DEPARTMENT OF INFORMATION ENGINEERING

MASTER'S DEGREE IN
COMPUTER ENGINEERING

Deep Learning-Based 6-DoF Object Pose Estimation With Synthetic Data: A Case Study in Underwater Environments

Supervisor

Prof. ALBERTO PRETTO

Student

NICOLA VALZAN

ACCADEMIC YEAR 2021 - 2022

28/02/2022

ABSTRACT

In this thesis we aim to address the image based 6-DoF pose estimation problem, or 3D pose estimation problem, for Autonomous Underwater Vehicles (AUVs). The results of the object pose estimation will be used, for example, to estimate the global location of the AUV or to approach more accurately the underwater infrastructures. Actually, an autonomous robot or a team of autonomous robots need accurate location skills to safely and effectively move within an underwater environment, where communications are sparse and unreliable, and to accomplish high-level tasks such as: underwater exploration, mapping of the surrounding environment, multi-robot conveyance and many other multi-robot problems.
Several state-of-the-art approaches will be analysed and tested on real datasets. Collecting underwater images and providing them with an accurate ground-truth estimate of the object's pose is an expansive and extremely time-consuming activity To this end, we addressed the problem using only synthetic datasets. In fact, it was not possible to use the standard datasets used in the analyzed papers, since they are datasets with objects and conditions very different from those in which the AUVs operate. Hence, we exploited an unpaired image-to-image translation network is employed to bridge the gap between the rendered and the real images, producing photorealistic synthetic training images. Promising preliminary results confirm the goodness of the made choices.

Contents

1 Introduction **1**
- 1.1 Robotics . . . 1
  - 1.1.1 Sensors for Robotics . . . 1
  - 1.1.2 Actuators . . . 2
  - 1.1.3 Automation and Autonomy . . . 2
- 1.2 Underwater Robotics . . . 3
  - 1.2.1 Autonomous Underwater Vehicles . . . 3
- 1.3 The 6-DoF Object Pose Estimation Problem . . . 4

2 Object Pose Estimation: the Analyzed Frameworks **7**
- 2.1 Random Sample Consensus . . . 7
  - 2.1.1 Formulation . . . 8
  - 2.1.2 Improvements . . . 8
- 2.2 Perspective-n-Point . . . 8
  - 2.2.1 Formulation . . . 9
  - 2.2.2 General Case . . . 10
  - 2.2.3 Solution as Weighted Sum of Eigenvectors . . . 10
  - 2.2.4 Choosing the Right Linear Combination . . . 11
  - 2.2.5 Planar Case . . . 13
  - 2.2.6 Gauss-Newton Optimization . . . 13
- 2.3 Deep Underwater Relative Localization . . . 14
  - 2.3.1 Formulation . . . 15
  - 2.3.2 YOLO . . . 15
  - 2.3.3 Object Detection Stream . . . 17
  - 2.3.4 Pose Regression Stream . . . 18
  - 2.3.5 Pose refinement . . . 19
  - 2.3.6 Implementation details . . . 19
- 2.4 Pixel-wise Voting Network . . . 19
  - 2.4.1 Formulation . . . 20

2.4.2 ResNet . . . 20
2.4.3 Voting-based keypoint localization . . . 24
2.4.4 Keypoint selection . . . 24
2.4.5 Uncertainty-driven PnP . . . 25
2.4.6 Implementation details . . . 26
2.5 Dense Pose Object Detector . . . 26
2.5.1 Data Preparation . . . 26
2.5.2 Object Detection . . . 27
2.5.3 Pose Refinement . . . 27
2.5.4 Limitations . . . 28

3 Synthetic Dataset Generation **30**
3.1 Cycle Generative Adversarial Network . . . 31
3.1.1 Formulation . . . 32
3.1.2 Adversarial Loss . . . 32
3.1.3 Cycle Consistency Loss . . . 32
3.1.4 Full Objective . . . 33
3.1.5 Implementation details . . . 34
3.2 Synthetic Dataset . . . 34
3.2.1 Underwater Backgrounds . . . 34
3.2.2 Synthetic Objects . . . 35
3.3 Experiments . . . 35
3.4 Data Enhancement . . . 38
3.4.1 Histogram Equalization . . . 38
3.4.2 Contrast Limited Adaptive Histogram Equalization . . . 40

4 Experiments **44**
4.1 Pipeline . . . 44
4.2 Training Environment . . . 45
4.3 Approaches . . . 45
4.3.1 DeepURL . . . 45
4.3.2 PVNet . . . 46

5 Conclusion **51**
5.1 Future Work . . . 52

Bibliography **54**

Acknowledgements **59**

Chapter 1

Introduction

The 6DoF pose estimation problem consists of finding and estimating the rotation and translation of an object in space, i.e., determining the 2D corners of the object in the image and then find the 3D projections in space. Accurate pose estimation is crucial for many problems such as: augmented reality, autonomous driving, and robotic manipulation. Also, there are several environments, particularly in GPS-denied environments with limited features, where keeping track of the vehicle's position is a very challenging task. A common strategy to address the localization problem is to use the landmark based localization. This strategy consists in having objects, called landmarks, in the map whose position is known. In this way, when they are recognized, the position of the robot can be estimated. In this work, the problem of estimating the pose of 3D objects in underwater environments will be addressed, in particular for Autonomous Underwater Vehicles (AUVs). This problem is rather challenging from many perspectives: among others, the variability of light conditions and the turbidity of the water, and often the lack of accurate CAD models of the objects of interest. Also, it will not be possible to use the typical datasets (LINEMOD [18], Occlusion LINEMOD [3] and YCB-Video [45]) used in the analyzed papers, being datasets with objects very different from the ones addressed by AUVs.

1.1 Robotics

Robotics is the engineering discipline that studies and develops methods that allow a robot to perform specific tasks by automatically or autonomously reproducing human work. Robotics involves design, construction, operation, and use of robots. The goal of robotics is to design machines that can help and assist humans, in particular substitute for humans and replicate human actions. Important situations are in the use of jobs that are dangerous to humans, such as inspection of radioactive materials, bomb detection and deactivation. Robots are often used in hazardous environments or where humans cannot survive, such as in space, underwater, in high temperatures, and for cleaning and containment of hazardous materials and radiation.

The main components of a robot are the sensors, which allow it to sense the outside world and the state the robot is in, and the actuators, which allow the actual movement of the various parts of the robot.

1.1.1 Sensors for Robotics

Sensors allow robots to receive information about a certain measurement of a robot's condition and environment. Sensors in robots are based on the functions of human sensory organs and

require extensive information about their environment in order to function effectively. This is essential for robots to perform their tasks, and act upon any changes in the environment to calculate the appropriate response. Also robot's internal status are very important to give warnings about safety or malfunctions.

Specifically, sensors can be divided into three types:

Proprioceptive: sensors that sense the internal state of the robot, more precisely the internal state of the robot actuators, e.g. temperature, major position or battery power. Important for the self-control.

Exteroceptive: sensors that perceptive outside the robots, e.g. environment, landmarks or distances. Utilized for navigation and object recognition.

Exproprioceptive: sensors that perceptive positions of external objects relative to parts of the robot, so robot with respect that the environment, e.g. external camera that sees the position of the robot. Usually vision to understand the situation.

Another type of classification can be made based on the input received from the sensor:

Passive: sensor receives energy already in the environment, e.g. camera. They consume less energy, but often signal noise problems.

Active: sensor emits some form of energy and then measures the impact as a way of understanding the environment, e.g. ultrasonics or laser. They consume more energy and less signal noise problems, but often have restricted environments.

As you can see, sensors are a very important part of a robot, especially on autonomous robots.

1.1.2 Actuators

Actuators are the *muscles* of a robot, the parts which convert stored energy into movement that are responsible for moving and controlling a mechanism or system, acts upon to perform an operation or task. There are mainly three types of motors used as actuators:

Electric motor: uses electrical energy generated through a variable magnetic flux and is typically used where high speeds and low forces are needed.

Hydraulic motor: exploits the volume change due to a pressurized fluid, used in applications where large forces and low speeds are needed.

Pneumatic motor: uses pneumatic energy provided by a compressor.

1.1.3 Automation and Autonomy

Automation is about physically-situated tools performing highly repetitive, pre-planned actions for well-modeled tasks under the *closed world assumption*. The most obvious example of an automated robot are industrial robots. They focus on control theory, joint movement to get fastest and repeatable trajectory. They should have an high repeatability in a world where everything is fixtured to be in the right place at the right time. However engineers are adding sensors to reduce need for fixturing. Specifically the *closed world assumption* assumes:

- Everything relevant is known a priori, that there are no surprises
- Everything relevant can be completely modeled

- If world is modeled accurately enough, can create stable control loops to respond to all expected situations
- If world is controlled, can minimize or eliminate sensing

Instead, autonomy is about physically-situated agents who not only perform actions but can also adapt to the *open world* where the environment and tasks are not known a priori by generating new plans, monitoring and changing plans, and learning within the constraints of their bounded rationality. Also intelligent robot is situated as intelligent agent. Intelligent agent is a system that perceives its environment and takes actions which maximize its chances of success Therefore an intelligent agent is self-governing and autonomous, even it is a heavily bounded agent. Robot autonomy is often viewed as requiring the generation and execution of actions to meet a goal or carry out a mission, Execution of the plan may be confounded by the unmodeled events (open world), requiring the system to dynamically adapt or replan. This mean to maximaze the successes of a specific mission or goal. Autonomy robots are under the *open world assumption*:

- models may be available but are only partially (and unpredictably) correct
- must be able to sense relevant aspects of the world in order to dynamically adapt actions (e.g., act as an agent)

Since the robot is in an open world it will be very important what it perceives from it, that is, from the external environment around it. But above all the relationship between robot and environment. That is why in autonomous robots the perception part is very important and therefore the sensors that are used, both quantitatively and qualitatively. In fact, it is not a simple perception, but a more elaborate and complex *sensor fusion*. Sensor fusion is an high-level perception and world models, given by raw sensor data combined with algorithms to obtain features, called sensing.

1.2 Underwater Robotics

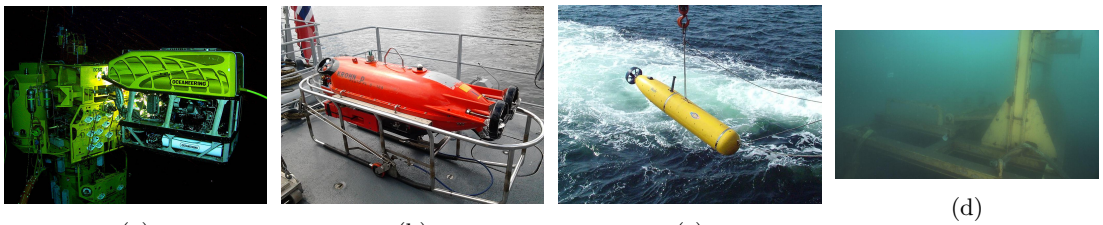
One particular type of robots are underwater robots, which are automated and autonomous. Specifically, automated underwater robots are called ROVs, remotely operated vehicles. While autonomous ones are called AUVs, autonomous underwater vehicles.

ROV is controlled and powered from the surface by an operator/pilot via an umbilical or using remote control. Figure 1.1 shows examples of ROV and AUVs.

1.2.1 Autonomous Underwater Vehicles

AUV is a robot that travels underwater without requiring input from an operator and capable of carrying out missions autonomously. AUVs are therefore able to save on the total cost of a mission, as they do not require an equipped ship and qualified personnel to remotely guide the robot. They also allow to complete missions that would be impossible due to the umbilical cord with the support ship, of which a typical example is the exploration under the ice. Research in this field can be divided into various areas, depending on the type of problem being examined. However, it is important to consider that the main difference between ROVs and AUVs is the absence of human action, so they are mainly studied techniques of artificial intelligence and autonomous robotics, which allow vehicles to fully perform the task for which they are intended, even in the presence of unexpected events or scenarios not necessarily known in advance.

AUVs carry sensors to navigate autonomously and map features of the ocean. Typical sensors include compasses, depth sensors, sidescan and other sonars, magnetometers, thermistors and



(a) (b) (c) (d)

Figure 1.1: (a) ROV at work in an underwater oil and gas field. The ROV is operating a subsea torque tool (wrench) on a valve on the subsea structure. (b) Pluto Plus AUV for underwater mine identification and destruction. From Norwegian minehunter KNM Hinnøy (c) Picture taken of the Battlespace Preparation Autonomous Underwater Vehicle (BPAUV) by an employee of Bluefin Robotics Corporation during a US Navy exercise. (d) Underwater platform examined in this thesis for the various experiments.

conductivity probes. Some AUVs are outfitted with biological sensors. Also radio waves cannot penetrate water very far, so as soon as an AUV dives it loses its GPS signal. Therefore, a standard way for AUVs to navigate underwater is through *dead reckoning*, the process of calculating current position of some moving object by using a previously determined position, or fix, and then incorporating estimates of speed, heading direction, and course over elapsed time.

Another important sensor used by AUVs is the camera: used to see the surroundings, recognize objects and estimate their position. The main problem is the underwater environment in which the robot is located, in fact this makes the pose estimation problem more challenge. Such as blurred AUV representation on the image, very bad lighting, image coloration and distinction between AUV and background. Lighting problem because it could be very bright or very dark, think if the robot is a few meters from the surface of the water or very deep. Image coloration problem because the color of the water can vary, some times it could be crystal blue and other times dark green with sand added. Also considering the general difficulties already known, including severe occlusions and truncations, variations in lighting and appearance, and cluttered background objects.

1.3 The 6-DoF Object Pose Estimation Problem

Early approaches to the 6-DoF object pose estimation problems employed a CAD-based template matching strategies [5, 6, 13] or extracted a set of local features such as SIFT [30] or SURF [2] to be matched with a features-based model of the object of interest. Multi-scale representation of the original image were also used to find objects at different distances.

State of the art focuses on deep-learning techniques. In fact, the logic behind neural networks, i.e., deep layers, is precisely to represent the image in different representations and thus obtain as much information as possible simultaneously. This simulates keypoint descriptors and makes the detection more robust and efficient. These deep-learning approaches are primarily data-driven, meaning that the dataset used is very important and influences the final result. Dataset that is one of the main problems of underwater object pose estimation, both because of worse image quality and because there are very few project-specific datasets addressed.

All methods considered and analyzed are data-driven, i.e., the data used to train the various models is a key part of the entire pipeline. In this case the data used are only images representing the object, the background and possible obstacles. In the underwater environment, it is difficult

to create a dataset of real images and even more difficult to get the exact location of the object. For these reasons an unpaired image-to-image transformation strategy is used to obtain a synthetic adapted images dataset, called CycleGAN [50], instead to use a real dataset. Dataset obtained from an initial synthetic dataset divided, mainly, into background images and target object representations. After creating the synthetic adapted datasets we go on to test the 6-DoF object pose estimation approaches considered. Specifically, approaches with different strategies were considered in order to evaluate which one is the most suitable for our problem. In particular the focus will be on state-of-the-art approaches: DeepURL[22], PVNet[34] and DPOD[48].

In summary, the pipeline that will be followed is as follows:

- First creation of a synthetic dataset from background images and object images
- Second will be used a strategy to generate a synthetic adapted dataset and consequently obtain the right ground-truth for the next step
- Third a data-driven deep-learning strategy will be applied to perform the object pose estimation problem

In the following chapter, the various methods used for the 6DoF object pose estimation problem will be presented in detail, trying to describe them in the same way and following a single division into sections for consistency in comparing them. Specifically, the sections will be divided as follows: an introduction, the technical formulation of the model, various strategies implemented and the details implemented. Next, a chapter dedicated to the specific data-driven technique used to create dataset for the models. In the fourth chapter, the strategy to generate synthetic adapted datasets and subsequently used by data-driven techniques will be presented. The last chapter will present the experiments performed: describing the changes adopted on the models and their reasons, concluding with comparisons and the results obtained.

Chapter 2

Object Pose Estimation: the Analyzed Frameworks

As previously introduced, the goal of this thesis is to analyze effective solutions to the 6-DoF object pose estimation problem in underwater environments. In particular the focus will be on data-driven deep-learning methods, as they represent the state-of-art approaches: in particular, we will analyze the DeepURL[22], PVNet[34] and DPOD[48] frameworks. We opted for these methods because each develops a different approach to solving the object pose estimation problem, so we can compare which strategy is the most suitable for the underwater scenario. Specifically, DeepURL is already oriented to the underwater environment, while PVNet and DPOD are general. In this chapter, we will present these methods in detail, first introducing the basic techniques employed, then describing the individual pipelines.

All these approaches use the Random Sample Consensus (RANSAC) [9] based Perspective-n-Point (PnP) [27] algorithm, strategy used to find the 3D object projections of the 2D points estimated in the image. Therefore, importance will also be given to these two strategies in this chapter.

2.1 Random Sample Consensus

Random Sample Consensus (RANSAC) [9] is a paradigm for fitting a model to experimental data. RANSAC is capable of interpreting and smoothing data containing a significant percentage of gross errors and is thus ideally suited for applications in automated image analysis where interpretation is based on the data provided by error-prone feature detectors.

Scene analysis is about interpreting the received data against predefined models. Conceptually, this interpretation can be divided into two distinct activities:

- the problem of finding the best match between the data and one of the available models, so the classification problem
- the problem of computing the best values for the free parameters of the selected model, so the parameter estimation problem

These two problems are not independent, in fact to obtain a solution to the parameter estimation problem is often required to solve the classification problem.

Techniques for parameter estimation problem used before RANSAC, such as least squares, optimize, given a specified objective function, the fit of a functional description to all the data.

The worst problem of these techniques is they have no internal mechanisms for detecting and rejecting gross errors. These "classical" techniques are averaging techniques that rely on the smoothing assumption, where the maximum expected deviation of any datum from the assumed model is a direct function of the size of the dataset, and thus regardless of the size of the dataset, there will always be enough good values to smooth out any gross deviations. However, in many practical parameter estimation problems the smoothing assumption does not hold, i.e., the data contain uncompensated gross errors.

2.1.1 Formulation

The revolution of the RANSAC procedure is in being opposite to those conventional smoothing techniques. Rather than using as much as possible data to obtain an initial solution and then attempting to eliminate the invalid data points, RANSAC uses a small initial dataset and then enlarges this initial set with consistent data when possible. The RANSAC paradigm is more formally stated as follows [9]:

Given a model that requires a minimum of n data points to instantiate its free parameters, and a set of data points P such that the number of points in P is greater than n, this mean $f(P) \geq n$, randomly select a subset S_1 of n data points from P and instantiate the model. Use the instantiated model M_1 to determine the subset S_1^* of points in P that are within some error tolerance of M_l. The set S_1^* is called the consensus set of S_1.
If $f(S_1^*)$ is greater than some threshold t, which is a function of the estimate of the number of gross errors in P, use S_1^* to compute (possibly using least squares) a new model M_1^*.
If $f(S_1^*)$ is less than t, randomly select a new subset S_2 and repeat the above process. If, after some predetermined number of trials, no consensus set with t or more members has been found, either solve the model with the largest consensus set found, or terminate in failure.

2.1.2 Improvements

There are two obvious improvements to the above algorithm:

- if there is a problem related to selecting points to form the S's, it uses a deterministic selection process instead of a random one
- once a suitable consensus set S^* has been found and a model M^* instantiated, add any new points from P that are consistent with M^* to S^* and compute a new model based on this larger set

RANSAC paradigm contains three unspecified parameters:

- the error tolerance used to determine if a point is compatible with a model or not
- the number of subsets to try
- the threshold t, which is the number of compatible points used to imply that the correct model has been found

2.2 Perspective-n-Point

The aim of the Perspective-n-Point problem (PnP) is to determine the position and orientation of a camera given its intrinsic parameters and a set of n reference points correspondences between 3D points and their 2D projections. It has many applications in Computer Vision, Augmented Reality and Robotics. In this project is used for feature point-based camera tracking,

that require dealing with many noisy feature points in real-time, which requires computationally efficient methods. The better PnP algorithm known until now is EPnP [27], the authors introduce a non-iterative solution with better accuracy and much lower computational complexity than other methods, and much faster than iterative ones with little loss of accuracy. This approach is $O(n)$ for $n \geq 4$ whereas all other methods known are either specialized for small fixed values of n, very sensitive to noise, or much slower (Figure 2.1).

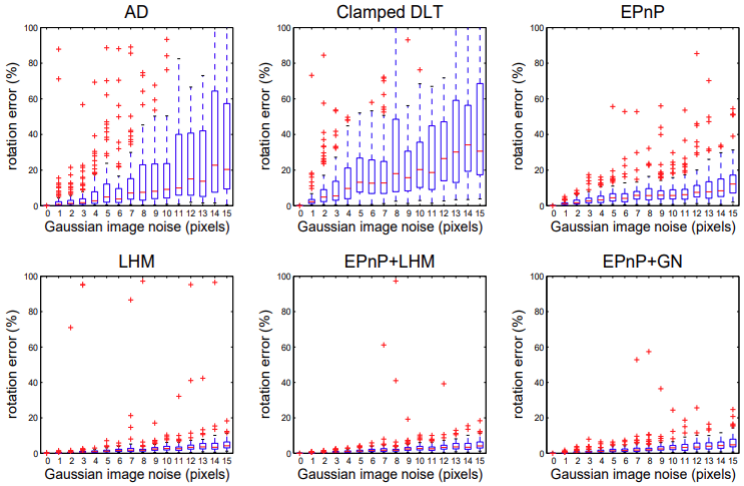


Figure 2.1: Comparing the accuracy of the authors method against state-of-the-art ones. They use the boxplot representation. The blue boxes denote the first and third quartiles of the errors, the lines extending from each end of the box depict the statistical extent of the data, and the crosses indicate observations that fall out of it. Top row. Accuracy of non-iterative methods as a function of noise when using $n = 6$ 3D-to-2D correspondences: AD is the method of Ansar and Daniilidis [1]; Clamped DLT is the DLT algorithm after clamping the internal parameters with their known values; and EPnP is this method. Bottom row. Accuracy of iterative methods using$n = 6$: LHM is Lu et al.'s method [31] initialized with a weak perspective assumption; EPnP+LHM is Lu et al.'s algorithm initialized with the output of this algorithm; EPnP+GN, this method followed by a Gauss-Newton optimization.

2.2.1 Formulation

Most of the approaches attempt to solve for the depths of the reference points in the camera coordinate system. Instead EPnP express their coordinates as a weighted sum of virtual control points. The key to this efficient implementation is the control points used: 4 non-coplanar for general configurations and 3 for planar configurations. The coordinates of these control points in the camera coordinate system became the unknown of the problem. In this way, for large n's, EPnP use much smaller unknown values rather than the traditional approaches that use more n depth values. The solution of the problem can be expressed as a vector that lies in the kernel of a matrix of size $2n \times 12$ for general configurations, or $2n \times 9$ for planar configurations. This matrix, called M, can be easily computed from the reference points between 3D world coordinates and 2D image projections. More precisely, it is a weighted sum of the null eigenvectors of M. Given that the correct linear combination is the one that preserve the distances that yields the 3D camera coordinates correspond to the control points. This can be done with a negligible computation cost by solving small systems of quadratic equations to find the best weights. In

fact, the most expensive computation, for n sufficiently large, is the matrix $M^T M$ computation, which grows linearly with n.

2.2.2 General Case

Let the n reference points in 3D coordinates known in the world coordinate system, be p_i , $i = 1, ..., n$. Similarly, let the 4 control points we use to express their world coordinates be c_j , $j = 1, ..., 4$. From now on will be used ^{w} for the point coordinates that are expressed in the world coordinate system and ^{c} for the camera coordinate system. Each reference point is expressed as a weighted sum of the control points

$$p_i^w = \sum_{j=1}^{4} \alpha_{ij} c_j^w \text{ , with } \sum_{j=1}^{4} \alpha_{ij} = 1 \,,$$

where the α_{ij} are homogeneous barycentric coordinates, which are uniquely defined and can easily be estimated. The same relation holds in the camera coordinate system and it is possible to write

$$p_i^c = \sum_{j=1}^{4} \alpha_{ij} c_j^c \,.$$

In theory the control points can be chosen arbitrarily. However, in practice, taking the centroid of the reference points as one increase the stability of the method, and select the rest in such a way that they form a basis aligned with the principal directions of the data. This normalization of the point coordinate amounts in the linear system of equations that are introduced below is very similar to the one recommended for the classic DLT algorithm [14].

2.2.3 Solution as Weighted Sum of Eigenvectors

The 2D projections of the reference points are known, in this way is possible to derive the solution inside the matrix M kernel. Let A be the internal camera calibration matrix and $\{u_i\}_{i=1,...,n}$ the 2D projection of the $\{p_i\}_{i=1,...,n}$ reference points, obtaining:

$$\forall i,\, w_i \begin{bmatrix} u_i \\ 1 \end{bmatrix} \,=\, A p_i^c \,=\, A \sum_{j=1}^{4} \alpha_{ij} c_j^c \,,$$

where the w_i are scalar projective parameters.

Considering for each control point c_j^c the specific 3D coordinates $\left[x_j^c,\, y_j^c,\, z_j^c\right]^T$. Also expand the matrix A: the 2D coordinates $[u_i,\, v_i]^T$ of the u_i projections, the f_u, f_v focal length coefficients and $(u_c,\, v_c)$ principal points. The previous equation becomes:

$$\forall i,\, w_i \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} \,=\, \begin{bmatrix} f_u & 0 & u_c \\ 0 & f_v & v_c \\ 0 & 0 & 1 \end{bmatrix} \sum_{j=1}^{4} \alpha_{ij} \begin{bmatrix} x_j^c \\ y_j^c \\ z_j^c \end{bmatrix} \,.$$

The unknown parameters of this linear system are the 12 control point coordinates $\{(x_j^c,\, y_j^c,\, z_j^c)\}_{j=1,...,4}$ and the n projective parameters $\{w_i\}_{i=1,...,n}$. The last row implies that $w_i = \sum_{j=1}^{4} \alpha_{ij} z_j^c$. Substituting this expression in the first two rows yields two linear equations for each reference point:

$$\begin{cases} \sum_{j=1}^{4} \left(\alpha_{ij} f_u x_j^c \,+\, \alpha_{ij} (u_c - u_i) z_j^c \,=\, 0 \right) \\ \\ \sum_{j=1}^{4} \left(\alpha_{ij} f_v y_j^c \,+\, \alpha_{ij} (v_c - v_i) z_j^c \,=\, 0 \right) \end{cases} \,,$$

where the w_i projective parameter does not appear anymore in those equations.

Concatenating the all n reference points, it is possible to generate a linear system of the form

$$Mx = 0\,,$$

where $x = [c_1^{c^T}, c_2^{c^T}, c_3^{c^T}, c_4^{c^T}]^T$ is an unknown 12-vector. Also M is a $2n \times 12$ matrix, generated by arranging the coefficients of the system for each reference point.

The solution therefore belongs to the null space, or kernel, of M, and can be expressed as

$$x = \sum_{i=1}^{N} \beta_i v_i\,,$$

where the set v_i are the columns of the right-singular vectors of M corresponding to the N null singular values of M. They can be found efficiently as the null eigenvectors of matrix $M^T M$, which is of small constant (12×12) size. Computing the product $M^T M$ has $O(n)$ complexity, and is the most time consuming step in the method if n is sufficiently large.

2.2.4 Choosing the Right Linear Combination

From the previous equation the solution is to compute the appropriate values for the coefficients $\{\beta_i\}_{i=1,\ldots,N}$, this is because the solution can be expressed as a linear combination of the null eigenvectors of $M^T M$. In theory, with the scale ambiguity, given perfect data from at least six reference points captured by a perspective camera, the dimension N of the null-space of $M^T M$ should be exactly one. Note that a orthographic camera instead of perspective one, the dimension of the null space increases to four, because changing the depths of the four control points would have no influence on where the reference points project. Figure 2.2 illustrates this behavior, given by the authors. For small focal lengths, $M^T M$ has only one zero eigenvalue. However, with the increment of the focal length and the camera becomes closer to being orthographic, all four smallest eigenvalues tend to zero. Furthermore, if the correspondences are noisy, the smallest eigenvalue of them will be tiny but not strictly equal to zero. Depending on the configuration of the reference points, the focal length of the camera and the amount of noise, the effective dimension N of the null space of $M^T M$ can be considered from 1 to 4. Figure 2.3 shows this.

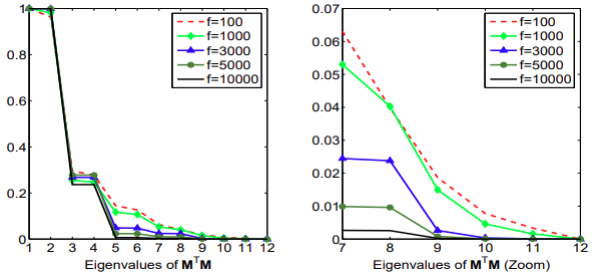


Figure 2.2: Left: Singular values of $M^T M$ for different focal lengths. Each curve averages 100 synthetic trials. Right: Zooming in on the smallest eigenvalues. For small focal lengths, the camera is perspective and only one eigenvalue is zero, which reflects the scale ambiguity. As the focal length increases and the camera becomes orthographic, all four smallest eigenvalues tend to zero.

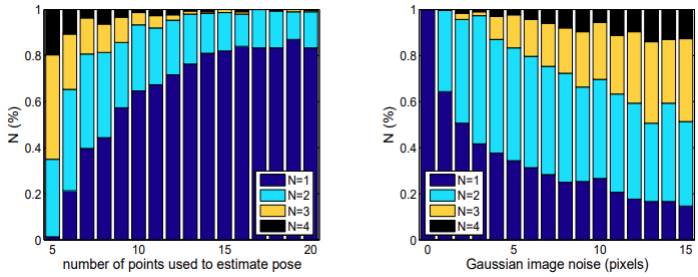


Figure 2.3: Effective number N of null singular values in $M^T M$. Each vertical bar represents the distributions of N for a total of 300 experiments, taken by the authors. Left: the results plot for a fixed image noise of $\sigma = 10$ pixels and an increasing number of reference points. Right: the results correspond to a fixed $n = 6$ number of reference points and increasing level of noise in the 2D projections.

In practice, several eigenvalues can have similar magnitudes, for this reason trying to pick a value of N among the set $\{1, 2, 3, 4\}$ can be error prone. Instead is a better idea compute solutions for all four values of N and keep the one that yields the smallest reprojection error:

$$\text{res} = \sum_i \text{dist}^2(A[R|t] \begin{bmatrix} p_i^w \\ 1 \end{bmatrix}, u_i) ,$$

where $\text{dist}(\widetilde{m}, n)$ is the 2D distance between point m expressed in homogeneous coordinates and point n. This improves robustness without any noticeable computational penalty, because the most expensive operation is the computation of $M^T M$, which is done only once, and not the solving of a few quadratic equations. The distribution of values of N estimated in this way is depicted by Figure 2.3.

The description for the quadratic constraints of $N = 1, 2, 3, 4$ is introduced:

Case $N = 1$: The solution is simply $x = \beta v$. β can be solved by computing the distances between control points in the camera coordinate system that should be equal to the ones computed in the world coordinate system when using the given 3D coordinates. Let $v^{[i]}$ be the sub-vector of v that corresponds to the coordinates of the control point c_i^c. Specifically, $v^{[1]}$ will represent the vectors made of the first three elements of v. Maintaining the distance between pairs of control points (c_i, c_j) implies that

$$\|\beta v^{[i]} - \beta v^{[j]}\|^2 = \|c_i^w - c_j^w\|^2 .$$

Since the distance $\|c_i^w - c_j^w\|^2$ is known, computing β in closed-form becomes:

$$\beta = \frac{\sum_{\{i,j\} \in [1;4]} \|\beta v^{[i]} - \beta v^{[j]}\| \cdot \|c_i^w - c_j^w\|}{\sum_{\{i,j\} \in [1;4]} \|\beta v^{[i]} - \beta v^{[j]}\|^2} .$$

Case $N = 2$: Now the solution is $x = \beta_1 v_1 + \beta_2 v_2$, and the distance constraints become:

$$\|(\beta_1 v_1^{[i]} + \beta_1 v_1^{[j]}) - (\beta_2 v_2^{[i]} + \beta_2 v_2^{[j]})\|^2 = \|c_i^w - c_j^w\|^2 .$$

β_1 and β_2 only appear in the quadratic terms and they can be solved using a technique called *linearization* in cryptography, which was employed by [1] to estimate the point depths. It involves to solve a linear system in $[\beta_{11}, \beta_{12}, \beta_{22}]^T$ where $\beta_{11} = \beta_1^2$, $\beta_{12} =$

$\beta_1\beta_2$, $\beta_{22} = \beta_2^2$. Since there are four control points, this produces a linear system of six equations in the β_{ab}, where indices a and b for the β's indicate the differentiate from the indices i and j used for the 3D points. So it is possible to write:

$$L\beta \;=\; \rho\,,$$

where L is a 6×3 matrix composed of the elements of v_1 and v_2, ρ is a 6-vector with the squared distances $\|c_i^w - c_j^w\|^2$, and $\beta = [\beta_{11}, \beta_{12}, \beta_{22}]^T$ is the vector of unknowns. To solve this system it can be used the pseudoinverse of L and choose the signs for the β_a so that all the p_i^c have positive z coordinates. This imply β_1 and β_2 values can be further refined by using the formula of Equation [11] to estimate a common scale β so that

$$c_i^c = \beta(\beta_1 v_1^{[i]} + \beta_2 v_2^{[i]})\;.$$

Case $N = 3$**:** As in the $N = 2$ case, it is possible to use the six distance constraints of Equation [12]. This yields again a linear system $L\beta \;=\; \rho$, although with larger dimensionality. Now L is a square 6×6 matrix formed with the elements of v_1, $v_2$and $v3$, and β becomes the 6D vector $[\beta_{11}, \beta_{12}, \beta_{13}, \beta_{22}, \beta_{23}, \beta_{33}]^T$. Now is possible to follow the same procedure as before, except that take the inverse of L instead of its pseudo-inverse.

Case $N = 4$**:** Now with four unknown β_a and the six distance constraints, in theory, should be sufficient to compute the solution. Unfortunately, the *linearization* procedure treats all 10 products $\beta_{ab} = \beta_a\beta_b$ as unknowns and there are not enough constraints anymore. Fortunately the problem can be solved with another strategy, the *relinearization* technique [16], whose principle is the same as the previous one used to determine the control points coordinates. The solution for the β_{ab} is in the null space of a first homogeneous linear system made from the original constraints. The correct coefficients are found by introducing new quadratic equations and solving them again by *linearization*, hence the name *relinearization*. These new quadratic equations are derived from the fact that there is, by commutativity of the multiplication

$$\beta_{ab}\beta_{cd} \;=\; \beta_a\beta_b\beta_c\beta_d \;=\; \beta_{a'b'}\beta_{c'd'}\;,$$

where $\{a', b', c', d'\}$ represents any permutation of the integers $\{a, b, c, d\}$.

2.2.5 Planar Case

In the planar case, that is, when the moment matrix of the reference points has one very small eigenvalue, is needed only three control points. The dimensionality of M is then reduced to $2n \times 9$ with 9D eigenvectors v_i, but the above equations remain mostly valid. The main difference is that the number of quadratic constraints drops from 6 to 3. As a consequence, the relinearization technique is introduced in the $N = 4$ case of the previous section for $N \geq 3$.

2.2.6 Gauss-Newton Optimization

This section will show that these closed-form solutions are more accurate than other state-of-the-art non-iterative methods. Also, this algorithm is faster than the best known by the authors [31]. However, it may be less accurate, especially when an iterative algorithm is initialized with good input. To overcome this problem, a refinement procedure will be introduced that will increase the accuracy at the expense of very little extra computational cost. The Figures 2.1 and 2.4 show that this closed-form approach with refinement has the same accuracy as the method examined by the authors [20], but still much faster.

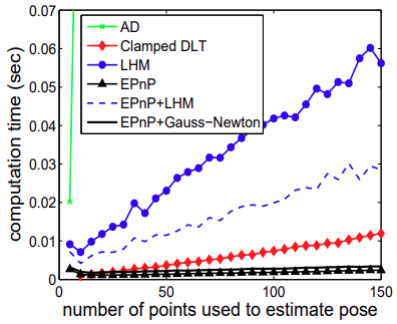


Figure 2.4: Comparing computation times of EPnP method against the state-of-the-art ones introduced in Figure 2.1. The computation times are plotted as a function of the number of correspondences. This method is both more accurate and faster than the other non-iterative ones, especially for large amounts of noise, and is almost as accurate as the iterative LHM. Furthermore, if maximal precision is required, the output of EPnP algorithm can be used to initialize a Gauss-Newton optimization procedure which requires a negligible amount of additional time

Then choosing the four values $\beta = [\beta_1, \beta_2, \beta_3, \beta_4]^T$ that minimize the change in distance between control points. Specifically, the use of Gauss-Newton algorithm to minimize:

$$\text{Error}(\beta) = \sum_{(i,j)\ s.t.\ i<j} (\|c_i^c - c_j^c\|^2 - \|c_i^w - c_j^w\|^2) \ ,$$

with respect β. The distances $\|c_i^w - c_j^w\|^2$ in the world coordinate system are known and the control point coordinates in camera reference are expressed as a function of the β coefficients as

$$c_i^c = \sum_{j=1}^{4} \beta_j v_j^{[i]} \ .$$

Since the optimization is performed only over the four β_i coefficients, its computational complexity is independent of the number of input 3D-to-2D correspondences. This yields fast and constant time convergence since, in practice, less than 10 iterations are required. As a result, the computational burden associated to this refinement procedure is almost negligible as can be observed in Figure 2.4. In fact, the time required for the optimization may be considered as constant, and hence, the overall complexity of the closed-form solution and Gauss-Newton remains linear with the number of input 3D-to-2D correspondences.

2.3 Deep Underwater Relative Localization

The big picture of DeepURL is a Convolutional Neural Network (CNN), in particular a modified version of YOLOv3 [37], that predicts the 2D projections of the 8 corners of the AUV 3D model, like [36, 42], and an object detection bounding box. Some approaches divide an image into grid cells, they use global estimates of 2D keypoints for the object with the highest confidence value. In DeepURL approach, each grid cell inside the bounding box predicts the 2D projections of keypoints along with their confidences focusing on local regions belonging to the object. These predictions of all cells are then combined based on their corresponding confidence scores using RANSAC-based PnP during 6DoF pose estimation.

2.3.1 Formulation

The proposed network consists of an encoder, Darknet53 [37], and two decoders: Detection Decoder and Pose Regression Decoder. The detection decoder detects objects with bounding boxes, and the pose regression decoder regresses to 2D corner keypoints of the 3D object model. The decoders predict a 3D tensor with a spatial resolution of S $\times$ S and a dimension of D_{det} and D_{reg}, respectively. Figure 2.5 show an overview scheme.

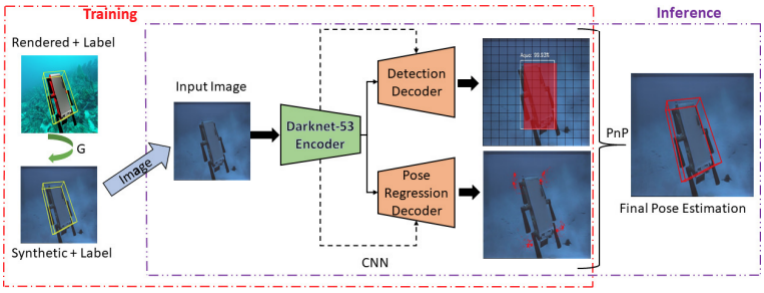


Figure 2.5: In training (outlined in red), the rendered images are translated to the synthetic images resembling the AUV in a pool or a ocean environment. The synthetic images are then fed to a common encoder, which is connected to two decoder streams: Detection Decoder (object detection) and Pose Regression Decoder (6D pose regression). Only in inference (outlined in purple), are the predicted 2D keypoint projections of 8 corners of the AUV model processed and utilized to obtain a 6D pose using the RANSAC-based PnP algorithm.

2.3.2 YOLO

DeepURL uses as backbones encoder network Darknet-53 from YOLOv3, the bounding boxes prediction system. YOLOv3 is the third version of YOLO system [38], "You Only Look Once". In general, this object detection is a regression problem to spatially separated bounding boxes and associated class probabilities. This is done by a single neural network that predicts directly from full images in one evaluation. Since the whole detection pipeline is a single network has several benefits. First, YOLO is extremely fast, since the frame detection is a regression problem there are no need for a complex pipeline. Furthermore, it can be optimized end-to-end directly on detection performance. Second, YOLO reasons globally about the image when making predictions. In fact, it sees the entire image during the training, so it implicitly encodes contextual information about classes as well as their appearance. Third, YOLO learns generalizable representations of objects. Since it is highly generalizable, with new domains or unexpected inputs it is less likely to break down.

In summary, this system takes a classifier for that object and evaluate it at various locations and scales in the image. Systems like deformable parts models (DPM) use a sliding window approach where the classifier is run at evenly spaced locations over the entire image. Figure 2.6 shows the model.

The neural network used by YOLOv3 is Darknet-53. It is a standard CNN with a first part of convolutional layers and at the end a fully-connected layers follow by a softmax function. From the previous version that used a 19 convolutional layers network, called Darknet-19, this third version uses a 53 convolutional layers network. Figure 2.7 shows the architecture. This increase in the number of layers does not reduce performance; in fact, it remains faster than many other state-of-the-art networks.

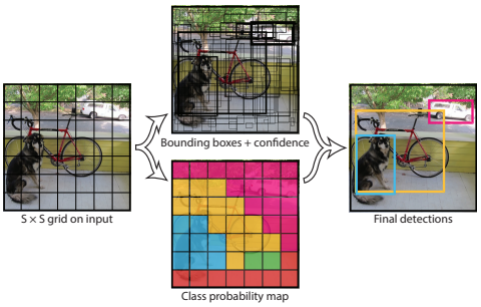


Figure 2.6: The Model detection as a regression problem. It divides the image into an S × S grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 2.7: Darknet-53 architecture.

After this data extraction by Darknet-53, the estimation of bounding boxes is done through a system by YOLOv3, that predicts boxes at 3 different scales. In particular, extracts features from those scales using a similar concept to feature pyramid networks [28]. From the base feature extractor it adds several convolutional layers, the last of these predicts a 3-d tensor encoding bounding box, objectness, and class predictions. YOLO predicts 3 boxes at each scale so the tensor in output is $N \times N \times [3 \cdot (4 + 1 + C)]$ for the 4 bounding box offsets, 1 objectness prediction and C class predictions. Next it takes the feature map from 2 layers previous and upsample it by 2×. Also it takes a feature map from earlier in the network and merge it with the upsampled features using concatenation. This method allows to get more meaningful semantic information from the upsampled features and finer-grained information from the earlier feature map. Then add a few more convolutional layers to process this combined feature map, and eventually predict a similar tensor, although now twice the size. The same design is performed one more time to predict boxes for the final scale. Thus predictions for the 3rd scale benefit from all the prior computation as well as finegrained features from early on in the network.

This bounding boxes prediction system is used by DeepURL, as you will see in the next sections.

2.3.3 Object Detection Stream

The object detection stream is like the detection stream of YOLOv3 [37] which predicts object bounding box. For each grid cell at offset (cx, cy) from the top left corner of the image, the network predicts 4 coordinates for each bounding box: t_x, t_y, t_w, t_h. Following Darknet53, it uses 9 anchor boxes obtained by k-means clustering on COCO dataset [29] of size (10×13), (16×30), (33×23), (30×61), (62×45), (59×119), (116×90), (156×198), (373×326) divided among three scales. The width and height are predicted as the fraction of the anchor box priors (p_w, p_h) and the actual bounding box values are obtained as

$$b_x = \sigma(t_x) + c_x,$$

$$b_y = \sigma(t_y) + c_y,$$

$$b_w = p_w e^{t_w},$$

$$b_h = p_h e^{t_h}$$

where σ represents the sigmoid function. Figure 2.8 shows the scheme.

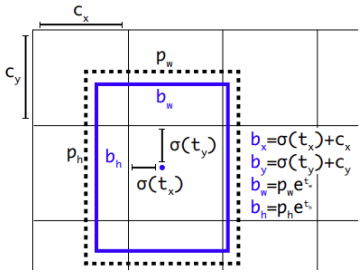


Figure 2.8: Bounding boxes with dimension priors and location prediction. The model predicts the width and height of the box as offsets from cluster centroids and it predicts the center coordinates of the box relative to the location of filter application using a sigmoid function.

The sum of square of error between the ground truth t_* and coordinate prediction $\hat{t}_*$ is used as the loss function. The ground truth values t_* can be obtained by inverting equation Eq. (3). The object detection stream also predicts the objectness score of each bounding box by calculating its intersection over union with anchor boxes and class prediction scores using independent logistic classifiers. The total object detection loss L_{det} is the sum of coordinate prediction loss, objectness score loss, and class prediction loss [38]:

$$L_{det} = \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbf{1}_{ij}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbf{1}_{ij}^{obj} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbf{1}_{ij}^{obj} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbf{1}_{ij}^{noobj} \left(C_i - \hat{C}_i \right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbf{1}_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2$$

where $\mathbf{1}_i^{obj}$ denotes if object appears in cell i and $\mathbf{1}_{ij}^{obj}$ denotes that the j^{th} bounding box predictor in cell i is "responsible" for that prediction. Also this object detection loss increases the loss from bounding box coordinate predictions and decrease the loss from confidence predictions for boxes that do not contain objects. Parameters λ_{coord} and λ_{noobj} is used to accomplish this. The authors for better results set $\lambda_{coord} = 5$ and $\lambda_{noobj} = 0.5$. Note that the loss function penalizes classification error if an object is present in that grid cell and also penalizes bounding box coordinate error if that predictor is "responsible" for the ground truth box.

2.3.4 Pose Regression Stream

The pose regression stream predicts the location of the 2D projections of the 3D keypoints associated with the 3D object model of the AUV. The 8 corner points of the model bounding boxes is used as keypoints. The pose regression stream predicts a 3D tensor with size S × S × D_{reg}, where $D_{reg} = 3 \times 8$ is the (x, y) spatial locations for the 8 keypoint projections along with their confidence values. So, the 2D coordinates of the 2D keypoints are not directly predict, instead it is estimated the offset of each keypoint from the corresponding grid cell as in Figure 2.9(b). Let c be the position of grid cell from top left image corner. For each i^{th} keypoint, it predicts the offset $f_i(c)$ from grid cell, so that the actual location in image coordinates becomes $c + f_i(c)$, which should be close to the ground truth 2D locations g_i . The residual is calculated as

$$\Delta_i(c) = c + f_i(c) - g_i$$

and it defines the offset loss function, L_{off}, for spatial residual:

$$L_{off} = \sum_{c \in B} \sum_{i=1}^{8} \|\Delta_i(c)\|_1,$$

where B consists of grid cells that fall inside the object bounding box. Apart from the 2D keypoint locations, the pose regression stream also calculates confidence value for each predicted keypoint $v_i(c)$, which is obtained through the sigmoid function on the network output. The confidence value is a representation of the distance between the predicted keypoint and ground truth values and is used a sharp exponential function of the 2D euclidean distance. The confidence loss is calculated as

$$L_{conf} = \sum_{c \in B} \sum_{i=1}^{8} \|v_i(c) - exp(-\alpha \|\Delta_i(c)\|_2)\|_1,$$

where parameter α defines the sharpness of the exponential function. The final pose regression loss is:

$$L_{reg} = \lambda_{off} L_{off} + \lambda_{conf} L_{conf},$$

for numerical stability, the authors down-weight the confidence loss for cells that do not contain objects by setting λ_{conf} to 0.1, as suggested in [38]. For the cells that include the object, λ_{conf} is set to 5.0 and λ_{off} to 1.

Therefore, the total loss of the network is:

$$L = L_{det} + L_{reg}.$$

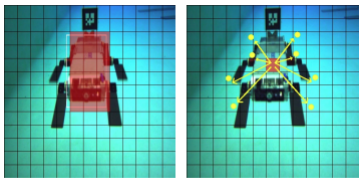


Figure 2.9: (a) The object detection stream predicts the bounding box and assigns each cell inside the box to the AUV object. (b) The regression stream predicts the location of 8 bounding box corners as 2D keypoints from each grid cell.

2.3.5 Pose refinement

To achieve a better results is apply a pose refinement strategy. During inference, the object detection stream predicts the coordinate locations of the bounding boxes with their confidences and the class probabilities for each grid cell. Then, the class-specific confidence score is estimated for the object by multiplying the class probability and confidence score. To select the best bounding box, it is used a non-max suppression [39] (Expand this?) with an IOU threshold of 0.4 and a class-specific confidence score threshold of 0.3.

Simultaneously, the pose regression stream produces the projected 2D locations of the object's 3D bounding box, along with their confidence scores for each grid cell, as shown in Figure 2.10(b). The 2D keypoints estimated for each grid cells that fall outside the bounding box, as shown in Figure 2.10(a), predicted by the object detection stream are filtered out. In an ideal case, the remaining 2D keypoints should cluster around the object center. The 2D keypoints does not belong to a cluster are removed using a pixel distance threshold of 0.3 times the image width. The keypoints with confidence scores less than 0.5 are also filtered out. To balance the computation time and accuracy, the authors of the paper empirically found that using the 12 most confident 2D predictions for each 3D keypoint produces an acceptable pose estimate after RANSACbased PnP [27], as visualize in Figure 2.10(c). To obtain a robust pose estimation is used a RANSAC-based PnP method on $12 \times 8 = 96$ 2D-to-3D correspondence pairs between the image keypoints and the object's 3D model, as shown in Figure 2.10(d).

2.3.6 Implementation details

Create the synthetic dataset following the training procedure of CycleGAN and let the training continue until it generated acceptable reconstruction. Once CycleGAN can reasonably reconstruct for the target domain, use the model weights of that epoch to translate all rendered images to synthetic images. Then, the synthetic to the CNN.

Darknet-53 is trained on the synthetic dataset, where the first 3 epochs are part of a warmup phase: the learning rate gradually increases from 0 to 1e-4. Is utilized the SGD optimizer with a momentum of 0.9 and a piecewise decay to decrease the learning rate to 3e-5 at an intermediate number of epochs and 1e-5 for the last remaining epochs. Also it uses minibatches to avoid the overfitting(, minibatches of size 8 were produced by applying data augmentation techniques, including randomly changing hue, saturation, and exposure of the image up to a factor of 1.5).

2.4 Pixel-wise Voting Network

PVNet instead of estimating keypoints directly creates a vector-field representation for keypoint localization. The basic idea is illustrated in Figure 2.11.

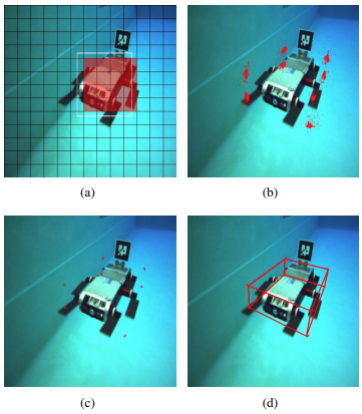


Figure 2.10: Inference strategy for combining pose candidates. (a) Grid cells inside the detection box belonging to AUV object overlaid on the image. (b) Each grid predicts 2D locations for corresponding 3D keypoints shown as red dots. (c) For each keypoints, 12 best candidates are selected based on the confidence scores. (d) Using $12 \times 8 = 96$ 2D-to-3D correspondence pairs and running RANSAC-based PnP algorithm yield accurate pose estimate as shown by the overlaid bounding box.

2.4.1 Formulation

The strategy of predicting the vector-field representation for the keypoints localization enforces the network to focus on local features of objects and spatial relations between object parts. As a result, this vector-field representation can represent the location of an invisible part inferred from the visible parts, so the object keypoints that are even outside the input image. All these advantages make it an ideal representation for occluded or truncated objects. Another advantage of PVNet approach is that the dense outputs provide rich information for the PnP algorithm to deal with inaccurate keypoints estimation. Firstly RANSAC-based voting prunes outlier predictions and gives a spatial probability distribution for each keypoint. Then the PnP solver has more freedom from the uncertainty keypoint locations to identify consistent correspondences for predicting the final pose. In summary PVNet uses a two-stage pipeline: first detect 2D object keypoints using CNNs and then compute 6D pose parameters using the PnP algorithm, as shows by the Figure 2.12.

2.4.2 ResNet

PVNet uses a pretrained ResNet-18 as the backbone network for the first stage. The big picture behind ResNet [15] is a residual learning framework to simplify the training of networks and reformulate the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions.

Deep networks naturally integrate low/mid/high level features and classifiers in an end-to-end multilayer fashion, and the "levels" of this features can be enriched by the deeper layers. Network depth is a crucial importance, but this increase even the number of the problems. The notorious problem of vanishing and exploding gradients is an hamper convergence from the beginning. However, it has been largely addressed by normalized initialization and intermediate normalization layers, which enable networks start converging for stochastic gradient descent (SGD) with backpropagation. Another problem related to deeper networks is a degradation problem: accuracy gets saturated and then degrades rapidly. Unfortunately, such degradation

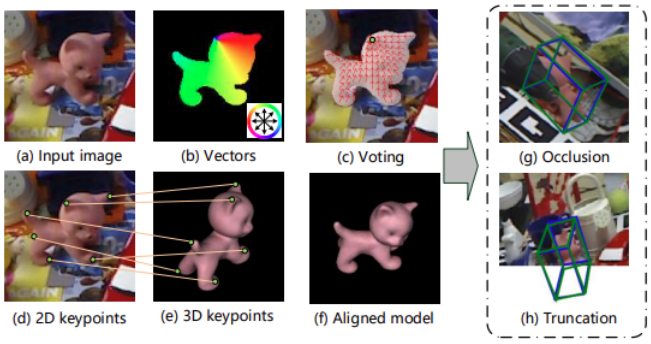


Figure 2.11: The 6D pose estimation problem is formulated as a Perspective-n-Point (PnP) problem, which requires correspondences between 2D and 3D keypoints, as illustrated in (d) and (e). PVNet predicts vectors pointing to keypoints for each pixel, as shown in (b), and localize 2D keypoints in a RANSAC-based voting scheme, as shown in (c). Most advantages of this method is robustness to occlusion (g) and truncation (h), where the green bounding boxes represent the ground truth pose and the blue bounding boxes represent the prediction.

is not caused by overfitting, and adding more layers to a suitably deep model leads to higher training error. In fact deeper network has higher training and test error, as shows Figure 2.13.

The degradation problem, of training accuracy, indicates that not all models are similarly easy to optimize. Let is consider a shallow architecture and its deeper counterpart that has more hidden layers onto it. To avoid an higher training error of deeper model than its shallower counterpart exists a construction solution: the added layers are identity mapping, and the other layers are copied from the learned shallower model. However experiments conducted by the authors show that current solvers are unable to find solutions that are comparably good or better than the constructed solution, or are unable to do so in feasible time 2.14.

ResNet addresses the degradation problem by introducing a deep residual learning framework.

Consider an underlying mapping to be fit by a few stacked layers $H(x)$, not necessarily represent the entire network, with x denoting the inputs to the first layer. Hypothesize that multiple nonlinear layers can asymptotically approximate complicated functions, then it is equivalent to hypothesize that they can asymptotically approximate the residual functions, i.e., $H(x) - x$, assuming the input dimension is equivalent to the output dimension. So rather than expect stacked layers approximate $H(x)$, the model can explicitly let these layers approximate a residual function $F(x) := H(x) - x$. The original mapping becomes $F(x) + x$. Although both forms should be able to asymptotically approximate the desired functions, the ease of learning might be different. So the added layers can be constructed as identity mappings and a deeper model should have training error no greater than its shallow counterpart. The degradation problem suggests that the solvers might have difficulties in approximating identity mappings by multiple nonlinear layers. To approach identity mappings with the residual learning reformulation, the models should simply drive the weights of the multiple nonlinear layers toward zero, if the identity mappings are optimal. In real cases, it is unlikely that identity mappings are optimal, but it may help to precondition the problem. Also, if the optimal function is closer to an identity mapping than to a zero mapping, it should be easier for the model to find the perturbations with reference to an identity mapping, than to learn the function as a new one.

For every few stacked layers, considering the building blocks, ResNet uses the residual learn-

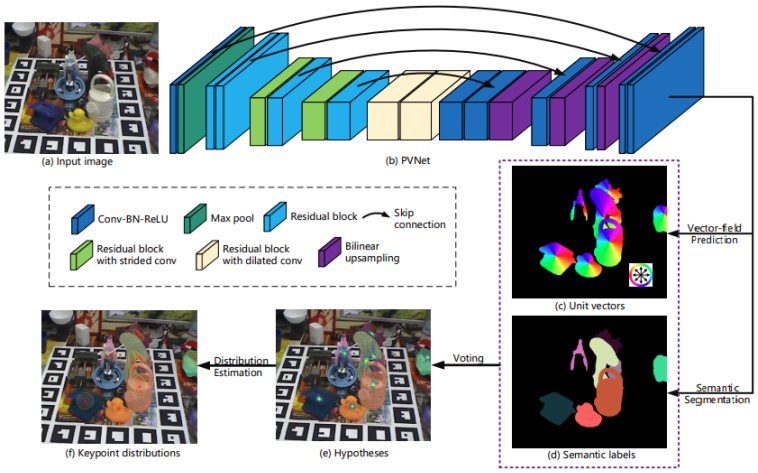


Figure 2.12: Overview of the keypoint localization: (a) An image of the Occlusion LINEMOD dataset. (b) The architecture of PVNet. (c) Pixel-wise vectors pointing to the object keypoints. (d) Semantic labels. (e) Hypotheses of the keypoint locations generated by voting. The hypotheses with higher voting scores are brighter. (f) Probability distributions of the keypoint locations estimated from hypotheses. The mean of a distribution is represented by a red star and the covariance matrix is shown by ellipses.

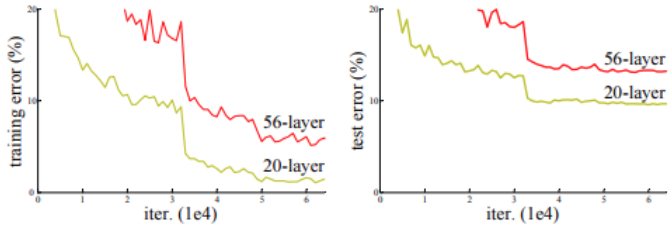


Figure 2.13: Training error (left) and test error (right) on CIFAR-10 [24] with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Figure 2.14. Results obtained by the authors of ResNet.

ing. Formally, a building block is defined as:

$$y = F(x, W_i) + x,$$

where x and y are the input and output vectors of the layers considered. The function $F(x, W_i)$ represents the residual mapping to be learned. Figure 2.15 shows an example. In this case there are two layers, where $F = W_2\sigma(W_i x)$ in which σ denotes RELU [resnet cite 29] and the biases are omitted for simplifying notations. Specifically, the operation $F + x$ is performed by a shortcut connection and element-wise addition, this mean F and x must have the same dimension. Then apply the nonlinearity after the last layer. If the two dimensions are not equal (e.g., when changing the input/output channels), it is possible to perform a linear projection W_s by the shortcut connections to match the dimensions:

$$y = F(x, W_i) + W_s x$$

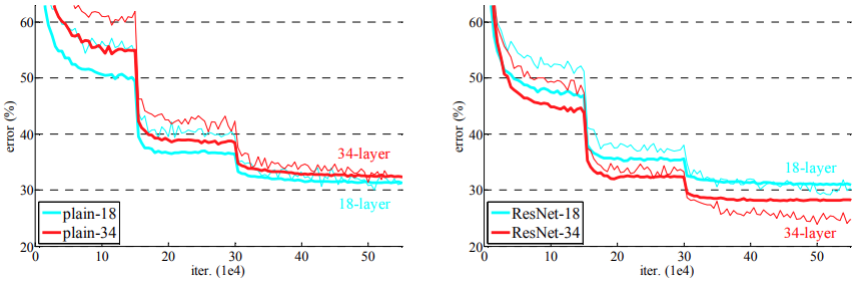


Figure 2.14: Training on ImageNet [40]. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts. Results obtained by the authors of ResNet.

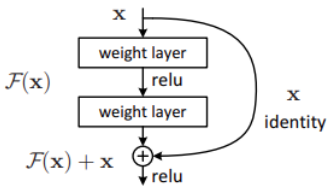


Figure 2.15: Residual learning: a building block.

The shortcut connections introduce neither extra parameter nor computation complexity. This is not only attractive in practice but also important in the comparison between plain and residual networks. In fact, plain and residual networks can simultaneously have the same number of parameters, depth, width, and computational cost, except for the negligible element-wise addition.

The form of the residual function F can be flexible. Experiments from the authors involve a function F that has two or three layers, as shown on Figure 2.16, but more layers are possible. Instead with only a single layer, the building block is similar to a linear layer: $y = W_1 x + x$, that not take advantages.

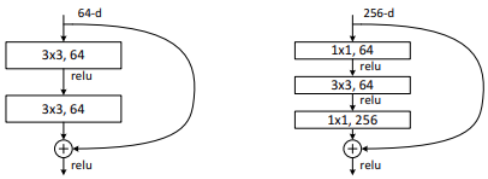


Figure 2.16: Left: a building block (on 56×56 feature maps) for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

This strategy can be apply to fully-connected and convolutional layers. The above notations are about fully-connected layers for simplicity, but the mapping $F(x, W_i)$ can represent multiple convolutional layers, where element-wise addition is performed on two feature maps, channel by channel.

2.4.3 Voting-based keypoint localization

Given an RGB image, PVNet predicts pixel-wise object labels and vectors that represent the direction from every pixel to every keypoint. In this way, it generates hypotheses of 2D locations for that keypoints like the confidence scores of RANSAC-based voting. Based on these hypotheses, the mean and covariance of the spatial probability distribution for each keypoint are estimate. More specifically, PVNet performs two tasks: semantic segmentation and vector-field prediction. For a pixel p, PVNet outputs the semantic label that associates it with a specific object and the vector $v_k(p)$ that represents the direction from the pixel p to a 2D keypoint x_k of the object. In addiction a vector $v_k(p)$ could be the offset between the pixel and the keypoint. In this way, it obtains the target object pixels and add the offset to generate a set of keypoint hypotheses. However, scale changes of the object penalize these offset, which limits the generalization ability of PVNet. For this reason the vector computation is scale-invariant

$$v_k(p) = \frac{x_k - p}{\|x_k - p\|_2},$$

which only cares the relative direction between objects parts.

To a better results the keypoint hypotheses are generated by a RANSAC-based voting scheme, given target object pixels and unit vectors. First, choose randomly two pixels and take the intersection of their vectors as a hypothesis $h_{k,i}$ associated to the keypoint x_k. This step is repeated N times to generate a set of hypothesis $\{h_{k,i}|i = 1, 2, ..., N\}$ that represent possible keypoint locations. Then, all pixels of the object vote for these hypothesis. Specifically, the voting score $w_{k,i}$ of a hypothesis $h_{k,i}$ is defined as

$$w_{k,i} = \sum_{p \in O} \mathbb{I}\left(\frac{(h_{k,i} - p)^T}{\|h_{k,i} - p\|_2} v_k(p) \geq \theta \right),$$

where $\mathbb{I}$ represents the indicator function, θ is a threshold (0.99 usually), and $p \in O$ means that the pixel p belongs to the object O. Intuitively, a higher voting score means that a hypothesis is more confident as it coincides with more predicted directions.

The resulting hypotheses characterize the spatial probability distribution of a keypoint in the image, as shown by Figure 2.12(e). Finally, the mean μ_k and the covariance Σ_k for a keypoint x_k are estimated by:

$$\mu_k = \frac{\sum_{i=1}^{N} w_{k,i} h_{k,i}}{\sum_{i=1}^{N} w_{k,i}},$$

$$\Sigma_k = \frac{\sum_{i=1}^{N} w_{k,i} (h_{k,i} - \mu_k)(h_{k,i} - \mu_k)^T}{\sum_{i=1}^{N} w_{k,i}},$$

which are used for uncertainty-driven PnP solver.

2.4.4 Keypoint selection

As shown by Figure 2.17(a) the bounding box corners may be far from the object pixels in the image and this longer distance can generate a larger localization error. Other methods like DeepURL, use the eight corners of the 3D bounding box of the object as the keypoints, so they are more error driven. This mean the keypoints need to be defined based on the 3D object model. Figure 2.17(b) and (c)) show the hypotheses of a bounding box corner and a keypoint selected on the object surface, respectively, which are generated by our PVNet. The keypoint on the object surface usually has a much smaller variance in localization. For this reason PVNet

(a) (b) (c)

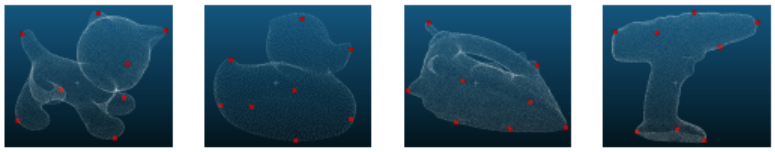
Figure 2.17: (a) A 3D object model and its 3D bounding box. (b) Hypotheses produced by PVNet for a bounding box corner. (c) Hypotheses produced by PVNet for a keypoint selected on the object surface. The smaller variance of the surface keypoint shows that it is easier to localize the surface keypoint than the bounding box corner in our approach.

select keypoint on the onject surface. In addiction, these keypoints should spread out on the object to make the PnP algorithm more stable.

To select K keypoint is used the farthest point sampling (FPS) algorithm:

1. initialize the keypoint set by adding the object center
2. find a point on the object surface, which is farthest to the current keypoint set
3. add the keypoint to the set
4. repeat until the size of the set reaches K

Considering both accuracy and efficiency, the authors suggest K = 8 according to their experiment results. Figure 2.18 visualizes examples of selected keypoints of some objects.



cat duck iron driller

Figure 2.18: Keypoints of four objects in the LINEMOD [18] dataset.

2.4.5 Uncertainty-driven PnP

Given 2D keypoint locations for each object, its 6D pose problem can be computed by solving the EPnP problem, like DeepURL. However, with different keypoints selection PVNet have different confidences and uncertainty patterns, which effects the PnP solver.

Given the estimated mean μ_k and covariance matrix Σ_k for $k = 1, ..., K$ the related 6DoF pose problem (R, t) by minimizing the Mahalanobis distance is:

$$\underset{R,t}{\text{minimize}} \sum_{k=1}^{K} (\widetilde{x}_k - \mu_k)^T \Sigma_k^{-1} (\widetilde{x}_k - \mu_k),$$

$$\widetilde{x}_k = \pi(RX_k + t),$$

where X_k is the 3D coordinate of the keypoint, $\widetilde{x}_k$ is the 2D projection of X_k and π is the perspective projection function. The parameters R and t are initialized by EPnP based on four

keypoints, whose covariance matrices have the smallest traces. Then, it solves the minimization problem using the Levenberg-Marquardt algorithm.

2.4.6 Implementation details

Assuming there are C classes of objects and K keypoints for each class, PVNet takes as input the H $\times$ W $\times$ 3 image, processes it with a fully convolutional architecture, and outputs an H $\times$ W $\times$ (K $\times$ 2 $\times$ C) tensor representing vectors and an H $\times$ W $\times$ (C + 1) tensor representing class probabilities. It uses a pretrained ResNet-18 [cite] as the backbone network with three modifications on it. First, when the feature map of the network has a size of H/8 $\times$ W/8, PVNet does not downsample the feature map anymore by discarding the subsequent pooling layers. Second, to keep the receptive fields unchanged, the subsequent convolutions are replaced with suitable dilated convolutions [47]. Third, the fully connected layers in the original ResNet-18 are replaced with convolution layers. Then, this model repeatedly perform skip connection, convolution and upsampling on the feature map, until its size reaches H $\times$ W, as shown in Figure 2.12(b). By applying a 1 $\times$ 1 convolution on the final feature map, we obtain the unit vectors and class probabilities.

PVNet uses the smooth $\mathbf{l}_1$ loss proposed in Fast R-CNN [10] for learning unit vectors. The corresponding loss function is defined as

$$\mathbf{l}(w) = \sum_{k=1}^{k} \sum_{p \in O} \left(\mathbf{l}_1(\Delta v_k(p; w)|_x) + \mathbf{l}_1(\Delta v_k(p; w)|_y) \right),$$

$$\Delta v_k(p; w) = \widetilde{v}_k(p; w) - v_k(p),$$

where w represents the parameters of PVNet, $\widetilde{v}_k$ is the predicted vector, v_k is the ground truth unit vector, and $\Delta v_k(p; w)|_x$ and $\Delta v_k(p; w)|_y$ represent the two elements of Δv_k, respectively. Note that during testing, there is no need of the predicted vectors to be unit because the subsequent processing uses only the directions of the vectors.

2.5 Dense Pose Object Detector

Dense Pose Object Detector (DPOD) estimates dense multi-class 2D-3D correspondence maps between an input RGB image and available 3D models. Given the correspondences, a 6DoF pose object estimation is computed via PnP and RANSAC. Also a RGB pose refinement of the initial pose estimation is performed using a custom deep learning-based refinement scheme.

2.5.1 Data Preparation

DPOD, like DeepURL, introduces a strategy to create a specific dataset to train the model. First the 3D model of the object is rendered in all possible viewpoints so that it is covered sufficiently. Then for each camera position the object is rendered on a black background and the RGB and depth channels are stored. In addition, a depth mask is created for each position in order to create a box containing the depicted object. For detector training, masks are used to cut out objects from images and then stored as patches for the online augmentation phase. In addition, new in-plane poses are artificially simulated by adding additional rotations. For training the refinement, objects are left as they are.

The custom part of the dataset concerns the generation of images with a correspondence mapping. To to learn a dense 2D-3D correspondences, each 3D object model of the dataset is textured with a correspondence map, shown on Figure 2.19. A correspondence map is a

2-channel image with values ranging from 0 to 255, where the objects are textured using either simple spherical or cylindrical projections. In this way, there is a bijective mapping between the model's vertices and pixels on the correspondence map. This provides an easy-to-read 2D-3D correspondences since given the pixel color, it is instantaneously to estimate the object position on the model surface by selecting the vertex with the same color value. Given the predicted correspondence map, the pose estimation stage estimates the object pose with respect to the camera.

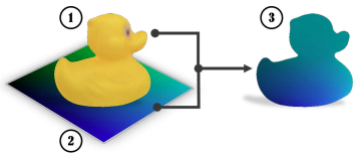


Figure 2.19: Correspondence model: (1) 3D model of interest, (2) apply a 2 channel correspondence texture to the object. The resulting correspondence model (3) is then used to generate ground-truth maps and estimate poses.

2.5.2 Object Detection

The pipeline is divided into two blocks: the correspondence block and the pose block, Figure 2.20.

The correspondence block is an encoder-decoder convolutional neural network with three decoder which regress the mask of the object and dense 2D-3D correspondence map from an RGB image. The encoder model is based on a 12-layer ResNet-like architecture [15]. The decoders upsample the feature up to its original size using a stack of bilinear interpolations followed by convolutional layers. Finally, images are generated with a single channel, where each pixel contains the maximum estimated probability of the class, forming the ID mask, U and V channels of the correspondence image. For the 2D-3D matches is used the color regression problem as discrete color class classification problem, this is useful for much faster convergence and for the superior quality. The network parameters are optimized subject to the composite loss function:

$$L = \alpha L_m + \beta L_u + \gamma L_v \,,$$

where L_m is the mask loss, and L_u and L_v are the losses associated to the U-V channels of the correspondence image. α, β, and γ are weight the factors. Both L_u and L_v losses are defined as multi-class cross-entropy functions, whereas L_m uses a weighted version.

The pose block is responsible for the pose prediction. Given the estimated mask, the 2D locations are estimated, whereas the correspondence map maps each 2D point to a coordinate on an actual 3D model. The 6D pose is then performed using the EPnP method [27]. Also RANSAC [9] is used in conjunction with PnP to make camera pose prediction more robust to possible outliers.

2.5.3 Pose Refinement

Analogous to the object detection stage, for the pose refinement stage is used a ResNet-based architecture. The network has two parallel input branches: one branch receives an RGB image as input, while the other one extracts features from the rendering of the object in the predicted pose from the previous stage. From these two networks the two feature vectors obtained are subtracted and fed into the next ResNet-based block producing the feature vector. The network

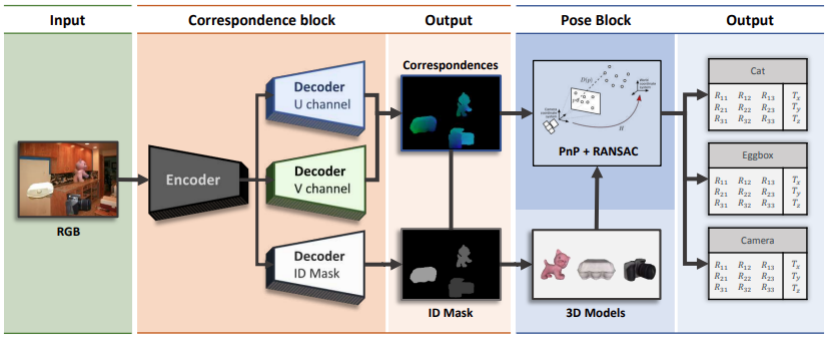


Figure 2.20: Dense Object Detection Pipeline.

ends with three separate output head blocks: one for regressing the rotation, one for regressing the translation in X and Y directions, and one for regressing the translation in Z direction. Each head block is implemented as two fully connected layers.

2.5.4 Limitations

At first glance, this method seemed very callous in that it used an efficient strategy that was different from both DeepURL and PVNet. This would have allowed further comparison of different state-of-the-art strategies that best represented the underwater object pose estimation problem. However after several attempts in building the dataset a complication presented itself. Specifically in the creation of the correspondence map images. This is because the authors of the paper use objects with simple and convex shapes, i.e. the correspondence mapping projection that is done is immediate and each point of the object is mapped with a different pair (U,V). However industrial objects, as in this case, can be very complex and with a low density of vertices representing the 3D model. This implies, for example, taken any square it can be represented with four vertices and two triangles, the basic geometric form used to model 3D objects. The correspondence map associated with the square is simply an interpolation, which makes the pixels less distinguishable from each other, unlike the objects used by the authors with a high density of vertices. Another limitation is the presence of pixels with equal (U,V) values in the same line through the center of reference of the object. Last issue is the presence of a blind part that remains without (U,V) values and that the authors have masked by mapping it to the base of the object.

Chapter 3

Synthetic Dataset Generation

All methods considered and analyzed are data-driven, i.e., the data used to train the various models is a key part of the entire pipeline. In this case the data used are only images representing the object, the background and possible obstacles. General difficulties already known on general image detection include severe occlusions and truncations, variations in lighting and appearance, and cluttered background objects. Adding that we are in an underwater environment where images can be even more difficult to analyze and extract key parts, the dataset used is even more fundamental. This obviously leads to greater difficulties and a more challenging scenario.

Most recent RGB-based detectors can be divided in two groups based on the type of data they use for training: synthetic-based and real-based, where both types of data generation have their pros and cons. When the object is sufficiently covered by real images, it is more convenient to use it. In fact, the close resemblance to the actual object allows for better training, faster convergence and better results. However, training with real images presents some problems, such as: illuminations, poses, scales and occlusions, which could lead the generalization in new environments. Also in some cases, as in this project, the poses of the object, both 2D points and 3D projections in the real world are not available. This could happen because of an inability to get them or because the process is too expensive.

With synthetic rendering of the dataset, a disproportionate number of images can be generated to sufficiently cover the object from all viewpoints. Despite being advantageous in terms of the pose coverage, one as to deal with the domain gap problem severely hindering the performance if no additional data augmentation is applied. Theoretically, one can benefit from the advantages of both types of data by mixing real and synthetic adapted images in the training set. In fact approaches that can be trained with both methods are more desirable.

The models analyzed in this thesis could use both methods, but there is the problem of not having the poses of the real data. From this issue in the following chapter, the problem of generating a synthetic adapted or corrupt dataset from a dataset of false rendered images will be addressed. Specifically, an unpaired image-to-image transformation strategy will be analyzed: Cycle Generative Adversarial Network (CycleGAN) [50].

The next section describes the strategy used to obtain the final synthetic adapted dataset with CycleGAN. In the second section will present the entire pipeline to generate a synthetic adapted dataset. In particular how to generate a first dataset of synthetic images, obtained by a specific algorithm. Then will present the experiments done with CycleGAN to obtain different synthetic adapted datasets and subsequently compare them. Cloncluding with data enhancement strategies to check if this leads to real improvements in the final results.

3.1 Cycle Generative Adversarial Network

From the DeepURL paper the best method used is the generation of rendered image with a tool to create the dataset, then use an image-to-image transform to obtain the synthetic adapted or corrupted images, that they can be used for the approaches taken. A method that can learn to capture special characteristics of one image collection and figure out how these characteristics could be translated into the other image collection, all in the absence of any paired training examples. Converting an image from one representation of a given scene, x, to another, y.

Years of research in computer vision, image processing, computational photography, and graphics have produced powerful translation systems in a supervised scenario, where are available image pairs $\{x_i, y_i\}_{i=1}^{N}$, e.g. [8, 16, 21, 25, 46], as Figure 3.1(a) shown. However, obtaining paired training data can be difficult and expensive. For example, exists only a couple of datasets for tasks like semantic segmentation, and they are relatively small. For the AUV 6DoF object pose estimation is significant an algorithm that can learn to translate between domains without paired input-output examples, Figure 3.1(b).

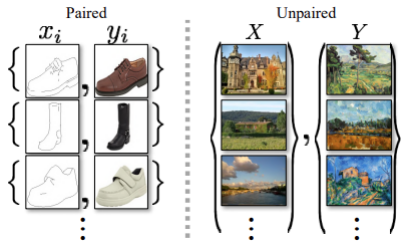


Figure 3.1: (a) Paired training data consists of training examples $\{x_i, y_i\}_{i=1}^{N}$, where the correspondence between x_i and y_i exists. (b) unpaired training data, consisting of a source set $\{x_i\}_{i=1}^{N}(x_i \in X)$ and a target set $\{y_j\}_{j=1}^{M}(y_j \in Y)$, with no information provided as to which x_i matches which y_j.

Where the supervision in the form of paired examples, can be exploit supervision at the level of sets: given one set of images in domain X and a different set in domain Y. Also it assumes there is some underlying relationship between the two domains and learns that relationship. For example, that they are two different renderings of the same underlying scene.

To do that, CycleGAN trains a mapping function $G : X \rightarrow Y$ such that the output $\hat{y} = G(x)$, $x \in X$, is indistinguishable from images $y \in Y$ by an adversary trained to classify $\hat{y}$ apart from y. The optimal mapping function G thereby translates the domain X to a domain $\hat{Y}$ distributed identically to Y. However, a translation does not guarantee that an individual input x and output y are paired up in a meaningful way, because there are infinitely many mappings G that will induce the same distribution over $\hat{y}$. Moreover, in practice, it is difficult to optimize the adversarial objective in isolation. Therefore, CycleGAN exploits the property of "cycle consistent" [49]. Where a translator $G : X \rightarrow Y$ and another translator $F : Y \rightarrow X$, with G and F inverses of each other, and both mappings should be bijections. With this structural assumption training both the mappings G and F simultaneously and adding a cycle consistency loss is encouraged $F(G(x)) \approx x$ and $G(F(y)) \approx y$. Combining this cycle consistency loss with adversarial losses on domains X and Y yields full objective for unpaired image-to-image translation.

3.1.1 Formulation

The goal is to learn mapping functions between a source domain X and a target domain Y given training samples $x_i \in X$, $\forall i = 1 \dots N$ and $y_j \in Y$, $\forall j = 1 \dots M$. The data distribution is denoted as $x \sim p_{data}(x)$ and $y \sim p_{data}(y)$. As illustrated in Figure 3.2(a), the model includes two mappings $G : X \rightarrow Y$ and $F : Y \rightarrow X$ as said before and in addition, two adversarial discriminators D_X and D_Y. Specifically(/in detail) D_X aims to distinguish between images $\{x\}$ and translated images $\{F(y)\}$. In the same way, D_Y aims to discriminate between $\{y\}$ and $\{G(x)\}$. So, the objective contains two types of terms: adversarial losses [11] for matching the distribution of generated images to the data distribution in the target domain; and cycle consistency losses [49] to prevent the learned mappings G and F from contradicting each other.

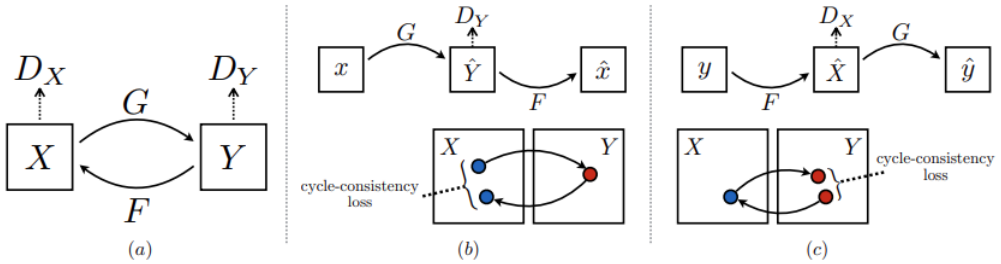


Figure 3.2: (a) The model contains two mapping functions $G : X \rightarrow Y$ and $F : Y \rightarrow X$, and associated adversarial discriminators D_Y and D_X. D_Y encourages G to translate X into outputs indistinguishable from domain Y, and vice versa for D_X and F. To further regularize the mappings, is introduced two cycle consistency losses that capture the intuition that if the model translates from one domain to the other and back again it should arrive at where it started. (b) Forward cycle-consistency loss: $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$. (c) Backward cycle-consistency loss: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$

3.1.2 Adversarial Loss

The adversarial losses are applied to both mapping functions. For the mapping function $G : X \rightarrow Y$ and its discriminator D_Y, the objective is:

$$L_{GAN}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{data}(y)}[\log D_Y(y)] + \mathbb{E}_{x \sim p_{data}(x)}[\log(1 - D_Y(G(x)))],$$

where G tries to generate images $G(x)$ that look similar to images from domain Y, while D_Y aims to distinguish between translated samples $G(x)$ and real samples y. Also, the G aims to minimize this objective against adversary D_Y that tries to maximize it. Same for the mapping function $F : Y \rightarrow X$ and its discriminator D_X, i.e.,

$$\min_G \max_{D_Y} L_{GAN}(G, D_Y, X, Y)\,,$$

$$\min_F \max_{D_X} L_{GAN}(F, D_X, Y, X)\,.$$

3.1.3 Cycle Consistency Loss

In theory, the adversarial training can learn mappings G and F that produce outputs identically distributed as target domains Y and X respectively. However, with large enough capacity,

a network can map the same set of input images to any random permutation of images in the target domain, this imply that adversarial losses alone cannot guarantee that the learned function can map an individual input x_i to a desired output y_i. For this reason, CyceGAN implements the cycle consistency losses, where for each source image x the translation image should bring back to the original one: i.e., $x \to G(x) \to F(G(x)) \approx x$. This is the forward cycle consistency as shown in Figure 3.2(b).

Similarly, as illustrated in Figure 3.2(c), for each image y, G and F should also satisfy the backward cycle consistency: i.e., $y \to F(y) \to G(F(y)) \approx y$. The cycle consistency loss is:

$$L_{cyc}(G, F) = \mathbb{E}_{x \sim p_{data}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{data}(y)}[\|G(F(y)) - y\|_1]$$

As verified by the authors of the paper, replacing the L1 norm does not lead to any improvement.

Figure 3.3 shows an example of cycle consistency loss, made available by the authors

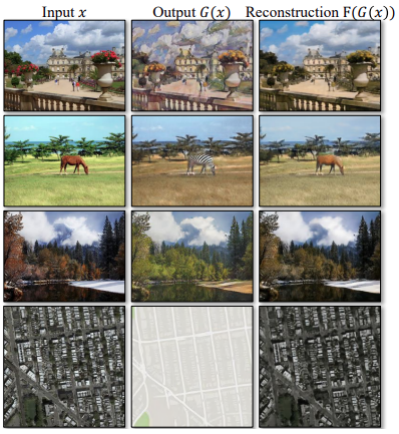


Figure 3.3: The input images x, output images $G(x)$ and the reconstructed images $F(G(x))$ from various experiments. From top to bottom: photo↔Cezanne; horses↔zebras; winter↔summer Yosemite; aerial photos↔Google maps.

3.1.4 Full Objective

The full objective is:

$$L(G, F, D_X, D_Y) = L_{GAN}(G, D_Y, X, Y) + L_{GAN}(F, D_X, Y, X) + \lambda L_{cyc}(G, F),$$

where λ controls the relative importance of the two objectives.

The problem solved by CycleGAN is:

$$G^*, F^* = \arg \min_{G,F} \max_{D_X, D_Y} L(G, F, D_X, D_Y)$$

This model can be viewed as training two "autoencoders" [20], where it learns one autoencoder $F \circ G : X \to X$ jointly with another $G \circ F : Y \to Y$. However, they have a special internal structure: they map an image to itself via an intermediate representation that is a translation of the image into another domain. Obviously, the target distribution for the $X \to X$ autoencoder is the domain Y and for the $Y \to Y$ autoencoder is the domain X.

3.1.5 Implementation details

CycleGAN adopts the architecture from Johnson et al. [21], which according to the authors of the paper, it has shown impressive results for neural style transfer and super-resolution. This network contains three convolutions, several residual blocks [15], two fractionally-strided convolutions with stride 1/2, and one convolution that maps features to RGB. The model uses 6 blocks for 128×128 images, 9 blocks for 256×256 and higher-resolution training images. Also it uses instance normalization [44]. As discriminator networks CycleGAN uses 70x70 PatchGANs [20, 7, 26], which aim to classify whether 70×70 overlapping image patches are real or synthetic. Such a patch-level discriminator architecture has fewer parameters than a full-image discriminator and can work on arbitrarilysized images in a fully convolutional fashion [20].

To stabilize the model training procedure is applied two specific techniques. First, for L_{GAN} equation, replace the negative log likelihood objective by a least-squares loss [32]. This loss is more stable during training and generates higher quality results. In particular, for a GAN loss $L_{GAN}(G, D, X, Y)$, train X to minimize $\mathbb{E}_{x\sim p_{data}(x)}[(D(G(x)) - 1)^2]$ and train D to minimize $\mathbb{E}_{y\sim p_{data}(y)}[(D(y)-1)^2]+\mathbb{E}_{x\sim p_{data}(x)}[D(G(x))^2]$. Second, to reduce model oscillation [11], apply Shrivastava et al.'s strategy [41] and update the discriminators using a history of generated images rather than the ones produced by the latest generators. Introduce an image buffer that stores the 50 previously created images. Authors suggest to set $\lambda = 10$ in the full objective equation, use the Adam solver [23] with a batch size of 1, and keep the same learning rate for a first part of the training and linearly decay the rate to zero over the next epochs.

3.2 Synthetic Dataset

The generation of a synthetic dataset is divided, mainly, into two phases: search or creation of the backgrounds images and generation of the object views.

From now on, the following terminology will be used: *synthetic dataset* as the source dataset of CycleGan, *real dataset* as the target dataset of CycleGAN, and *synthetic adapted or corrupted dataset* as the result of CycleGAN and input for 6DoF object pose estimation approaches.

3.2.1 Underwater Backgrounds

The first step is to search for or create what will be the backgrounds of the synthetic dataset. In this case, existing underwater backgrounds were used, regardless of coloration, brightness and subjects present. This phase was quite time consuming, as each image was downloaded from the web manually. In fact even if the main constraint was the underwater scenario, there were some limitations to be taken into account, such as having some subjects (peaches, wrecks, stones and other) present but not too many, having a certain variety of coloration of the sea (crystal blue, midnight blue, green, brown sand), general variation of brightness (day and night) and blurred backgrounds to simulate the unfocusing of the lens. As you can guess this took time, also because initially CycleGAN was tested with simple underwater images with different subjects: species of fish, mollusks, crustaceans, wrecks and divers. Two datasets depicting subjects in underwater and non-underwater environments were taken. This was done to test if the network was able to recognize and identify the various subjects in the images and change between environments automatically.

In summary at the beginning two datasets of underwater subjects were created, one synthetic and one real, to test CycleGAN and then a dataset of underwater backgrounds for the final synthetic dataset. Figure 3.4 shows some examples of underwater backgrounds.

3.2.2 Synthetic Objects

Synthetic representations of the object are made through a 3D model representing the object. In particular the PLY format [12] is used.

Polygon File Format (PLY) or Stanford Triangle Format is a format for storing graphical objects that are described as a collection of polygons. The main advantage is simple and easy to implement, but it is general enough to create a wide variety of objects. The PLY format describes an object as a collection of vertices, faces and other elements, along with properties such as color and normal direction that can be attached to these elements. A PLY file contains the description of exactly one object. A typical PLY object definition is simply a indexed list of (x, y, z) triples for each vertex and a list of faces that are described by indices that identify the vertices. In addition PLY provides a number of properties to enrich the polygon: color, surface normals, texture coordinates, transparency, range data confidence, and different properties for the front and back of a polygon. Moreover there is a certain degree of customization, in fact one can create a new element type and define the properties associated with this element. This brief description has been devoted to PLY since this polygon format is also used by the three object pose estimation approaches examined.

With the 3D model of the object we are going to create a series of 2D views of the model. First we assume that the object is at the center of a sphere, so that a coordinate system can be defined with respect to the spherical parameters of longitude (or roll), latitude (or pitch), and distance. Starting with this assumption, we place a series of virtual cameras around the 3D model and project the object in the image plane of each virtual camera. In this way the different 2D views of the object are generated.The next step is to arrange the virtual cameras uniformly around the 3D model of the object. To do this we use an icosphere, which is a particular convex polyhedron made of triangles, where each virtual camera is associated with a vertex of the polyhedron.From this method we have two important advantages: uniform vertices around the object and each triangle can be subdivided into three other triangles. The number of recursive subdivisions, called the subdivision level, consequently increases the density of the vertices of the polyhedron and thus increases the number of views obtained from the 3D model of the object. In addition, to make the algorithm more efficient, some non-essential vertices can be avoided, such as those in the southern hemisphere. In addition we can set some parameters of the algorithm: minimum and maximum distance from the object along the optical axis of the virtual cameras, sample the distance interval with a certain sampling step, for computational efficiency set the maximum number of sampled points and the sampling granularity of the polyhedron. Once all the different views of the object are created, they are overlaid on the backgrounds to get a synthetic images. To increase the number of images and variety, we also perform a translation of the synthetic objects with respect to the background.

In this way we obtain the synthetic dataset to be used as the source dataset for CycleGAN and obtain the synthetic adapted datasets that will be used for the object pose estimation problem. Figure 3.4 shows examples of synthetic images rendered from synthetic backgrounds and synthetic object.

3.3 Experiments

For the generation of the synthetic adapted dataset we need two important datasets: the synthetic one, obtained in the previous section and the real one, for this thesis is used an underwater platform. Figure 3.5 shows some images of the real dataset.

CycleGAN has several training and test flags:

- *load_size*: scale images to the size selected.

Figure 3.4: Examples of synthetic images rendered from synthetic backgrounds and synthetic object.

- *crop_size*: after *load_size* a crop is performed to the size selected.
- *preprocess*: scaling and cropping of images at load time, this parameter activates/deactivates *load_size* and *crop_size* parameters. The default option *'resize_and_crop'* resizes the image to the load size and does a random crop size. The option *'crop'* skips the resizing step and only performs random cropping. The option *'scale_width'* resizes the image to have width equal to the crop size while keeping the aspect ratio. The option *'scale_width_and_crop'* first resizes the image to have width equal to the load size and then does random cropping of crop size. The option *'none'* tries to skip all these preprocessing steps. However, if the image size is not a multiple of some number depending on the number of downsamplings of the generator, you will get an error because the size of the output image may be different from the size of the input image. Therefore, *'none'* option still tries to adjust the image size to be a multiple of 4. You might need a bigger adjustment if you change the generator architecture.
- *pool_size*: the size of image buffer that stores previously generated images.
- *epoch, batch_size, lr*: respectively to choose the number of epochs, batch set size, and training policy.

During all training a default learning rate equal to 0.0002 was used, this is because after some tests with was noticed that with a high learning rate created too many variations and at the end could not converge, on the contrary with a too low learning rate could not learn enough and increased the time. As a learning policy was used the default one: linear. The other policies made available did not give a real improvement.

The other parameters are very useful if you want to speed up training by doing a resize, add perturbation by cropping, or focus on a certain part of the image. In the beginning they were widely used to get different variations and speed up the training, but this brought a change on

Figure 3.5: Example of real images.

the final size of the image and the object inside. Obviously this is to the detriment of the final result you want to obtain, because the synthetic adapted dataset must be perfectly consistent with the synthetic one. The reason is quite obvious, since all the various 2D keypoints and 3D projections are done with respect to the synthetic dataset.

The most important experiment concerns the addition of a new value to the full objective loss function, called *shape loss*. The shape loss, summarizing, is the gradient magnitude difference between the synthetic image taken as input by CycleGAN and the synthetic adapted one created in output. Specifically, the two images are converted to gray-scale, because we care about the shape of the object and not the colors. Then a Gaussian blur is applied to smooth the image. We derive the gradients with respect to the two dimensions by applying Sobel's method, improved by the previous blur. We compute the normalized gradient magnitude, L1, with respect to the two previous gradients. At the end the gradient magnitude difference between the two images is computed and normalized with respect to the object mask. In addition, the shape loss is adjusted by a hyper-parameter λ_{sl}.

The first experiment conducted was to simply train CycleGAN with synthetic and real datasets. Then shape loss strategy was applied with different λ_{sl} values: 1.5, 3, 5. Figure 3.6 shows some examples of images obtained from the different synthetic adapted datasets.

As we can see almost all the synthetic adapted images obtained are good, in fact CycleGAN is able to distinguish quite easily the object from the background. In addition, even if we used different colors for the object and different backgrounds, very different from each other, CycleGAN was able to generalize quite well. The first thing we notice is the accentuation, almost excessive, of the $\lambda_{sl} = 1.5$, in fact this could be more a disadvantage than an advantage, because AUV finding in the real world images much less accentuated could have difficulty in recognizing the object. On the contrary, using a $\lambda_{sl} = 5$, the object is blurred and is less recognizable than the other strategies. With $\lambda_{sl} = 3$ there is no more excessive accentuation and there is a certain similarity with $\lambda_{sl} = 5$, but perhaps in general with less blurring. The absence of shape loss could be a valid decision from what we see, in fact in general the object is distinguishable from the background and also the shape of the object remains intact. In general

all four strategies have acceptable results and some very negative ones if we look at the whole datasets obtained. What was found is the validity of $\lambda_{sl} = 3$ or without, so use an intermediate value or don't use it at all.

3.4 Data Enhancement

Following the experiments done on CycleGAN, data augmentation strategies were applied to obtain better results or try to obtain them. In particular we used Histogram Equalization (HE) [33] strategy and then a natural improvement, Contrast Limited Adaptive Histogram Equalization (CLAHE) [51]. We opted for these strategies because they require little computational power and they are fast, since the AUVs must process images online. In fact, consideration was given to using a deep-learning based method: Holistically-Nested Edge Detection [46], but later discarded to allow for online execution.

3.4.1 Histogram Equalization

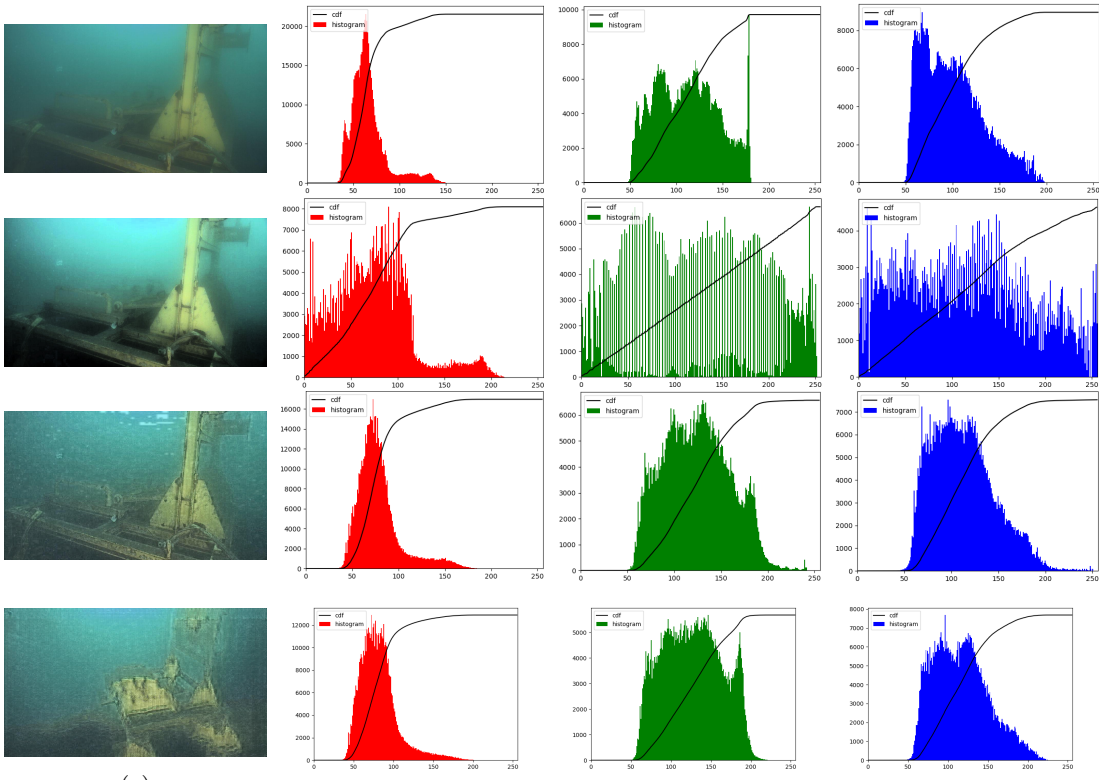
A histogram of an image is the graphical interpretation of the image's pixel intensity values. It can be interpreted as the data structure that stores the frequencies of all the pixel intensity levels in the image. Figure 3.7 first row shows an histogram example.

Histogram Equalization is an image processing technique that adjusts the contrast of an image by using its histogram, especially when the image is represented by a narrow range of intensity values. To enhance the image's contrast, it spreads out the most frequent pixel intensity values or stretches out the intensity range of the image, in this way the intensities can be better distributed on the histogram utilizing the full range of intensities evenly. By accomplishing this, histogram equalization allows the image's areas with lower contrast to gain a higher contrast, while spreading out the highly populated intensity values which use to degrade image contrast. The method is useful in images with backgrounds and foregrounds that are both bright or both dark, which look washed out because they do not have sufficient contrast. Also, the light and dark areas blend together creating a flatter image that lacks highlights and shadows. In particular, the method can lead to better views and to better detail, for example, in photographs that are either over or under-exposed. A key advantage of the method is that it is a fairly straightforward technique adaptive to the input image and an invertible operator. So in theory, if the histogram equalization function is known, then the original histogram can be recovered. The calculation is not computationally intensive. A disadvantage of the method is that it is indiscriminate. It may increase the contrast of background noise, while decreasing the usable signal. Histogram equalization often produces unrealistic effects in photographs; however it is very useful for scientific images like thermal, satellite or x-ray images, often the same class of images to which one would apply *false-color*. Also histogram equalization can produce undesirable effects, like visible image gradient, when applied to images with low color depth. For example, if applied to 8-bit image displayed with 8-bit gray-scale palette it will further reduce color depth of the image, the number of unique shades of gray. Histogram equalization will work the best when applied to images with much higher color depth than palette size, like continuous data or 16-bit gray-scale images.

The above describes histogram equalization on a grayscale image. However it can also be used on RGB images by applying the same method separately to the Red, Green and Blue channels of the RGB color values of the image. However, applying the same method on the Red, Green, and Blue channels of an RGB image may yield dramatic changes in the image's color balance since the relative distributions of the color channels change as a result of applying the algorithm. However, if the image is first converted to another color space, Lab color space,

Figure 3.6: Example of synthetic adapted images obtained.

or HSL/HSV color space in particular, then the algorithm can be applied to the luminance or value channel without resulting in changes to the hue and saturation of the image [33]. There are several histogram equalization methods in 3D space, like [43], but it results in "whitening" where the probability of bright pixels are higher than that of dark ones. Figure 3.7 second row shows an example of histogram equalization on HSV color space, in particular on the Value channel.



(a) (b) (c) (d)

Figure 3.7: First row shows an example of RBG color space histograms. Second row shows HE example on HVS color space. Third row shows CLAHE example on Lab color space. Fourth row shows a CLAHE example on a synthetic adapted image. This compares the real and synthetic adapted dataset and those with data enhancement strategies. (a) image example. (b), (c) and (d) are the RGB channels histograms associated.

3.4.2 Contrast Limited Adaptive Histogram Equalization

Adaptive histogram equalization (AHE) [19, 35] is a computer image processing technique used to improve contrast in images. It differs from ordinary histogram equalization in the respect that the adaptive method computes several histograms, each corresponding to a distinct section of the image, and uses them to redistribute the lightness values of the image. It is therefore suitable for improving the local contrast and enhancing the definitions of edges in each region of an image. Ordinary histogram equalization uses the same transformation derived from the image histogram to transform all pixels. This works well when the distribution of pixel values is similar throughout the image. However, when the image contains regions that are significantly lighter

or darker than most of the image, the contrast in those regions will not be sufficiently enhanced. AHE improves on this by transforming each pixel with a transformation function derived from a neighbourhood region. In its simplest form, each pixel is transformed based on the histogram of a square surrounding the pixel, as in the figure below. The derivation of the transformation functions from the histograms is exactly the same as for ordinary histogram equalization. The transformation function is proportional to the cumulative distribution function (CDF) of pixel values in the neighbourhood. Pixels near the image boundary have to be treated specially, because their neighbourhood would not lie completely within the image. This applies for example to the pixels to the left or above the blue pixel in the figure. This can be solved by extending the image by mirroring pixel lines and columns with respect to the image boundary. Simply copying the pixel lines on the border is not appropriate, as it would lead to a highly peaked neighbourhood histogram.

Adaptive histogram equalization in its straightforward form presented above, both with and without contrast limiting, requires the computation of a different neighbourhood histogram and transformation function for each pixel in the image. This makes the method very expensive computationally. Interpolation allows a significant improvement in efficiency without compromising the quality of the result [35], where the image is partitioned into equally sized rectangular tiles. Figure 3.8 shows the procedure. A histogram, CDF and transformation function is then computed for each of the tiles. The transformation functions are appropriate for the tile center pixels, correspond to the black squares in the left part of the figure. All other pixels are transformed with up to four transformation functions of the tiles with center pixels closest to them, and are assigned interpolated values. Pixels in the bulk of the image: shaded blue, are bilinearly interpolated; pixels close to the boundary, shaded green, are linearly interpolated; and pixels near corners, shaded red, are transformed with the transformation function of the corner tile. The interpolation coefficients reflect the location of pixels between the closest tile center pixels, so that the result is continuous as the pixel approaches a tile center.

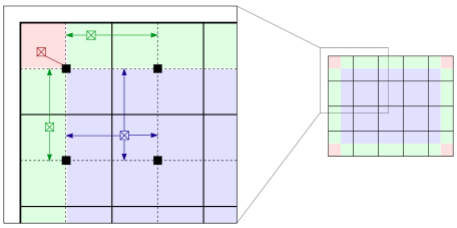


Figure 3.8: Rappresentation of efficient AHE computation by interpolation.

Ordinary AHE tends to overamplify the contrast in near-constant regions of the image, since the histogram in such regions is highly concentrated. As a result, AHE may cause noise to be amplified in near-constant regions. Contrast Limited AHE (CLAHE) is a variant of adaptive histogram equalization in which the contrast amplification is limited, so as to reduce this problem of noise amplification [51]. In CLAHE, the contrast amplification in the vicinity of a given pixel value is given by the slope of the transformation function. This is proportional to the slope of the neighbourhood CDF and therefore to the value of the histogram at that pixel value. CLAHE limits the amplification by clipping the histogram at a predefined value before computing the CDF. This limits the slope of the CDF and therefore of the transformation function. The value at which the histogram is clipped, the so-called clip limit, depends on the normalization of the histogram and thereby on the size of the neighbourhood region. Common values limit the resulting amplification to between 3 and 4. It is advantageous not to discard the part of the histogram that exceeds the clip limit but to redistribute it equally among all

histogram bins. The redistribution will push some bins over the clip limit again (region shaded green in the figure), resulting in an effective clip limit that is larger than the prescribed limit and the exact value of which depends on the image. If this is undesirable, the redistribution procedure can be repeated recursively until the excess is negligible. Figure 3.7 third row shows an example of contrast limited adaptive histogram equalization on Lab color space, in particular on the perceptual Lightness channel. Also the fourth row shows a CLAHE example on a synthetic adapted image. From this example we can immediately see that the original histogram is not balanced. HE, on the contrary, equalizes too much, going to balance the intensity indifferently with respect to the whole image and the result is a brighter image and histograms consistently spread over all values. CLAHE, on the other hand, equalizes different areas of the image while preserving brightness, but creating a pixel effect. From the synthetic adapted image we can see that after applying CLAHE the histograms remain faithful to the real image, which means that the various intensities are not abnormally modified and the various areas of the image are preserved. Moreover, this same shape of the histograms consolidates the effectiveness of CycleGAN, because even if the object is in a different pose and view-point the variety of intensities and colors is maintained.

Chapter 4

Experiments

In this chapter we will test the estimation approaches considered: DeepURL and PVNet, DPOD as explained will not be taken into account for reasons already explained and widely discussed above. In particular we will discuss the experiments carried out and the various results achieved, positive and not. Before this section, we want to dedicate a section to the entire pipeline considered and the working environments used to perform the various tests. Going to illustrate the aspects both positive and negative that led from their use.

4.1 Pipeline

Figure 4.1 shows the entire pipeline considered for the underwater 6-DoF object pose estimation problem. Specifically, at first an algorithm is run to obtain the synthetic dataset from the 3D model of the object and an underwater background. Then CycleGAN creates the corrupted dataset given the synthetic and real dataset. In this step, 4 datasets are created: three with shape loss and one without. At this point we apply CLAHE to the dataset without shape loss, thus obtaining 5 datasets in total. Finally, a 6-DoF object pose estimation approach is applied.

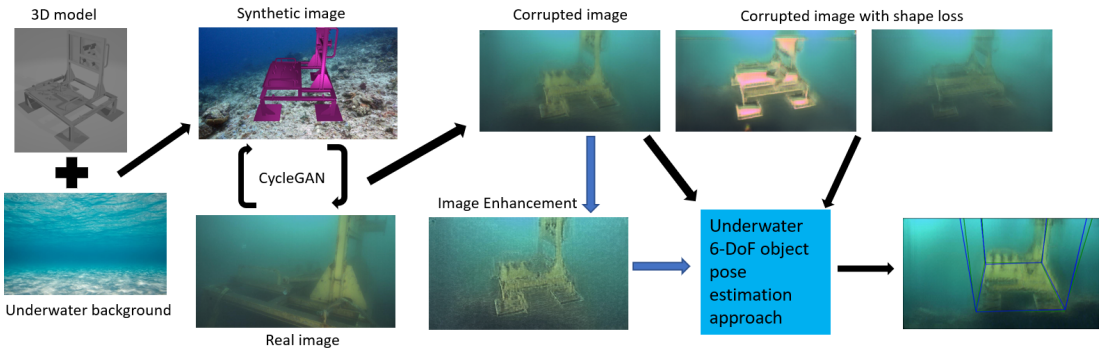


Figure 4.1: The entire pipeline considered

4.2 Training Environment

The cluster of the Department of Information Engineering of the University of Padua was used for the various trainings. In particular, the cluster is composed of graphics cards less performing, such as nvidia Quadro p2000 and other very performing, such as nvidia TitanX and RTX3090. The training were performed through containers specifically created for each considered model, this is because each one had its own specifications to be met. In particular was used SingularityCE, which is a container platform. The advantage of using containers is to separate the hardware and software components. This has allowed to create different environments on a single piece of hardware without them conflicting with each other.

Of course, there were also problems during the creation of these containers, in particular:

- The fact that you need to compile the trained modules on your computer and then upload them to the cluster
- Meet the dependencies of both the models and the target hardware

The first problem is related to the container loading phase, because in my case a slow connection took a lot of time to the other phases of the thesis project. The second problem is related to the specific dependencies of the various models and those of the graphics cards, which in rare cases matched. This in a nutshell means wasting time in finding the right combination in creating a container that satisfies everyone. Also each time the container had to be tested, of course, in the cluster and that implied a new upload. In addition, SingularityCE builds containers from Docker containers or other pre-built environments, which means a download phase.

In summary there is an initial download phase to download the pre-built environment and the specific dependencies of the model used, build of the final container and upload to the cluster, and all this to be repeated over and over again if the dependencies did not satisfy both the training environment and the models. This process involved great waste of time, time taken away from training or implementing other methods not considered in this thesis.

4.3 Approaches

As already explained DPOD will not be considered in this chapter, as it does not satisfy the requirements for our problem. Therefore DeepURL and PVNet will be considered. DeepURL as it will be explained will have a qualitative accuracy, while PVNet will have a quantitative accuracy.

4.3.1 DeepURL

DeepURL provides several parameters to model the training and try to improve the final result:

- *corners_number*: The number of the corners used by the model during the training, the 8 bounding box corners or include the centroid as well and make 9 corners.
- *img_size*: Images will be resized and fed to the network, we can choose two different dimension and take a rectangular images.
- *lr, optimizer and batch_size*: a set of parameters to manage the learning rate (initial, decay and bound) and the training policy.

Different combinations of these parameters were used during training, but without noticing, at least for our specific problem, any real advantage or improvement over the default settings.

This means a learning rate equal to 0.0001, a simple Momentum optimization, batch size at 4, 200 epochs and image resizing while keeping the original aspect ratio.

To evaluate the pose estimation capability of the DeepURL results the authors use a specific metrics. Calculated the mean translation error as the Euclidean distance between the predicted and the ground truth translation and rotation. For individual angle errors in terms of Yaw, Roll, and Pitch, the authors decomposed the rotation matrices into Euler angles and calculated their absolute difference. To evaluate the pose accuracy, it is used standard metrics: 2D reprojection error and ADD metric, the average 3D distance of the model corners. These two metrics will be described in the next section on PVNet.

Before presenting the results obtained, we want to point out an issue that may have negatively affected the training and the final result. The code was developed with Tensorflow version 1, the problem is that this version is no longer maintained and also the dependencies are quite difficult to satisfy. In fact several containers have been tried to satisfy this version and at the same time meet the requirements of the working environment. But there was always an issue that prevented the correct execution of the code. The only working solution found was to use the latest available versions and convert the code to Tensorflow version 2 using a script and a patient search in the documentation. Hence, the training was performed on partially modified code and this may have affected its correct functioning.

Figure 4.2 shows some results obtained with DeepURL. The results obtained are not satisfactory, in fact we see errors with respect to both translation and rotation. In fact here we find the limitation of using a network that estimates the bounding box and from this derives the 3D projections. But actually the 2D bounding box predicts the position quite well, this means that it is the next step of 3D pose estimation that is not performed adequately. A possible consideration is that the 2D bounding box is predicted by trying to fill the target object as best as possible. On the contrary in the 3D bounding box estimation many points can very often be in the surroundings of the object, this means that the network has to predict corners that are not pixels object and consequently has no real measure on the distance. Moreover the model should take into account all possible positions and viewpoints.

Since the results were discouraging already with a synthetic dataset, where the AUV is distinguishable from the background and all the issues of the real dataset are not present, no further training was performed with other datasets. In fact the subject is different, since we were still in an initial phase of evaluation of the various approaches. Keeping in mind, however, different combinations of the given parameters were tried to get acceptable results. Somewhat surprising are the results obtained by the DeepURL authors, as they show acceptable results. It can be assumed that it is for the dataset used, in fact they use an AUV in a pool and in an ocean. As we can guess the pool is a very favorable environment even if underwater, as there are no other objects and the water is crystal blue, so there is no particular problem related to this scenario. It is also quite favorable, because the AUV is well distinguishable from the background and allows a remarkable contrast between the two. Also in the case of the ocean, a very clean and recognizable water was used with respect to the target.

However, DeepURL remains an inspiring approach regarding the use and generation of synthetic adapted datasets to train models and counteract the insufficiency of real datasets for underwater environments

4.3.2 PVNet

Before presenting the experiments done, we want to discuss the training environment, which especially in this case did not make our life easy. In fact, the only way to train PVNet was to use the nvidia Quadro P2000 graphics card, i.e., the low performance one, because the high performance ones required dependencies that PVNet could not support. Also, the same

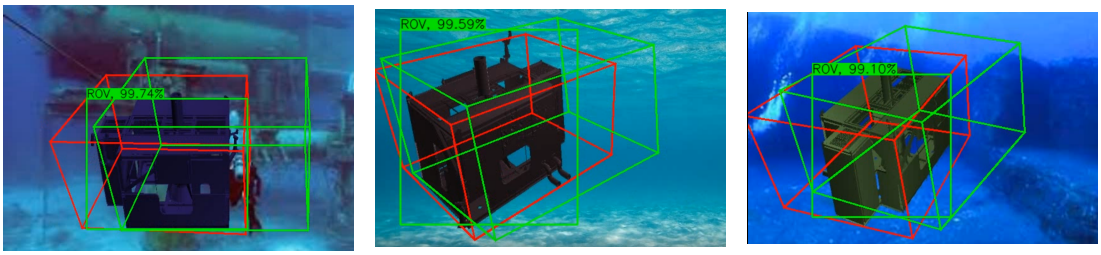


Figure 4.2: DeepURL results with synthetic dataset.

container created by the authors did not work in any graphics card.

PVNet provides several parameters to model the training and try to improve the final result:

- *lr, optimizer and batch_size*: a set of parameters to manage the learning rate (initial and decay) and the training policy.
- *cropresize_rate*: indicates the ratio to keep with respect to the original image, if less than one a resize and crop is performed; values [0-1].
- *rotate*: rotation to be applied to the object (rate, min and max)

Again, the default configuration was retained, as other combinations gave no real improvement. Learning rate equal to 0.001, Adam optimization, batch size at 2, 200 epochs and original image size.

Two metrics are used to evaluate this approach:

2D Projection metric [4]: this metric computes the mean distance between the projections of 3D model points given the estimated pose and the ground-truth pose. A pose is considered as correct if the distance is less than 5 pixels.

ADD metric [17]: compute the mean distance between two transformed model points using the estimated pose and the ground-truth pose. When the distance is less than 10% of the model's diameter, it is claimed that the estimated pose is correct. For symmetric objects, it is used the ADD-S metric [45], where the mean distance is computed based on the closest point distance.

Figure 4.4 shows some results obtained with PVNet. The first line was included as a comparison with DeepURL and we can see that in this approach the estimation is much better, almost perfect. Obviously this case has no real validity, but still it can serve as an example if the target is well recognizable and distinguishable from the background. In the second row, containing images without shape loss strategy, we can see in general that the box is predicted well even in those cases where the target is not very distinguishable. However, in cases where there is an accentuation of the object, some rotation is added. In the remaining three rows we have the datasets with the shape loss strategy. Also in these three cases the estimates are acceptable, but with a small translation compared to no shape loss. In all four tests there is additional rotation when the object is near the edge, this could be caused by incorrect pixel voting in determining the corners. Taking into account the results obtained with respect to the metrics and the visible bounding boxes, the best strategies seem to be: no shape loss strategy or with $\lambda_{sl} = 3$ and then use an intermediate value. A valid alternative remains shape loss with $\lambda_{sl} = 1.5$, since that excessive accentuation could lead to unexpected improvements.

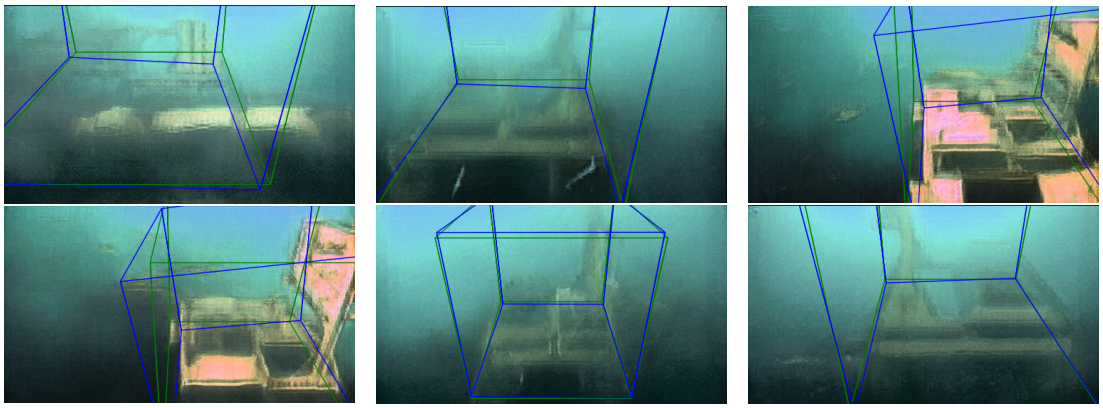


Figure 4.3: PVNet results with CLAHE dataset

Figure 4.3 shows some results obtained apply CLAHE to the dataset without shape loss strategy. The presence of CLAHE does not deviate much from the previous results, however it remains an excellent strategy to consider for adding perturbation to the training.

Table 4.1 shows the performance obtained. It can be seen immediately the big difference between the two metrics of accuracy, in fact for all tests conducted we have ADD very high and 2D Projection very low. This might seem an error in the correctness of the tests, but also in some cases reported by the authors of PVNet we find the same results. We hypothesize that this is related to the difference in the two metrics. In fact, 2D Projection metric measures the difference between bounding boxes within 5 pixels, while ADD metric measures the difference within 10% of the object diameter. Consequently, even a small variation of the estimated bounding box by 5 pixels from ground truth greatly decreases the accuracy of the 2D Projection metric. Instead ADD metric remains high probably due to the large diameter of the object and the 10% of it is large enough to cover the error between the two bounding boxes.

PVNet	Proj-2D	ADD
$\lambda_{sl} = 0$	5.2	99.7
$\lambda_{sl} = 0 + CLAHE$	4.1	99.5
$\lambda_{sl} = 1.5$	5.9	99.8
$\lambda_{sl} = 3$	4.4	99.9
$\lambda_{sl} = 5$	2.7	99.8

Table 4.1: Performance of PVNet with different datasets

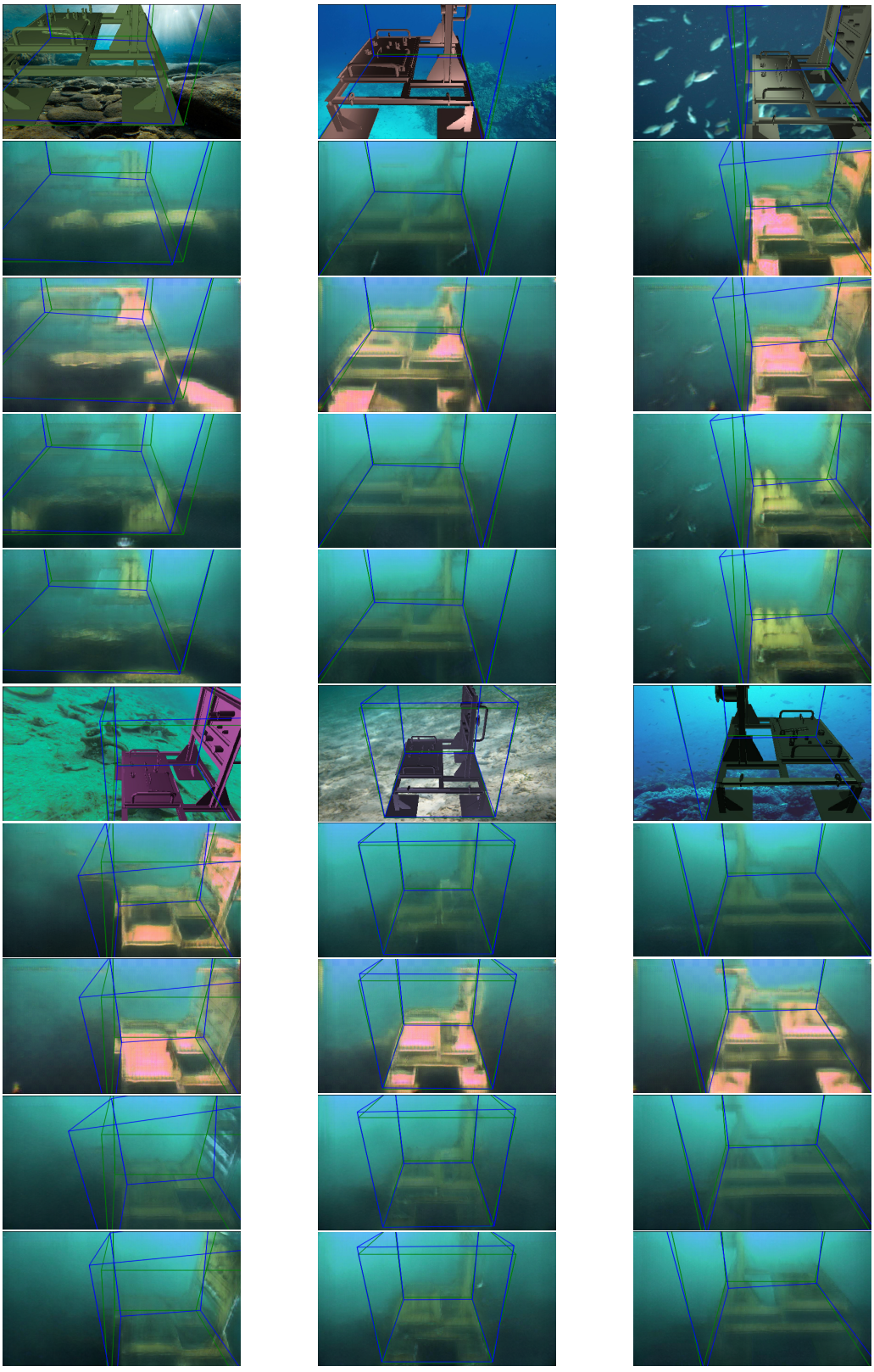


Figure 4.4: PVNet results with synthetic and synthetic adapted datasets

Chapter 5

Conclusion

Data-driven 6-DoF object pose estimation methods require accurate ground truth data, but in underwater scenarios, it is very difficult to get both real images and ground truth data. Therefore it is very useful to be able to create synthetic images and then transform them into photo realistic images to be used to effectively train the data-driven methods. In particular in this thesis an unpaired image-to-image transformation strategy is used to obtain a synthetic adapted images dataset, called CycleGAN. In this way is "easily" to create a specific dataset for this particular problem and as needed. CycleGAN tries to capture special features of one image collection and figuring out how these characteristics could be translated into the other image collection, all in the absence of any paired training examples. This speeds up the creation of the synthetic dataset, at least in part since there is no need to create one-to-one pairs, which by the way is very difficult in our specific case. The results obtained are more than acceptable, even those obtained with the introduction of shape loss, as they could lead to randomness and unexpected improvements during the training. Obviously also thanks to the synthetic dataset, in particular to the careful selection of the various backgrounds, but taking into account also in this case to insert the perturbation.

After creating the adapted synthetic datasets, we tested the considered 6-DoF object pose estimation approaches. Specifically, approaches with different strategies were considered in order to evaluate which one is the most suitable for our problem. DPOD, as it has been explained, is not able to handle complex objects or objects that do not have a certain level of detail of the 3D model.

DeepURL estimates the 3D pose of an AUV from a single image as projected 2D image keypoints, that represent the 8 corners of the 3D model of the AUV. Then the 3D pose in the camera coordinates is determined using RANSAC-based PnP. In other words, the detected keypoints serve as an intermediate representation for pose estimation. Such two-stage approaches achieve state-of-the-art performance, thanks to robust detection of keypoints. However, these methods have difficulty in tackling occluded and truncated objects, since part of their keypoints are invisible. But, as it turns out, DeepURL is not suitable for us, as having to directly predict the corners of the bounding box has limitations.

PVNet predicts vectors that represent directions from each pixel of the object towards the keypoints. These directions then vote for the keypoint locations based on the RANSAC algorithm. This voting scheme is motivated from a property of rigid objects that once it sees some local parts, it can infer the relative directions to other parts. In particular is oriented to create a flexible representation for localizing occluded or truncated keypoints. Another important feature of this representation is that it provides uncertainties of keypoint locations that can be further leveraged by the PnP solver. It also tries to better predict under several occlu-

sions, variations in lighting and appearance, and cluttered background objects. In addition, PVNet has proven reliable in low and bad light conditions, in case of clipping and difficulty in distinguishing the object from the background.This corner voting strategy was inspiring, as it brought very positive results, pointing out that it is not enough to apply methods already known and extensively studied in some cases, but new ideas are needed.

5.1 Future Work

We discuss several improvements that could lead to further enhance the performance of the underwater object pose estimation pipeline. Summarizing:

- Choice of backgrounds images used for the synthetic dataset
- Improvements on the corrupted datasets generation related to CycleGAN
- Using other data enhancement strategies
- Improvements in object pose estimation approaches

The first improvement concerns the generation of synthetic images. Instead of using random backgrounds images or downloaded from the web, and then slide the models of the object of interest, it would be possible to extract images from a simulator developed with a rendering engine, taking care to bring the virtual camera very close to the relevant portion of the objects. In fact, the synthetic images extracted from the simulator have the advantage of not having a background with objects in the foreground, which could confuse CycleGAN during the corruption process. Moreover, using the simulator it is possible to adjust the type of water and turbidity levels in order to generate synthetic images even more similar to the real ones.

The second addresses the SHAPE_LOSS custom model. This model, in fact, uses a too aggressive technique to extract gradients from both the synthetic and corrupted images. Indeed, a smoother gradient extraction technique than Sobel could allow an increase in the details. In other words, it would be possible to obtain synthetic adapted images that are more faithful to the originals in terms of shape and detail, which would probably lead to an increase in accuracy in the testing phase of object pose estimation problem.

The third improvement is to apply other data enhancement strategies to the synthetic adapted datasets, in particular also testing state-of-the-art deep-learning based approaches.

The fourth improvement involves the last part of the pipeline. Specifically using DeepURL with the most recent version of YOLO, seeing if this can bring about any marked improvements. Adapt DPOP for complex objects as in our case, as the strategy behind it seemed very valid. Use approaches that use PVNet itself with an additional layer of pose refinement. But also analyze and test additional strategies that have not been considered so far.

All these aspects could allow to further increase the level of accuracy achieved by the underwater 6-DoF object pose estimation pipeline.

Bibliography

[1] A. Ansar and K. Daniilidis. “Linear pose estimation from points or lines”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 25.5 (2003), pp. 578–589.

[2] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. “Surf: Speeded up robust features”. In: *In ECCV*. 2006, pp. 404–417.

[3] Eric Brachmann et al. “Learning 6D Object Pose Estimation Using 3D Object Coordinates.” In: *ECCV (2)*. Vol. 8690. Lecture Notes in Computer Science. Springer, 2014, pp. 536–551.

[4] Eric Brachmann et al. “Uncertainty-Driven 6D Pose Estimation of Objects and Scenes From a Single RGB Image”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016.

[5] J. H. M. Byne and J. A. D. W. Anderson. “A CAD-based computer vision system.” In: *Image Vis. Comput.* 16.8 (1998), pp. 533–539.

[6] “CAD-Based Vision: Object Recognition in Cluttered Range Images Using Recognition Strategies”. In: *CVGIP: Image Understanding* 58.1 (1993), pp. 33–48.

[7] Li Chuan and Wand Michael. “Precomputed real-time texture synthesis with markovian generative adversarial networks”. In: *Computer Vision – ECCV 2016*. 2016.

[8] David Eigen and Rob Fergus. “Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-scale Convolutional Architecture.” In: *ICCV*. IEEE Computer Society, 2015, pp. 2650–2658.

[9] Martin A. Fischler and Robert C. Bolles. “Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography”. In: *Commun. ACM* 24.6 (June 1981), pp. 381–395.

[10] Ross B. Girshick. “Fast R-CNN.” In: *ICCV*. IEEE Computer Society, 2015, pp. 1440–1448.

[11] Ian Goodfellow et al. “Generative Adversarial Nets”. In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani et al. Vol. 27. Curran Associates, Inc., 2014.

[12] Turk Greg. “The PLY Polygon File Format.” In: 1994.

[13] Christian Hansen, Thomas C. Henderson, and Roderic A. Grupen. “CAD-based 3-D object recognition.” In: *SMC*. IEEE, 1989, pp. 168–172.

[14] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Second. Cambridge University Press, ISBN: 0521540518, 2004.

[15] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778.

[16] Aaron Hertzmann et al. “Image analogies”. In: *SIGGRAPH ’01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM Press, 2001, pp. 327–340.

[17] Stefan Hinterstoisser et al. “Model Based Training, Detection and Pose Estimation of Texture-Less 3D Objects in Heavily Cluttered Scenes”. In: *Computer Vision – ACCV 2012*. Ed. by Kyoung Mu Lee et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 548–562.

[18] Stefan Hinterstoisser et al. “Model Based Training, Detection and Pose Estimation of Texture-Less 3D Objects in Heavily Cluttered Scenes.” In: *ACCV (1)*. Vol. 7724. Lecture Notes in Computer Science. Springer, 2012, pp. 548–562.

[19] Robert Hummel. “Image enhancement by histogram transformation”. In: *Computer Graphics and Image Processing* 6.2 (1977), pp. 184–195.

[20] Phillip Isola et al. “Image-to-Image Translation with Conditional Adversarial Networks”. In: *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*. 2017.

[21] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. “Perceptual Losses for Real-Time Style Transfer and Super-Resolution”. In: *Computer Vision – ECCV 2016*. Ed. by Bastian Leibe et al. 2016.

[22] Bharat Joshi et al. *DeepURL: Deep Pose Estimation Framework for Underwater Relative Localization.* 2020.

[23] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization." In: *ICLR.* 2015.

[24] Alex Krizhevsky. "Learning Multiple Layers of Features from Tiny Images". In: *Tech Report* (2009), pp. 32–33.

[25] Pierre-Yves Laffont et al. "Transient attributes for high-level understanding and editing of outdoor scenes." In: *ACM Trans. Graph.* 33.4 (2014), 149:1–149:11.

[26] C. Ledig et al. "Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network". In: *Computer Vision and Pattern Recognition.* 2017.

[27] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. "EPnP: An Accurate O(n) Solution to the PnP Problem". In: *International Journal of Computer Vision* (2008).

[28] Tsung-Yi Lin et al. "Feature Pyramid Networks for Object Detection." In: *CVPR.* IEEE Computer Society, 2017, pp. 936–944.

[29] Tsung-Yi Lin et al. "Microsoft COCO: Common Objects in Context". In: *Computer Vision – ECCV 2014.* Ed. by David Fleet et al. Cham: Springer International Publishing, 2014, pp. 740–755.

[30] David G. Lowe. "Object Recognition from Local Scale-Invariant Features". In: *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2.* ICCV '99. IEEE Computer Society, 1999, pp. 1150–1157.

[31] C.-P. Lu, G.D. Hager, and E. Mjolsness. "Fast and globally convergent pose estimation from video images". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.6 (2000), pp. 610–622.

[32] Xudong Mao et al. "Least Squares Generative Adversarial Networks". In: Oct. 2017, pp. 2813–2821. DOI: 10.1109/ICCV.2017.304.

[33] S.K. Naik and C.A. Murthy. "Hue-preserving color image enhancement without gamut problem". In: *IEEE Transactions on Image Processing* 12.12 (2003), pp. 1591–1598.

[34] Sida Peng et al. “PVNet: Pixel-wise Voting Network for 6DoF Pose Estimation”. In: *CVPR*. 2019.

[35] Stephen M. Pizer et al. “Adaptive Histogram Equalization and Its Variations”. In: *Comput. Vision Graph. Image Process.* 39.3 (Sept. 1987), pp. 355–368.

[36] Mahdi Rad and Vincent Lepetit. “BB8: A Scalable, Accurate, Robust to Partial Occlusion Method for Predicting the 3D Poses of Challenging Objects without Using Depth.” In: *ICCV*. IEEE Computer Society, 2017, pp. 3848–3856.

[37] Joseph Redmon and Ali Farhadi. “YOLOv3: An Incremental Improvement”. In: (2018).

[38] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2016.

[39] Rasmus Rothe, Matthieu Guillaumin, and Luc Van Gool. “Non-maximum Suppression for Object Detection by Passing Messages Between Windows.” In: *ACCV*. Vol. 9003. 2014, pp. 290–306.

[40] Olga Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: (2014). arXiv: 1607.08022.

[41] Ashish Shrivastava et al. “Learning from Simulated and Unsupervised Images through Adversarial Training.” In: *CVPR*. IEEE Computer Society, 2017, pp. 2242–2251.

[42] Bugra Tekin, Sudipta N. Sinha, and Pascal Fua. “Real-Time Seamless Single Shot 6D Object Pose Prediction.” In: *CVPR*. IEEE Computer Society, 2018, pp. 292–301.

[43] P.E. Trahanias and A.N. Venetsanopoulos. “Color image enhancement through 3-D histogram equalization”. In: *Proceedings., 11th IAPR International Conference on Pattern Recognition. Vol. III. Conference C: Image, Speech and Signal Analysis,* 1992, pp. 545–548.

[44] Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. “Instance Normalization: The Missing Ingredient for Fast Stylization”. In: *CoRR* abs/1607.08022 (2016). arXiv: 1607.08022.

[45] Yu Xiang et al. “PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes.” In: *Robotics: Science and Systems*. 2018.

[46] Saining Xie and Zhuowen Tu. “Holistically-Nested Edge Detection.” In: *ICCV*. IEEE Computer Society, 2015, pp. 1395–1403.

[47] Fisher Yu and Vladlen Koltun. “Multi-Scale Context Aggregation by Dilated Convolutions.” In: *ICLR*. 2016.

[48] Sergey Zakharov, Ivan Shugurov, and Slobodan Ilic. “DPOD: 6D Pose Object Detector and Refiner”. In: *International Conference on Computer Vision (ICCV)*. Oct. 2019.

[49] Tinghui Zhou et al. “Learning Dense Correspondence via 3D-guided Cycle Consistency”. In: *CoRR* abs/1604.05383 (2016).

[50] Jun-Yan Zhu et al. “Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks”. In: *Computer Vision (ICCV), 2017 IEEE International Conference on*. 2017.

[51] Karel J. Zuiderveld. “Contrast Limited Adaptive Histogram Equalization.” In: *Graphics Gems*. Ed. by Paul S. Heckbert. Elsevier, 1994, pp. 474–485.

Acknowledgements

The realization of this thesis represents the end of a journey that began several years ago, but at the same time the beginning of a completely new path, to which several people have contributed.
First of all, I would like to thank Riccardo Bastianello and Luca Signorato, two colleagues and true companions of adventure who have accompanied me during my journey at the University of Padua, always ready to lighten the most difficult moments of university life. In particular, I would like to thank Riccardo for all the precious discussions I had during the development of my thesis, working on different sections of the same project.
Secondly, I would like to thank my thesis supervisor, Alberto Pretto, for having been always present and available during the development of my thesis and for having believed in my abilities. His advice and His way of doing things made the work very pleasant and stimulating, turning it into a valuable source of discovery and personal enrichment. I would also like to thank my supervisor for having contributed to the generation of the synthetic data used to create the various tests.
Finally, a special thanks to my family. To my brother, for all the hours spent together to share some common passions and the special bond we have, a bond that I hope will last forever. To my parents, who supported me and encouraged me to do my best, continuing to believe in me until the end.

Padua, 28 February 2022

Nicola Valzan