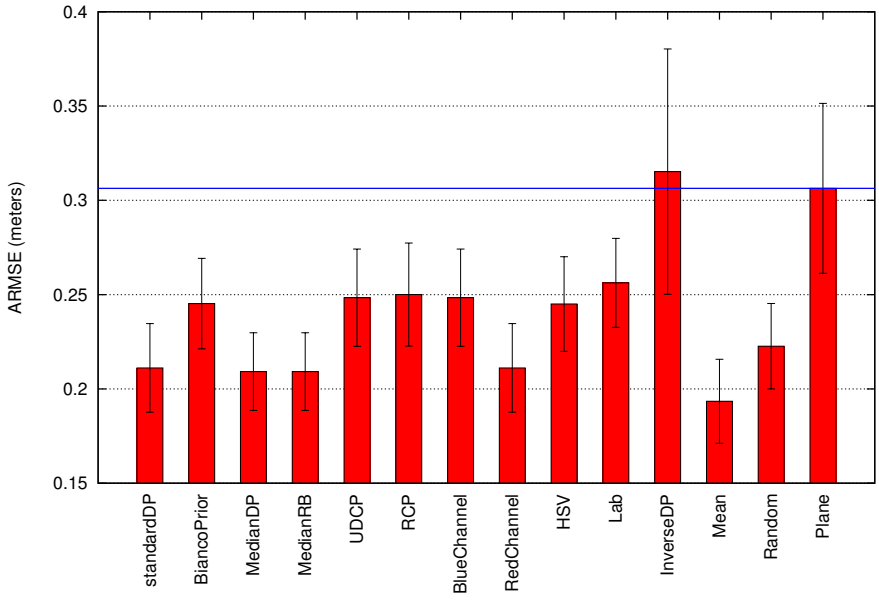

Figura 6.6: DCP variants comparison for rocks dataset and ARMSE metric.

information from a single image. Furthermore, the use of strobes assures a better illumination, reducing the errors caused by unexpected shadows.

The results for the ARMSE metric can be seen in figure 6.6, graph bars show the mean value for the whole dataset, and the errorbars is the 95 % confidence interval of the images mean. The blue horizontal line marks the boundary of the plane estimation. Anything above this error is worse than not estimating anything. The comparison includes the complete set of DCP based versions already presented.

As expected, inverse DCP obtains the worst result which is even worse than a constant estimation. Regarding other priors performance, it is possible to order them in two groups depending on their performance. From higher to lower error, the first group is around 0.25 meters and is formed by lab, RCP, blue channel, UDCP, HSV and Bianco prior. The second group has a slightly smaller error, around 0.21 meters, including DCP, red channel, MDCP and UMDCP. Choosing a random pixel in the neighbourhood performance is in the middle of both groups at 0.225 meters error. Finally, the best algorithm according to ARMSE metric is the mean prior with an error of 0.19 meters.

Both median priors perform exactly the same, this means the green and blue channels of the image are not being used as they are never the minimum value in the pixel. This is reasonable as the red channel attenuates much faster than blue and green channels, thus the channel minimization is of little use in this case. The minimization keeps choosing the red channel as can be seen in the “only red” performance, which is very similar to standard DCP and median priors.

Avoiding the use of the red channel of the image, such as UDCP and blue

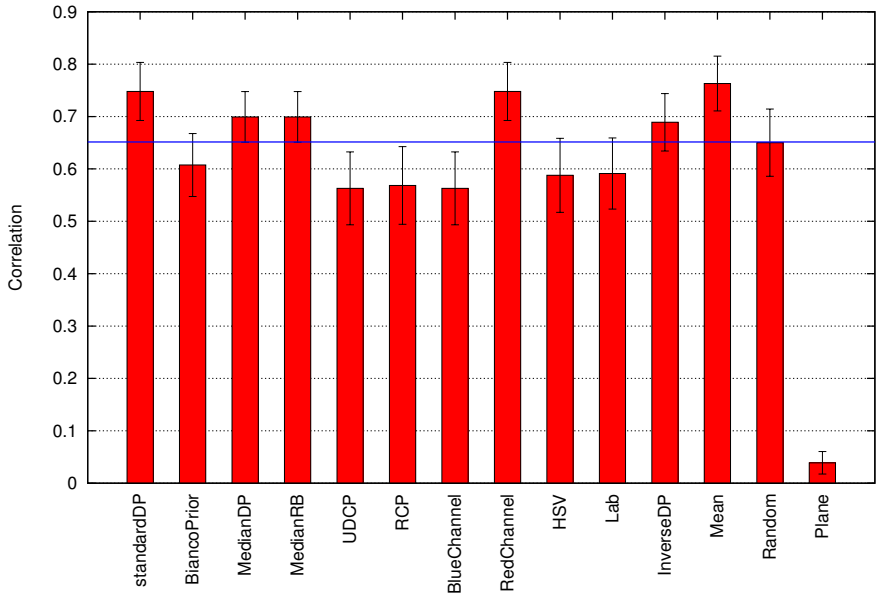

Figura 6.7: DCP variations comparison using correlation metric for rocks dataset.

channel priors do, or using the inverse of the red channel as in RCP, seems to have a negative effect on the performance. Thus, it can be concluded that the red channel contains valuable depth information.

The random performance shows that the image is already related to depth due to the light attenuation. In terms of the ARMSE metric, only 5 out of 12 algorithms, boundary algorithms are not considered, performed better. However, this good result of the random choice of the dark channel is enhanced by the remaining steps of the dark channel prior dehazing algorithm, specially refinement as will be discussed below.

In the case of correlation, the results confirm this first impression as figure 6.7 shows. The correlation data is shown as an absolute value as it is not important if the relation is positive or negative. However, the mean correlation is made from the raw results, so if an algorithm obtains positive and negative correlations the result will be close to 0. The average and confidence interval for the dataset are shown as in the case of the ARMSE metric using bar and errorbar. A horizontal line is drawn with the random result as any estimation below this line is worse than the already correlated image.

In this case, every variation of the DCP except the constant estimation shows a medium to strong correlation with the groundTruth. The main drawback is that the random also shows a good 0.65 correlation. This proves the raw data is already correlated with the desired output and/or other stages of the process, mainly refinement, which are also an important part of the method. Light attenuation may be causing this effect as pixels closer to the camera are brighter than deep ones. As a

consequence, choosing a random pixel in the case of underwater imagery, which is not the set up that the DCP was originally designed for, is a good depth estimation.

It is important to remark that although a good correlation usually means a good ARMSE, this is not always true. Inverse DCP has a fairly good correlation value, near 0.69, but it is the worst algorithm in terms of the ARMSE metric. This means the scale correction is wrong, although the ARMSE metric equals the mean and is able to compare transmission and depth in absolute terms, the scale of both measures is not adjusted.

Taking this into account, only 7 out of 12 approaches perform better than the random algorithm: standard DCP, Mean, MDCP, UMDCP, RedChannel, InverseDCP and mean. The best result is obtained by the Mean, as happened in ARMSE with 0.7629 correlation, more than 10 % better than random selection.

The main conclusion of this experiment is the DCP is working because the data is already showing some correlation with the transmission. However, the use of certain priors and other steps of the dark channel prior dehazing algorithm enhance the results, reducing the errors and increasing the correlation. The two following experiments explore the atmospheric light estimation and the refinement step impact in these results.

6.4.2. Atmospheric light estimation

The second experiment explores the atmospheric light A^c estimation of the dark channel prior. Due to the difficulty of obtaining a valid groundtruth for this parameter, the overall performance is compared using the proposed estimation for each variant and omitting it. If the atmospheric light estimation is correct the metrics should show an improvement when using it to dehaze the image.

Analysing the importance of this estimation in the dehazing algorithm in equation 6.8, it can be seen it is almost a change in the scale. Some points may change as the atmospheric light intensity is different for each channel, but in practice this does not happen for many pixels. As a consequence, this change should be more noticeable in the ARMSE metric because correlation is invariant to scale changes.

As in the previous experiment, the rocks dataset has been used because it is the easiest case for the dark channel prior estimation.

The results shown in figure 6.8 demonstrate that there is no big difference between the raw transmission estimation and the one after atmospheric light estimation in the case of AUV imagery. The only noticeable difference is in the case of inverse DCP, where a bad choice of atmospheric light is producing big errors. As can be seen, omitting the atmospheric light estimation only has a big impact in the case of a wrong estimation, this case can be seen in the inverse DCP algorithm.

There is nothing to be compared with Bianco Prior because it never computes the atmospheric light. Plane estimation has exactly the same transmission estimation because it is independent from the atmospheric light estimation as it returns a constant value. Now the plane estimation is the worst as expected.

The original DCP, mean channel and red channel work, take advantage of the atmospheric light estimation. On the other hand cielab and the previously mentioned inverse DCP work better without it, meaning both techniques estimate the atmospheric light wrongly and these errors lead to bad results according to the ARMSE metric.

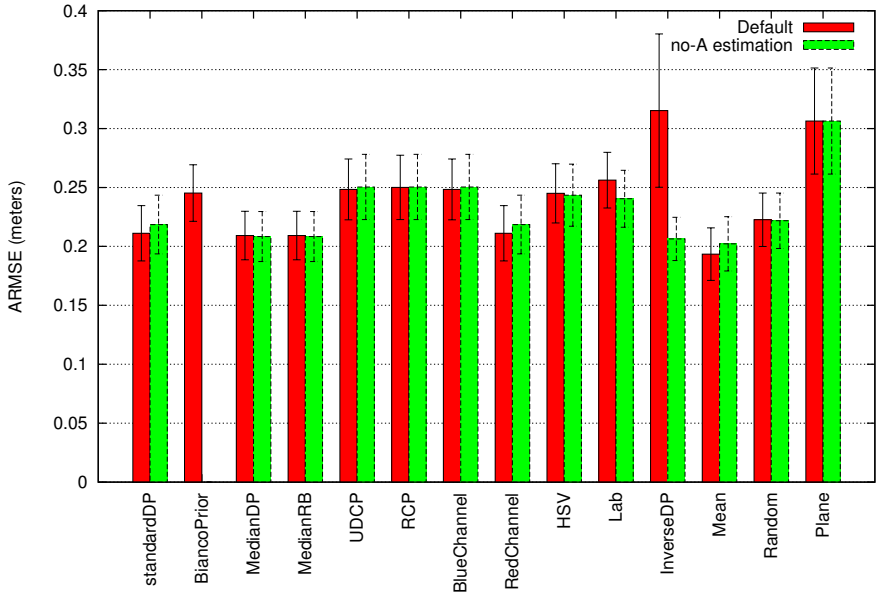

Figura 6.8: DCP transmission estimation comparison depending on atmospheric light estimation using rocks dataset.

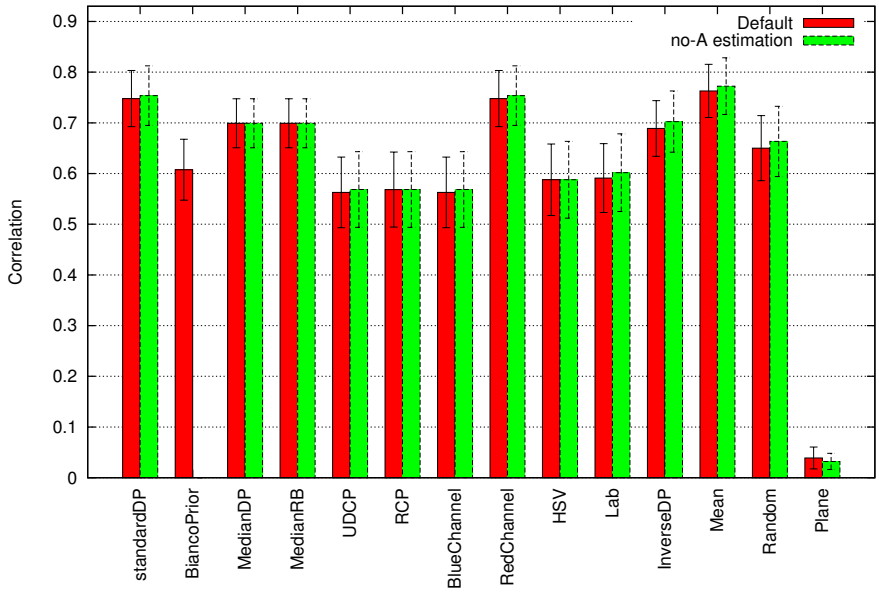

Figura 6.9: DCP transmission estimation comparison depending on atmospheric light estimation using correlation metric on rocks dataset.

Figure 6.9 shows the results for the correlation metric. As expected, the difference between estimating the ambient light or omitting this step is negligible. This result confirms the atmospheric light estimation is almost a scale change and correlation is invariant to scale. Consequently, only small changes due to the recalculation of the algorithm after the scale change are produced, and the correlation suffers minor changes.

In conclusion, the ambient light estimation is important for the dark channel prior image dehazing so that the amount of haze that needs to be removed from the image can be established. As this benchmark is based in comparing the transmission with a depthmap, an unknown scale transform is required, making it very difficult to decide if the ambient light estimation is good.

However, the ARMSE metric that assumes a perfect scale estimation in order to compare the transmission estimation and the groundtruth depthmap, shows when the ambient light estimation is not correct. In this situation the best implementation is obtaining 0.19 meters of mean error from a dataset of an average 3.64 meters that is 5 % of mean relative error.

This is a good result, given the input is a single image, and means it is possible to use the light degradation in water as a cue to estimate image depth. However, a random choice of the pixel that forms the dark channel obtains a 0.22 mean error that is around 6 % relative error. The conclusion of this results is the image intensity is already related to depth, and the DCP enhances these results.

But in any case, a mean error of 20 centimetres is acceptable for image dehazing as it means the resulting image will only have a haze of 20 centimetres of water. So, in this case the DCP is a good alternative for a real time underwater dehazing when only a single image is available.

6.4.3. Refinement step

The last step of the dark channel prior algorithm is the refinement step. Until this step, the method obtains a coarse estimation that usually suffers from "halos" around objects. These halos are caused by the minimums applied in the neighbourhood in equation 6.9, and make the estimation more imprecise along the borders of the objects. In order to obtain a higher quality result, it is necessary to refine this coarse estimation and retrieve the lost borders around objects at different depths.

This experiment compares different alternatives in the literature for implementing the refinement step. The main reason for this experiment is not only to find the performance differences, but to compare the running time as this is the most time consuming step of the DCP. Although the DCP is able to run in a short time, in the proposed use case of enhancing the robot performance in the intervention this is a key parameter as the algorithm must run as fast as possible.

The original DCP used a soft matting, but this step is extremely slow due to its computational complexity. For this reason many authors use a much faster "guided image filter" as presented in [He et al., 2010]. In this experiment, the performance of these techniques is compared in terms of the error produced in the transmission to see if the faster execution is worthy or not, and if it can be used in real time applications. An unrefined output has also been added to the comparison in order to see the real impact of this step on the algorithm.

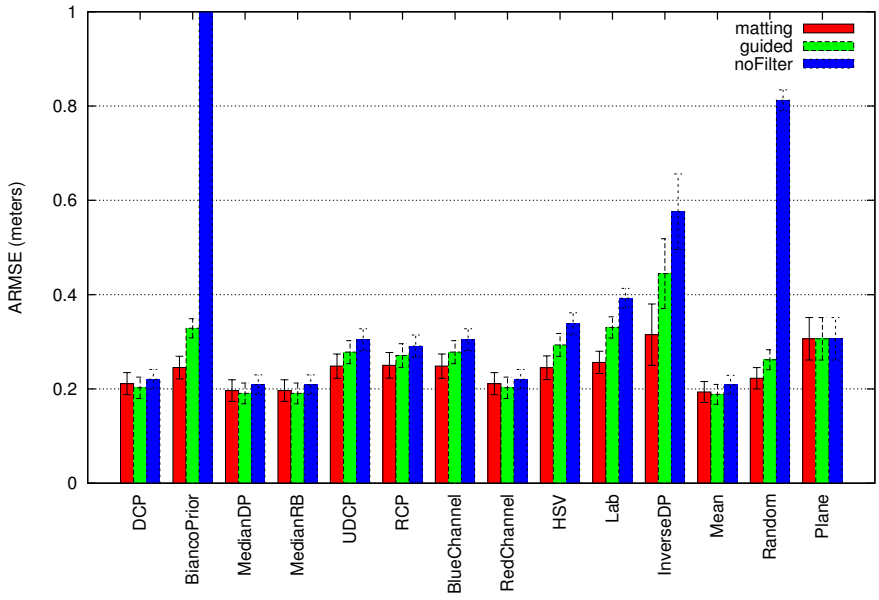

Figura 6.10: Filtering step comparison using ARMSE metric for different DCP variations.

The results of this experiment for the ARMSE metric can be seen in figure 6.10. As in the previous experiments, the graph shows a bar with the mean error for each algorithm, and errorbars for the 95 % confidence interval. The blue bars are used for the unrefined estimation, green bars for the guided filter and red corresponds to the image matting version.

As can be seen in the results, the refinement step is a key stage that reduces the error in all cases except the plane estimation that cannot be refined due to its characteristics. In the cases of Bianco Prior and random estimation the filtering step is crucial, reducing the error by more than 70 %. There is also a lot of benefit for the inverse DCP lab and HSV in this process, reducing the mean error by around 40 % for the ARMSE metric. The rest of the compared alternatives also benefit from this error reduction ranging from 18 % in RCP and UDCP cases to 6 % in median and only red channel.

Given these results, it seems clear that the refinement step is an important process and the impact on the final performance is high. Even in cases where the initial coarse estimation is not good, it reduces the error to acceptable levels as in the case of the Bianco prior, random estimation or inverseDP. In the rest of the cases, even when the initial estimation is already good, it helps to enhance the results decreasing the errors.

In the comparison of guided filter and image matting, the results are not conclusive. The image matting procedure seems to reduce the error to a higher degree when the initial estimation was bad, as in the cases of Bianco Prior, HSV, CIELAB, inverse DCP or random. However, when the initial estimation is more precise the

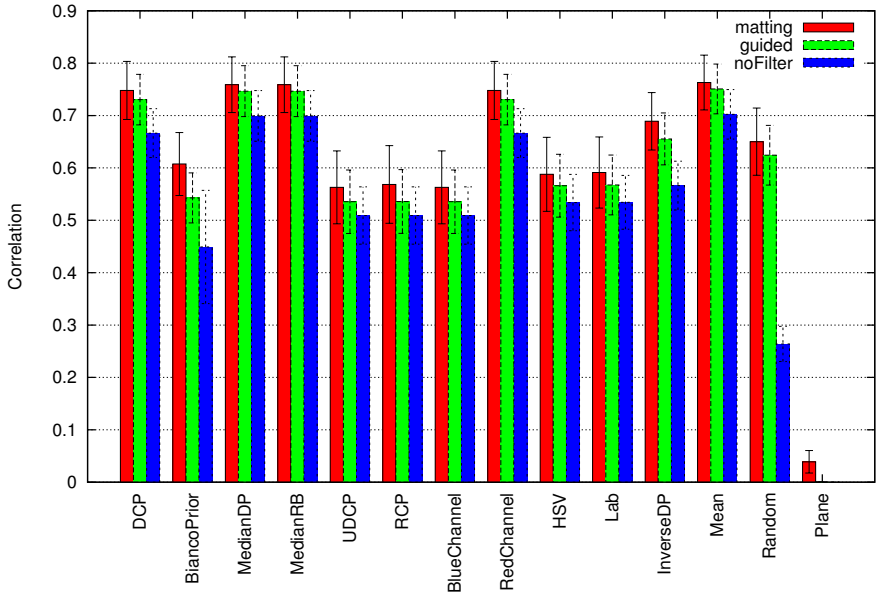

Figura 6.11: Filtering step comparison for different DCP variations.

guided filter is able to enhance the results better than image matting. This happens in the median priors, original DCP, only red channel and mean algorithms. In other cases the results of both refinement filters is really similar.

A possible cause for this behaviour is that the image matting makes more use of the guide image, the raw input in this case, to enhance the estimation. As a consequence, the estimation is not as important as in the guided filter. However, if the initial estimation is good the guided filter obtains more benefit from it.

The authors of MDCP suggest the refinement step is not needed due the nature of the median operation. The results suggest that this is true as median and mean priors are the ones with the smaller difference between not filtering and using image matting or guided filter. Nevertheless, the use of these techniques still reduced the error in the conducted experiments.

Figure 6.11 shows the correlation results for the various implemented priors. As can be seen, image matting is the best filtering option, obtaining around a 10 % increase in most cases and much more when the initial coarse estimation was bad. Guided filter performance increment follows the same pattern, but it is slightly smaller, around 7.5 % increase in most cases.

As happened with the ARMSE metric, the most interesting thing about this step is that even bad initial estimations, such as the random prior that has 0.26 correlation before the refinement, ends with a good result, 0.65 in the case of matting and 0.62 in the case of guided filter. Probably, this is the reason why random choice is getting a good result, the filtering step is greatly enhancing the results, using the information in the raw image.

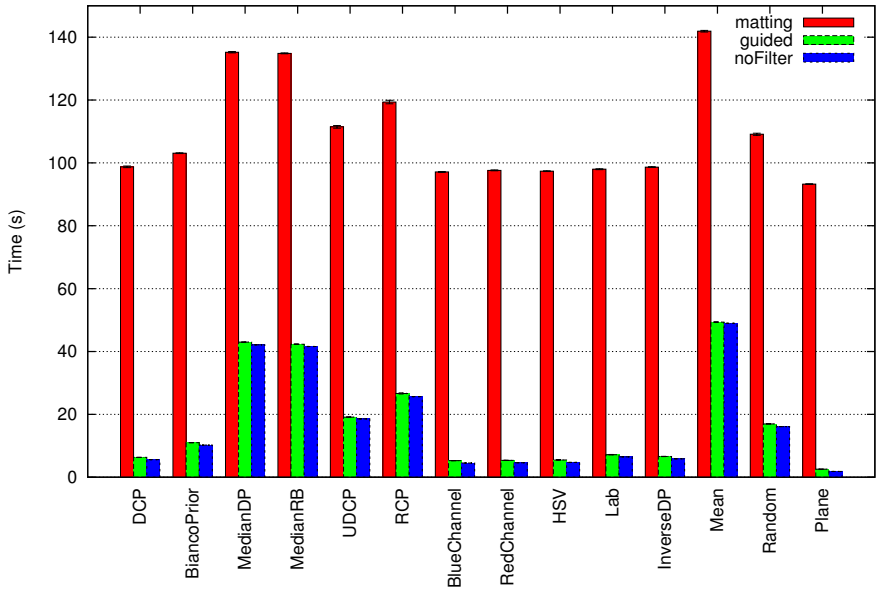

Figura 6.12: Filtering step computation time comparison for different DCP variations.

The main difference between this and the previous metric is the image matting seems to perform better in all cases. The performance difference is still higher when the coarse estimation is not good, but guided filter does not outperform image matting in any case. On the other hand, median and mean priors get less benefit from the refinement step due to the nature of the operation that already avoids halos in the borders as happened with the ARMSE metric.

These last two figures show the need for a refinement step that makes use of the raw image to increase the precision of the estimation. It is possible to see that even in cases where the coarse estimation would be considered a bad result, these techniques are able to refine it and finally obtain a fairly good result. As a consequence, this is one of the most important steps in the dark channel prior dehazing scheme.

As discussed before, the image matting procedure is extremely slow in terms of computation. Consequently, it is difficult to add it to a dehazing pipeline in real time that requires short processing times in order to be able to control the vehicle. Furthermore, the image matting cpu and memory complexity depends on the image size, making it impossible to use it on the raw images from modern cameras.

For this reason, the execution time has also been measured for the three refinement alternatives in each implemented prior. The computing time results can be seen in figure 6.12. The machine used for this experiment is a Intel Core i5 650 at 3.2Ghz with 10GB of DDR3 RAM using Matlab. The graph shows the processing time per image for each refinement option and algorithm.

As the results show, the difference between matting and guided filtering is huge,

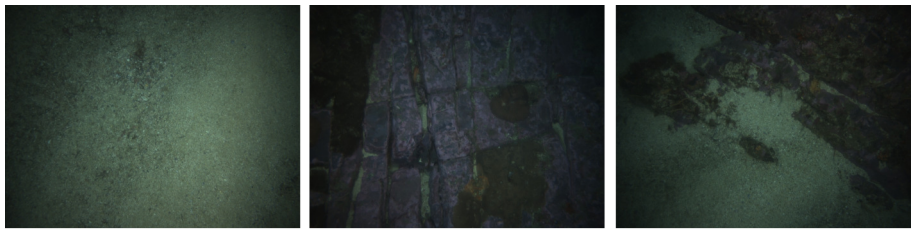

Figura 6.13: Different groups of images in texture dataset.

in fact guided filter only took 0.8 seconds more per image than no filtering at all. Thus there is no doubt that the guided filter is a good alternative in terms of processing time. Furthermore, the required time for the guided filter increases linearly with the number of pixels while the image matting is a squared progression.

About the different prior implementations, it is noticeable that although median and mean performed better without filter, they require more than 40 seconds to run while other alternatives such as DCP, blue channel or red channel only require 5 seconds. UDCP, RCP and random are in the middle of them requiring around 20 seconds to compute.

The results show that image matting is not feasible in real time applications. Although guided filter obtains slightly worse results in the correlation metric, it is the only option that mixes good performance and execution time, so it is the best choice for the case of enhancing AUV interventions. With reference to the different priors, median and mean offer good results but also a higher computation time, thus DCP or red channel seem to be better options for a real time application.

6.4.4. Texture benchmarking

So far, experiments show that using DCP or some of its variants is feasible for AUV imagery in terms of precision and computing time. But the dataset used for validation is the best case scenario as it is only formed by rocks with a uniform texture. In the following experiment the rocks and sand dataset, is used to test the algorithms in a more challenging environment.

In order to understand the performance in a different environment, the dataset was manually tagged into three groups of images, that can be seen in figure 6.13, depending on the objects appearing in it. The first group, sand, is made of images that do not show plants or rocks but only sand. The second group, rocks, is formed by rocky images similar to the ones in the rocks dataset, but with a different color scheme corresponding to the location of this dataset. Finally, the rest of the images were included in a mixed group of images that contain rocks and sand textures.

In this experiment, the number of compared techniques has been reduced to 7, discarding the techniques used to validate the results (plane, inverse DCP and random) and the ones with bad performance in the first dataset. The DCP variants tested are DCP, Bianco, Mean, MedianRB, UDCP, RCP and only red channel. Furthermore, the results shown are limited only to the use of image matting as it was the configuration that produced better results, although other configurations

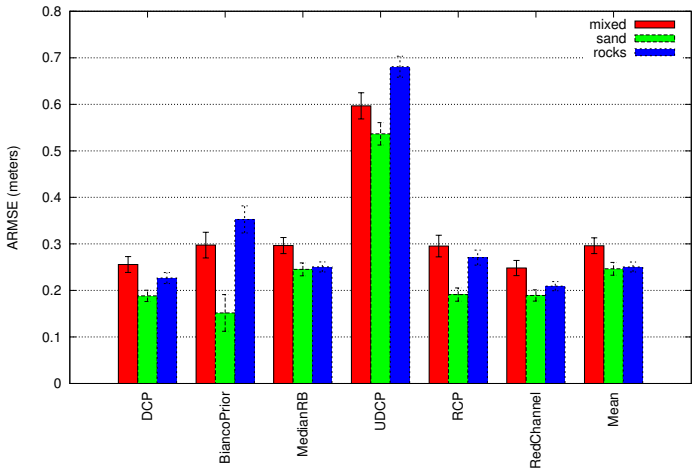

Figura 6.14: DCP variations results using ARMSE metric for rocks and sand dataset.

were tested.

Results for the ARMSE metric are shown in figure 6.14. As in the previous cases, the mean error for each group of images is shown in a bar and the error bars correspond to the 95 % confidence interval. The red bars show the error for the images with more than one texture, the ones with only sand are shown in green and images with only rocks in blue color.

As can be seen, differences between the different group of images are significant. All the compared algorithms obtain better results when images only contain a kind of texture, except the Bianco and RCP priors for rocky images. This is caused by the special treatment of the red color in these priors that negate its value together with the reddish color of the rocks.

These two DCP variants expect the red color will attenuate at an extremely fast rate, however the rocks in this dataset have a strong purple color. As the assumption is not true for this group of images, the error in this cases is higher than in other priors that do not rely on this assumption.

In the rest of the tested variants, the group of images that contains more than one texture obtains a significantly higher error. The best results are for the sand images, that are the ones with a more uniform texture that makes easier to detect changes in the amount of haze due to its uniformity. The rocks dataset error is smaller than the one of the mixed set as it contains only one texture, but still higher than the sand because the texture has more color variation.

This behaviour can be explained if it is assumed that there is a constant attenuation $b(\lambda)$ independent from wavelength in the transmission estimation, equation 6.8. This approach does not consider that the light attenuates faster depending on its wavelength, so objects with different colors, or textures in this case, should be treated differently. This makes it difficult to achieve a good result in the underwater environment, although the system is still able to produce a decent result.

About the individual performance, DCP and only red channel are the best

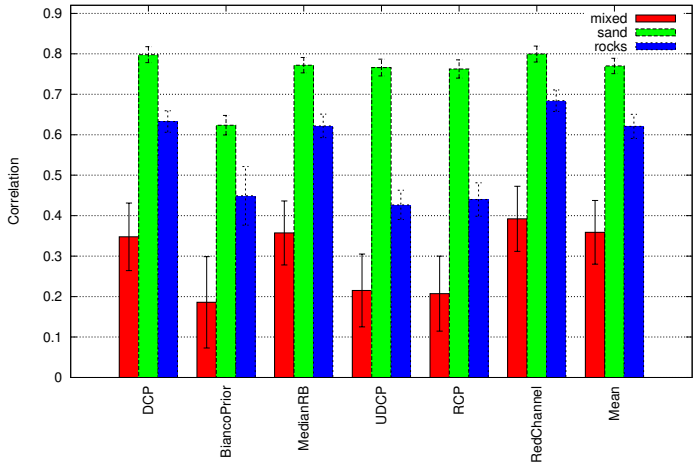

Figura 6.15: DCP variations results using correlation metric for rocks and sand dataset.

option, obtaining around 25 centimeters of error in the mixed group of images and close to 20 in the sand and rocks images. However, as this dataset has been taken from a shorter distance, it translates to a 10 % relative error in mixed and close to 8.5 % in the sand and rocks.

Median and Mean priors achieve a similar result, around 10 % relative error in the rock and sand groups of images and 13 % in the mixed dataset. Bianco prior obtains the best result for sand images, but it does not perform as good in the other group of images. UDCP results are extremely bad for this dataset, probably due to its use of the red channel.

The results for the correlation, showed in figure 6.15, are an even stronger proof of this fact. Although, in the ARMSE metric the compared algorithms are still able to achieve a good result for the mixed group of images, in the case of correlation this does not happen.

The results for the complete dataset show weak to no linear correlation. This means that the DCP algorithms were not able to extract a good transmission estimation for them, as it should be linearly correlated to it. If the estimation and the depthmap are represented as surfaces they will have a different shape.

In the case of only sand images, they show a strong relationship and rocky images a moderate to strong relationship. This proves the previously stated hypothesis: when the image textures are more uniform it is easier to extract a transmission estimation. This is a logical consequence because if the texture image is a plain color the only color changes would be caused by the haze, thus making the transmission estimation easier.

The confidence intervals show the metric is stable in the case of the single textured group of images, while the confidence interval is huge in the mixed group of images. This means the algorithms are not robust in this environment, some images will be correctly estimated and some others will not.

According to the correlation metric, DCP and only red channel achieve the

best results for every group of images. On the other hand, Bianco prior is the one which performs worst in this experiment. Even RCP and bianco priors present this behaviour, although their treatment of the red channel is probably not the best for red rocks.

6.5. Discussion and conclusions

Although enhancing underwater images for autonomous underwater robots is a highly interesting feature, real time single image dehazing is still a challenging task. Several dehazing alternatives have been reviewed in this chapter, analysing the feasibility of using them as a preprocess step for vision algorithms in underwater robotics.

These dehazing frameworks have been classified depending on their hardware and software needs, selecting those suitable for the use in a real time application. Interestingly, the more suitable family of algorithms are based in the dark channel prior. Motivated by the lack of objective benchmarking, different experiments have been conducted to analyse the performance of these kind of solutions in real AUV imagery.

In order to do so, a complete set of datasets have been used, formed by the raw images in different locations, to cover a wide range of textures such as rocks, sand, kelp or corals. Additionally, a depthmap from a stereo camera linked to these images is also included so that it is possible to know the depth of most of the pixels in the image. This depthmap has been used as groundtruth for an objective evaluation, given that the estimated transmission in the DCP algorithms should be linearly related to it.

The metrics proposed to evaluate the approaches include an adjusted root mean squared error and the pearson correlation. These metrics are able to measure the performance of the DCP based algorithms, exploiting the linear relation between transmission and depthmap. Using this experimental setup four different experiments have been carried out to test different parts of the methodology.

In the first of them, several different alternatives are compared as described in the bibliography, concluding the DCP based solutions are feasible for underwater image dehazing. In the second experiment, the ambient light estimation is benchmarked showing it is important to determine the amount of haze in an image.

The third experiment focuses in the refinement step, and the use of guided filter as an alternative to run in real time. The conclusion of this experiment is that image matting is too slow for use in a robot, as images need about 90 seconds to be processed using this method. However, the refinement step has a big impact on the final performance. For this reason the use of a compromise solution is suggested, using a guided filter the performance of which is slightly worse than image matting but can be computed in 0.8 seconds.

The last experiment tests the most promising DCP alternatives in three groups of images containing different objects. The results show the DCP based algorithms have difficulties dealing with multitextured images reducing its performance. When the image contains only a uniform texture, the DCP variants are able to correctly estimate the transmission. However, images that contain multiple objects with different textures obtain worse results due to the increased difficulty.

Taking this into account, it can be concluded that dark channel prior dehazing is a good alternative for single image dehazing. However, there is room for improvement when the images show different textures. As a consequence the following chapter focuses on this problem and proposes a neural network based solution.

Capítulo 7

Deep learning for single image dehazing

Deep learning, presented in [LeCun et al., 2015], is an extremely fast growing methodology that has been reintroduced into many different contexts in recent years. Deep learning is a machine learning technology, based on Artificial Neural Networks (ANNs), obtained by composing simple but non-linear layers that transform the data so that it is represented in a more abstract way. Using this transformation compositions, it is possible to learn very complex functions from raw data and obtain a higher-level interpretation of that data. This chapter is devoted to the use of this technology in a single image dehazing problem.

Although the first ANN dates from 1943, when Walter Pitts modelled it using electrical circuits in [McCulloch and Pitts, 1943], the training algorithms known at that moment made training multilayer networks impossible, limiting the amount of applications. In the 1980's with the discovery of the back propagation algorithm, a method which can be used to automatically train a neural network used in [Williams and Hinton, 1986], the ANNs had a rebirth.

However, the multi layered back propagation network required too much iterations for the computers at that moment. Even though, some problems were "solved" using ANNs such as recognizing handwritten digits in [LeCun et al., 1989]. This back propagation enthusiasm continued through the 1990's, but the required computation and the lack of promising results discouraged researchers who started using other kinds of machine learning such as Support Vector Machines (SVM) or Random Forests (RF).

Finally, in 2006-2010 with the increase of pure computational power and a rebranding to deep learning, instead of the already forgotten ANNs, the field underwent a revolution. The use of Graphic Processing Units (GPU) made learning 70 times faster due to the parallel computing of millions of parameters. This allowed to bigger networks to be trained, and this in turn achieved results far beyond other state of the art approaches. Additionally, deep learning also benefits from another trend in computing: Big Data [McAfee et al., 2012]. This commonly used term refers to the huge amount of data available to train neural networks that helps to generalize the training, avoiding to learn specific samples of the data.

In Chapter 6 several underwater image dehazing algorithms are reviewed and compared in the context of autonomous intervention. However, most of the algorithms capable of restoring the image require a huge amount of data in order to estimate the parameters required to dehaze the image. On the other hand, neural networks are proven to be good at generalizing complex problems and learning complex functions to transform data. Furthermore, deep learning approaches have achieved good results in similar problems such as colorizing black and white images in [Iizuka et al., 2016] or defogging images for autonomous driving in [Hussain and Jeong, 2015].

For these reasons, in this chapter two deep learning approaches are proposed to dehaze underwater images from a still image input. Additionally, once trained the neural network estimations are produced quickly, allowing this methodology to be used in a real system to enhance other vision algorithms that may lead to more robust autonomous robots.

7.1. The fundamentals of Deep learning

All the deep learning methods are based on a common concept: find patterns and model high level abstractions in data. The term "Deep learningïs a rebranding for artificial neural networks inside the machine learning field. The goal of this methodology is to replace handcrafted algorithms with efficient solutions learned from raw data.

The main motivation for this, is the difficulty of characterizing every single case in problems where the input is not restricted. A classic example that reflects this situation is the handwritten text recognition problem. Several non learning approaches have been proposed to recognize handwritten text through time, but it is necessary for every possible situation where a character or letter is to be recognised to be explicitly expressed. On the other hand, neural network solutions are able to outperform classical approaches without any prior knowledge about the language used, and nowadays are achieving performances comparable to human efforts.

In order to do so, neural networks take a large amount of handwritten text images, known as a training set. A system which is able to learn from it and infer rules is then used so that characters and words can be automatically recognised. Furthermore, increasing the number of examples increases the accuracy, as the system is able to generalize and generate even more precise rules.

7.1.1. Architecture

The data can be represented in many different ways depending on its nature and the purpose of the neural network. Some representations may be more effective than others, for instance image representations work better using matrices than vectors, because matrices keep the logical structure better. This input can be labelled as in the case of supervised learning, attaching information related to the solution of the problem to the dataset, or not in which case is known as unsupervised learning.

In this thesis, the proposed solutions are based on supervised learning. This means that besides the input, a "label" that contains the desired output is also

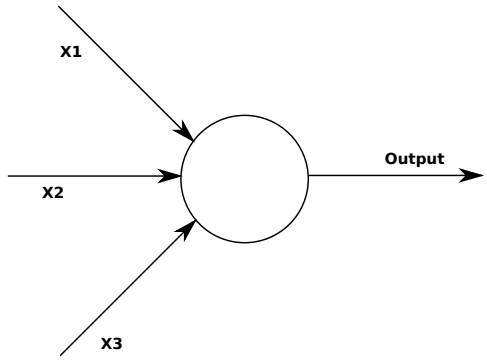

Figura 7.1: Basic representation of a perceptron.

needed in the learning process. This information will be used to correctly train the neural network, so that it produces the desired output for each input, and also test the system once trained. In the case of unsupervised learning, the neural network tries to find patterns or statistically relevant information.

Another key characteristic of deep learning is the layered structure, from which the word "deep" is derived. The neural networks are formed by small units simulating real neurons called perceptrons, usually represented as figure 7.1 shows. A perceptron is a small processing unit that takes several inputs and produces a single output. Different functions can be coded into these units, a basic one is a weighted sum of the inputs x plus a bias b and a threshold value to decide a binary output as can be seen on equation 7.1.

$$output = \begin{cases} 0, & \text{if } \sum_j w_j x_j + b \leq threshold \\ 1, & \text{if } \sum_j w_j x_j + b > threshold \end{cases} \tag{7.1}$$

The weights, bias and threshold in the perceptron will decide the function in it. As an example, the decision to take an umbrella when going to work can be expressed in an artificial neuron depending on three inputs: is it raining, will it rain when I come back according to forecast and I will drive to the office. Depending on the weights assigned to each input, the bias and the threshold the answer to the problem will change accordingly. For instance, assigning a negative weight to the car input will result in a decrease in the possibilities of taking an umbrella while a big number on the raining weight will increase them.

However, a single perceptron is not able to model a complex system, in fact it is only able to model a single decision. For this reason, perceptrons are usually combined forming totally connected layers from the inputs to the outputs as figure 7.2 shows. The first layer is commonly known as input layer, as it has the raw inputs of the problem, and it needs as many neurons as the input of the problem. Similarly, the last layer is the output layer requiring the same outputs as the expected output. All the layers between these two are called hidden layers, and may have any number of perceptrons.

The real power of this methodology lies in the fact that, using this architecture and a different number of hidden layers and units in them, it is possible to create

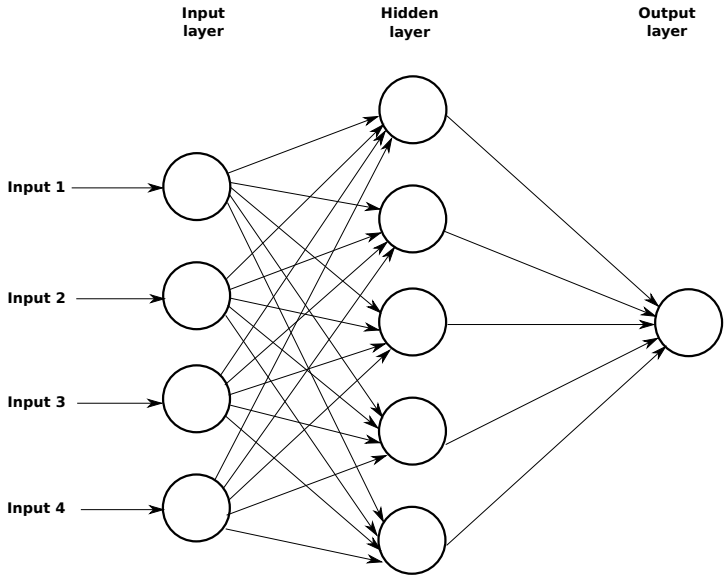

Figura 7.2: Basic representation of a neural network.

an architecture capable of expressing a huge number of complex functions. Consequently, the next tool required is a way to automatically decide the weights of the network so that it chooses the right function to solve a specific problem. This process is known as learning because it uses examples or raw data to automatically estimate the optimal weights of the network for them.

For this purpose, the available input data is usually organized in two sets. The first set is the training dataset that will be used to obtain the best possible network weights. The second set, test dataset, is used to measure the performance and decide the best network architecture, such as the optimal number of layers or neurons. The results obtained are always referenced to this test dataset, ensuring that the neural network has not learned specific details of the data, a situation known as overfitting. Furthermore, when there is enough data a third validation dataset that will only be used in the last evaluation of the neural network is created, thus avoiding overfitting the system unintentionally through different training iterations within the test dataset.

7.1.2. Automatic learning

The next step in the training process is to define a function to quantify how well the neural network is achieving the goal, known as cost or loss function. In the case of supervised learning it usually measures the distance of the neural network result with respect to the "label" or groundtruth. Unsupervised learning cost functions focus on measuring desired characteristics of the output.

Nevertheless, the used function depends on the problem and different functions may change the training result, thus is an important design decision. For instance, it is advisable to use smooth cost functions that reflect the consequences of small changes in the weights rather than functions that require big changes in order to

see a change in the result.

Now, an algorithm is needed to compute the neural network parameters in a way that minimizes the cost function for the training data. A possible way to solve the problem is using calculus to analytically find the minimum. However, neural network cost functions often depend on billions of parameters, making it impossible to use calculus to minimize it. For this reason, the most commonly used technique for this is **gradient descent** and its different variants or optimizations.

Basically, what gradient descent does is to initialize randomly all the weights in the neural network and to modify them in the direction that minimizes the cost function through iteration. It is possible to imagine this as finding the minimum of a valley, the deepest point, starting at any point and moving down through the steepest slope. But in the case of a neural network, instead of a 3D valley the space may have millions of dimensions.

$$\Delta C \approx \nabla C \cdot \Delta \vec{w} = \frac{\partial C}{\partial w_1}\Delta w_1 + \frac{\partial C}{\partial w_2}\Delta w_2 + ... + \frac{\partial C}{\partial w_n}\Delta w_n \tag{7.2}$$

Equation 7.2 express the cost variation ΔC depending on the weight change $\Delta \vec{w}$. What gradient descent does is choose the right weights $\vec{w}$ so as to make the cost variation ΔC negative. The gradient vector ∇C can be seen as an operation that relates changes in the weights with changes in C.

As a consequence, using equation 7.3 it is possible to obtain the right weights to make the cost function negative. The η is a small positive value, commonly referred to as learning rate, that controls the "step" moved in the minimization direction. In other words, the gradient indicates the direction of the movement and the learning rate the amount of movement, so that this will be another parameter which has to be chosen in the neural network training.

$$\Delta \vec{w} = -\eta \nabla C \tag{7.3}$$

Using this, it is just necessary to iteratively compute new gradients for the current weights and update them with the new values that are obtained. This algorithm will constantly decrease the cost function until it reaches a minimum.

It is possible to prove that this weight computation always decreases the cost function by substituting it in equation 7.2. This leads to equation 7.4, where $\|\nabla C\|^2 \geq 0$ guaranteeing $\Delta C \leq 0$ due to the previous definition of η, small positive value. This is true for the limits of the approximation.

$$\Delta C \approx -\eta \nabla C \cdot \nabla C = -\eta \|\nabla C\|^2 \tag{7.4}$$

The limits of this approximation are mainly dependant on the learning rate. If the learning rate has a high value, the computed gradients may not be a good approximation with the result that the cost function variation will increase. However, choosing a small learning rate will require too many iterations, making the optimization too slow.

Several research works try to deal with this problem computing second order derivatives, that describe the variation of the gradient, in order to automatically obtain the learning rate that it is better at each moment of the training. Although the automatic computation of the learning rate is a great advantage, the time consumed by this computation can be an important drawback.

Adding this to an already computationally complex gradient descent may be impossible. It is important to remember the gradient computation for the cost function requires the training data as parameter, with labels in the case of supervised learning. Consequently, if the training set is big, it requires a lot of time to compute all the gradients required to update the network weights. This step may be needed thousands or even hundreds of thousands times until the network is trained and reaches a minimum, making the training stage extremely slow.

A simple optimization to handle the slow computation time drawback is the **stochastic gradient descent**. In this case, instead of using the whole training dataset a small subset of randomly selected samples, usually known as batch or minibatch, is used. This is a fairly good optimization, as the gradient direction may not be completely accurate, but as the algorithm is already taking small steps towards the minimum it is not important. In the long run it may require a higher number of iterations through the whole training set, known as epochs, but the faster computation of the gradients make it worthwhile.

It is important to remember this method finds a local minimum, where the gradient vanishes. The search for an absolute minimum in a reasonable period of time is still an active research field. However, this problem can be off set by initiating the weights randomly several times and choosing the best performing trained network. It may be still a local minimum, but the chances of finding a minimum close to the optimal increase.

Finally, the meaning of these discovered weights is hard to interpret. Being able to understand it could help to enhance the software, leading to better performance. Unfortunately, in most cases how the neural network is capable of producing a good result is still unknown. Several researchers are working on understanding how the artificial intelligence of a neural network works. Although some advances have been recently presented such as [Yosinski et al., 2015] this is still an active research field.

7.1.3. Computing gradient descent: backpropagation

To compute the previously presented optimization requires a gradient ∇C that describes the changes in the cost function depending on the changes in the weights. The most common method to obtain it in a neural network is the backpropagation algorithm introduced in the 1970s but popularized after the presentation in [Williams and Hinton, 1986].

Unfortunately, it is not possible to directly apply backpropagation to any cost function, the cost function must meet two requirements. It must be possible to compute it for a single training sample and average the results of them obtaining an average cost, thus it is possible to obtain the gradients from them. Secondly, it must be a function of the output from the neural network, and eventually the desired output of the sample.

Taking into account the previous description of a perceptron, its output is the result of an activation function that will be noted as σ. This activation function is computed from the weights of the l layer w^l, and the activations of the neurons of the previous layer a^{l-1} plus the bias b^l. The equation 7.5 express this relation in a vectorized form, note the weights and activation are vectors.

$$a^l = \sigma\left(w^l \cdot a^{l-1} + b^l\right) \tag{7.5}$$

Considering this, the error in the output layer OL, ϵ^{OL} can be obtained using the derivative of the activation expression for each neuron activation function. The equation 7.6 shows the error for the j activation function of the output layer. Similarly, a vector expression can be derived to obtain an error vector ϵ^{OL} that covers all the neurons in the output layer.

$$\epsilon_j^{OL} = \frac{\partial C}{\partial a_j^{OL}} \sigma' \left(w^{OL} \cdot a^{OL-1} + b^{OL} \right) \tag{7.6}$$

In this expression, the first term corresponding to partial derivatives, measure the influence of the neuron j in the cost function. On the other hand, the second term controls how fast the activation function is changing. The multiplication of both terms produces an indicator of the part of the error that the neuron is responsible for in the cost function.

Both terms should be easy and fast to compute. Specially the second term that is partially known in the direct computation of the neuron. The first term depends on the cost function used as the partial derivatives are required.

Using this cost error equation it is possible to obtain the cost for each neuron in the output layer. The next step consists in propagating this error backwards, backpropagation, obtaining an error for each neuron in the network. In order to obtain the errors in the layer l from the next layer it is possible to use equation 7.7.

$$\epsilon^l = \left(\left(w^{l+1} \right)^T \epsilon^{l+1} \right) * \sigma' \left(w^l \cdot a^{l-1} + b^l \right) \tag{7.7}$$

As happened with the error in the output layer expression in this case the first term is a measure for the importance of this neuron in the next layer. The second term is equal to the one in equation 7.6, indicating the weight of the activation function of the neuron. As can be seen the transposed matrix is actually moving backward the error of $l + 1$ layer to the layer l.

Using this last two expressions it is possible to compute the error ϵ^l for all the neurons in every layer in the network. Starting from the cost function the error is backpropagated layer by layer until the input layer. However, the rate of change of the cost with respect to biases and weights is still needed.

In order to obtain the error, equation 7.8 shows the error computed for a neuron j is actually the bias rate needed for this neuron j. So using the backpropagation means it can be computed directly and be able to train the biases of the neural network.

$$\frac{\partial C}{\partial b_j^l} = \epsilon_j^l \tag{7.8}$$

This computation is logical as the bias of the neuron does not depend on any weight coming from earlier stages. As a consequence, its learning rate is dependant on the error produced in this neuron only, not from earlier stages of the neural network.

Finally, the gradient of the cost with respect to a weight of the neuron k in layer $l - 1$ to neuron j in the l layer can be computed using equation 7.9. In this case, only the activation function of the neuron from which the weight comes is required besides the neuron error.

$$\frac{\partial C}{\partial w_j k^l} = a_k^{l-1} \epsilon_j^l \tag{7.9}$$

An interesting consequence is that if the activation is close to 0, the weight will learn slowly. This makes sense, because if the previous neuron is not activated the final impact of this weight in the result is small. It can only be compensated by a high error coming from the neuron that would require big changes in the weights that get to this neuron.

These four equations described in this section can be derived from the chain rule from multivariable calculus. Further details about this can be consulted in the original work [Williams and Hinton, 1986] or in the online resource of [Nielsen, 2015], where most of the information summarized in this section was extracted from.

Another important characteristic of backpropagation is that it allows all the weight derivatives to be computed with just a forward and a backward pass through the network. Other methods require a single pass for each weight, consuming a lot of time and making it impossible to train big networks. The backpropagation method does a smart computation, sharing results so it can be done in just one pass, and permitting its use in bigger networks.

Using this knowledge it is possible to calculate the gradients needed for the gradient descent optimization in a neural network. The equations show a sigmoid activation function and fully connected layers, however other activation functions and different network architectures may be used changing the equations accordingly.

7.1.4. Convolutional networks

Since the introduction of backpropagation, several network architectures have been designed and used in different problems. The recurrent neural networks family, described in [Karpathy et al., 2015], are used for sequential data such as machine translation [Sutskever et al., 2014], speech recognition [Graves et al., 2013] or image captioning [Karpathy and Fei-Fei, 2015]. Another popular architecture is the autoencoder, capable of reducing the dimensionality of data, commonly known as compressing and decompressing, using a neural network as presented in [Hinton and Salakhutdinov, 2006].

However, when dealing with image inputs, the best performing, and probably the most popular architecture, is the biologically inspired convolutional network described in [LeCun et al., 2010]. These types of networks are able to detect features hierarchically and are invariant to position changes. This makes it possible to abstract from position details and focus on recognizing objects and forms in the input images such as lines, circles in the first layers or animals, faces and objects in the last layers. The most popular application is image classification [Krizhevsky et al., 2012], but any problem based on image processing such as face detection [Farfade et al., 2015] or image upscaling [Dong et al., 2014] is suitable for this architecture.

For this reason, in the underwater image dehazing application proposed in this work, two different convolutional neural networks are used in order to solve the problem. The main advantage of these networks is they actually take advantage

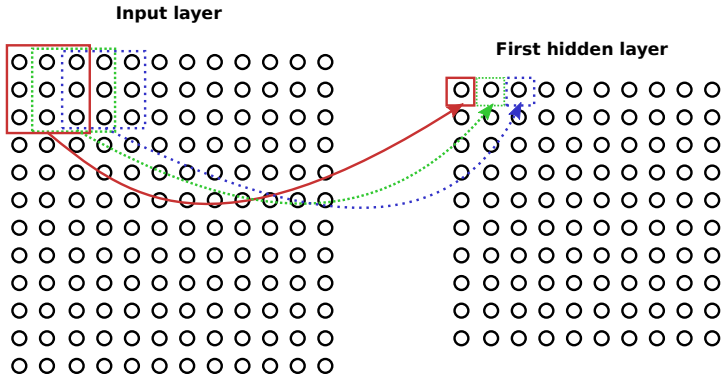

Figura 7.3: Connections of the input of a convolutional network with the next layer.

of the spatial structure of the images. The convolutional network does the same process for every pixel independently of its location in the image.

The main difference with regular neural networks, known as Multi Layer Perceptron (MLP), is the connection between neurons. In a convolutional network, the neurons are organized as matrices whose value corresponds to the pixel intensities in the input. In the case of MLP, each neuron was connected to every other neuron in the next layer. In contrast, in a convolutional network only the local receptive field, also called kernel, is connected to the next layer. This local field is a small region of neurons around the center of the corresponding neuron in the next layer.

An example of these connections can be seen in figure 7.3. The neurons in the top left red square with continuous lines of the input layer are connected with the top left neuron of the first hidden layer. Then the receptive field window is moved to the right, and the neurons in the green dashed square are connected to the second neuron of the first hidden layer. Similarly the third neuron connections are depicted with a blue double dashed square.

As happened in the MLP, each neuron in the receptive field of the input layer will be multiplied by a weight and added to the result together with a bias. The result of this operation will be the input to an activation function that will decide if the output neuron should be activated.

The example shows a window size of 3x3 neurons that will be connected to the next layer. However, bigger sizes are used as 5x5, 7x7. The size of this receptive field causes the output to be smaller, in this case the input has 12x12 neurons and the first hidden layer has 10x10. If the window size is bigger the next layer size will be even smaller. In order not to reduce the size of the network, it is possible to use zero padding in the margins, that is by adding fake neurons with 0 value.

Another important parameter is the stride length, which defines how many neurons the receptive field moves each time. In this case a stride of one has been used, in the case of using stride 2 the first hidden layer will be half its size and the green dashed neuron will not exist.

The second difference with this method and MLP is the weights and bias in the receptive field are shared for each of the hidden layer neurons. For this reason, all the neurons in the first hidden layer are detecting the same feature, and consequently

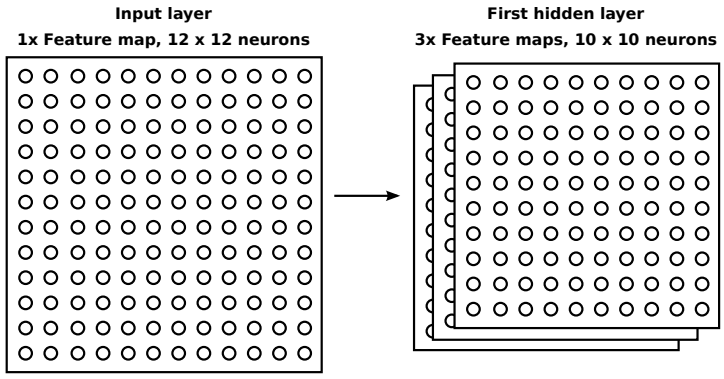

Figura 7.4: Convolutional layer producing 3 feature maps.

called features instead of layers. For example, a receptive field able to find horizontal lines will find it in all the image independently of its position.

Consequently, several features will be needed in each layer in order to detect different characteristics of the image, as figure 7.4 shows. Each feature map will be defined by a shared receptive field of weights and a bias. That is the reason why each layer of a convolutional neural network is formed by multiple feature matrices. It should also pointed out that the input layer is commonly formed by three features of the image, the red, green and blue channels.

The weight sharing has another important characteristic, the number of parameters of the network is greatly reduced. As each feature map only needs a receptive field to produce an input size layer of resulting neurons, the number of parameters is small compared with an MLP. This will make training faster and allow deeper models to be created assuming the translation invariance of the model.

The last difference between MLP and convolutional networks that is relevant for this work, is that of the the pooling layers. In addition to the already described convolutional layers, convolutional neural networks also use pooling layers. These layers reduce the amount of information in the layer, retrieving a summary of the activated neurons. The resulting network is consequently reduced in size depending on the pooling receptive field. For instance, a 2x2 pooling will halve the feature map size.

These types of layers are typically used after a convolution to reduce the information, the main idea behind them is, once a feature has been found the exact position is not important. So it is a good idea to reduce the information, keeping only that which is most relevant at each moment. There are different pooling functions, but the most common is max pooling that retrieves the maximum of the activated neurons in the receptive field.

$$\widetilde{t}(x) = 1 - \min_{y \epsilon \Omega(x)} \left(\min_{c} \frac{E^c(y)}{A^c} \right) \tag{7.10}$$

It is interesting to compare the convolution operations to the dark channel prior methodology described in 6.2.1. The transmission $\widetilde{t}(x)$ estimation of the DCP can be expressed as equation 7.10 shows. It basically searches for the minimum intensity channel in the window around the pixel $E^c(y)$ and divides it by the airlight A^c.

On the other hand, in a convolutional network it is possible to express it with a convolution followed by a pooling. The convolution operation can be modelled as equation 7.11, where WS is the window size of the receptive field, w and b the weights and bias of the convolution kernel and x the pixel input. If the convolution weights are set to 0 for every pixel but the central, that has a $-1/A^c$, the max pooling naturally returns the negative DCP.

$$Conv(x) = \sigma\left(\sum_{i=1}^{WS}\sum_{j=1}^{WS}(x_{i,j}w_{i,j}) + b\right) \tag{7.11}$$

As can be seen, both operations are really similar as they iterate in a reduced window around the pixel to detect a feature, and then reduce the information using a maximization. Similarly, every variation of the dark prior described in equation 6.2.1 can be expressed using convolutions and poolings easily, except the median.

With reference to the image matting step, it is not possible to directly express it in a convolution. However, the alternative to it, guided filter, can be expressed in a few convolutional and pooling layers. Thus, the DCP could be easily implemented with convolutional operations, this suggests that this is a good architecture for an image dehazing solution.

Furthermore, many other haze feature based methods such as the contrast operations can also be naturally modelled with convolutions. For this reason, it seems a natural step to let a convolutional network automatically learn the best features for haze detection and restore the degraded image. If the DCP is the best option, the neural network should discover it and use it in the dehazing scheme. What is more, if the best option is a combination of different haze detectors, the neural network should be able to learn it.

7.2. State of the art

The growth of available data in computers for processing [McAfee et al., 2012], combined with the increasing processing capabilities of computers, initiated the deep learning revolution. Although applications are not restricted to image processing tasks, this is the domain that has seen the biggest change in response to the introduction of the deep learning methods. Classic problems that have seen little progress for years have been solved, as explained in [LeCun et al., 2015], and new applications are continually being proposed in other contexts such as the discovery of new pharmaceutical products [Ma et al., 2015] or even discover exotic physics particles [Baldi et al., 2014].

In the case of learning approaches for image dehazing, only a few papers have been published and none of them, to the best of the author's knowledge, are tested in underwater environments. In [Cai et al., 2016] the authors propose a deep learning solution to remove haze from outdoor air free images. The first step of this approach is to use a convolutional neural network to estimate the medium transmission, $\widetilde{t}$ of equation 6.8.

Due to the difficulty of obtaining good pairs of hazy and haze free images the authors use synthetically generated images to train and validate the network. A

medium transmission is generated using the image formation model. In a second step, using the medium transmission predicted by the neural network the images are dehazed using the image formation model. The main drawback of this method is the use of synthetic images that often ignore many of the problems of real images.

However, the results compared with other methods in real images show the neural network is able to learn from the synthetic images obtaining reasonably good results. Furthermore, the required computation time is smaller than other approaches due to the parallelism of neural networks.

In [Hussain and Jeong, 2015], the authors propose a classical MLP network to directly produce a dehazed image from a fogged input. The system is tested using synthetic gray scale images with and without fog. The application proposed in the paper is automatic dehazing and is designed to help drivers in foggy conditions.

As happened in the previous approach, the use of synthetic images may ignore many other problems present in real images that do not follow exactly the image formation model. Moreover, the validation only includes synthetic images of the same dataset and does not show real application results. Finally, the use of black and white images may be interesting for some cases but reduce the difficulty of the problem avoiding color correction problems.

Other learning approaches use random forests as in the case of [Tang et al., 2014]. This approach uses a list of haze feature detectors from other works, in particular dark channel prior, local max contrast, hue disparity and local max saturation, to use them in a random forest regression. The trained random forest decides from the input of these features the best transmission estimation to dehaze the image afterwards.

Once again, the difficulty of obtaining good training images makes authors to use synthetic images. In this case, a transmission map is artificially created together with a depthmap in order to train the random forest. An interesting conclusion of the paper is that the dark channel prior is the most relevant feature for predicting the transmission in images.

However, the random forest also relies on other techniques to refine the estimation and finally outperforms any of the initially used estimations. In some way, it is similar to a sensor fusion as it includes several approximations to produce one that is better than any single estimation. The work also shows results in real images, demonstrating that the training with synthetic images is valid.

The work in [Zhu et al., 2015] uses a slightly different learning approach. Firstly, a linear model that relates the depth of the scene with brightness and color saturation is derived from statistical information of hazy images. Then, this linear model is trained with synthetic images with artificial depthmaps in order to obtain an approximation of the correct parameters. Finally, the estimated depth is used in the image formation model to obtain a haze free image.

Besides the training with synthetic images, the approach relies on a depth estimation that may be incorrect. The system is tested with real images again showing it is possible to obtain good results after training with synthetic images.

The authors in [Mai et al., 2014] perform a similar computation estimating the depth of the image as an input to the dehazing step. In order to do so, the first step is a depth estimation from the colors of the image using a MLP neural network that takes an RGB pixel as input and produces a depth value. The depthmaps are artificially created to obtain a valid training set for the neural network. After that,

the depth information is used to dehaze the image through the image formation model.

The results show that the method actually dehazes images. However, the results of the depth estimation from a single RGB value are not validated and might be far from the real ones as they depend on the intensity of a single pixel. As a consequence, an image of a plane with different colors will have different depths.

The main drawback of these learning approaches arises from the fact that they use synthetic images created from non-hazy images to train a neural network that estimates transmission because of the difficulty of finding hazy and non-hazy pairs. These synthetic images ignore many problems of real images. Furthermore, none of them tries to use them in an underwater environment, as is proposed in this thesis.

As can be seen, the learning based dehazing approaches can be classified in two types of image processing described in the previous chapter: image restoration and enhancement. Some works try to estimate different parameters to restore the image such as the transmission or the depthmap. While others try to enhance the degraded image directly in order to create a more visually pleasing image.

To overcome the problem of synthetic images, many of these works mention the possibility of using a depthmap groundtruth in order to train the networks as transmission is linearly related to it. But obtaining a good depthmap in open air hazy images is also a challenging task given the distance to the targets.

There is a very active research line that focuses on obtaining a 3D representation of the environment due to the importance of this kind of information when interpreting a scene. This is the reason why there is so much prior work on sensors able to sense this kind of information such as stereo cameras, motion cameras or infrared camera/projector, popularly known as Kinect. Unfortunately, these sensors are not suitable for open air images, stereo cameras and motion cameras require the objects to be close, and infrared light projection is undetectable in sun light.

However, in the case of underwater imagery, where images need to be close to the seafloor so that interesting objects or features can be detected, it is possible to use stereo and motion cameras. This allows a training dataset with real depth information as groundtruth to be obtained so that a neural network capable of estimating it in new samples can be trained. The infrared projector is still not viable since the infrared light suffers a big attenuation due the water absorption.

Nevertheless, the depthmap information may still be impossible to retrieve in many situations, for example where the seafloor is not textured enough to obtain features that can be recovered, and even when the system is capable of doing this, the depthmap will still have gaps or zones without depth information. For these reasons, being able to estimate 3D from still images is a nice feature to have beyond the single image dehazing capability.

7.2.1. Depthmap estimation from a still image

As the depthmap is a key value in dehazing restoration techniques, it is interesting to review different alternatives in the literature to estimate it from a still image. Using this estimation of the depthmap, it will finally be possible to dehaze the image.

One of the first works to address this problem from a learning perpective was [Saxena et al., 2008], which uses a linear regression and a MRF for predicting the

depth in a set of images. The system starts by training with a set of monocular images, including indoor and outdoor environments, and their corresponding groundtruth depthmaps. The approach relies on different features to obtain relative and absolute depth, so that the MRF can model the relationship between this parts to obtain a global spatial structure.

The model was recently extended in [Saxena et al., 2009] to create and include a system for 3D model generation. However, the main drawback of this system is that it relies on the horizontal alignment of patches, and suffers a performance drop in less controlled settings.

The work in [Ladicky et al., 2014] simultaneously performs depth estimation and semantic labelling on the same dataset. It can be concluded that doing it at the same time benefits both processes improving the results, knowing the label helps to estimate the depth and vice versa. However, this method uses hand crafted features that may not be useful in other contexts.

In [Eigen et al., 2014] a deep convolutional network is used to estimate 3D depth in different images. The work introduces an interesting concept of coarse estimation that predicts a global scale and then refines it in a second iteration. The network is divided in two, the first coarse network predicts a global depth with 5 convolutional and pooling layers followed by two fully connected, non convolutional, layers. This produces an initial estimation at a small resolution that lacks details, but has a general idea of the structure of the scene.

The result of this network is concatenated in another one that uses the whole image in 3 convolutional and 3 pooling layers to obtain an output of 1/4 resolution. This second network refines the result of the previous one, increasing the resolution and recovering details lost in the first estimation.

Another interesting addition is a scale invariant mean squared error for the cost function in the training. Due to the nature of still images, it is impossible to obtain the correct scale from it unless there is an object with a known size. A proof of this is the fact that it is sometimes impossible to distinguish a photo of a dolls house from a real house. In order to solve this, the authors introduce a loss function that considers relative distances between pixel depths instead of absolute distances.

The results of this work outperform previous alternatives obtaining a 0.215 relative error in the test images. It may not seem an impressive result but it is important to consider previous approaches achieved 0.349 in the case of Make3D for the same dataset.

This work was extended in [Eigen and Fergus, 2015], maintaining the idea of coarse and fine estimation but adding a new scale network for even higher resolution, obtaining an output of half the input resolution. This new network receives the feature maps from the second fine estimation, scales them up and convolutes with the raw image to produce a higher resolution, more precise than the previous ones.

In addition, this work adds gradients to the scale invariant mean squared error. The aim of this is also to minimize the local structure of the image using the difference of the vertical and horizontal gradients of the estimation and groundtruth. Furthermore, the neural network is able to predict normals and per pixel semantic labeling besides depth.

The introduction of these enhancements, results in a performance increase in the depth prediction, reducing the relative error to 0.158 and an absolute error of

0.65 meters.

The authors in [Baig and Torresani, 2015] also use a coarse estimation followed by a refining step. But in this case a slightly different approach for refining the results is used while maintaining a convolutional deep network for coarse estimation. The local refinement regresses directly to pixel depth using the global estimate as a feature. Both steps are trained over a depth dictionary and a regression mapping simultaneously producing better results than separated learning.

An interesting feature of this approach is instead of using a scale invariant loss to avoid the scale prediction problems, the authors propose a zero mean depthmap. In order to obtain it, the original depthmap is preprocessed subtracting the mean depth map computed from the entire dataset. Thus the training predicts deviations from the mean depth map instead of absolute distances. After the prediction, the depth map is added to the prediction to generate the final output.

The results of this work are slightly worse than the previous network for the same dataset. But the output image sizes are bigger than the approach in [Eigen and Fergus, 2015] obtaining images closer to the input size.

The approach in [Liu et al., 2015] proposes a different alternative from the previous coarse and fine estimations. The images are segmented in superpixels containing image patches that will be processed to obtain the depth map. A convolutional neural network connected to a MLP takes these superpixels to produce a single depth value for it. Simultaneously, a fully connected network uses the superpixels similarities as input to produce a 1 dimensional similarities vector for each pair.

Both network outputs are taken by a conditional random field (CRF), a model for structured prediction, and finally predict the depth of the image. This approach obtains similar results to the coarse fine alternatives for outdoor and indoor images demonstrating that alternatives to this concept are able to obtain state of the art results.

As can be seen, none of the reviewed methods have been used in an underwater environment. To the author's knowledge there is no previous work using deep learning in this context, although other dehazing alternatives obtain this information as a side result. Although monocular images do not contain direct data on the distance to objects in the scene, within a constrained set of environmental parameters various visual cues, such as shading and variations in contrast caused by light moving through different distances in the water column, may provide information about scene depth.

Taking this into account, two deep learning solutions for underwater single image dehazing are proposed. The first is an end-to-end approach that directly dehazes the images from the raw input without considering the image formation model. In the second algorithm presented, the deep learning step estimates a depthmap to obtain a transmission estimation and finally dehaze the image using the image formation model.

These approaches have two important advantages with respect to other state of the art alternatives such as dark channel prior or image restoration alternatives. Due to the parallelism of neural networks, it is possible to run the algorithms fast enough to include them in a real time performing system, making it possible to use them in a intervention. The approach does not require any hardware or additional information, all that is required is the raw image that needs to be enhanced, thus

making possible to use it in any robotic platform.

7.3. Direct underwater dehazing using deep learning

The first approach to still underwater image dehazing is an image enhancement procedure that directly produces the result without taking into account the image formation model. A convolutional neural network is trained to produce visually pleasing images from raw degraded inputs. The main goal is to improve the performance of other vision algorithms such as trackers or grasp planning algorithms using this output instead of the acquired image.

However, the approach proposed is not completely an image enhancement method due to the learning stage. The groundtruth images used to train the network are introduced from a restoration technique described in [Bryson et al., 2016]. Here it is shown that even though the image formation model is not explicitly included in the methodology, the neural network may be able to learn the intrinsics of it from the samples used to train.

For this reason, the solution may not be completely labelled as image enhancement method. On the other hand, as discussed in the previous section it is difficult to understand exactly how the neural network produces the results. So it cannot fairly be classified as image restoration also, because it is not specifically using the image formation model or any parameter estimation.

Basically, the proposed approach is an image enhancement algorithm inferred from image restoration. Furthermore, it make the best of both worlds: single image as input and real time performance like image enhancement methods, while trying to obtain results as close as possible to image restoration techniques.

7.3.1. Image datasets

The image datasets to train and evaluate the system require a pair of raw and restored images. In this experiment, six datasets have been used to train, test and validate the results of the proposed approach. A detailed description of the dataset, as well as acquisition details, can be consulted in the Appendix A. As depth information is not required, all the images in the datasets can be used to train the system, including the kelp dataset where the depth information is not reliable due to its nature.

The images cover a wide variety of textures and objects at different depths in order to generalize the learned transformation, being able to work in new environments instead of learning fixed transformations for the trained textures. Sample images for each dataset can be seen in figure 7.5.

The images have been taken by an underwater camera mounted in an autonomous underwater vehicle [Williams et al., 2012] during different real underwater interventions in different locations. The images have been divided into 6 groups depending on the characteristics of the images and the location of the acquisition.

The dataset division allows the use of only some sets of images in the training stage and to keep others for the validation stage. Thus it is possible to use images from different interventions, locations and conditions for the validation, and test the system generalization. Besides this, each dataset has been grouped in a training

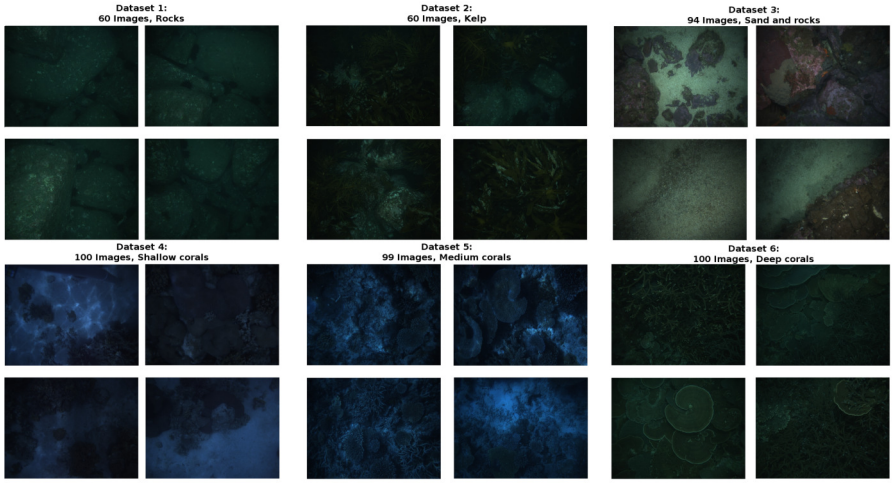

Figura 7.5: Images of the different datasets used in the direct underwater dehazing.

set and a testing set with images randomly selected to measure the performance in each dataset.

7.3.2. Neural network architecture

A convolutional neural network was the architecture chosen because of the advantages that these types of layers bring to the processing of images. The network hyperparameters that best fit the problem have been decided experimentally, comparing results for different network architectures. Figure 7.6 shows a schematic representation of the final network configuration.

As can be seen, the image input is a raw degraded image at high resolution, 1360 x 1024 pixels. Initially, the RGB intensities are converted from 0-255 to the 0-1 range in order to facilitate the network processing. This image is processed through the neural network to produce a reduced 454 x 343 pixels enhanced image. In order to do so, six consecutive convolutional layers are introduced with an increasing number of features to produce the final result.

As a result, after the first layer the image is transformed into 10 feature maps, that increase to 25 after the second convolution and keep increasing until reaching 55 after 5 stages. The last convolution needs to reduce this feature maps to 3 that will become the RGB channels of the resulting image.

Each convolution operation uses a Rectifier Linear Unit (ReLU) activation function, except the last one. This activation function can be seen in equation 7.12. It just returns the maximum between the result of the convolution x and 0. Although this activation function does not directly produce a performance increase, it reduces the training time. Furthermore, as the output needs to be in the range of 0 to 1 the activation function just discards the negative values, making it easier to find a valid model.

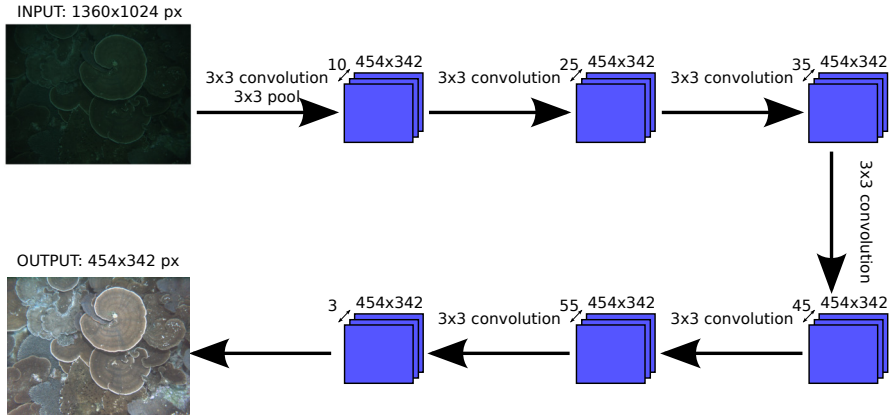

Figura 7.6: Architecture of the convolutional network used to dehaze underwater images.

$$f(x) = max(0, x) \tag{7.12}$$

In the first layer, additionally to the convolution operation, a pooling takes place to reduce the amount of information to the most relevant features. The convolution window has been set to 3x3 pixels with zero padding to maintain the image borders in order not to crop the resulting image.

In order to train the weights and biases of the neural network, the second order gradient descent Adam (Adaptative Moment Estimation) optimizer, described in [Kingma and Ba, 2014], has been used. This optimizer estimates second order moments to automatically set the best learning rate in each moment and decrease the required iterations to train the network.

The cost function used to minimize the error is commonly known as the $l2$ loss function. The $l2$ loss function computes the squared sum of the differences between the estimated y' and groundtruth y values, as can be seen in equation 7.13. In this case minimizing the l2 loss means minimizing the differences of intensities between the restored image, and the ones estimated by the neural network.

$$l2 = \sum_{i=0}^{n} (y_i - y_i')^2 \tag{7.13}$$

As a consequence, if the neural network perfectly learns to perform the same transformation applied with the restoration methodology, it will produce exactly the same result. However, the restoration method used for training requires a depth-map and a whole dataset of images, while the neural network will need to do it with just a single image. For this reason, it is unlikely to achieve the perfect result, but it should be able to obtain a good approximation.

Furthermore, the neural network will try to mimic illumination, contrast and other details present in the groundtruth results, although strictly they do not form part of dehazing. This fact is also a reminder of the need to provide high quality datasets to train the network, and obtain a good trained neural network. In this

case, the datasets only contain a dehazing transformation so further work is not necessary. Another possible alternative to deal with this is to modify the loss function to ignore unnecessary image characteristics, such as illumination or contrast, and learn only the dehazing transformation.

The neural network has been trained with TensorFlow, described in [Abadi et al., 2016], using an Nvidia Geforce GTX 960 with 4GB of RAM. TensorFlow is an open source software library focused on deep learning developed by researchers and engineers working on the Google Brain Team. It provides numerical computation using data flow graphs that represent mathematical operations on multidimensional data arrays known as tensors.

7.3.3. Compared algorithms

Two experiments have been conducted to evaluate the precision of the neural network estimations. In the first case, all the datasets have been used to train and see if the system is able to learn and correctly predict the test images of these datasets. However, the experiment does not show a realistic situation: training images for the intervention location are usually not available at training time.

Consequently, a second experiment has been designed to simulate this situation. In the second experiment, a dataset is left out in the training stage, that is performed with the five remaining datasets. Then, the trained network is validated with images from a location that it has never seen in the training. This experiment tests the validity of the system, generalizing the solution to be used in a different location.

Additionally, four simple commonly used image enhancement techniques have been added to the comparison in order to see the real performance of the learning approach. In first place, a histogram equalization has been compared with the proposed solution. Histogram equalization is based on the analysis of the histogram of a raw image, displacing it to follow a desired distribution that maximizes certain parameters.

There are different implementation options depending on the desired final distribution of the histogram, in this work the most widespread, normal distribution for each channel, is used. The main drawback of these kinds of techniques is they tend to produce unrealistic effects in photographs, creating false colors and contrast, known as overcorrecting the image. Furthermore, it may increase the noise while decreasing the original signal depending on the input.

A simple variation of the histogram equalization that deals with these kinds of problems is the Contrast Limited Adaptive Histogram Equalization (CLAHE). These types of techniques have been used in underwater image dehazing in [Hitam et al., 2013], and in other situations to locally increase contrast.

The main difference with the previous technique is that it divides the images in tiles and computes several histograms to redistribute the intensity values in the image. The contrast limited addition is an approach to prevent the overamplification of noise typical in these kinds of techniques. The algorithm has been applied in the Lab colorspace only to the luminosity component, in order to preserve colors and reduce overcorrecting, and in the case of RGB colorspace to the three components separately.

Cuadro 7.1: Required time to process a single image for image enhancement algorithms

Proposed	ACE	histeq	CLAHE	CLAHERGB
0.013s	1.5s	0.013s	1.7s	0.3s

Finally, the last compared algorithm is an Automatic Color Enhancement (ACE), as explained in [Getreuer, 2012], that is also used in underwater environments in [Iqbal et al., 2007] for dehazing. This technique enhances the image based on a simple model of the human visual system, inspired by different techniques such as gray world transformation, white patch assumption, lateral inhibition and local global adaptation. The main drawback of this technique is it is computationally complex, each image requires around 1.5 seconds in a Intel i5 at 3.2Ghz with a Geforce 960GTX while the time to process a single image in a neural network is 0,013 seconds.

Even though all the compared algorithms are classified as image enhancement, they show different computational complexities, thus making it difficult to the use of some of them in a real time applications. In table 7.1 the required time to process a 1360 x 1024 pixel single image for each of them is shown.

As can be seen the neural network and the histogram equalization require a similar amount of time since they are the fastest to compute by far. The CLAHE on the RGB colorspace is still a fast algorithm, and could be easily included in a real time solution. However, the CLAHE on lab workspace is noticeably slower, requiring 1.7 seconds per image due to the colorspace change before and after the adaptative histogram equalization takes place. The ACE is also a slow process that requires almost 1.5 seconds per image.

7.3.4. Experiment 1: Test from same dataset

In this experiment, all the datasets have been used to train the neural network, keeping a few images of each in order to test its performance. In order to train the images, each epoch of the training set is divided in randomly selected smaller batches of 10 images so it is possible to execute them in the GPU memory. During every epoch, the batches are randomized automatically choosing a different set of images in the group, avoiding overfitting for the batch.

Furthermore, the test images are periodically checked to detect an early stopping scenario. This is the case when the training error decreases, but the test error increases due to an overfit with the training set. However, this did not happen in this the experiment so the training stopped when the training error reached a minimum over 100 epochs.

The system has been trained for 1700 epochs, reaching a 5.6 % training error. This error corresponds to the cost function, the mean difference between each intensity pixel and its groundtruth counterpart. In order to show percentage errors and facilitate network learning, intensities are transformed from 0-255 to 0-1 range.

The results for the test images of each dataset can be seen in table 7.2, together with the image enhancement techniques. As can be seen, the proposed method obtains the best results in all cases. This is not surprising as it is training with images

Cuadro 7.2: Results for the direct neural network dehazing experiment 1: Training with all the datasets.

Technique	Rocks	Kelp	Rocks-Sand	Deep	Medium	Shallow
Proposed	**3.5 %**	**6.5 %**	**5.1 %**	**4.1 %**	**3.4 %**	**3.2 %**
ACE	6.8 %	9.1 %	15.3 %	15.7 %	7.5 %	9.1 %
histeq	25.6 %	37.9 %	29.4 %	20.5 %	27.6 %	27.5 %
CLAHE	16.2 %	6.7 %	9.8 %	27.1 %	8.2 %	9.0 %
CLAHERGB	16.5 %	6.6 %	9.7 %	27.2 %	9.2 %	9.3 %

from the same survey, so it has similar examples that help to dehaze the raw image. But it is important that the neural network is able to learn the transformation and correctly apply it to new images never seen before, even though similar to others in the training set.

With regard to the image enhancement algorithms, it can be seen that they perform very differently depending on the dataset. Furthermore, while some of them may have a good result for a concrete dataset it does not mean this dataset is easiest to dehaze, because other enhancement methods produce bad results for it. For instance, the ACE dehazing has a result close to the neural network performance for the rocks datasets while CLAHE results are far from it. In the case of deep corals, ACE and CLAHE alternatives obtain the worst performance while the histogram equalization is the best case.

This means each technique can be more or less appropriate according to the type of image. As the datasets cover different types of textures, and different visibility and illumination conditions, there is no perfect solution for all of them. For this reason, developing a technique able to treat the images differently depending on their characteristics seems a good alternative. This is probably the reason why the neural network is outperforming the image enhancement methods, it is able to adapt dynamically to the image characteristics and dehaze it depending on them.

The ACE and both CLAHE applications perform best, besides the neural network solution. CLAHE algorithms seems to work better in kelp and rocks and sand datasets, while ACE is better in rocks and deep corals. In the medium and shallow corals datasets both approaches perform similarly. But in any case, the results are still far from the neural network performance.

Although using CLAHE in the L of lab colorspace seems to be a more commonly used approach, the results show there is not a big difference with a direct application on the RGB components. The results of both configurations are similar for all the datasets due to the similarities in the procedure. For this reason, the CLAHERGB may be a better option for real time applications due to its fastest computation time.

The histogram equalization is by far the worst performing in all the datasets except the deep corals where it is better then the CLAHE alternatives. Anyway, it may be an interesting alternative to other techniques for real time applications given the faster computation.

In any case, the difference with the restored image may be a bad metric of the dehazing process, different illumination, contrast, etc, may result in a high difference but still be good at dehazing. Taking this into account, the visual results

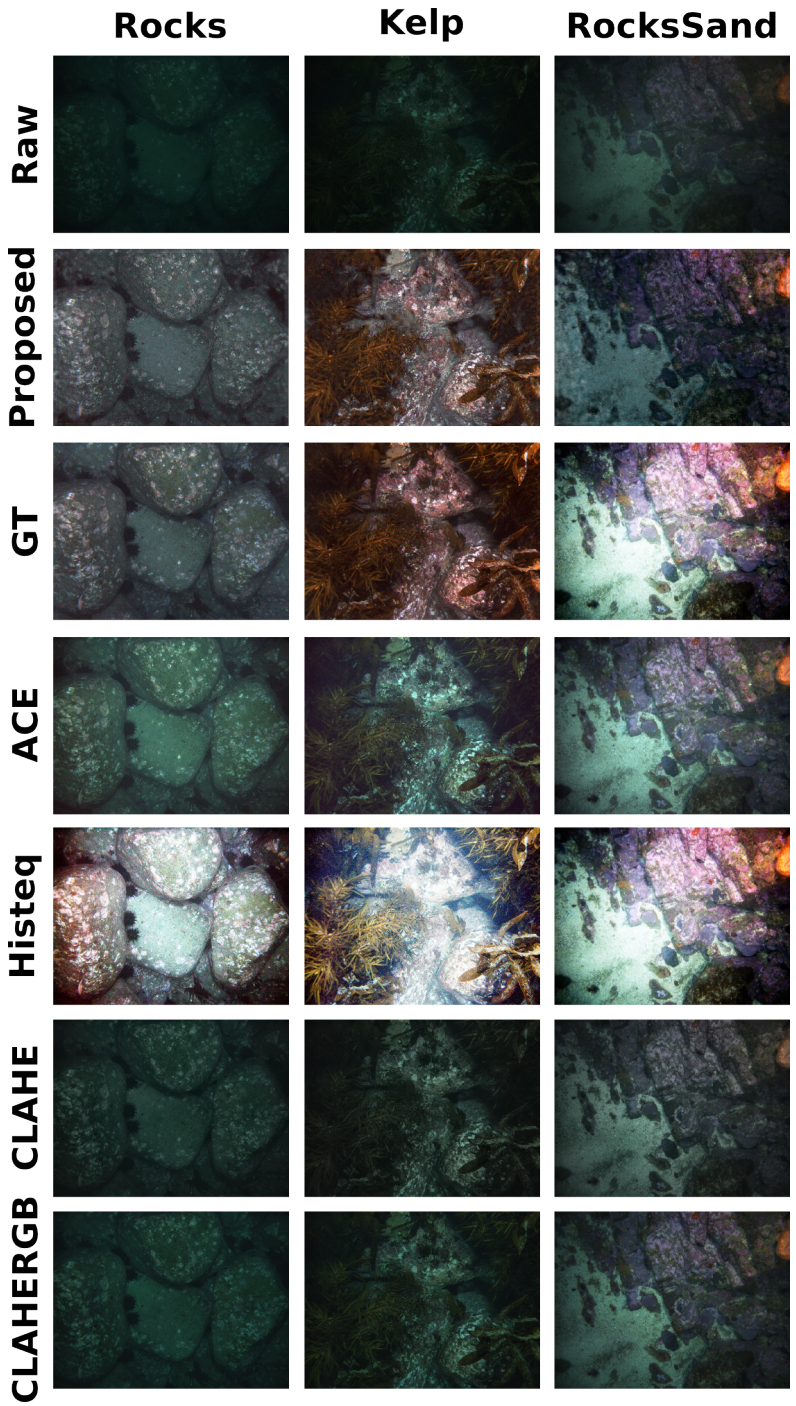

Figura 7.7: Comparison of the direct image dehazing using different techniques when using all the datasets in training (part 1).

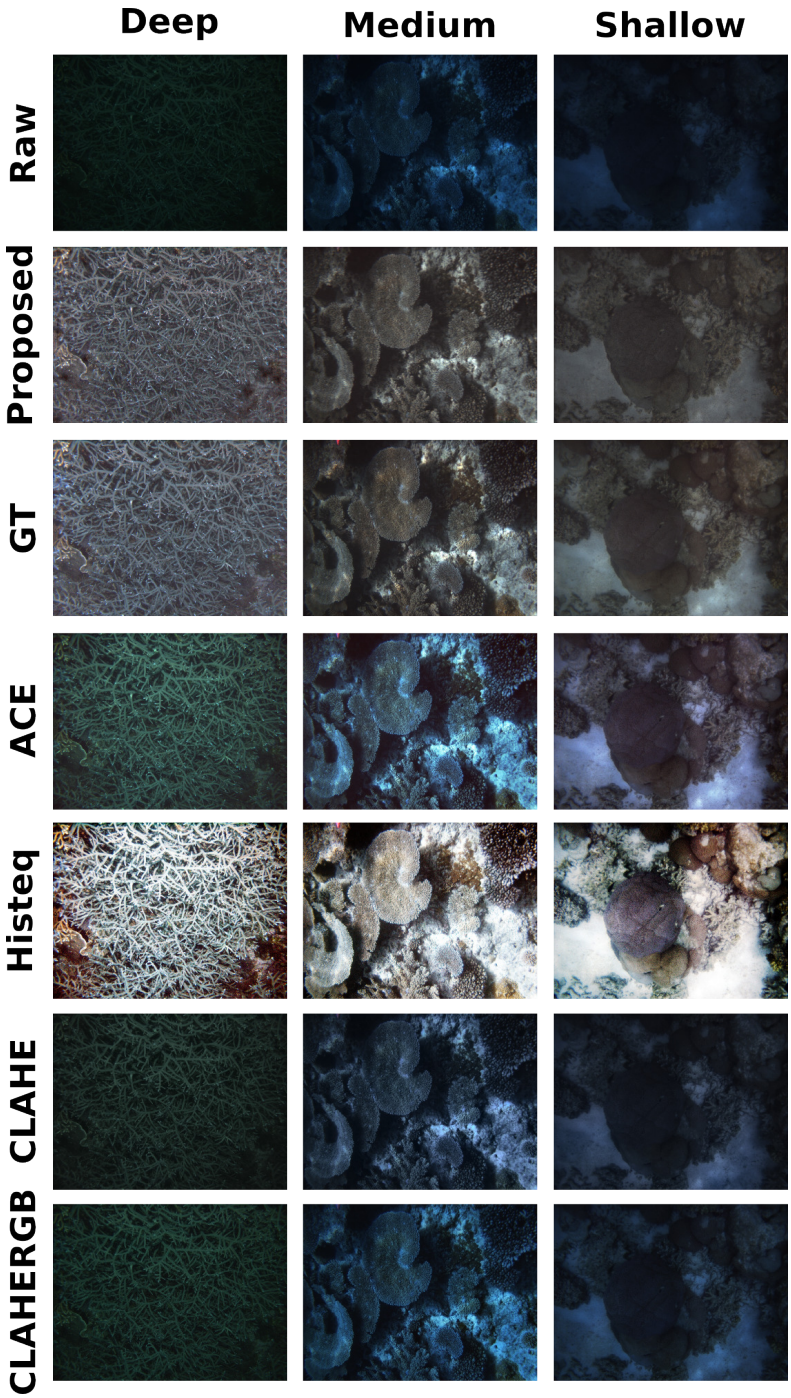

Figura 7.8: Comparison of the direct image dehazing using different techniques when using all the datasets in training (part 2).

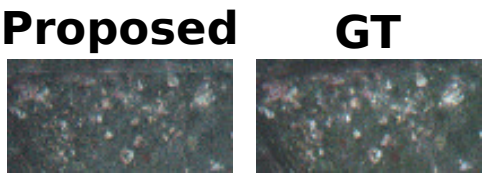

Figura 7.9: Halos in the border of the images present in neural network dehazing.

of the dehazing can be seen in figure 7.7 and figure 7.8. In the figures, a test image of every dataset for each compared technique is shown together with the raw and groundtruth (GT) images.

As can be seen, the proposed method and the groundtruth images are indistinguishable in most cases, and in the cases that are different, such as RocksSand dataset test image, it is difficult to decide which one is better. The neural network dehazing is darker, probably resulting in a higher difference, but providing a more visually pleasing result than the groundtruth image that it is overilluminated. However, in terms of restoring the original image without the water degradation effects, the ground truth may produce a better dehazing result.

About the image enhancement methods, the ACE methodology obtains slightly uncorrected images. The algorithm is not completely able to remove the haze, obtaining bluish, in the case of medium and shallow corals, or greenish images depending on the input. Even though the neural network and groundtruth techniques were able to completely correct the water color in the images, the ACE algorithm greatly enhances the image from the raw input.

On the other hand, histogram equalization overcorrects the images, producing unnatural images with extreme colors in some cases, such as the kelp dataset. In the case of the rocks dataset, it exacerbates the colors absorption, showing a reddish tone for the rocks closer to the camera that diminishes with the distance. Besides this, it demonstrates that although the numeric difference with the groundtruth is high, part of this difference is caused by the brightness of the equalized images.

In the case of CLAHE, the images of both configurations are indistinguishable as numerical results suggested. Even though images are enhanced with respect to the raw image, the colors are not corrected and still show a bluish or greenish tone. The obtained images are similar to the ones from the ACE algorithm but slightly darker.

A noticeable drawback of the proposed method is the halo in the borders of the image that can be seen on figure 7.9. As can be seen in the zoom of the rocks dataset image, blurred lines parallel to the border appear surrounding the image. These unnatural borders are caused by the zero padding configured in the convolutional operations. The zero padding convolutes with a 0 input in the borders that is misinterpreted by the network, resulting in halos in the borders of the image.

As a conclusion, the proposed algorithm is clearly the best alternative in numeric and visual results. Furthermore, it can be computed extremely quickly being the best option for a real time dehazing application. However, the results need further validation as the neural network was trained with images from the same survey as the test, making it easier to learn the dehazing transformation for them.

7.3.5. Experiment 2: Validating with a different dataset

Taking into account the results of the previous experiment, this test simulates the situation of using a trained neural network in a different environment. The increased difficulty is caused by the fact that the system has never seen images from this location, so it is impossible to use past knowledge of the location in order to produce a restored image. This will test the generalization capabilities of the neural network.

In order to test this generalization algorithm, one of the available dataset must be left for validation. The deep corals dataset has been chosen because its bad results with the best image enhancement techniques make it a more challenging scenario. For this reason, the remaining datasets are included in the training scheme and a new neural network has been trained with them.

The training procedure is exactly the same as in the previous experiment. In this case, the neural network required 2500 epochs to reach a similar amount of training error, 5.5 %. This was probably caused by a inferior random initialization of the parameters, as the neural network used the same datasets and reached a similar training error.

The numerical results for the validation dataset show that the difference with the groundtruth image is closer to the ACE performance, but it is still performing better. The mean intensity difference for the validation dataset is 14.1 %, while the ACE technique obtains 15.7 %, histogram equalization 20.5 % and CLAHE is around 27.2 % far from the groundtruth images. As expected, the test images for the resting datasets maintain the distance error to the groundtruth images around 4.5 %.

This results are logical as the neural network was trained with similar images in the previous experiment. However, the system is still able to generalize and produce a fairly good result when dehazing completely unknown images. Furthermore, the proposed approach performed better than any of the image enhancement alternatives and it is possible to process the images much faster than most of them.

As the previous experiment showed, although the cost function is good for training, sometimes the difference with the restored image does not mean it is a good dehazing. For this reason, samples of the validation dataset for the proposed method, input image, groundtruth and compared enhancement methods can be seen in figure 7.10. The images for the best and worst performances for the neural network, according to the difference with the restored image, are displayed together with two images close to the mean difference.

These results show the neural network solution slightly overcorrects the images with respect to the groundtruth. It is possible to see that because the corals in the image have a higher red component than the restored images. But in any case, the images look natural with no signs of the greenish tone of the raw acquired images. Moreover, the two images close to the mean difference are just darker than the restored images having a more natural appearance.

Besides this, the neural network processed images still suffer from small halos around the borders of the image. This effect is caused by the zero padding and may be removed croping the image when processing it or applying a specific postprocess to remove it. However, it only affects a few pixels in the borders that are not important for the image appearance or further automatic processing for autonomous

Figura 7.10: Comparison of the image dehazing using different techniques for the validation dataset.

robotics.

Beyond the dehazing process, an unexpected result is the neural network also learned to deal with vignetting. As can be seen in the raw, histogram equalization and specially in ACE processed images, the corners of the images are darker than the rest of the image. On the other hand, the restoring methodology used to train images, described in [Bryson et al., 2016], takes into account the vignetting, and this effect has been translated to corrected images that do not have this degradation. The trained neural network was able to learn this correction and the enhanced images do not have dark corners.

Regarding histogram equalization, it also overcorrects the image but in a much higher amount, producing unnatural images far from the desired result. Furthermore, it is possible to observe the different light wavelength attenuation as the objects closer to the camera are mainly red, the intermediate have a greenish tone and objects further away are blue. Additionally, some parts of the image are too bright and some others too dark, making it difficult to interpret the image.

In the case of ACE, images are not completely corrected and processed images have a greenish color compared with raw images. The image has improved from the acquired source increasing the brightness and reducing the degradation from the water. However, the colors of the corals in it are still attenuated making it difficult to distinguish them.

Finally, the CLAHE and CLAHERGB offer very similar performance being impossible to see any difference between them. The enhanced images are far from the groundtruth target showing a dark green tone.

This experiment proves the neural network performance is better than the compared alternatives, although numerical results seem to be close to the ACE performance, visual results look much more natural. The system was able to learn a dehazing transformation successfully, correcting different degradation effects such as attenuation, scattering and vignetting at the same time.

From the tested algorithms, it was the only one able to produce naturally looking images without the underwater haze that are visually pleasing at the same time. Furthermore, the computation time is much shorter making it possible to include it in a real time system as a preprocessing step. Finally, it is important to remark that the training never saw an image of the validation dataset demonstrating that it is possible to use images to train in a different location from the intervention.

7.4. Estimating depth for still image dehazing

As discussed previously, an estimation of the depthmap of the scene is a key parameter for underwater image restoration techniques. The depth is directly related to the transmission that degrades the image. For this reason, many approaches estimate transmission with a depthmap, as the restoration alternatives, or directly estimate the transmission as in the case of DCP. Obtaining 3D information is possible through a stereo camera, however untextured terrain or the need to dehaze single images requires the use of other techniques.

For this reason, an alternative to the previous approximation that directly enhances the image without taking into account the image formation model is to make an estimation of the depthmap and then dehaze it accordingly. In order to

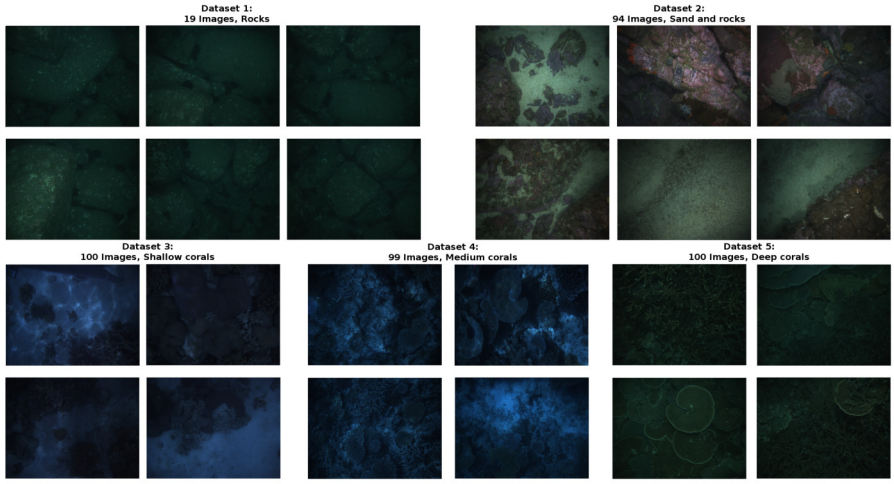

Figura 7.11: Images of the different datasets used in the deep learning approach for estimating 3D depth.

do so, the proposed approach makes use of deep learning and datasets with depth information to train a neural network able to predict depth from still images.

A key idea in this approach is that although single images do not contain depth information, visual cues such as shading and specially the light attenuation as it travels through different distances may allude to it. As a consequence, a deep learning approach may be able to learn to detect features and extract a depthmap useful for single image dehazing.

Furthermore, as described in section 7.2.1 using deep learning for depth estimation has already been done in air free images obtaining promising results. These approaches may obtain even better results in an underwater environment, where the image degradation caused by the light transmission underwater is a strong depth cue.

7.4.1. Image datasets

In this case, the deep learning solution needs to be trained and validated with images that have 3D information. For this reason, the kelp dataset was discarded, and the rocks dataset is used only for the cases where a depthmap is available. Finally, five datasets where used: rocks, rocks and sand and the three corals datasets. The dataset acquisition method and details about the characteristics can be consulted on Appendix A.

Example images from each dataset used in this approach are shown in figure 7.11. As can be seen, images cover a different range of textures, illumination conditions and distance to camera. This will help the neural network to generalize and be able to correctly predict depth independently of the location and conditions of the image.

In order to augment the data used in the experiment, the datasets are augmented in the training stage. The augment techniques consist of the application

of simple transformations to the input data apparently increasing the number of examples, helping to generalize and learn a more abstract solution to the problem. In this case, the training images are randomly flipped horizontally and vertically with 0.5 probability, multiplying the effective training set by $4x$. These random flips abstract local details like specific image positions of depth cues or common camera angle with respect to the ground.

7.4.2. Proposed approach

The proposed approach is based on the restoration algorithms, as a consequence parameters derived from the water state are estimated to use them in the image formation model explained in section 6.1. One of the most important parameters is the transmission that attenuates the light and is directly related to depth. As discussed before this can be done with a convolutional neural network, consequently this parameter will be estimated using a deep learning approach.

The proposed method consists of three steps: (1) Obtain a coarse estimation of the depthmap using a neural network, (2) refine the estimation with a guided filter and (3) the final application: dehaze the image using this estimation. The first two steps are similar to neural networks for 3D estimation explained in section 7.2.1, firstly obtaining a coarse estimation and then refining it to obtain more detail.

The first step, coarse estimation, is performed through a convolutional neural network. Once trained, the neural network receives an RGB image as input and produces a rough depthmap. The training stage also requires depth estimation so the network is able to learn depth cues from the example pairs: image and depthmap. Although a previously trained network may be able to make a good estimation in a different environment allowing its use in monocular cameras, it is recommended the neural network to be trained with images similar to the environment where the AUV will finally work for better performance. This step may be necessary if the textures and features are extremely different from the ones used for training, as the neural network needs to learn depth cues in this new environment. For this reason, the datasets used include a wide variety of textures and environments.

The refinement step is made through a guided filter as described in [He et al., 2010]. However, merging it with the previous step in another neural network, similar to other works in the state of the art, has also been tested. In this case, an initial estimation of the depthmap obtained in the previous step and the original RGB image serve as input to produce a more precise and detailed depthmap than the rough estimation from the neural network.

The final step involves image dehazing. This step uses the previously estimated depthmap and the original image to produce the enhanced image. The transmission of the image is estimated using the depthmap, and it is applied with a histogram equalization in order to recover the original colors of the image.

As in the previous case, the system has been trained with TensorFlow, described in [Abadi et al., 2016], using an Nvidia Geforce GTX 960 with 4GB of RAM.

Coarse estimation

The coarse estimation is made using a convolutional neural network that receives a single raw image, and returns a depthmap of the same size. In order to do so, the

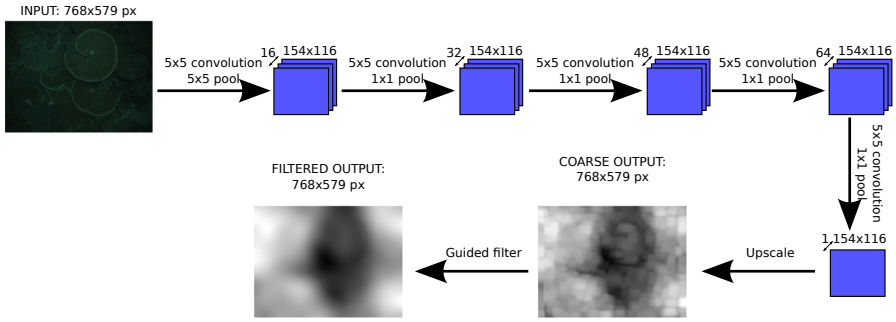

Figura 7.12: Architecture of the convolutional network used to estimate depth.

image is processed through multiple convolutional and pooling steps. As discussed before, these kind of operations to obtain a depthmap are reasonable as many heuristics such as DCP or contrast operations can be expressed as convolutions. The main advantage of this approach is the neural network will have the freedom to learn the best weights and parameters to extract the correct features for depth estimation.

The use of fully connected layers has been discarded because it requires a number of neurons that depend on the image size. This fact makes it impossible to use in high resolution images or requires the output to be resized. Furthermore, the type of transformation required should be spatially invariant, thus a convolutional neural network appears to be the more logical approach.

Several neural network configurations have been tested, but the architecture proposed here can be seen in figure 7.12. The neural network is made of five consecutive operations of convolution and pooling, and a final upscale to recover the original size. After this, a coarse output is obtained that will be refined to obtain a filtered output.

As figure 7.12 shows, five layers of convolution and pooling have been used. The first layer reduces the image size with a 5x5 pooling, while the others maintain the size. The number of filters increases in each layer until the last one, that is reduced to one, the output.

The initial layer is a $5x5$ followed by a $5x5$ max pooling that takes a $768x579$ image with RGB channels and reduces it to 16 feature maps with size $154x116$, containing the most interesting features. The input images are previously scaled to 0-1 range to make the training easier in the neural network. This is an important step as it helps to reduce the amount of information that will be processed in the subsequent layers of the neural network, making possible to train it faster without loosing precision.

As a consequence, after the last convolutional layer is computed a biliniar upscale is required to restore the original image size at the output. This step is included inside the neural network, the cost function is computed using the upscaled image, thus it can adjust the cost using this knowledge.

Alternatives to this drastic reduction of the feature size have been tested, but the results showed no benefit. For this reason, working with a reduced feature map

appears to be a better option as it increases the training speed without reducing the precision. Using a max pooling lets the neural network decide the best features to keep in order to estimate the depth.

After these initial convolutional layers, the next three layers increase the number of features at each layer until 64 in the fourth layer. Then, the features will be reduced to one at the final layer, corresponding to the depthmap prediction. In these layers, the max pooling is made using a strides configuration of 1 pixel, as a consequence the size of the feature maps maintain the same size through the rest of the network. Furthermore, every convolutional layer is configured to use zero padding and maintain the size of the feature maps and consequently the output size.

The influence of the number of layers has also been compared. The conclusion in this case is that 5 layers is the best option to avoid overfitting while obtaining the best possible results with this architecture. The details of this comparison are included in the experiments results.

As in the previous approach, all the hidden layers use a ReLu activation function with the exception of the last convolutional layer which is linear. This is also a natural option as the negative values are not possible, thus discarding the models that produce negative values will decrease the training time.

Similar to the previously presented neural network, the optimization function used is the Adam optimizer. This is a second order gradient descent algorithm that estimates second order moments to automatically set the best learning rate in each moment, and decrease the required iterations. It usually makes the system converge faster than regular backpropagation methods, slightly increasing the computational complexity.

With regard to the cost function, the l2 loss function, squared difference between estimation and desired output, is not the best option due to the impossibility of determining the right scale from a single image. In other words, it is impossible to estimate real distances from a single image with a single image without a known object or scale, thus direct difference of distances is not a good choice for the cost function. For instance, it is not possible to tell the difference between a picture of a real house from a perfect mock-up or model, that is the reason why a scale is needed.

For this reason, using the absolute or squared difference of the estimated depth and groundtruth depth may make the training stage more difficult. In the state of the art mainly two different alternatives have been presented to solve this problem. The first one is preprocessing the data to use relative distances in the training set and learn to predict relative distances. The main disadvantage of this alternative is the system will need to transform back these relative distances back again.

The second option, described in [Eigen et al., 2014], uses a scale invariant error when training. This cost function controls the amount of error coming from a wrong scale choice, thus a prediction with the same mean squared error will have a smaller cost if it is linearly related to the groundtruth. The advantage of this approach is that it tries to estimate an absolute distance but the training also minimizes the relative estimation, thus assuring a good relative prediction but still producing a scale guess even though it might be wrong.

Taking this into account, a scale invariant error cost function has been used to train the neural network. Additionally, a common transformation is measuring

the error in log scale as it will help in the error computation. As a result, the loss function used in the deep learning approach can be seen in equation 7.14.

$$Loss\,(gt, y) = \frac{1}{n}\sum_i diff_i^2 - \frac{\alpha}{n^2}\left(\sum_i diff_i\right)^2 \tag{7.14}$$

The loss function depends on the groundtruth depthmap gt and the neural network prediction y. The $diff_i$ term is defined as the difference between a predicted point y_i and the corresponding groundtruth point gt_i in logarithmic scale: $diff_i = \log y_i - \log gt_i$. Besides this, n is the number of pixels in the prediction, basically $height \cdot width$, and $\alpha \in [0, 1]$ is a coefficient that controls the scale invariant term.

The first part of the cost function is the Root Mean Squared Error (RMSE), which penalises individual predictions far from the groundtruth values in absolute terms. However, the scale invariant addition is the second term of the equation that counteracts the first term when the prediction mean is different from the groundtruth mean.

For example, assuming α is 1 if $gt = 0{,}5y$ then $diff_i = \log 2$ for every number. When $diff_i = k$ for every number in the series the equality in equation 7.15 is true. This means when the prediction and groundtruth are linearly related both terms are equal and the cost function is 0, ignoring scale changes.

$$\frac{1}{n}\sum_i k_i^2 = \frac{1}{n^2}\left(\sum_i k_i\right)^2 = \frac{1}{n^2}\left(\sum_i k_i^2 + \sum_{j\neq i} k_i k_j\right), \forall_{i,j} k_i = k_j \tag{7.15}$$

On the other hand, if pair predictions are $gt = \frac{1}{k}y$ and the other half are $gt = ky$ then $diff_i = \log k$ if $i\,\%2 = 0$ and $diff_i = \log \frac{1}{k}$ otherwise. This makes the second term of the cost function 0 as equation 7.16 shows, thus the error will be the MSE.

$$\frac{1}{n^2}\left(\sum_i diff_i\right)^2 = \frac{1}{n^2}\left(\sum_{i\,\%2=0} \log k + \sum_{i\,\%2\neq 0} \log \frac{1}{k}\right)^2 = 0 \tag{7.16}$$

Taking this into account, the cost function will measure if the neural network output and the groundtruth values are linearly related, returning a value close to 0 when they are. In other words, the scale invariant term added to the MSE detects when both depthmaps are the same with a different scale factor and counteracts the MSE. As a consequence, minimizing equation 7.14 finds the best linearly related depthmap, independent of the absolute values.

However, it is also interesting to obtain accurate absolute predictions for the depth. In order to achieve this, the coefficient α has been introduced, when it is close to 1 the neural network will search for scale independent solutions, and when it is close to 0 it will search for absolute predictions. Taking this into account, an intermediate 0.5 value has been used, being able to achieve a good absolute and relative prediction minimizing both terms.

Lastly, the groundtruth depthmaps do not provide a valid depth for every point in the image, because they were acquired using stereo cameras that require texture

to retrieve 3D. In order to deal with this, the loss function is only evaluated in points with valid depth, adjusting n for each image, and performing the summations only when a valid depth point is available.

In [Eigen and Fergus, 2015] adding gradient differences to the loss function is proposed, but experiments in these datasets showed no benefit. Calculating image gradients consumes a lot of time and memory, furthermore using it in a reduced dataset did not enhance the results. For these reasons, it is discarded from the proposed solution, but is still a feasible area for future work and the results of this alternative are compared with the proposed solution. In that case, the cost function is the one expressed in equation 7.17.

$$Loss\left(gt, y\right) = \frac{1}{n}\sum_{i} diff_i^2 - \frac{\alpha}{2n^2}\left(\sum_{i} diff_i\right)^2 + \frac{1}{n}\sum_{i}\left[\left(\Delta_x diff_i\right)^2 + \left(\Delta_y diff_i\right)^2\right] \tag{7.17}$$

Guided filtering

Once the coarse estimation is obtained using the neural network, a blurred image containing the estimated depthmap can be estimated. The coarse map comes from an upscale of a neural network, so it does not have enough details and proper borders to be useful. In the state of the art an additional convolutional network is proposed to refine the results using the coarse estimation together with the input image.

However, this problem is similar to the refinement to prevent halos in the dark channel prior explained in section 6.2.1. The solution proposed by the DCP authors in [He et al., 2011] is an image matting able to smooth the results while maintaining the borders. Unfortunately, this is an extremely slow non linear process. For this reason, other authors substitute this by a guided filer as described in [He et al., 2010]. In the experiments of Chapter 6 both alternatives are compared, concluding that the small performance gain of image matting is not worth the required computing time. Furthermore, as the application studied, enhancing underwater images for increasing autonomous capabilities of underwater robotcs, requires almost real time processing only guided filtering is considered.

In this case both alternatives, additional convolutional network and guided filtering, have been tested. The results showed guided filtering was able to refine the image better than a convolutional neural network. Taking this into account, the approach described uses a guided filter to smooth the coarse estimation and finally obtain a depthmap. The results of this alternative are included in the comparison.

The guided filter is an image filter that computes the filtering output considering the content of a guidance image, as figure 7.13 shows. This is supposed to filter the image, smoothing it but preserving the edges already present in the guide image. This effect can be seen in the figure 7.13, the estimation is smoothed respecting the edges in the guide image. As can be seen, the random noise in the estimation surface has almost disappeared in the filtered image, but it still preserves the edges already present in the guide image. In this case, the input of the guided filter is the depthmap estimation calculated by the neural network, and the guide used is the original image.

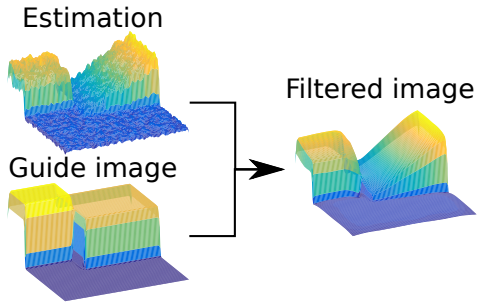

Figura 7.13: Guided filter smoothing of two surfaces example.

Dehazing

The final step is the use of the estimated refined depthmap in an application, in this work it is used in the problem of image dehazing. A simple solution analogue to DCP has been chosen to work in real time, although more advanced techniques may achieve better results. The DCP used a heuristic called dark channel in order to estimate the image transmission that describes part of the degradation of the image. This transmission is directly related to image depth, thus it is possible to use the previously estimated depth instead of the heuristic used in DCP.

As discussed before, this is a reasonable assumption as the convolutional neural network should be able to naturally learn the DCP heuristic using convolutional and pooling operations. Furthermore, if the DCP is the optimal solution to transmission estimation it is highly probable that the deep learning solution will find it, but it has enough freedom to learn the operation that produces the best results in the training set of images.

In this case, the depthmap estimation is used as transmission $\widetilde{t}$, in order to do so the exponentiated depth d is multiplied by a constant attenuation $b(\lambda)$. This attenuation is the density of haze in the environment, that can be approximated to be constant in a reduced zone. In this dehazing scheme, the value of the most distant pixel (RGB) proved to be a good estimator for attenuation $b(\lambda)$ as proposed by the DCP algorithm.

Using this transmission, the image can be processed to recover the original colors from the attenuation. Thus the inverse of the simplified attenuation function 7.18 is used as described in equation 6.7, being I the image acquired by the camera, and J the image without noise.

$$I = J\widetilde{t}(x) = Je^{-b(\lambda)d} \tag{7.18}$$

It is only necessary to use the previously estimated values for depth d, attenuation $b(\lambda)$ and the captured image I to restore the "original" image J. Finally, a histogram equalization is applied to enhance the colors of the image that may be displaced due to the attenuation estimation.

Other image formation models may be used to achieve more precise results, for instance considering the sunlight or introducing a more complex estimation of the attenuation coefficient. This simple model was chosen to show the validity of the

depth estimation in this context, and allow real time single image dehazing.

7.4.3. Metrics and evaluation.

In order to evaluate the performance of the neural network, 20 % of the images are left in a validation set. This set of images is chosen randomly from each dataset. This approach ensures that the different environments used for testing are represented equally in the validation set. Consequently, all the datasets have enough images in the validation set in order to measure the performance of the proposed approach.

The validation set is only used for final results, while the training stage uses the rest grouped in minibatches small enough to fit in GPU memory, in this case 10 images, but the number may be different depending on the memory requirements of the network architecture. As in the previous neural network approach, the mini batches are randomized after each epoch avoiding the selection of the same group of images together. Furthermore, as described before each time an image is used to train it may be flipped horizontally or vertically to augment the number of training images and avoid overfitting.

Although the cost function used for training is a good function for this purpose, it is not a good measure for the system performance. It is a good idea to separate the absolute estimation and the relative estimation from the cost function. This will make it possible to decide which kinds of applications can benefit from the 3D estimation, and facilitate comparing with other alternatives such as the DCP or other alternatives that estimate depth.

Due to the nature of the application, three different metrics are used to check the validity of the estimation: Correlation, Root Minimum Squared Error(RMSE) and Adjusted Root Minimum Squared Error(ARMSE). Each of them will measure a different aspect of the result. In the first case, the correlation measures the linear similarity between estimation and groundtruth, which is a measure of the structure. The RMSE evaluates the absolute difference of the prediction of the neural network and the real value, describing the precision. Finally, ARMSE is an adjusted RMSE useful to compare with other dehazing approaches that estimate transmission instead of depthmaps.

In order to measure linear correlation, the sample Pearson correlation coefficient that can be seen in equation 7.19 is used. In it, x and y are estimated and groundtuth depth values respectively. It measures the linear correlation between two variables X,Y producing a value between -1 and +1 where 1 is total positive correlation, 0 is no correlation and -1 is total negative correlation.

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y}, r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{7.19}$$

This metric is useful to know if both surfaces are similar in terms of appearance. As neural networks are trained to minimize the error between surfaces, the result should be close to 1 the higher the better. However, two depthmaps may be really similar and not be linearly related, for this reason it should be complemented with other metrics. This correlation is not dependant on scale, so it is a good measure to use with transmission estimations after taking logarithms as well as depthmaps.

The second proposed metric is a Root Minimum Squared Error. It is actually one of the terms of the minimization function, so the lower the better. Although, the cost function has also an invariant scale term that can make this result increase. In this case, it is measuring the point-to-point error so surfaces could be different in shape but still have a good RMSE value. For this reason, it is reasonable to also measure the correlation or the ARMSE error.

Finally, an Adjusted Root Minimum Squared Error is proposed. This measure equals the means of the estimation and groundtruth depthmap so the scale uncertainty problem, that is the impossibility to distinguish a real house from a dollhouse in a 2D picture, is avoided given that both depthmaps now have the same scale. This is a good measure for the dehazing problem as the depthmap is multiplied by the attenuation, which has to be estimated afterwards. As a consequence, the scale of the estimation is not so important as in other applications. Furthermore, this measure can be extracted from the transmission estimation allowing the results to be compared with other techniques that do not directly estimate depthmaps.

In addition, these last two metrics are also computed relative to the groundtruth depth, so a percentage error is obtained. This helps to understand the precision of the proposed approach in relative terms. The equations to calculate this error for an estimation x and groundtruth y measures can be seen in equation 7.20, using an adjusted x provides ARMSE and RARMSE metrics.

$$\begin{aligned} RMSE &= \sqrt{\frac{1}{n}\sum_{x_i \in y_i} (x_i - y_i)^2} \\ RRMSE &= \frac{1}{n}\sum_{x_i \in y_i} \left| \frac{(x_i - y_i)}{y_i} \right| \end{aligned} \tag{7.20}$$

In terms of visualization, 3D plots of the estimated and groundtruth depthmaps, pixel error histograms, and RMSE can be obtained similar to the one shown in figure 7.14. This is valuable information for interpreting the performance of the 3D estimation and checking if the neural network is providing good results.

For instance, in this figure the top left bar chart shows a histogram of the depth estimation errors for the top right image. This histogram accumulates the number of pixels with similar errors in bars representing the amount of pixels within a range of error. If the estimation was accurate, it should show a gaussian distribution centered in the 0 error. A displacement in the error axis will show an incorrect scale estimation. Furthermore, the smaller the standard deviation the better it is at estimating the depthmap. It is also possible to generate this type of graphics with ARMSE and relative errors.

The bottom surfaces are the stereo groundtruth depth map on the right side, and the neural network estimation on the left side. This kind of visualization has been used to check the validity of the metrics and the performance of the neural network. In this case, it can be seen that both surfaces are similar in the structure, thus it is a good estimation of the depth of the image.

In order to compare with other state-of-the-art algorithms, the DCP technique presented in [He et al., 2011] has been used in the image datasets. The correlation and ARMSE metrics are also computed and shown. In order to compute the ARMSE metric it is necessary to use the transmission estimation of the DCP shown in

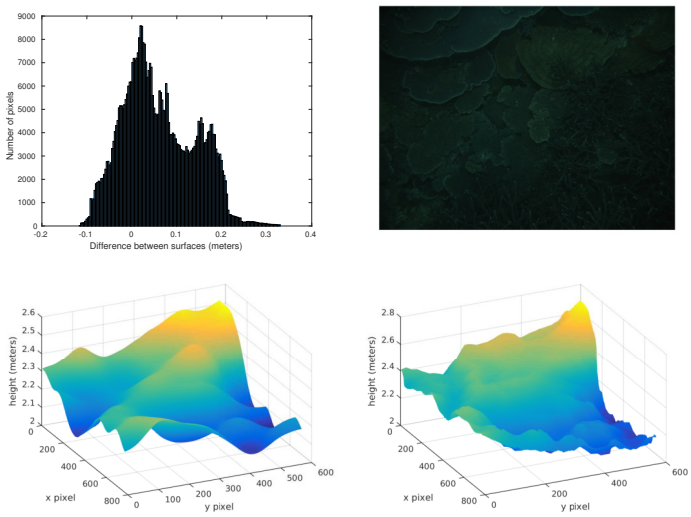

Figura 7.14: Visual metrics and evaluation for depthmap estimation: (top right) Visualization of the image, (top left) error histogram, (bottom right) groundtruth depthmap surface and (bottom left) neural network estimation surface.

equation 7.21, where $b(\lambda)$ is the attenuation and d the depthmap. The optimal $b(\lambda)$ has been used to obtain the depthmap to be compared with the groundtruth.

$$\widetilde{t}(x) = e^{-b(\lambda)d} \tag{7.21}$$

In the case of the dehazing step only qualitative results are shown due to the difficulty of using an appropiate measure that focuses on the dehazing instead of illumination, contrast, colors, etc. However, in this case other alternative methods that require the same input as the proposed approach are included so that they can be easily compared with the dehazing.

To show the validity of the depth estimation in the context of still underwater dehazing, three experiments have been conducted. In the first experiment, different neural network architectures and other non neural network solutions are compared in a small scale test using only deep coral dataset. This initial experiment helps to decide the most promising parameter configurations that are tested in a second experiment that uses all the datasets to validate the previous results.

These two experiments only compared the depthmap estimation stage, coarse and refined estimations. However, the final image dehazing was not compared or evaluated as the depth estimation can be singularly benchmarked. The final experiment focuses on comparing the results of the proposed depthmap estimation in a simple image dehazing system with other dehazing techniques from the literature that require the same input.

7.4.4. Experiment 1: Testing on deep corals dataset

The objective of this experiment is to decide the optimal neural network that will be used to estimate depthmaps in the rest of the datasets and validate the approach. In order to test the training of the network, 20 % of the images are randomly selected to test the system while training and detect possible overfitting of the data. The results presented correspond to the evaluation of the previously described metrics on this test samples.

Each neural network architecture has been trained until the test cost function showed no improvement for 100 epochs. The test error stabilized around 5000 to 6000 epochs, depending on the initialization and the onset of increase caused by the overfitting in the training set. For this reason, the training used early-stopping in order to obtain the maximum performance in images besides the training set.

The deep corals dataset was used in this initial experiment to decide which was the best possible architecture in terms of its difficulty. The deep corals dataset is the one with the greater quantity of different textures and depth variations, so it is a good environment to do reduced scale experiments that can be moved to a bigger dataset. Furthermore, using a simple dataset such as the rocks images could lead to specific solutions with non realistic results in a more generic case.

Different network architectures have been tested in this experiment. In the following a description of each of the implemented architectures can be seen:

- **Proposed:** The previously described architecture with 5 convolutional layers plus a guided filter for refining the estimation. The first convolutional layer reduces the information with a max polling with strides $5x5$ and before the cost function evaluation is upscaled back to the original size. The cost function is the one in equation 7.14.
- **Noguided:** This approach uses exactly the same neural network and training as the previously compared system but it does not use the guided filtering refining step. This approach shows the importance of the refining step and the improvement of the guided filtering.
- **Coarsefine:** In this case the neural network described in [Eigen et al., 2014] is implemented with a small variation to compare it with the same dataset of underwater images. Due to the image sizes it was not possible to use fully convolutional layers at the end of the first coarse networks. For this reason, these layers have been substituted with convolutional layers. Thus the architecture uses 6 convolutional layers that reduce the image size to a coarse estimation that is upscaled back to original size and convoluted with 3 additional layers with the original image in order to refine the estimation.
- **Coarsefine guided:** This network shares the same training as the previous one but it also makes use of the guided filter refinement used in the proposed methodology. This checks if the refinement step in the network proposed by [Eigen et al., 2014] can be further improved using a guided filter as proposed in the DCP techniques.
- **7 layers:** This approach increases the number of convolutional and pooling layers to 7. The main goal of this network is to see if a deeper solution helps to solve the problem or increases the overfitting problems.

Cuadro 7.3: Results of different neural network architectures on dataset 5.

Network	Corr.	RMSE(m)	RRMSE(%)	ARMSE(m)	RARMSE(%)
Proposed	**0.7909**	**0.1003**	3.9471	**0.0819**	**3.1481**
Noguided	0.7506	0.1099	4.2939	0.0933	3.5524
coarsefine [Eigen et al., 2014]	0.7101	0.1196	4.6446	0.1015	3.8186
coarsefine guided	0.7585	0.1074	4.2184	0.0877	3.3598
7 layers	0.7597	0.1089	4.1839	0.0923	3.5171
6 layers	0.7692	0.1101	4.3343	0.0894	3.4692
4 layers	0.6705	0.1399	5.8538	0.1044	4.0513
gradients [Eigen and Fergus, 2015]	0.7888	0.1015	**3.9469**	0.0834	3.2055
nostrides	0.7448	0.1207	5.0060	0.0878	3.3865
Dark prior [He et al., 2011]	0.0464	-	-	0.1893	7.4065

- **6 layers:** As in the previous case, this approach increases the number of layers to six in order to see the impact in the final results.
- **4 layers:** In this case, the number of convolutional and pooling layers is reduced in order to see if a smaller neural network is still capable of learning a valid solution with similar performance as the 5 layers version.
- **Gradients:** In this experiment the cost function used is that proposed in [Eigen and Fergus, 2015], which can be seen in equation 7.17, adding gradients to learn local structure. Forcing the gradients to be similar ensures that local structure, such as depth changes, is learned but greatly increases computational complexity introducing the gradient calculation for estimated depthmaps. Furthermore, the groundtruth does not provide a valid depth for every pixel thus making it impossible to compute the gradients in some points. For this reason a pre processing stage was added to compute the valid gradient points for each groundtruth image and accelerate the computation.
- **Nostrides:** This network maintains the image size through the convolutional layers configuring strides 1 in the operations. This is a hybrid between the coarsefine network and the proposed approach as it tries to obtain a refined estimation using the original image size but it does not downscale and upscale the image as in the proposed approach. This is also a slower neural network due to the higher feature maps managed by the network.
- **Dark Prior:** The dark channel prior presented in [He et al., 2011] is compared with the previous deep learning solutions obtaining a transmission estimation that can be compared in the correlation and adjusted metrics.

The results of this experiment are shown in table 7.3 for the different network architectures described. The results of the five proposed metrics are included for every alternative but the DCP, because it is not possible to compute unadjusted metrics as it directly estimates transmissions instead of depthmaps. The best results for each metric are highlighted in the corresponding column using bold numbers.

As can be seen, the best performing approach is the proposed methodology as it obtains the best results for each metric except the relative root mean squared error (RRMSE) that is 0.0002 % worse than the gradients approach. For this reason, it can be considered the best option for further validation experiments.

Regarding specific parameter comparison, the first noticeable thing is that using a guided filter helps to improve the results. Even if the network has a part devoted to it, as in the case of the coarsefine network, using the guided filter enhances the results. The proposed network and the coarsefine approach are enhanced when using a guided filter with the original image. Furthermore, the rest of neural networks also experience the same behaviour although these results are not included to highlight the most interesting findings.

In terms of the number of layers, 5 convolutional and pooling layers appears to be the optimum value as it obtains the best results in the metrics. When using less than 5 layers, the system is not able to model correctly the complexity of the problem and does not overfit independently of the number of epochs. However, the results achieved are far from those of the 5 layer approach. On the other hand more than 5 layers start to overfit when reaching the optimum, reducing the train error but increasing the test error that does not improve the proposed network results. For this reason 5 layers where kept as the optimum number of layers.

About the loss function, the use of gradients to force the learning of local structure has been tested. However, the results showed no benefit, obtaining similar results to the regular scale invariant cost function trained network. The main drawback of this technique is the increased training time of computing gradient errors, around 50 %, even with precalculated gradients for ground truth images. Furthermore, it also increased memory consumption by 400 %, due to the precomputed gradients and valid gradients in X and Y axis.

Similarly, as the nostrides solution works with the complete image through all the layers, the training time is much higher than the pooling version. However, according to the metrics used, this longer training time does not pay off in terms of results. It seems that using reduced feature maps helps the neural network to produce a higher quality coarse estimation that can be further refined to produce better results, despite the upscale required after the convolutions.

The dark channel prior results are added to compare the deep learning approaches with a state of the art dehazing method suitable for single image dehazing. It is only possible to obtain adjusted results because of the nature of the algorithm, that estimates the transmission instead of the depthmap. The ARMSE results are computed assuming perfect attenuation estimation, thus this is a best case for the DCP technique.

Taking this into account, the DCP is still far from the results achieved by deep learning solutions. The correlation metric shows the transmission estimated by the DCP is not linearly related to the depthmap when it should be, according to the transmission definition. Furthermore, the adjusted root mean squared error results are 200 % worse than those of the neural network ones.

Finally, the numbers achieved by the neural network are really good in terms of retrieving 3D information from a single image. The measured correlation for the test images, 0.7909, is a strong correlation between the estimation and the groundtruth, demonstrating the linear relation. Moreover, the absolute unadjusted mean error is around 10 centimeters, and the relative is slightly under 4 %, which means it is possible to retrieve a really precise depthmap with just a single image. The scale adjusted error is above 8 centimeters that means a 3.15 % error which is also an impressive estimation from a single image

Comparing these results with the ones in [Eigen et al., 2014] that achieved a

Cuadro 7.4: Results of different neural network architectures using all the datasets

Network	Corr.	RMSE(m)	RRMSE(%)	ARMSE(m)	RARMSE(%)
Proposed	**0.8181**	**0.1293**	**5.2632**	**0.0982**	**3.6668**
Noguided	0.7878	0.1430	5.6955	0.1141	4.2351
coarsefine [Eigen et al., 2014]	0.7047	0.1653	6.0099	0.1342	4.5998
coarsefine guided	0.7447	0.1508	5.5593	0.1169	3.9949
nostrides	0.6904	0.1816	7.3358	0.1214	4.6501
Dark prior [He et al., 2011]	0.0737	-	-	0.2788	10.8783

21 % unadjusted error in open air images, proves the underwater haze is a strong depth cue that can be used to estimate depth. Using a really similar neural network architecture produced much better results. For this reason, using a deep learning solution to extract 3D information is an interesting alternative not only for dehazing but for any other application that requires a depth estimation from a single camera.

7.4.5. Experiment 2: Validation with the complete dataset

However, the previous results were only for images from a single location, and the neural network is specialized in images from that specific location characteristics such as, water color, textures, illumination... So, trying to estimate depth in images from another location will probably fail. Consequently, a more generic experiment must be conducted to analyse the capabilities of 3D estimation.

In this experiment, the five datasets with reliable depth information have been used to train a generic network capable of estimating depth from any underwater image. For this purpose, the neural network has been trained and validated using the previously described procedure.

Taking into account the previous results, the neural network has been trained using all the datasets in a second experiment, but only for the most promising architectures. The four, six and seven layers architectures have been discarded as the previous experiment showed five layers were enough to model the problem without overfitting. Additionally, the tests with different cost functions have also been discarded as the gradient computation was too computationally expensive and without any beneficial results.

The results obtained can be seen in table 7.4. As in the previous experiment, the complete metrics results are showed for proposed, noguided, coarsefine, coarsefine guided, nostrides and DCP. The best results for each metric are highlighted in bold numbers.

As can be seen the proposed architecture offers the best results when all the datasets are mixed. Furthermore, the proposed deep learning solution clearly achieves the best results for each metric. All the compared alternative results are considerably worse than the one presented in this section.

As happened in the previous experiment, the guided filter is proved to be a good filtering alternative to an additional convolutional network. The cause of this may be the additional convolutional layers to be trained make it more difficult to learn the proper weights and biases thus requiring a bigger amount of data to become properly trained. In any case, the guided filter has already a good performance thus there is no need to train an additional network.

Another interesting result is that the network which does not reduce the image size, labelled as nostrides, is the worst performing. Probably the need to work with the whole image makes it difficult to learn a convenient function. When the range of different images that are used as input increases, as in this experiment, generalizing with bigger images gets more difficult. As a consequence, the performance of this solution notably decreases with respect to the previous experiment.

In the case of coarsefine, the results are still worse than the proposed architecture. The addition of images from different locations did not affect the results with respect to the proposed solution. Nevertheless, using a guided filter after the neural network still enhances the results proving the refinement step is not properly working, or at least, it could work better.

In comparison with the previous results corresponding to only one dataset the RMSE error increases significantly due to the difficulty of choosing the right scale. In the previous experiment, the difference of this error with respect to the adjusted metric was really small. Using images from different locations showed choosing the right scale is more challenging, increasing from 3.9 % error to 5.2 %.

Nevertheless, this results are still far from the achieved in open air images where the light attenuation does not provide a strong depth cue. Errors in that case are close to 20 % while the use in an underwater environment reduces it in 400 %. However, it is important to remember that the underwater images used only show seafloor images in small distance ranges compared to the open air use case, for this reason comparing absolute errors is not valid.

On the other hand the adjusted metric shows a slight increase caused by the different sources of images in this experiment. The adjusted error increases from 3.1 % to 3.6 % which in absolute terms is around 10 centimeters, still a very accurate estimation for many applications. Because in this experiment there is a greater range of textures, water characteristics, etc the difficulty has increased and the errors have increased.

Regarding the correlation, surprisingly the results show an increase from 0.79 to 0.81. This is a really good result as the estimated depthmap is highly correlated with the groundtruth demonstrating the estimation is very similar in terms of structure.

Finally dark channel prior estimation results are also far from the ones achieved by neural networks. This is especially true in the correlation metric where dark prior shows no correlation with depthmap while the learning solution shows a strong relationship. This proves that although the DCP may produce visually pleasing images it is not a good depthmap estimator, so it should not be used for other purposes. This result is also caused by the fact that the DCP is sometimes directly related and some others is inversely related causing the mean to be close to zero.

Additionally to numeric results it is also possible to display histograms for the relative errors that show the performance difference with respect to other techniques. In figure 7.15 adjusted errors and relative errors for each pixel in each image are depicted in a histogram chart comparing DCP, blue color, and neural network, red color.

These histograms show the amount of pixels that have an error within a range. In order to compute this, all the pixels in every validation image have been used to count the number of pixels that have an error within a specific range. The optimal case should show 100 % pixels with zero error.

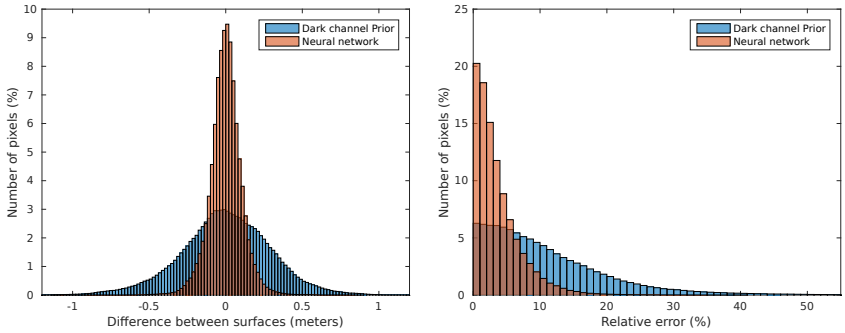

Figura 7.15: Error histograms, ARMSE (left) relative ARMSE (right), for all the images pixels comparing neural network and dark channel prior.

The histogram on the left represents the percent of pixels of every validation image in the y axis and the depth estimation error in the x axis. The histogram on the right corresponds to the relative error instead of absolute error. As can be seen the neural network obtains much better results showing a smaller deviation in results.

Both algorithms show a gaussian distribution of the error around the zero as it is an adjusted metric. However, the neural network has almost no errors below 0.5 meters or 20 % while DCP may obtain errors over 1 meter or 50 %. This means, the neural network is also much more robust than the DCP.

7.4.6. Experiment 3: Dehazing comparison

Finally, the last experiment focuses on an application of the 3D estimation: single image dehazing. The previously described depthmap estimation can be used in other applications that do not require a high precision estimation but would require an estimation for every pixel in the image or an estimation from a mono camera. Some applications that would be feasible are:

- 3D stereo completion: The stereo cameras usually retrieve an incomplete estimation of 3D when the objects are not textured enough to find matches between both cameras or are out of the range. Using a deep learning solution to estimate the pixels where the stereo system was unable to obtain an estimation may be a nice feature to have. Futhermore, the system could be simultaneously trained using the pixels with valid information, adapting to the current location.

- Basic obstacle avoidance: Although the depth estimation is not precise enough to directly allow object manipulation it should be sufficient for obstacle avoidance. Many budget ROV's or platforms lack of stereo cameras and rely on a single camera to navigate. A 3D estimation from single image would allow possible obstacles to be detected and automatically avoid them while moving close to the seafloor.

The performance of the image restoration from the neural network 3D estimation can be seen in figure 7.16. In this figure one representative image and groundtruth depthmap of each dataset can be seen with the results of depthmap estimation and an image dehazed using the neural network estimation.

As can be seen the groundtruth and estimated depthmaps are really similar in appearance and local structure. The main difference is the estimation has less details than the groundtruth depthmaps that may detect smaller changes in depth.

In figure 7.17 and figure 7.18 the dehazing results are compared with other state of the art techniques. The figure shows the raw image, neural network dehazing, restoration from [Bryson et al., 2016] as groundtruth, histogram equalization, ACE and CLAHE. Although other dehazing alternatives may produce better results, the proposed method is only compared with dehazing solutions that share the same inputs.

As can be seen the proposed approach produces visually pleasing images similar to the groundtruth restoring technique of the previous section. The methodology was able to successfully restore the colors for images of every dataset showing it is capable of generalizing and producing good results in different environments.

Comparing with other techniques the results are similar to the direct underwater dehazing neural network. The histogram equalization obtains overcorrected images with colors that are too bright and showing the effects of attenuation. For instance in the rocks example images, the parts that are closer to the camera are reddish while the parts further from it are bluish. However, depending on the image it is able to produce good results such as the medium corals sample.

In the case of ACE the algorithm is not able to remove completely the original colors of the water, thus images for rocks, rocks and sand and deep corals have a greenish tone and medium and shallow corals look bluish.

With the CLAHE dehazing, the results are similar but the images look darker than the ACE algorithm. On the other hand the proposed approach is able to remove this water color in most of the images while dehazing the inputs.

Finally, it is important to remember the dehazing algorithm uses the DCP approach. However, other dehazing processes may make use of the depthmap in a different way to correctly estimate attenuation and other important parameters to achieve better results. But in this work, a simple fast solution that is easy to compute was chosen so it can be carried out it in real time.

7.5. Results discussion

In the two previous sections, two deep learning solutions have been presented in order to dehaze underwater images from still images. Both of them have their own advantages and drawbacks. In the case of direct dehazing, resulting images may have colors different from the original image. On the other hand, dehazing using depth estimation from a neural network does not completely remove the water haze color.

Figure 7.19 shows this situation. Although both methods produce results beyond the state of the art in the field of single image dehazing, none of them is perfect. Images are greatly enhanced from the raw acquired input making it possible to use them in a real time robot vision pipeline as a preprocessing step.

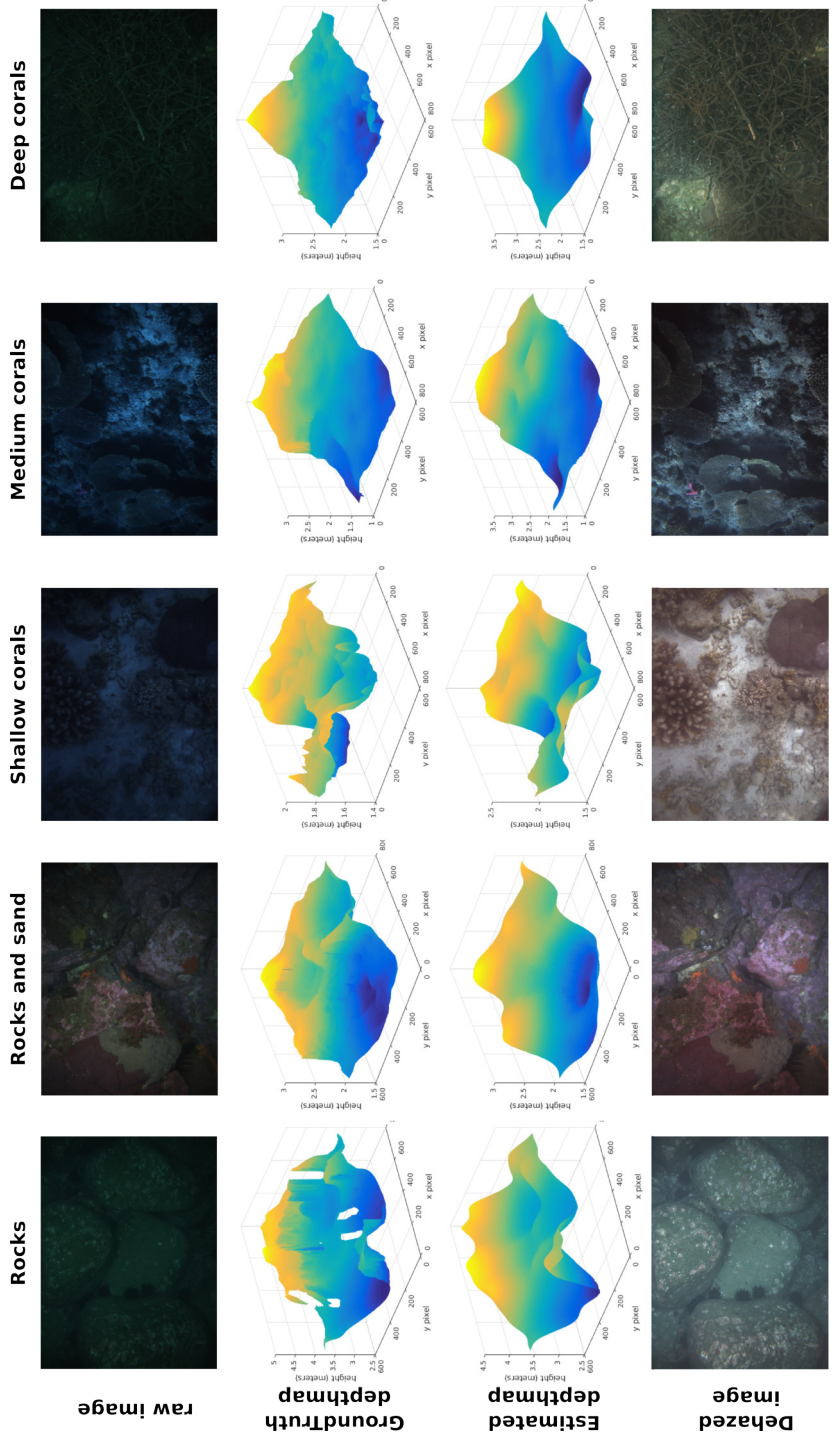

Figura 7.16: Neural network results applied to image dehazing, from left to right dataset images, from top to bottom raw image, groundTruth depthmap, neural network estimated depthmap and dehazed image.

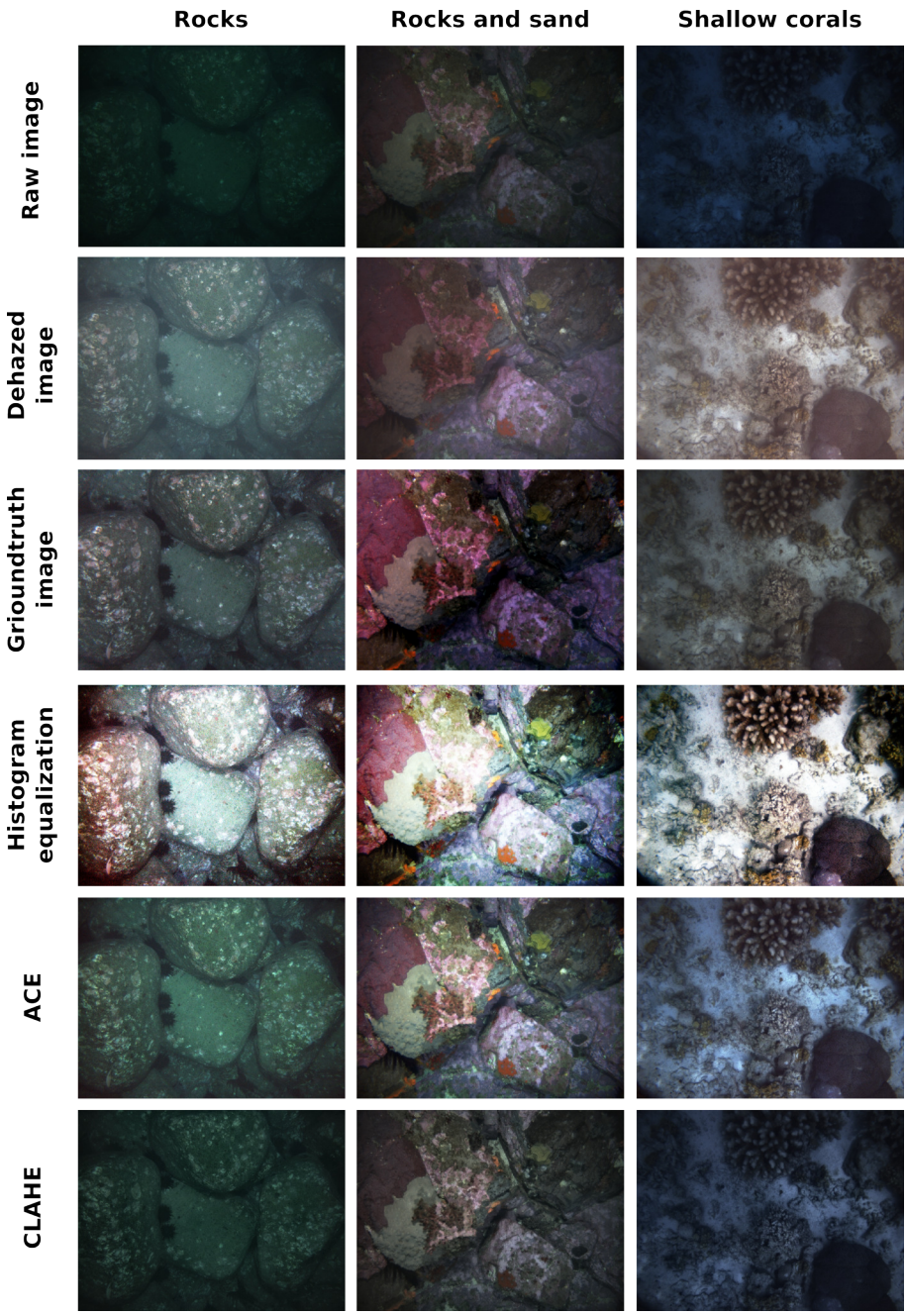

Figura 7.17: Image dehazing results using deep learning depth estimation compared with other still image dehazing techniques (part 1).

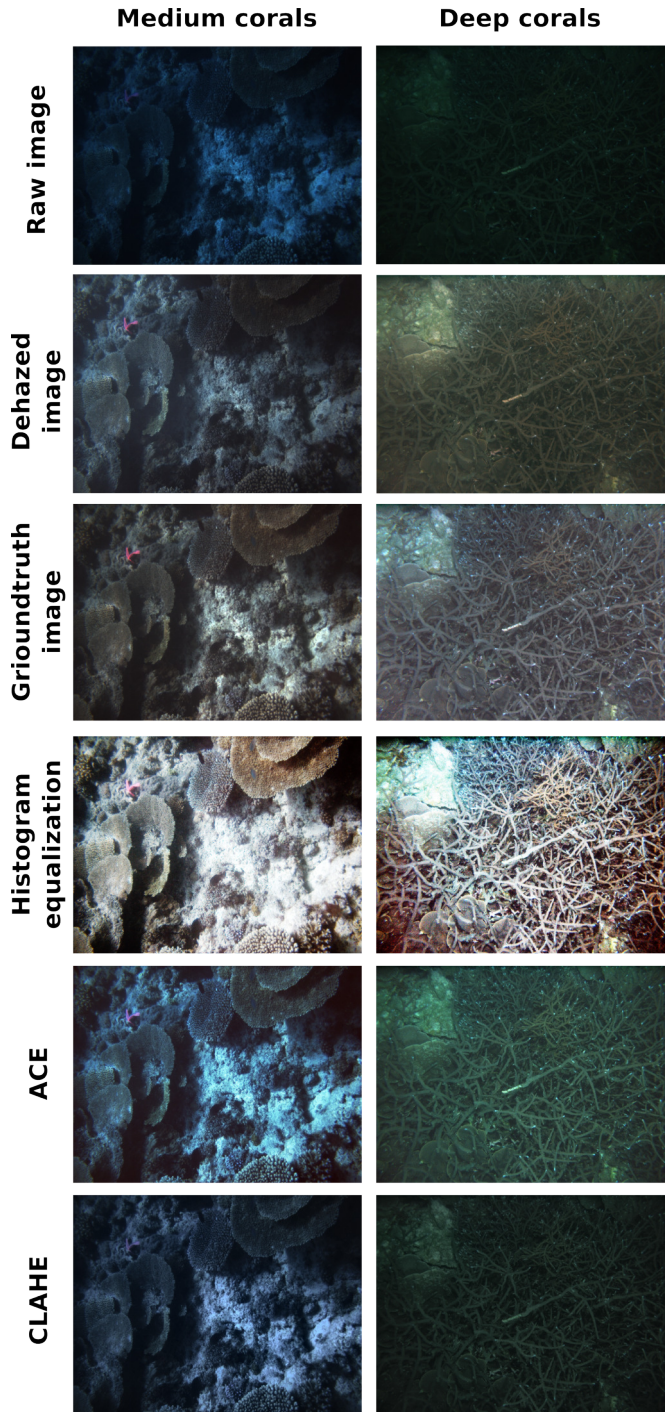

Figura 7.18: Image dehazing results using deep learning depth estimation compared with other still image dehazing techniques (part 2).

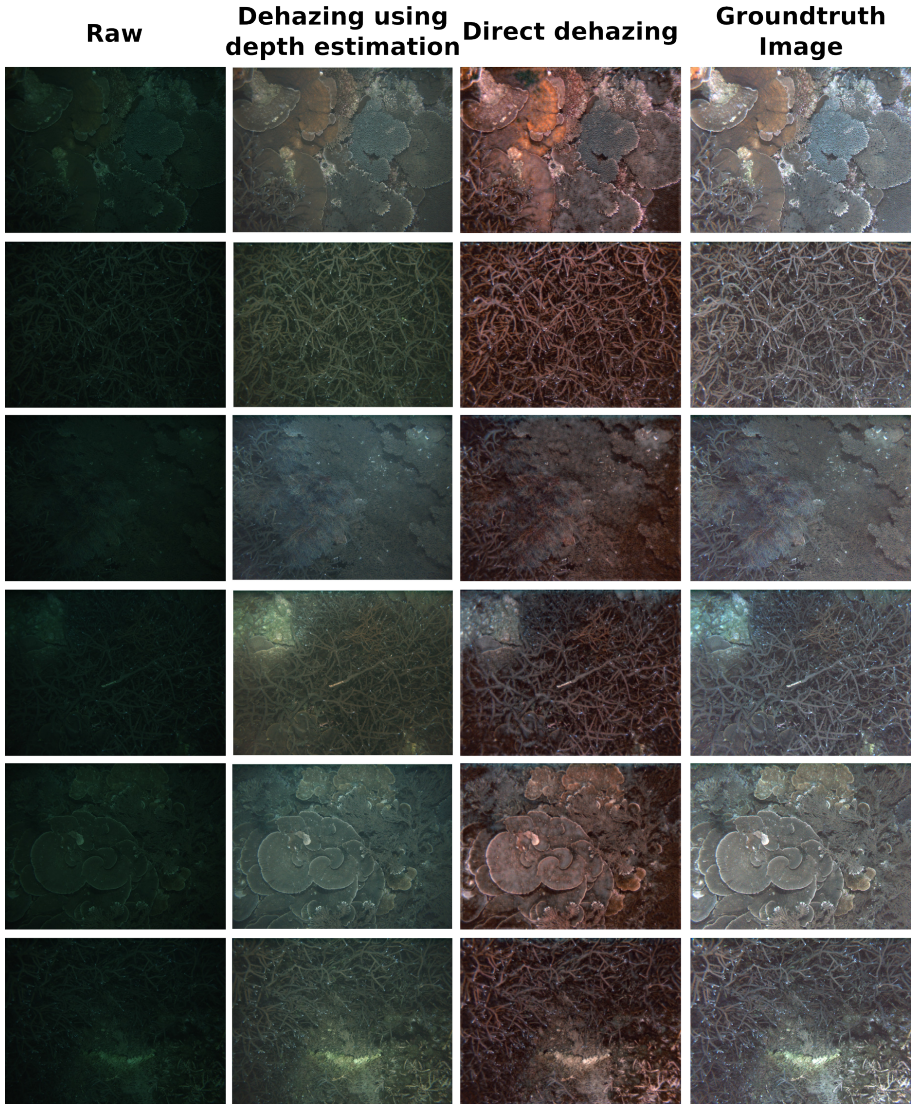

Figura 7.19: Comparison of the single image deep learning dehazing methods proposed in this chapter.

Taking this into account, the proposed solutions may have different applications. When it is not necessary to produce very realistic colors, but a haze-free image is required, direct dehazing is a better option. However, in the cases where real colors are important dehazing through depth estimation produces better results.

Nevertheless, both methods can be improved through training with ground-truth images acquired in a bigger number of different locations, allowing the neural network to generalize even more.

Besides this improvement, depth estimation may improve using additional information such as sonars or stereo cameras, and the dehazing can be further improved using the estimated depth information. On the other hand, direct dehazing can also make use of the depth information to produce more precise results.

Unfortunately, the use of this additional information to improve the results is out of the scope of this thesis, that focuses on dehazing with the minimum information.

7.6. Conclusions

In this chapter two new approaches for single image dehazing using deep learning have been presented. The approaches are based on emerging methodologies of deep learning and convolutional neural networks capable of learning complex transformations and producing results beyond the state of the art of classic algorithms.

The two new approaches are inspired by the main two points of view of underwater images processing: image restoration and image enhancement. In the image enhancement approach a direct transformation from the raw image to a dehazed image is learned using a state of the art image restoration technique as groundtruth. The image restoration approach uses a depthmap gathered with stereo cameras to learn the estimation of relevant parameters and dehaze the image using the image formation model.

However, direct underwater dehazing using deep learning should not be considered a full image enhancement technique because it learns from a restoration method. The neural network is trained using restored images from an approach that uses a big set of images from the same location, depth information and the image formation model. For this reason, the neural network is capable of learning an estimation of this transformation and produce visually pleasing images.

The results show this solution outperforms other state of the art alternatives such as histogram equalization, Automatic Color Enhancement (ACE) or Contrast Limited Adaptative Histogram Equalization (CLAHE). Moreover, the deep learning solution is able to learn other image enhancements such as vignetting removal and simultaneously apply them to the input images. Furthermore, the proposed neural network is much faster due to the GPU processing of convolutional operations allowing it to run in real time environments.

The option of image restoration presented focuses on estimating the depthmap of the image which is a key parameter to accurately dehaze the image. The main idea is that the haze is a strong depth cue that can be used to estimate the depth at each point similar to DCP or other heuristics. Furthermore, the state of the art image transmission estimation operations can be modelled as convolutions with fixed weights and biases, thus training the neural network will naturally discover

them if they are the optimum estimators.

The results show it is possible to estimate the 3D structure of an underwater still image with a fairly good error close to 4 %. These results are far from those obtained in open air images where the haze does not provide reliable depth information. This proves the light attenuation underwater can be used to extract the depthmap and use it in other applications such as completing stereo depthmaps, basic obstacle avoidance or image dehazing.

Finally, the estimated depthmap is used in an image dehazing application showing it is possible to obtain better results using a good parameter estimation. However, it would be possible to enhance the results even more using a more complex dehazing algorithm that exploits the depthmap estimation to characterize the water characteristics such as attenuation, backscattering or ambient light.

Capítulo 8

Conclusions

This chapter concludes the thesis presented in this document. It first comments on the main contributions of the research summarizing the most relevant contents in each chapter and describing the achieved results. Additionally, it points towards interesting future lines of work that have not been addressed throughout the thesis. Finally, the publications related to this work are listed.

The current technology of underwater interventions requires a great deal of development in order to satisfy the needs of those working in this environment. In order to perform an intervention in the ocean, the robot has to perform several steps correctly. The first is the scene interpretation, the robot needs to perceive the surroundings through all the possible sensors in order to understand its situation in the world. After that, the robot needs to decide the best possible action according to its interpretation of the world to achieve the specified goal. Finally, the robot has to perform the action while monitoring the environment for possible changes or unexpected situations. This thesis focuses on the first step, the interpretation of the scene, as it is a key process that determines subsequent actions.

Furthermore, the images acquired in underwater robotics are affected by different phenomena such as attenuation or scattering that degrade them making even more difficult to understand and, as a consequence, decide what action should be taken with regard to the information contained in them. However, humans are capable of correctly interpret these degraded images and can even form an idea of how the images would look like out of the water. For this reason, a dehazing approach is implemented in this work, restoring the original appearance of images in order to make the scene easier to understand.

However, the first step in this work corresponds to the design and implementation of a suitable framework for underwater robotics research. A software capable of simulating underwater interventions is a useful tool for the to study and analysis of the influence of underwater visibility on other vision algorithms. Consequently, the development of an underwater simulator, UWSim, is described in Chapter 2. This software is not only designed for simulation but supervision and monitoring when the vehicle goes deep into the sea and it is not possible to have a direct view of the system.

Moreover, the capacity of evaluating the robot actions with respect to a given metric or groundtruth is of the utmost importance. So a benchmarking suite is

also presented in Chapter 2. The suite allows the results of simulated as well as real interventions to be measured, obtaining a valuable log of results that can be analysed.

These tools have also been made accessible through internet using a cloud service as described in Chapter 3. This service makes possible the use of the tools without previous installation and using any device able to connect through a web browser.

These tools have been extensively used in various applications besides the study of underwater visibility. Chapter 4 includes a summary of the most interesting applications of the developed benchmarking software. In first place, the characterization of the TRIDENT experiments is described, explaining how the experiments were supervised and replayed in the simulation tool and afterwards the benchmark of both interventions is shown. Secondly, a simulated dredging intervention is evaluated through the benchmarking suite. Finally, the software is used to compare different devices to control autonomous underwater vehicles.

In Chapter 5 the previously developed software is used to analyse the performance of various types of vision software in decreasing visibility conditions. Basically three types of algorithms are evaluated: trackers, station keeping and 3D reconstruction. The results show the water turbidity greatly influences the performance making it necessary to discard some of the compared algorithms that may be feasible in air conditions. Furthermore, as the water turbidity increases, the precision and reliability of the compared solutions decreases, showing its importance.

Motivated by this results, a comparison of single image dehazing algorithms is presented in Chapter 6. Dehazing solutions are discussed from the perspective of autonomous underwater vehicles: real time performance and single image input. The conclusion is that none of the state of the art alternatives is suitable for the case under study. The solutions that achieve the best performance are too slow or require a big amount of information not available at the intervention time while the rest do not work in every situation.

Finally, in Chapter 7 two dehazing approaches are proposed for enhancing the results in autonomous underwater vehicles. Due to the recent enormous increase of deep learning solutions in the field of image processing, results far beyond the state of the art of classical algorithms have been obtained. The proposed approaches use deep learning to process the images. In the first case a direct transformation is learned from the raw image to the desired dehazed output of a slower image restoration technique. The second case uses deep learning to estimate the required parameters to use an image formation model that restores the original colors.

8.1. Contributions

The main goal of the thesis has been fulfilled developing a dehazing approach capable of enhancing the images for real time applications from a single image. Furthermore, the problem has been solved from two points of view obtaining promising results. The proposed approach is simple to use and can be extended by training with new images to produce better results. However, in the research for this goal various research contributions were achieved. These contributions are listed below:

- **AUVs simulator, UWSim:** A simulator for autonomous underwater vehicles have been developed. The simulator is offered as an open source tool available for researchers to develop AUV's. It is also possible to use the software to supervise interventions providing an integrated view of the robot sensors when it is underwater. The tool has a medium sized community as some European Community funded projects have decided to use it and at least 56 universities or institutes have used UWSim.

- **Generic benchmarking platform:** An abstract suite for evaluating and comparing algorithms have been developed. The tool makes it possible to evaluate through configured metrics any type of intervention in an objective manner using ROS as middleware. The suite has been extensively used through the thesis to evaluate vision algorithms, control, real interventions and even interfaces. Furthermore, the generic nature allows it to be used in simulated, hardware in the loop and real experiments.

- **Cloud simulation and benchmarking:** Besides the local simulation and benchmarking software, cloud services have been developed to provide remote simulation and benchmarking capabilities. This allows experiments to be launched with any device connected to internet independently of its processing power. The service also provides a comparison interface where researchers can directly compare the results of their experiments. As an application example, the cloud services have been used in education where students participated in a competition to obtain the best possible results in path following.

- **Analysis of the effects of turbidity:** The benchmarking module has also been used to study the consequences of underwater image degradation in vision algorithms. The study reveals that turbidity is a major issue that may make it impossible to correctly find an object or reconstruct in 3D. Furthermore, some alternatives such as the laser stripe projector are more interesting in the underwater environment than stereo reconstruction due to the immunity to haze.

- **Benchmark of single image dehazing algorithms:** A review of suitable techniques for single image dehazing on real time is provided. The study is made with objective metrics that prove none of the reviewed options offers precision and robustness at the same time. The precise methods require too much computation time or inputs to be used in real time strategies while faster methods do not work in every situation.

- **Deep learning single image dehazing:** Two different approaches for single image dehazing are proposed. In the first case, a direct enhancement neural network is trained using restored images from other slower algorithms that require several inputs. The trained neural network is able to generalize and produce dehazed images from degraded images never seen in the training stage producing results beyond the state of the art. The second alternative estimates the depth map from single RGB images using the haze as a depth cue. Using this estimation it is able to restore the colors of the image.

- **Underwater 3D estimation from single image:** As a side result from the single image dehazing proposed approach 3D estimation, a 3D depthmap estimation is obtained. The results show the relative error of this estimation is around 4 % demonstrating that underwater image degradation provides very useful information for this task. Furthermore, this fact can be used in other applications beside image dehazing as discussed in the next section.

8.2. Future lines

This thesis has produced several related publications dealing with simulation, benchmarking, underwater visibility and dehazing. However, every aspect presented in this thesis can be improved and there are still several open questions that can expand the research presented.

Autonomous underwater simulation

Although the developed simulator UWSim is used in several research institutions and universities numerous features could be added or improved. The simulator offers dynamic simulation of vehicles, taking into account water physics and external multibody simulation, but makes it necessary to create the vehicle characteristics and relies on external software for the multibody simulation. For this reason a generic complete dynamic simulation should be addressed as part of the simulator to help researchers with autonomous navigation.

New sensors and devices are continuously being developed in the robotics field, for this reason the maintenance of a simulator involves constantly developing new simulated devices. Furthermore, the implementation of some of the simulated sensors can be improved, for instance adding housing aberrations to cameras, camera motion blur or noise in arm joint articulations.

The implementation of interactive markers in OSG is still incomplete, the basic functionality of movement, spawning or deleting is available but advanced features such as menus are still not developed. This would help with the integration with ROS which has become a standard de facto in robotics and allowing the compatibility with software developed for other purposes.

Improving the supervision features is also an interesting line of work as the simulator can be used to monitor a real intervention. The simulator already shows the frames of the different objects in the scene, but it would be interesting to be able to add more objects via ROS transforms so it would be possible to easily detect errors in the intervention more easily. Another interesting addition in this line of work is being able to measure distances between objects or pointclouds.

Finally, the simulator lacks a simple interface that makes it more user friendly. Even though most actions can be done through keys (i.e. showing cameras), or console commands (i.e. moving vehicles), the development of a user interface to perform simple actions would be an important addition.

Cloud simulation and benchmarking

Besides the development of the simulation capabilities, the online features are also an interesting future line of work. As discussed in Chapter 3, the cloud services

have many advantages allowing experiments to be launched from any device without installation or configuration. For this reason, extending the current capabilities is an important step that some simulators are already working on.

The main issue of the current implementation lies on the reduced set of environments accessible through the web interface. Only a small group of scenes are available and nowadays is not possible to remotely configure them. A simple way to include new scenes, vehicles and models is needed to increase the usability of the system.

The benchmarking experiments need to be manually created. An automatic way to create new benchmarks that can be shared with other researchers could be interesting so that a common place to compare the results could be provided. An example of this situation is the pipe following educational application previously described, where students entered a competition to improve their software.

The dehazing strategies described in Chapters 6 and 7 should be benchmarked together with the tracking and station keeping algorithms, so they can be analysed in detail. This would prove the dehazing approaches proposed are capable of enhancing autonomous underwater vehicles.

Single image dehazing

In Chapter 7 two deep learning single image dehazing strategies are proposed. Even though both strategies are tested in real dataset images it would be interesting to test them in a real intervention when the vehicle is moving in a new environment.

Furthermore, both neural networks are trained with a relatively small set of images, in order to make the system more robust it would be necessary to train with larger sets of images that include several locations. This would make the neural network more general and capable of obtaining good results in any possible location.

Additionally, the use of stereo cameras or sonar information is common in the underwater robotics field. However, this information is not used in the deep learning approaches presented. Introducing, depthmaps from stereo images or sonar information to the neural network will probably improve the results obtained. Different neural networks architectures can be tested using information from other sensors that can help in the dehazing problem.

Underwater depthmap estimation from single image

As a required parameter for the image dehazing the second neural network proposed in Chapter 7 obtains a depthmap from single images. The results show that the obtained depthmap is far more precise than that obtained in air images using a similar network architecture. This demonstrates the light transmission in water can be used to estimate a depth estimation just using a single image.

This finding can be used in other applications besides dehazing. For instance it is possible to fill the gaps in a stereo reconstruction when there is not enough texture to find matching pixels in both cameras. Furthermore, it would be possible to retrain with the existing depth pixels in the camera and obtain the rest of the estimation through a neural network.

Although the precision of the depthmap estimation does not allow precise manipulation of objects it would be possible to navigate using mono cameras. The estimated depthmap is accurate enough to produce rough information about obstacles, that may be sufficient for safe navigation.

Finally, it is possible to use it in any application that requires a depth estimation for every pixel in the camera but which does not require a high precision.

8.3. Publications

The work in this thesis has been previously published in the following journal and conference papers:

1. Perez, J., Sales, J., Penalver, A., Fornas, D., Fernandez, J. J., Garcia, J. C., Sanz, P. J., Marin, R., and Prats, M. (2015). Exploring 3-d reconstruction techniques: A benchmarking tool for underwater robotics. *IEEE Robotics & Automation Magazine*, 22(3):85-95.

   Impact Factor 2015: 1.822

2. Perez, J., Sales, J., Penalver, A., Fernandez, J. J., Sanz, P. J., Garcia, J. C., Marti, J. V., Marin, R., and Fornas, D. (2015). Robotic manipulation within the underwater mission planning context. In *Motion and Operation Planning of Robotic Systems*, pages 495-522. Springer.

3. Sanz, P., Perez, J., Sales, J., Penalver, A., Fernandez, J., Fornas, D., Marin, R., and Garcia, J. (2015). A benchmarking perspective of underwater intervention systems. *IFAC Workshop on Navigation, Guidance and Control of Underwater Vehicles, Girona (Spain)*, 48(2):8-13.

4. Perez, J., Sales, J., Marin, R., and Sanz, P. J. (2014). Online tool for benchmarking of simulated intervention autonomous underwater vehicles: Evaluating position controllers in changing underwater currents. In *Artificial Intelligence, Modelling and Simulation (AIMS), 2014 2nd International Conference on*, pages 246-251. IEEE.

5. Perez, J., Sales, J., Marin, R., and Sanz, P. J. (2014). Web-based configuration tool for benchmarking of simulated intervention autonomous underwater vehicles. In *Autonomous Robot Systems and Competitions (ICARSC), 2014 IEEE International Conference on*, pages 279-284. IEEE.

6. Perez, J., Sales, J., Prats, M., Marin, R., and Sanz, P. J., (2013). Benchmarking on uwsim, an open source tool for developing and evaluating underwater robots experiments. In *Workshop on Human Robot Interaction (HRI), 2013 IEEE International Conference on Robotics and Automation (ICRA'13), Karlsruhe, Germany.*

7. Sanz, P., Perez, J., Penalver, A., Fernandez, J., Fornas, D., Sales, J., and Marin, R. (2013). Grasper hil simulation towards autonomous manipulation of an underwater panel in a permanent observatory. In *Oceans-San Diego, 2013*, pages 1-6. IEEE.

8. Perez, J., Sales, J., Prats, M., Marti, J. V., Fornas, D., Marin, R., and Sanz, P. J. (2013). The underwater simulator uwsim-benchmarking capabilities on autonomous grasping. In *ICINCO* (2), pages 369-376.

9. Prats, M., Perez, J., Fernandez, J. J., and Sanz, P. J. (2012). An open source tool for simulation and supervision of underwater intervention missions. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 2577-2582

10. Perez, J, Fornas, D., Marin, R., and Sanz, P. J., (2016). UWSim, un simulador submarino como herramienta educacional. In *XXXVII Jornadas de Automatica, Madrid*.

11. Sanz, P., Perez, J., Sales, J., Fernandez, J., Penalver, A., Fornas, D., Garcia, J., and Marin, R. (2015). Mermanip: Avances recientes en la manipulacion autonoma cooperativa submarina. In *XXXVI Jornadas de Automatica, Bilbao*.

12. Perez, J., Sales, J., Marin, R., Cervera, E., and Sanz, P. J.(2014). Configuracion y ejecucion de benchmarks de intervencion robotica submarina en uwsim mediante herramientas web. In *XXXV Jornadas de Automatica, Valencia*.

13. Sanz, P. J., Fernandez, J. J., Perez, J., Penalver, A., Garcia, J. C., Fornas, D., Sales, J., Bernabe, J. Antonio, and Marin, R. (2013). GRASPER: Un Proyecto Dirigido a Incrementar la Autonomia de la Manipulacion Submarina. In *XXXIV Jornadas de Automatica, Terrassa*.

Furthermore, it is scheduled for publication or under consideration in the following conference or journal papers:

1. ACCEPTED: Perez, J, Sales, J., Penalver, A., Fernandez, J.J., Fornas, D., Garcia, J. C., Marin, R., Sanz, P.J (2017).Benchmarking water turbidity effect on tracking algorithms. In *20th IFAC World Congress, Toulouse*.

2. ACCEPTED: Perez, J., Bryson, M., Williams, S.B, Sanz, P.J., (2017). A benchmarking study on single image dehazing techniques for underwater autonomous vehicles. In *Oceans-Aberdeen, 2017*, IEE.

3. ACCEPTED Perez, J., Attanasio, A.C., Nechyporenko, N., Sanz, P.J (2017). A Deep Learning Approach for Underwater Image Enhancement. In *International Work-Conference on the Interplay Between Natural and Artificial Computation, IWINAC 2017*.

4. Perez, J., Fornas, D., Marin, R., Sanz, P.J., (2017*). UWSim, un simulador submarino conectado a la nube como herramienta educacional. In *Revista Iberoamericana de Automática e Informática Industrial RIAI*. Elsevier.

5. Perez, J., Bryson, M., Williams, S.B, Sanz, P.J., Recovering depth from still images for underwater dehazing using deep learning. In *to be determined*.

Apéndice A

Image datasets for dehazing

Through the chapters 6 and 7 a group of image datasets is repeatedly used to test and compare different image dehazing alternatives. These datasets were acquired by the AUV Sirius [Williams et al., 2012] of the Australian Centre for Field Robotics in the University of Sydney, Australia who also provided and prepared the data.

Each image dataset consists of a group of images taken at the same survey in specific locations. Along with each image a dense stereo depthmap has also been gathered providing information about the depth of almost every pixel. Additionally, a restored image using the methodology described in [Bryson et al., 2016] is also included showing a haze free image derived from the raw image acquired by the camera.

The 3D information was retrieved using stereo cameras that were post-processed and combined with adjacent image pairs to produce a feature-based stereo depth map as described in [Johnson-Roberson et al., 2010]. In order to increase the accuracy of the depth computation, the images are processed using adjacent pairs to correct the light using a form of "gray world" transformation. Using these corrected images increases the number of feature matching stereo pairs in the underwater environment, increasing the amount of 3D points.

Then they are reprojected back into each camera, with a spatial resolution of 2.5 cm, and sub-centimeter depth accuracy, based on analysis of residual feature errors. However, due to the characteristics of the 3D estimation not every image has a completely reliable depthmap available. For this reason, some of the datasets were not used depending on the needs of the experiment such as the kelp dataset for depth estimation.

In order to try different environments, illumination, textures and water conditions six different datasets have been created to use in the thesis. This is necessary to benchmark the dehazing algorithms in different conditions and generalize in the deep learning approaches. The only thing in common is that the images are taken in an AUV and are pointing at the seafloor. The specific details of each dataset are shown in table A.1.

As can be seen the data has been taken at different depths ranging from 2 meters in the case of the shallow corals to 27 meters. The image range is the maximum and minimum distance in the dataset images, the objects which are closer and further away in the dataset. The mean distance is the average distance of objects in the

Cuadro A.1: Description of datasets used in the thesis.

Description	images	depthmaps	depth	image range	mean distance	strobes
Rocks	60	19	27 m	2.18 - 4.78 m	3.64 m	Yes
Kelp	60	0	27 m	-	-	Yes
Rocks sand	94	94	27 m	0.46 - 4.17 m	2.29 m	Yes
Shallow corals	100	100	2 m	0.34 - 2.73 m	1.91 m	No
Medium corals	99	99	7 m	1.06 - 4.77 m	1.50 m	No
Deep corals	100	100	18 m	0.97 - 4.83 m	4.03 m	Yes

image, for instance in the medium corals dataset the vehicle is closer to the seafloor, 1.5 meters average distance, but in deep corals the vehicle flies higher. Finally the strobes column indicates the use of artificial lightning mounted in the vehicle.

Images were acquired by the AUV Sirius [Williams et al., 2012] at five different field locations across Australia. The first three datasets of rocks, kelp and rocks and sand were acquired over boulderfields at St Helens, Tasmania. The resting datasets, corals at different depths, were acquired over coral reefs at One Tree Island (shallow corals) and Heron Island (medium corals), on the southern Great Barrier Reef and Houtman Abrolhos Islands, Western Australia (deep corals). All images were captured using a calibrated stereo-pair consisting of two prosilica 1.3 MPix cameras. Given the depth in the first three datasets and deep corals, artificial lighting was provided by two xenon strobes mounted to the AUV, in the case of shallow and medium depth corals they were captured in shallow waters and illuminated by sunlight.

In the following pages several images of each dataset are included to show the different environment and their characteristics.

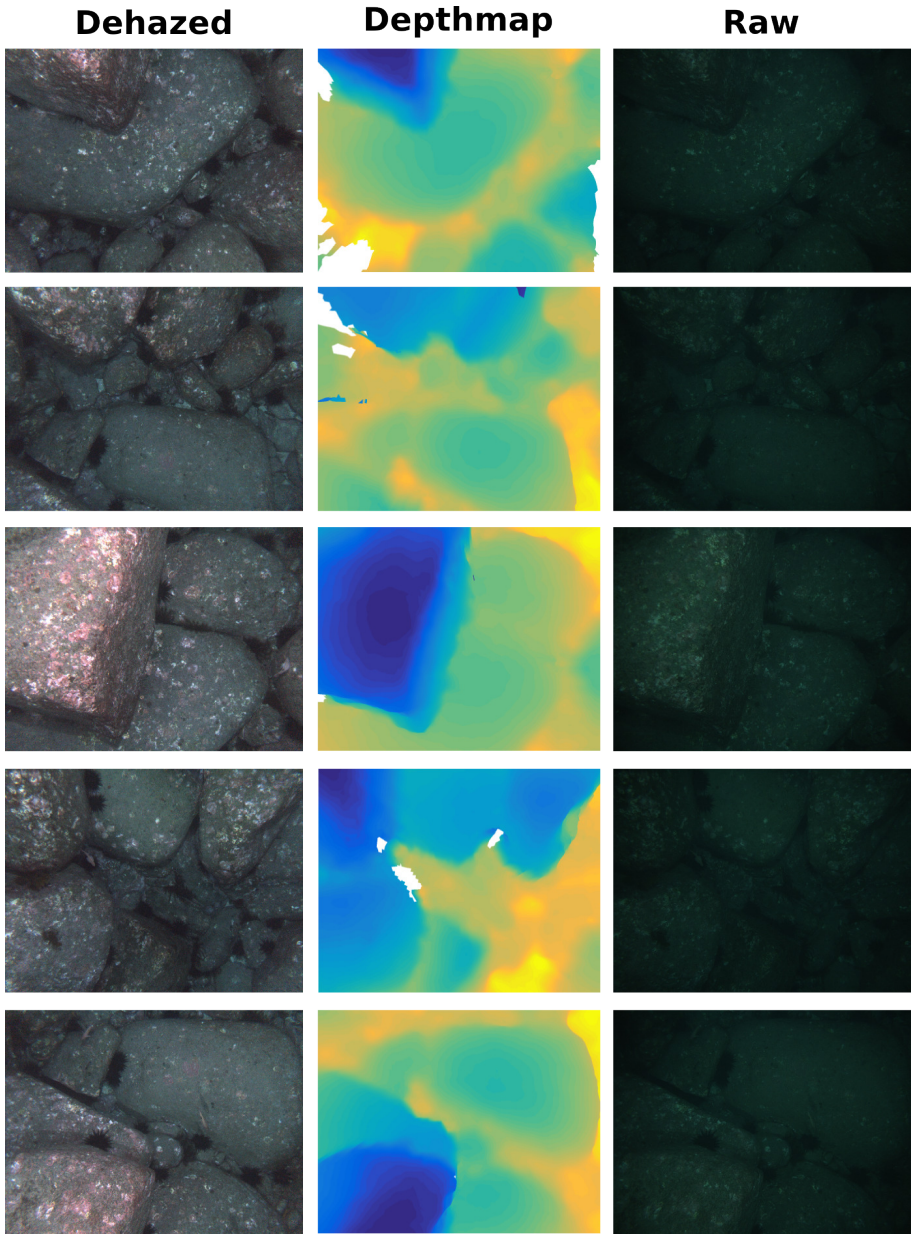

Figura A.1: Rocks dataset samples.

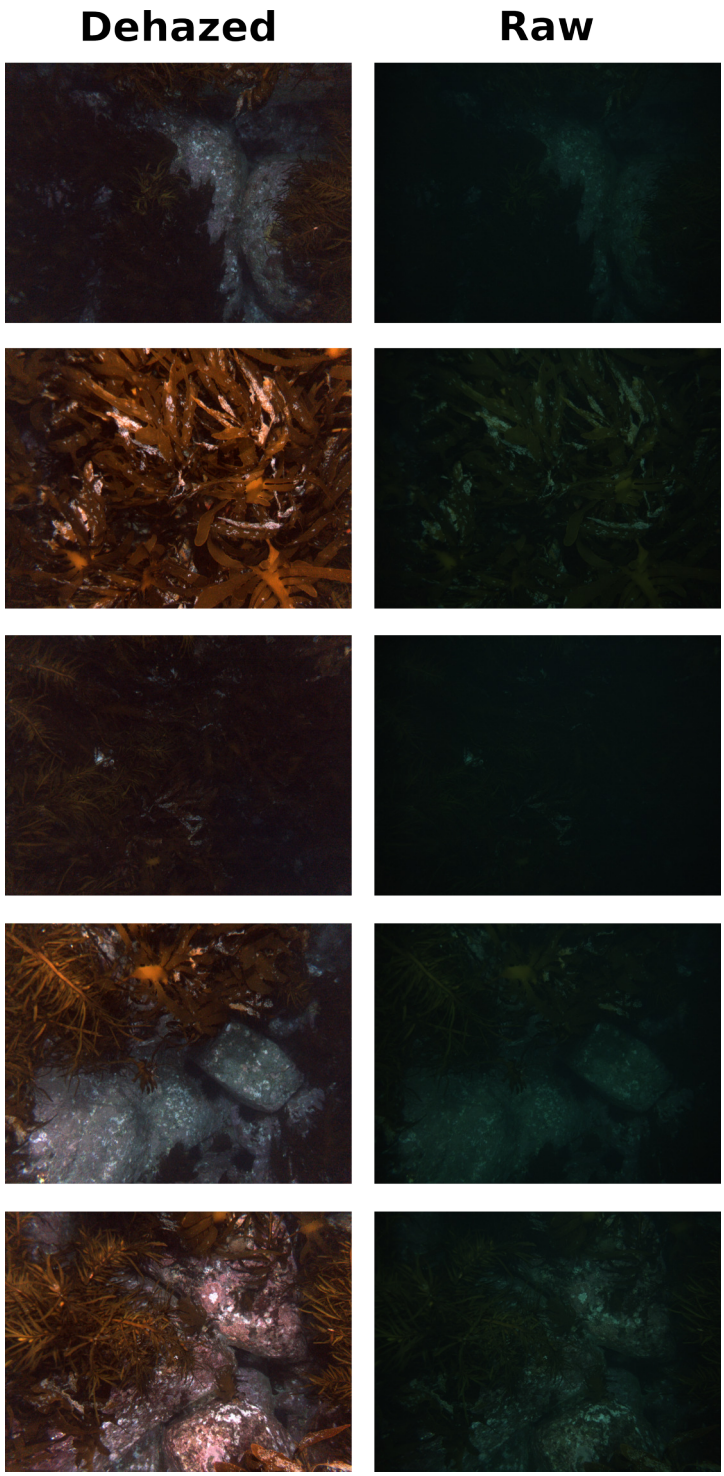

Figura A.2: Kelp dataset samples.

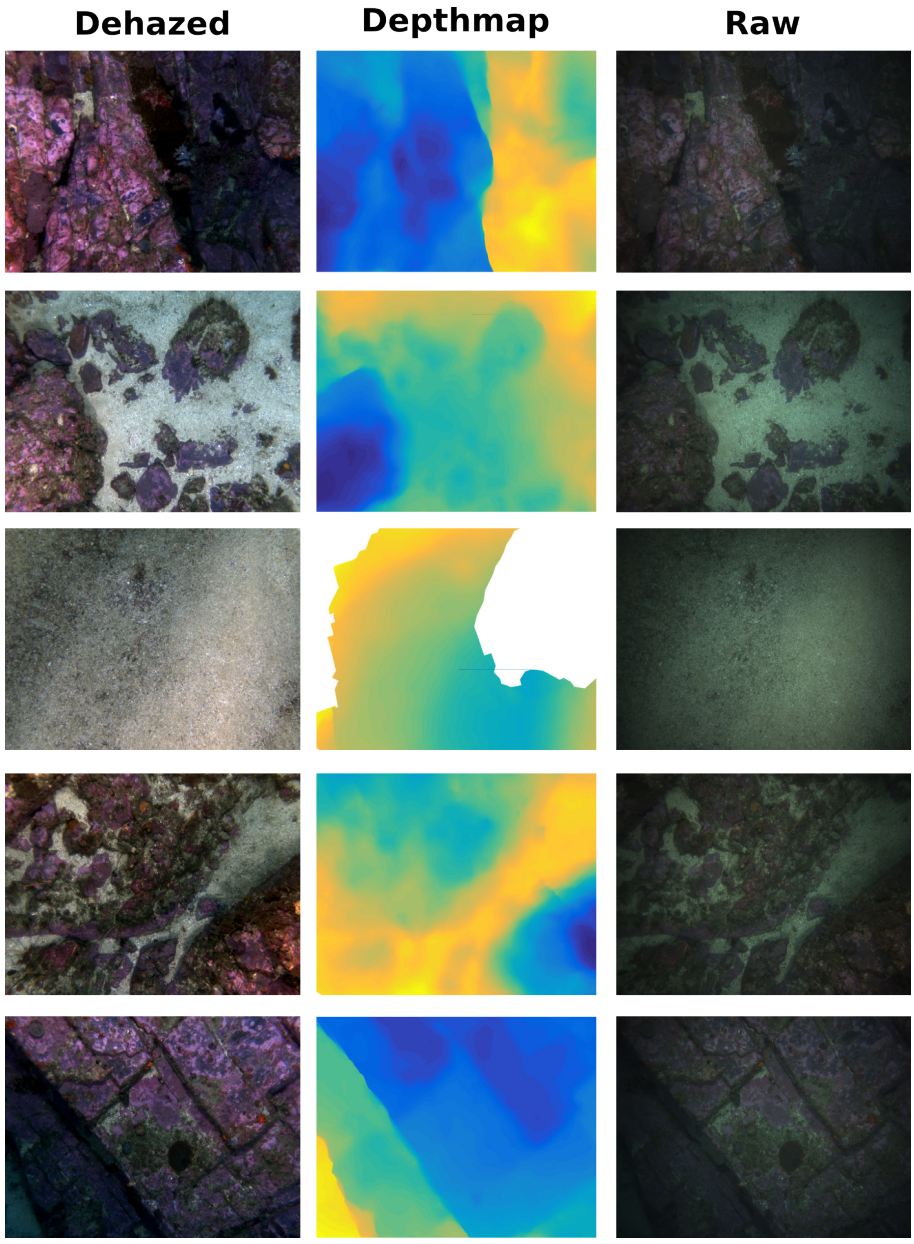

Figura A.3: Rocks and sand dataset samples.

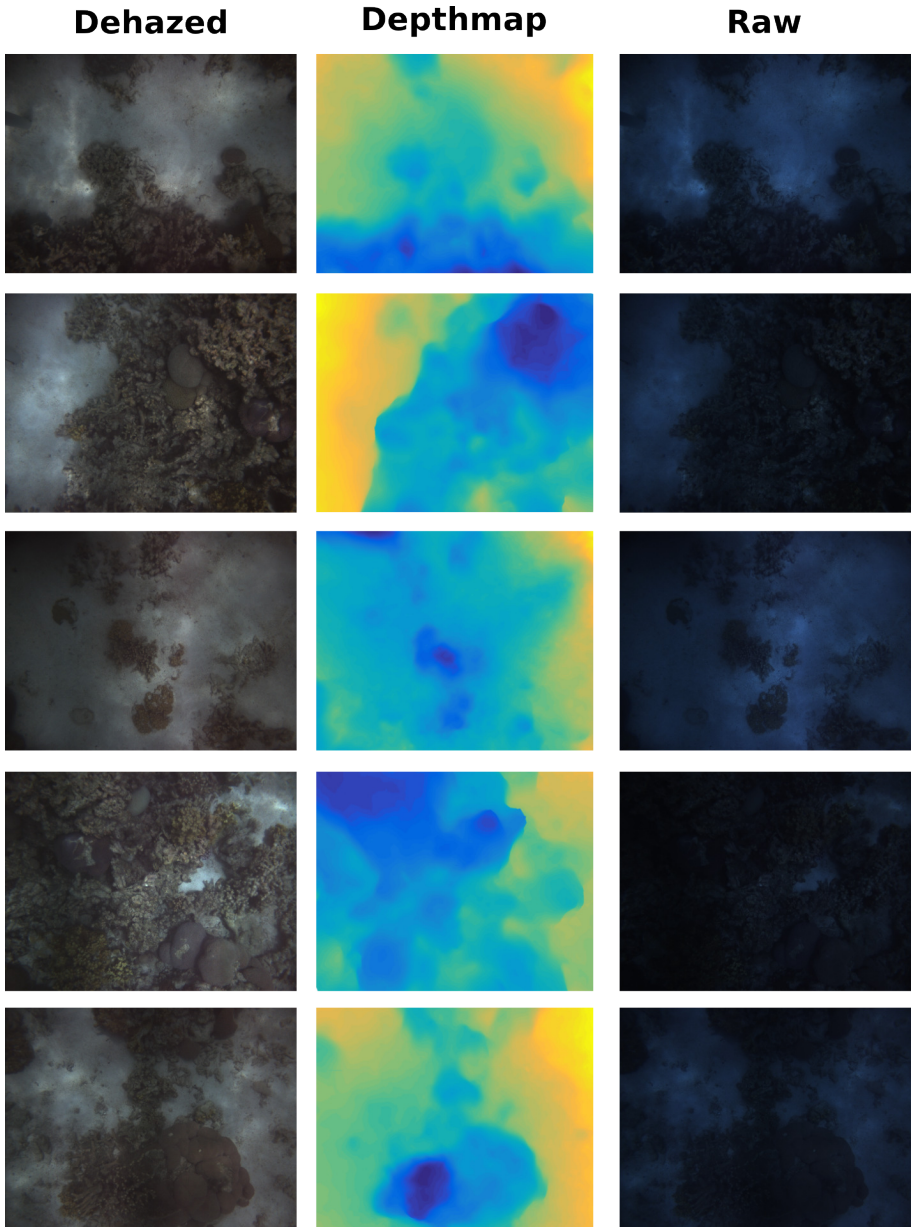

Figura A.4: Shallow depth corals dataset samples.

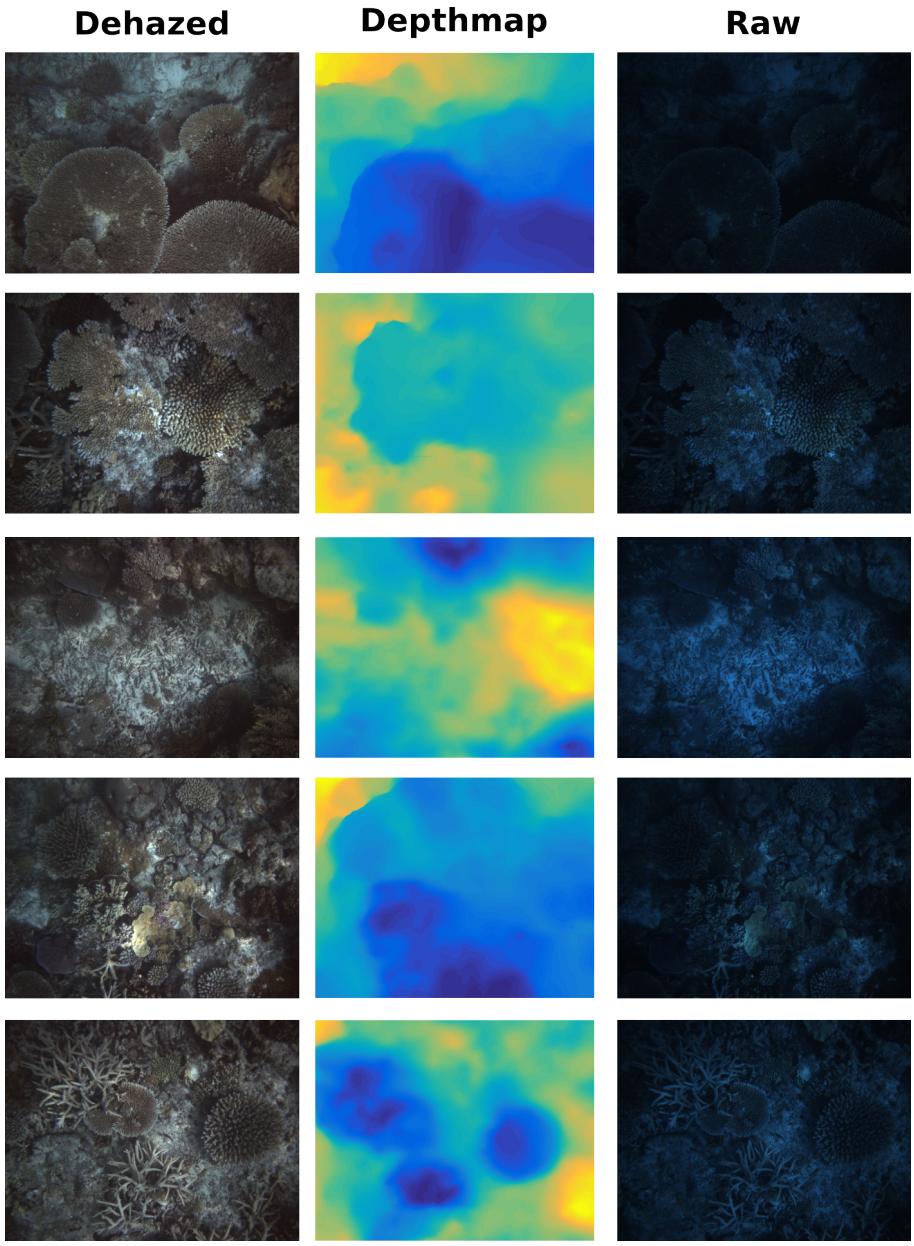

Figura A.5: Medium depth corals dataset samples.

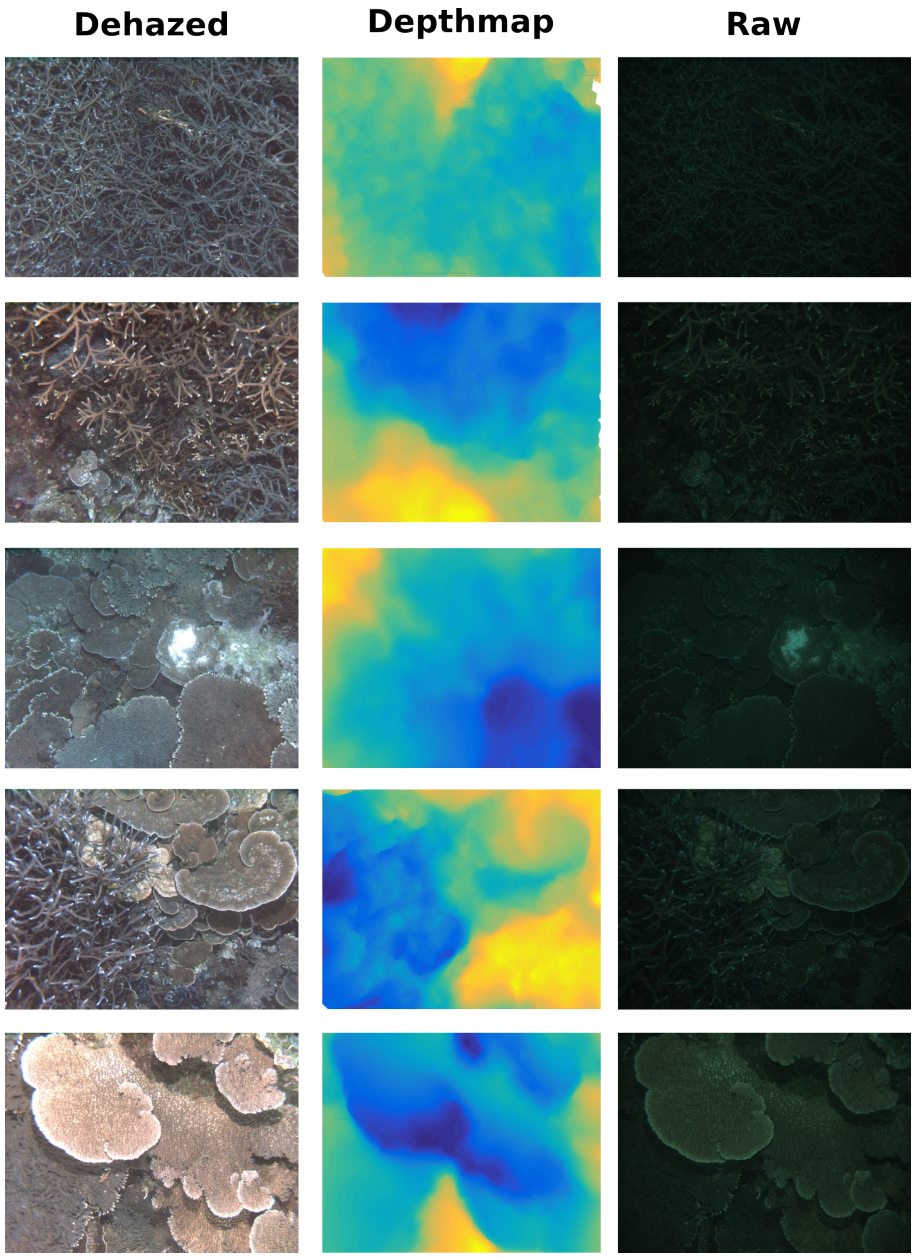

Figura A.6: Deep depth corals dataset samples.

Bibliografía

[Abadi et al., 2016] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., et al. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*. (Cited on pages 129 and 139.)

[Alimisis, 2013] Alimisis, D. (2013). Educational robotics: Open questions and new challenges. *Themes in Science and Technology Education*, 6(1):63–71. (Cited on page 38.)

[Amigoni et al., 2013] Amigoni, F., Bonarini, A., Fontana, G., Matteucci, M., and Schiaffonati, V. (2013). Benchmarking through competitions. In *European Robotics Forum–Workshop on Robot Competitions: Benchmarking, Technology Transfer, and Education*. (Cited on page 15.)

[Ancuti et al., 2011] Ancuti, C., Ancuti, C., Hermans, C., and Bekaert, P. (2011). A fast semi-inverse approach to detect and remove the haze from a single image. *Computer Vision–ACCV 2010*, pages 501–514. (Cited on page 86.)

[Ancuti et al., 2016a] Ancuti, C., Ancuti, C. O., De Vleeschouwer, C., and Bovik, A. C. (2016a). Night-time dehazing by fusion. In *Image Processing (ICIP), 2016 IEEE International Conference on*, pages 2256–2260. IEEE. (Cited on page 86.)

[Ancuti et al., 2016b] Ancuti, C., Ancuti, C. O., De Vleeschouwer, C., Garcia, R., and Bovik, A. C. (2016b). Multi-scale underwater descattering. (Cited on pages 86 and 87.)

[Ancuti and Ancuti, 2013] Ancuti, C. O. and Ancuti, C. (2013). Single image dehazing by multi-scale fusion. *IEEE Transactions on Image Processing*, 22(8):3271–3282. (Cited on page 86.)

[Antonelli, 2014] Antonelli, G. (2014). Simurv 4.0. In *Underwater Robots*, pages 257–265. Springer. (Cited on pages 20 and 49.)

[Antonelli et al., 2008] Antonelli, G., Fossen, T. I., and Yoerger, D. R. (2008). Underwater robotics. In *Springer handbook of robotics*, pages 987–1008. Springer. (Cited on page 1.)

[Baig and Torresani, 2015] Baig, M. H. and Torresani, L. (2015). Coarse-to-fine depth estimation from a single image via coupled regression and dictionary learning. *arXiv preprint arXiv:1501.04537*, 5. (Cited on page 125.)

[Baldi et al., 2014] Baldi, P., Sadowski, P., and Whiteson, D. (2014). Searching for exotic particles in high-energy physics with deep learning. *Nature communications*, 5. (Cited on page 121.)

[Bale, 2012] Bale, K. (2012). osgocean. (Cited on page 16.)

[Benitti, 2012] Benitti, F. B. V. (2012). Exploring the educational potential of robotics in schools: A systematic review. *Computers & Education*, 58(3):978–988. (Cited on page 38.)

[Bigg et al., 2003] Bigg, G., Jickells, T., Liss, P., and Osborn, T. (2003). The role of the oceans in climate. *International Journal of Climatology*, 23(10):1127–1159. (Cited on page 1.)

[Bischoff et al., 2010] Bischoff, R., Guhl, T., Prassler, E., Nowak, W., Kraetzschmar, G., Bruyninckx, H., Soetens, P., Haegele, M., Pott, A., Breedveld, P., Broenink, J., Brugali, D., and Tomatis, N. (2010). BRICS - Best practice in robotics. In *Proc. IFR Int. Symp. Robotics*, pages 968–975. (Cited on page 14.)

[Bonsignorio and Del Pobil, 2015] Bonsignorio, F. and Del Pobil, A. P. (2015). Toward replicable and measurable robotics research [from the guest editors]. *IEEE Robotics & Automation Magazine*, 22(3):32–35. (Cited on page 45.)

[Bonsignorio et al., 2008] Bonsignorio, F., Hallam, J., and del Pobil, A. (2008). Gem guidelines, euron gem sig report. (Cited on page 45.)

[Bryson et al., 2016] Bryson, M., Johnson-Roberson, M., Pizarro, O., and Williams, S. B. (2016). True color correction of autonomous underwater vehicle imagery. *Journal of Field Robotics*, 33(6):853–874. (Cited on pages 83, 85, 87, 126, 137, 154, and 169.)

[Cai et al., 2016] Cai, B., Xu, X., Jia, K., Qing, C., and Tao, D. (2016). Dehazenet: An end-to-end system for single image haze removal. *arXiv preprint arXiv:1601.07661*. (Cited on page 121.)

[Carlevaris-Bianco et al., 2010] Carlevaris-Bianco, N., Mohan, A., and Eustice, R. M. (2010). Initial results in underwater single image dehazing. In *OCEANS 2010*, pages 1–8. IEEE. (Cited on page 90.)

[Centelles et al., 2015] Centelles, D., Rubino, E., Soler, M., Martí, J., Sales, J., Marin, R., and Sanz, P. (2015). Underwater radio frequency based localization and image transmission system, including specific compression techniques, for autonomous manipulation. In *OCEANS 2015-Genova*, pages 1–5. IEEE. (Cited on page 4.)

[Center, 2015] Center, U. D. C.-S. (2015). Roboblockly. (Cited on page 28.)

[Cervera et al., 2016] Cervera, E., Martinet, P., Marin, R., Moughlbay, A. A., del Pobil, A. P., Alemany, J., Esteller, R., and Casañ, G. (2016). The robot programming network. *Journal of Intelligent & Robotic Systems*, 81(1):77–95. (Cited on page 28.)

[Chapman et al., 2006] Chapman, P., Conte, G., Drap, P., Gambogi, P., Gauch, F., Hanke, K., Long, L., Loureiro, V., Papini, O., Pascoal, A., et al. (2006). Venus, virtual exploration of underwater sites. *Proceeding of joint event CIPA/VAST/EG/Euro-Med*. (Cited on page 16.)

[Chiang and Chen, 2012] Chiang, J. Y. and Chen, Y.-C. (2012). Underwater image enhancement by wavelength compensation and dehazing. *IEEE Transactions on Image Processing*, 21(4):1756–1769. (Cited on page 86.)

[Choi et al., 1994] Choi, S., Takashige, G., and Yuh, J. (1994). Experimental study on an underwater robotic vehicle: Odin. In *Autonomous Underwater Vehicle Technology, 1994. AUV'94., Proceedings of the 1994 Symposium on*, pages 79–84. IEEE. (Cited on page 2.)

[Codevilla et al., 2014] Codevilla, F., Botelho, S., Drews-Jr, P., Duarte Filho, N., and Gaya, J. (2014). Underwater single image restoration using dark channel prior. (Cited on page 91.)

[Cohen et al., 2012] Cohen, B., Sucan, I., and Chitta, S. (2012). A generic infrastructure for benchmarking motion planners. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, pages 589–595, Vilamoura, Algarve, Portugal. (Cited on page 15.)

[Cook et al., 2014] Cook, D., Vardy, A., and Lewis, R. (2014). A survey of auv and robot simulators for multi-vehicle operations. In *2014 IEEE/OES Autonomous Underwater Vehicles (AUV)*, pages 1–8. IEEE. (Cited on pages 11 and 25.)

[Coumans, 2012] Coumans, E. (2012). Bullet physics engine. (Cited on page 16.)

[Craighead et al., 2007] Craighead, J., Murphy, R., Burke, J., and Goldiez, B. (2007). A survey of commercial & open source unmanned vehicle simulators. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 852–857. IEEE. (Cited on page 11.)

[De Novi et al., 2009] De Novi, G., Melchiorri, C., García, J., Sanz, P., Ridao, P., and Oliver, G. (2009). A new approach for a reconfigurable autonomous underwater vehicle for intervention. In *Systems conference, 2009 3rd annual IEEE*, pages 23–26. IEEE. (Cited on page 2.)

[DEXMART, 2009] DEXMART (2009). Specification of benchmarks. In *Deliverable D6.1 from FP7-DEXMART Project (DEXterous and autonomous dual-arm/hand robotic manipulation with sMART sensory-motor skills: A bridge from natural to artificial cognition*. (Cited on page 14.)

[Diankov, 2010] Diankov, R. (2010). *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute. (Cited on page 12.)

[Dong et al., 2014] Dong, C., Loy, C. C., He, K., and Tang, X. (2014). Learning a deep convolutional network for image super-resolution. In *European Conference on Computer Vision*, pages 184–199. Springer. (Cited on page 118.)

[Drews et al., 2013] Drews, P., Nascimento, E., Moraes, F., Botelho, S., and Campos, M. (2013). Transmission estimation in underwater single images. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 825–830. (Cited on pages 86 and 91.)

[Echeverria et al., 2011] Echeverria, G., Lassabe, N., Degroote, A., and Lemaignan, S. (2011). Modular open robots simulation engine: Morse. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 46–51. IEEE. (Cited on page 11.)

[Eguchi, 2013] Eguchi, A. (2013). Educational robotics theories and practice: Tips for how to do it right. *Robotics: Concepts, Methodologies, Tools, and Applications: Concepts, Methodologies, Tools, and Applications*, page 193. (Cited on page 38.)

[Eguchi, 2016] Eguchi, A. (2016). Robocupjunior for promoting stem education, 21st century skills, and technological advancement through robotics competition. *Robotics and Autonomous Systems*, 75:692–699. (Cited on page 38.)

[Eigen and Fergus, 2015] Eigen, D. and Fergus, R. (2015). Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2650–2658. (Cited on pages 124, 125, 143, and 149.)

[Eigen et al., 2014] Eigen, D., Puhrsch, C., and Fergus, R. (2014). Depth map prediction from a single image using a multi-scale deep network. In *Advances in neural information processing systems*, pages 2366–2374. (Cited on pages 124, 141, 148, 149, 150, and 151.)

[Esteller-Curto et al., 2012] Esteller-Curto, R., del Pobil, A., Cervera, E., and Marin, R. (2012). A test-bed internet based architecture proposal for benchmarking of visual servoing techniques. In *Proc. Sixth Int. Conf. Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*, pages 864–867. (Cited on page 15.)

[Evans et al., 2001] Evans, J., Keller, K., Smith, J., Marty, P., and Rigaud, O. (2001). Docking techniques and evaluation trials of the swimmer auv: an autonomous deployment auv for work-class rovs. In *OCEANS, 2001. MTS/IEEE Conference and Exhibition*, volume 1, pages 520–528. IEEE. (Cited on page 3.)

[Evans et al., 2003] Evans, J., Redmond, P., Plakas, C., Hamilton, K., and Lane, D. (2003). Autonomous docking for intervention-auvs using sonar and video-based real-time 3d pose estimation. In *Oceans 2003. Proceedings*, volume 4, pages 2201–2210. IEEE. (Cited on page 3.)

[Farfade et al., 2015] Farfade, S. S., Saberian, M. J., and Li, L.-J. (2015). Multi-view face detection using deep convolutional neural networks. In *Proceedings of the 5th ACM on International Conference on Multimedia Retrieval*, pages 643–650. ACM. (Cited on page 118.)

[Fattal, 2008] Fattal, R. (2008). Single image dehazing. *ACM transactions on graphics (TOG)*, 27(3):72. (Cited on page 86.)

[Fernández et al., 2015] Fernández, J., Pérez, J., Peñalver, A., Sales, J., Fornas, D., and Sanz, P. (2015). Benchmarking using uwsim, simurv and ros: An autonomous free floating dredging intervention case study. In *OCEANS 2015-Genova*, pages 1–7. IEEE. (Cited on pages 4 and 55.)

[Fernández et al., 2013] Fernández, J., Prats, M., Sanz, P. J., García, J., Marín, R., Robinson, M., Ribas, D., and Ridao, P. (2013). Grasping for the seabed: Developing a new underwater robot arm for shallow-water intervention. *Robotics Automation Magazine, IEEE*, 20(4):121–130. (Cited on pages 3, 50, and 75.)

[Fontana et al., 2014] Fontana, G., Matteucci, M., and Sorrenti, D. G. (2014). Rawseeds: Building a benchmarking toolkit for autonomous robotics. In Amigoni, F. and Schiaffonati, V., editors, *Methods and Experimental Techniques in Computer Engineering*, SpringerBriefs in Applied Sciences and Technology. Springer International Publishing. (Cited on page 15.)

[Fornas et al., 2016] Fornas, D., Sales, J., Peñalver, A., Pérez, J., Fernández, J. J., Marín, R., and Sanz, P. J. (2016). Fitting primitive shapes in point clouds: a practical approach to improve autonomous underwater grasp specification of unknown objects. *Journal of Experimental & Theoretical Artificial Intelligence*, 28(1-2):369–384. (Cited on page 4.)

[Foundation, 2015] Foundation, O. S. R. (2015). Cloudsim. (Cited on page 28.)

[Fugro,] Fugro. Deepworks. (Cited on page 13.)

[Galdran et al., 2015] Galdran, A., Pardo, D., Picón, A., and Alvarez-Gila, A. (2015). Automatic red-channel underwater image restoration. *Journal of Visual Communication and Image Representation*, 26:132–145. (Cited on page 91.)

[Garcia et al., 2010] Garcia, J., Prats, M., Sanz, P., Marin, R., and Belmonte, O. (2010). Exploring multimodal interfaces for underwater intervention systems. In *Proceedings of the IEEE ICRA 2010 Workshop on Multimodal Human-Robot Interfaces*. (Cited on page 4.)

[Garg et al., 2011] Garg, R., Mittal, B., and Garg, S. (2011). Histogram equalization techniques for image enhancement. *International Journal of Electronics & Communication Technology*, 2(1):107–111. (Cited on page 87.)

[Getreuer, 2012] Getreuer, P. (2012). Automatic color enhancement (ace) and its fast implementation. *Image Processing On Line*, 2:266–277. (Cited on page 130.)

[Ghani and Isa, 2015] Ghani, A. S. A. and Isa, N. A. M. (2015). Underwater image quality enhancement through integrated color model with rayleigh distribution. *Applied Soft Computing*, 27:219–230. (Cited on page 87.)

[Gibson et al., 2012] Gibson, K. B., Vo, D. T., and Nguyen, T. Q. (2012). An investigation of dehazing effects on image and video coding. *Image Processing, IEEE Transactions on*, 21(2):662–673. (Cited on page 91.)

[Goldman et al., 2004] Goldman, R., Eguchi, A., and Sklar, E. (2004). Using educational robotics to engage inner-city students with technology. In *Proceedings of the 6th international conference on Learning sciences*, pages 214–221. International Society of the Learning Sciences. (Cited on page 38.)

[Graves et al., 2013] Graves, A., Mohamed, A.-r., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE. (Cited on page 118.)

[Grossberg and Nayar, 2004] Grossberg, M. D. and Nayar, S. K. (2004). Modeling the space of camera response functions. *IEEE transactions on pattern analysis and machine intelligence*, 26(10):1272–1282. (Cited on page 84.)

[Haynes and Edwards, 2015] Haynes, C. and Edwards, J. (2015). First robotics competition [competitions]. *Robotics & Automation Magazine, IEEE*, 22(1):8–10. (Cited on page 38.)

[He et al., 2010] He, K., Sun, J., and Tang, X. (2010). Guided image filtering. In *Computer Vision–ECCV 2010*, pages 1–14. Springer. (Cited on pages 90, 102, 139, and 143.)

[He et al., 2011] He, K., Sun, J., and Tang, X. (2011). Single image haze removal using dark channel prior. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(12):2341–2353. (Cited on pages 86, 87, 88, 89, 143, 146, 149, and 151.)

[Hinton and Salakhutdinov, 2006] Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507. (Cited on page 118.)

[Hitam et al., 2013] Hitam, M. S., Awalludin, E. A., Yussof, W. N. J. H. W., and Bachok, Z. (2013). Mixture contrast limited adaptive histogram equalization for underwater image enhancement. In *Computer Applications Technology (ICCAT), 2013 International Conference on*, pages 1–5. IEEE. (Cited on pages 87 and 129.)

[Holz et al., 2013] Holz, D., Iocchi, L., and Van Der Zant, T. (2013). Benchmarking intelligent service robots through scientific competitions: The robocup@ home approach. In *AAAI Spring Symposium: Designing Intelligent Robots*. (Cited on page 15.)

[Hussain and Jeong, 2015] Hussain, F. and Jeong, J. (2015). Visibility enhancement of scene images degraded by foggy weather conditions with deep neural networks. *Journal of Sensors*, 2016. (Cited on pages 112 and 122.)

[Iizuka et al., 2016] Iizuka, S., Simo-Serra, E., and Ishikawa, H. (2016). Let there be color!: joint end-to-end learning of global and local image priors for automatic image colorization with simultaneous classification. *ACM Transactions on Graphics (TOG)*, 35(4):110. (Cited on page 112.)

[Inglis et al., 2012] Inglis, G., Smart, C., Vaughn, I., and Roman, C. (2012). A pipeline for structured light bathymetric mapping. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, pages 4425–4432, Vilamoura, Algarve, Portugal. (Cited on page 69.)

[Iqbal et al., 2007] Iqbal, K., Abdul Salam, R., Osman, M., Talib, A. Z., et al. (2007). Underwater image enhancement using an integrated colour model. *IAENG International Journal of Computer Science*, 32(2):239–244. (Cited on page 130.)

[Iqbal et al., 2010] Iqbal, K., Odetayo, M., James, A., Salam, R. A., and Talib, A. Z. H. (2010). Enhancing the low quality images using unsupervised colour correction method. In *Systems Man and Cybernetics (SMC), 2010 IEEE International Conference on*, pages 1703–1709. IEEE. (Cited on page 87.)

[Jaffe, 1990] Jaffe, J. S. (1990). Computer modeling and the design of optimal underwater imaging systems. *IEEE Journal of Oceanic Engineering*, 15(2):101–111. (Cited on pages 82 and 83.)

[Jaffe, 2010] Jaffe, J. S. (2010). Enhanced extended range underwater imaging via structured illumination. *Optics express*, 18(12):12328–12340. (Cited on pages 85 and 87.)

[Johnson, 2003] Johnson, J. (2003). Children, robotics, and education. *Artificial Life and Robotics*, 7(1-2):16–21. (Cited on page 38.)

[Johnson-Roberson et al., 2010] Johnson-Roberson, M., Pizarro, O., Williams, S. B., and Mahon, I. (2010). Generation and visualization of large-scale three-dimensional reconstructions from underwater robotic surveys. *Journal of Field Robotics*, 27(1):21–51. (Cited on page 169.)

[Kalwa et al., 2012] Kalwa, J., Pascoal, A., Ridao, P., Birk, A., Eichhorn, M., Brignone, L., Caccia, M., Alvez, J., and Santos, R. (2012). The european r&d-project morph: Marine robotic systems of self-organizing, logically linked physical nodes. *IFAC Proceedings Volumes*, 45(5):349–354. (Cited on page 25.)

[Karpathy and Fei-Fei, 2015] Karpathy, A. and Fei-Fei, L. (2015). Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3128–3137. (Cited on page 118.)

[Karpathy et al., 2015] Karpathy, A., Johnson, J., and Fei-Fei, L. (2015). Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*. (Cited on page 118.)

[Kermorgant, 2014] Kermorgant, O. (2014). A dynamic simulator for underwater vehicle-manipulators. In *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, pages 25–36. Springer. (Cited on page 21.)

[Kim and Pollefeys, 2008] Kim, S. J. and Pollefeys, M. (2008). Robust radiometric calibration and vignetting correction. *IEEE transactions on pattern analysis and machine intelligence*, 30(4):562–576. (Cited on page 83.)

[Kingma and Ba, 2014] Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*. (Cited on page 128.)

[Koenig and Howard, 2004] Koenig, N. and Howard, A. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2149–2154. IEEE. (Cited on page 11.)

[Konolige, 1998] Konolige, K. (1998). Small vision systems: Hardware and implementation. In Shirai, Y. and Hirose, S., editors, *Robotics Research*, pages 203–212. Springer London. (Cited on page 68.)

[Kootstra et al., 2012] Kootstra, G., PopoviÄĞ, M., JÃÿrgensen, J., Kragic, D., Petersen, H., and KrÃijger, N. (2012). Visgrab: A benchmark for vision-based grasping. *Paladyn*, 3(2):54–62. (Cited on page 15.)

[Kratz and Nishino, 2009] Kratz, L. and Nishino, K. (2009). Factorizing scene albedo and depth from a single foggy image. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 1701–1708. IEEE. (Cited on page 87.)

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105. (Cited on page 118.)

[LABS,] LABS, C. Vortex. (Cited on page 13.)

[Ladicky et al., 2014] Ladicky, L., Shi, J., and Pollefeys, M. (2014). Pulling things out of perspective. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 89–96. (Cited on page 124.)

[Lane et al., 1997] 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., et al. (1997). Amadeus: advanced manipulation for deep underwater sampling. *IEEE Robotics & Automation Magazine*, 4(4):34–45. (Cited on page 2.)

[Lane et al., 2012] Lane, D. M., Maurelli, F., Kormushev, P., Carreras, M., Fox, M., and Kyriakopoulos, K. (2012). Persistent autonomy: the challenges of the pandora project. *IFAC Proceedings Volumes*, 45(27):268–273. (Cited on page 25.)

[LeCun et al., 2015] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444. (Cited on pages 111 and 121.)

[LeCun et al., 1989] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551. (Cited on page 111.)

[LeCun et al., 2010] LeCun, Y., Kavukcuoglu, K., and Farabet, C. (2010). Convolutional networks and applications in vision. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pages 253–256. IEEE. (Cited on page 118.)

[Levin et al., 2008] Levin, A., Lischinski, D., and Weiss, Y. (2008). A closed-form solution to natural image matting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2):228–242. (Cited on page 89.)

[Liang et al., 2014] Liang, Z., Liu, H., Zhang, B., and Wang, B. (2014). Real-time hardware accelerator for single image haze removal using dark channel prior and guided filter. *IEICE Electronics Express*, 11(24):20141002–20141002. (Cited on page 86.)

[Liu et al., 2015] Liu, F., Shen, C., and Lin, G. (2015). Deep convolutional neural fields for depth estimation from a single image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5162–5170. (Cited on page 125.)

[LLC, 2006] LLC, M. S. (2006). Rovsim. (Cited on pages 13 and 28.)

[Lu et al., 2015] Lu, H., Li, Y., and Serikawa, S. (2015). Single underwater image descattering and color correction. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 1623–1627. IEEE. (Cited on page 91.)

[Ma et al., 2015] Ma, J., Sheridan, R. P., Liaw, A., Dahl, G. E., and Svetnik, V. (2015). Deep neural nets as a method for quantitative structure-activity relationships. *Journal of chemical information and modeling*, 55(2):263–274. (Cited on page 121.)

[Mai et al., 2014] Mai, J., Zhu, Q., Wu, D., Xie, Y., and Wang, L. (2014). Back propagation neural network dehazing. In *Robotics and Biomimetics (ROBIO), 2014 IEEE International Conference on*, pages 1433–1438. IEEE. (Cited on page 122.)

[Malis, 2004] Malis, E. (2004). Improving vision-based control using efficient second-order minimization techniques. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 2, pages 1843 – 1848 Vol.2. (Cited on pages 34 and 63.)

[Marchand, 1999] Marchand, E. (1999). ViSP: a software environment for eye-in-hand visual servoing. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 4, pages 3224–3229 vol.4. (Cited on pages 34 and 59.)

[Marti et al., 2012] Marti, J. V., Sales, J., Marin, R., and Jimenez-Ruiz, E. (2012). Localization of mobile sensors and actuators for intervention in low-visibility conditions: the zigbee fingerprinting approach. *International Journal of Distributed Sensor Networks*, 2012. (Cited on page 4.)

[Matsebe et al., 2008] Matsebe, O., Kumile, C., and Tlale, N. (2008). A review of virtual simulators for autonomous underwater vehicles (auvs). *IFAC Proceedings Volumes*, 41(1):31–37. (Cited on page 11.)

[McAfee et al., 2012] McAfee, A., Brynjolfsson, E., Davenport, T. H., Patil, D., and Barton, D. (2012). Big data. *The management revolution. Harvard Bus Rev*, 90(10):61–67. (Cited on pages 111 and 121.)

[McCulloch and Pitts, 1943] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133. (Cited on page 111.)

[Mendonça et al., 2013] Mendonça, R., Santana, P., Marques, F., Lourenço, A., Silva, J., and Barata, J. (2013). Kelpie: A ros-based multi-robot simulator for water surface and aerial vehicles. In *2013 IEEE International Conference on Systems, Man, and Cybernetics*, pages 3645–3650. IEEE. (Cited on page 12.)

[Michel, 2004] Michel, O. (2004). Webotstm: Professional mobile robot simulation. *arXiv preprint cs/0412052*. (Cited on page 12.)

[Nardi et al., 2015] Nardi, L., Bodin, B., Zia, M. Z., Mawer, J., Nisbet, A., Kelly, P. H., Davison, A. J., Luján, M., O'Boyle, M. F., Riley, G., et al. (2015). Introducing slambench, a performance and accuracy benchmarking methodology for slam. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5783–5790. IEEE. (Cited on page 15.)

[Nielsen, 2015] Nielsen, M. A. (2015). Neural networks and deep learning. *Determination Press*. (Cited on page 118.)

[Nowak et al., 2010] Nowak, W., Zakharov, A., Blumenthal, S., and Prassler, E. (2010). Benchmarks for mobile manipulation and robust obstacle avoidance and navigation. In *Deliverable D3.1 from FP7-BRICS Project (Best Practice in Robotics)*. (Cited on page 14.)

[Osfield et al., 2004] Osfield, R., Burns, D., et al. (2004). Open scene graph. (Cited on page 16.)

[Oude Elberink and Vosselman, 2011] Oude Elberink, S. and Vosselman, G. (2011). Quality analysis on 3D building models reconstructed from airborne laser scanning data. *ISPRS Journal of Photogrammetry and Remote Sensing*, 66(2):157–165. (Cited on page 70.)

[Pavin et al., 2015] Pavin, A., Inzartsev, A., Eliseenko, G., Lebedko, O., and Panin, M. (2015). A reconfigurable web-based simulation environment for auv. In *OCEANS 2015-MTS/IEEE Washington*, pages 1–7. IEEE. (Cited on pages 14 and 28.)

[Peñalver et al., 2014] Peñalver, A., Pérez, J., Fernández, J. J., Sales, J., Sanz, P., García, J., Fornas, D., and Marin, R. (2014). Autonomous intervention on an underwater panel mockup by using visually-guided manipulation techniques. *IFAC Proceedings Volumes*, 47(3):5151–5156. (Cited on page 69.)

[Peñalver et al., 2015] Peñalver, A., Pérez, J., Fernández, J. J., Sales, J., Sanz, P. J., García, J., Fornas, D., and Marín, R. (2015). Visually-guided manipulation techniques for robotic autonomous underwater panel interventions. *Annual Reviews in Control*, 40:201–211. (Cited on page 3.)

[Pérez et al., 2014] Pérez, J., Sales, J., Marín, R., and Sanz, P. J. (2014). Online tool for benchmarking of simulated intervention autonomous underwater

vehicles: Evaluating position controllers in changing underwater currents. In *Artificial Intelligence, Modelling and Simulation (AIMS), 2014 2nd International Conference on*, pages 246–251. IEEE. (Cited on page 27.)

[Perez et al., 2014] Perez, J., Sales, J., Marin, R., and Sanz, P. J. (2014). Web-based configuration tool for benchmarking of simulated intervention autonomous underwater vehicles. In *Autonomous Robot Systems and Competitions (ICARSC), 2014 IEEE International Conference on*, pages 279–284. IEEE. (Cited on page 27.)

[Pérez et al., 2015] Pérez, J., Sales, J., Peñalver, A., Fernández, J. J., Sanz, P. J., García, J. C., Martí, J. V., Marín, R., and Fornas, D. (2015). Robotic manipulation within the underwater mission planning context. In *Motion and Operation Planning of Robotic Systems*, pages 495–522. Springer. (Cited on page 9.)

[Perez et al., 2015] Perez, J., Sales, J., Penalver, A., Fornas, D., Fernandez, J. J., Garcia, J. C., Sanz, P. J., Marin, R., and Prats, M. (2015). Exploring 3-d reconstruction techniques: A benchmarking tool for underwater robotics. *IEEE Robotics & Automation Magazine*, 22(3):85–95. (Cited on page 57.)

[Pérez et al., 2013] Pérez, J., Sales, J., Prats, M., Martí, J. V., Fornas, D., Marín, R., and Sanz, P. J. (2013). The underwater simulator uwsim-benchmarking capabilities on autonomous grasping. In *ICINCO (2)*, pages 369–376. (Cited on page 57.)

[Prats et al., 2012a] Prats, M., Fernández, J., and Sanz, P. (2012a). Combining template tracking and laser peak detection for 3D reconstruction and grasping in underwater environments. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, pages 106–112, Vilamoura, Algarve, Portugal. (Cited on page 69.)

[Prats et al., 2012b] Prats, M., Pérez, J., Fernández, J. J., and Sanz, P. J. (2012b). An open source tool for simulation and supervision of underwater intervention missions. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 2577–2582. IEEE. (Cited on page 9.)

[Prats et al., 2012c] Prats, M., Ribas, D., Palomeras, N., García, J. C., Nannen, V., Wirth, S., Fernández, J. J., Beltrán, J. P., Campos, R., Ridao, P., et al. (2012c). Reconfigurable auv for intervention missions: a case study on underwater object recovery. *Intelligent Service Robotics*, 5(1):19–31. (Cited on page 3.)

[Raimondo and Silvia, 2010] Raimondo, S. and Silvia, C. (2010). Underwater image processing: state of the art of restoration and image enhancement methods. *EURASIP Journal on Advances in Signal Processing*, 2010. (Cited on page 85.)

[Rathnam and Birk, 2011] Rathnam, R. and Birk, A. (2011). Co3auvs: D2.2 release of the final simulator with a rich set of modeled sensors, vehicles and scenarios. Technical Report 2011/12, Jacobs University, Bremen, Germany. (Cited on page 13.)

[Ribas et al., 2012] Ribas, D., Palomeras, N., Ridao, P., Carreras, M., and Mallios, A. (2012). Girona 500 auv: From survey to intervention. *IEEE/ASME Transactions on Mechatronics*, 17(1):46–53. (Cited on page 50.)

[Rigaud et al., 1998] Rigaud, V., Coste-Maniere, E., Aldon, M.-J., Probert, P., Perrier, M., Rives, P., Simon, D., Lang, D., Kiener, J., Casal, A., et al. (1998). Union: underwater intelligent operation and navigation. *IEEE Robotics & Automation Magazine*, 5(1):25–35. (Cited on page 2.)

[Rohmer et al., 2013] Rohmer, E., Singh, S. P., and Freese, M. (2013). V-rep: A versatile and scalable robot simulation framework. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1321–1326. IEEE. (Cited on page 12.)

[Roser et al., 2014] Roser, M., Dunbabin, M., and Geiger, A. (2014). Simultaneous underwater visibility assessment, enhancement and improved stereo. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 3840–3847. IEEE. (Cited on pages 85 and 87.)

[Sales et al., 2010] Sales, J., Marín, R., Cervera, E., Rodríguez, S., and Pérez, J. (2010). Multi-sensor person following in low-visibility scenarios. *Sensors*, 10(12):10953–10966. (Cited on page 4.)

[Sanchez et al., 2015] Sanchez, J. G., Patrao, B., Almeida, L., Perez, J., Menezes, P., Dias, J., and Sanz, P. (2015). Design and evaluation of a natural interface for remote operation of underwater robots. *IEEE computer graphics and applications*. (Cited on pages 4 and 55.)

[Sanz et al., 2013] Sanz, P., Perez, J., Penalver, A., Fernandez, J., Fornas, D., Sales, J., and Marin, R. (2013). Grasper hil simulation towards autonomous manipulation of an underwater panel in a permanent observatory. In *Oceans-San Diego, 2013*, pages 1–6. IEEE. (Cited on page 9.)

[Saxena et al., 2008] Saxena, A., Chung, S. H., and Ng, A. Y. (2008). 3-d depth reconstruction from a single still image. *International journal of computer vision*, 76(1):53–69. (Cited on page 123.)

[Saxena et al., 2009] Saxena, A., Sun, M., and Ng, A. Y. (2009). Make3d: Learning 3d scene structure from a single still image. *IEEE transactions on pattern analysis and machine intelligence*, 31(5):824–840. (Cited on page 124.)

[Schnabel et al., 2007] Schnabel, R., Wahl, R., and Klein, R. (2007). Efficient ransac for point-cloud shape detection. In *Computer graphics forum*, volume 26, pages 214–226. Wiley Online Library. (Cited on page 50.)

[simulations, 2009] simulations, G. (2009). Vrov. (Cited on page 13.)

[Sutskever et al., 2014] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112. (Cited on page 118.)

[Tan, 2008] Tan, R. T. (2008). Visibility in bad weather from a single image. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE. (Cited on page 87.)

[Tang et al., 2014] Tang, K., Yang, J., and Wang, J. (2014). Investigating haze-relevant features in a learning framework for image dehazing. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 2995–3002. IEEE. (Cited on page 122.)

[Tarel and Hautiere, 2009] Tarel, J.-P. and Hautiere, N. (2009). Fast visibility restoration from a single color or gray level image. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2201–2208. IEEE. (Cited on page 86.)

[Tellez, 2017] Tellez, R. (2017). A thousand robots for each student: Using cloud robot simulations to teach robotics. In *Robotics in Education*, pages 143–155. Springer. (Cited on pages 14 and 28.)

[Tomasi and Manduchi, 1998] Tomasi, C. and Manduchi, R. (1998). Bilateral filtering for gray and color images. In *Computer Vision, 1998. Sixth International Conference on*, pages 839–846. IEEE. (Cited on page 89.)

[Torres-Méndez and Dudek, 2005] Torres-Méndez, L. A. and Dudek, G. (2005). Color correction of underwater images for aquatic robot inspection. In *Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 60–73. Springer. (Cited on pages 86 and 87.)

[Tosik and Maehle, 2014] Tosik, T. and Maehle, E. (2014). Mars: A simulation environment for marine robotics. In *2014 Oceans-St. John's*, pages 1–7. IEEE. (Cited on page 12.)

[Treibitz and Schechner, 2009] Treibitz, T. and Schechner, Y. Y. (2009). Active polarization descattering. *IEEE transactions on pattern analysis and machine intelligence*, 31(3):385–399. (Cited on pages 85 and 87.)

[Treibitz and Schechner, 2012] Treibitz, T. and Schechner, Y. Y. (2012). Turbid scene enhancement using multi-directional illumination fusion. *IEEE Transactions on Image Processing*, 21(11):4662–4667. (Cited on pages 85 and 87.)

[Ulbrich et al., 2011] Ulbrich, S., Kappler, D., Asfour, T., Vahrenkamp, N., Bierbaum, A., Przybylski, M., and Dillmann, R. (2011). The opengrasp benchmarking suite: An environment for the comparative analysis of grasping and dexterous manipulation. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, pages 1761–1767. (Cited on page 15.)

[Vasilescu et al., 2011] Vasilescu, I., Detweiler, C., and Rus, D. (2011). Color-accurate underwater imaging using perceptual adaptive illumination. *Autonomous Robots*, 31(2-3):285–296. (Cited on pages 85 and 87.)

[Wang et al., 1995] Wang, H. H., Rock, S. M., and Lees, M. (1995). Experiments in automatic retrieval of underwater objects with an auv. In *OCEANS'95. MTS/IEEE. Challenges of Our Changing Global Environment. Conference Proceedings.*, volume 1, pages 366–373. IEEE. (Cited on page 2.)

[Weisz et al., 2016] Weisz, J., Huang, Y., Lier, F., Sethumadhavan, S., and Allen, P. (2016). Robobench: Towards sustainable robotics system benchmarking. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3383–3389. IEEE. (Cited on page 15.)

[Williams and Hinton, 1986] Williams, D. and Hinton, G. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–538. (Cited on pages 111, 116, and 118.)

[Williams et al., 2012] Williams, S. B., Pizarro, O. R., Jakuba, M. V., Johnson, C. R., Barrett, N. S., Babcock, R. C., Kendrick, G. A., Steinberg, P. D., Heyward, A. J., Doherty, P. J., et al. (2012). Monitoring of benthic reference sites: using an autonomous underwater vehicle. *IEEE Robotics & Automation Magazine*, 19(1):73–84. (Cited on pages 126, 169, and 170.)

[Yosinski et al., 2015] Yosinski, J., Clune, J., Nguyen, A., Fuchs, T., and Lipson, H. (2015). Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*. (Cited on page 116.)

[Yuh et al., 1998] Yuh, J., Choi, S., Ikehara, C., Kim, G., McMurty, G., Ghasemi-Nejhad, M., Sarkar, N., and Sugihara, K. (1998). Design of a semi-autonomous underwater vehicle for intervention missions (sauvim). In *Underwater Technology, 1998. Proceedings of the 1998 International Symposium on*, pages 63–68. IEEE. (Cited on page 3.)

[Zhang and Zhao, 2017] Zhang, B. and Zhao, J. (2017). Hardware implementation for real-time haze removal. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(3):1188–1192. (Cited on page 86.)

[Zhu et al., 2015] Zhu, Q., Mai, J., and Shao, L. (2015). A fast single image haze removal algorithm using color attenuation prior. *Image Processing, IEEE Transactions on*, 24(11):3522–3533. (Cited on page 122.)