Vision-based Navigation Solution for Autonomous Underwater Vehicles

Tiago Martim Gomes Alves
tiago.g.alves@tecnico.ulisboa.pt

Instituto Superior Técnico, Universidade de Lisboa, Portugal
October 2021

Abstract

Navigation is fully autonomous when a vehicle can plan its path and execute it without human intervention. This research aims at testing the impact of introducing an AI-based approach for visual navigation in underwater environments. To achieve this, several challenges have to be overtaken. First, an annotated dataset with pairs of input images and segmentation grounds truths is essential for training a state-of-the-art AI model. Second, choosing a model adequate for image segmentation and training it. Finally, evaluate if this methodology improves the accuracy of visual navigation and scene reconstruction algorithms, such as online and offline SLAM. This approach achieved state-of-the-art results on the segmentation task, with 93% *pixel accuracy and* 85% *IoU. Using segmentation masks also improves the performance of offline and online SLAM algorithms.*

1. Introduction

Autonomous underwater vehicles (AUVs) can revolutionize deep sea exploration, by changing the way data is acquired for further mapping and monitoring. Space exploration is another field that could benefit from AUVs, since one of the goals of several international organizations is to explore parts of our solar system potentially capable of hosting life, as is the case of ocean worlds, such as Enceladus, Europa or Titan. These ocean worlds could have conditions similar to those on the deep parts of our oceans, thus developing vehicles capable of operating autonomously in our seas could be a first step.

Developing fully autonomous systems is crucial to achieving the above goals. Tim Shank, a researcher from WHOI, explains his vision in a conference [1], where a series of vehicles equipped with sensors are capable of communicating with each other and with local base stations about positioning, sensing and samples they are collecting. Russel Smith, an engineer from NASA's Jet Propulsion Lab (JPL), spoke in the same conference about the importance of vehicles capable of gathering visual information and use it for localization and mapping, enabling fully autonomous navigation.

Developing a visual based navigation solution for underwater vehicles operating in deep ocean comes with several challenges. Since the feature detection of this solution is based in deep learning, which requires annotated datasets for training, finding or developing a good dataset was the first task. The second challenge would be training a neural network suited for segmentation, in order tohave a robust feature detection tool. At last, assess if the proposed methodology helps improve the performance of SLAM algorithms.

To tackle the first problem, a dataset with pairs of input images and labeled segmentation ground truth needs to be created.

To address the segmentation problem, we want to use state of the art technology, robust enough to endure the difficult navigation conditions of a deep underwater environment. One of the possibilities is to resort to neural networks, such as Fully Convolutional Network.

To assess if the developed network contributes to SLAM algorithms, the outputs were tested, coupled with the input images, on both offline and online SLAM algorithms. Offline SLAM is used in post-exploration situations to reconstruct and map the visited sites. Online SLAM is applied for real-time autonomous navigation. Thus, evaluating on both tools is crucial in the scope of deep sea - and future ocean planets - exploration.

The main contributions of this work are:

- Development of an annotated dataset of deep underwater environments rich in hydrothermal vents and sea bed footage;
- Training and testing a state of the art AI-based semantic segmentation algorithm with the created dataset;

[1] https://www.youtube.com/watch?v=BGD_oyPGT6w&ab_channel=oceanexplorergov

- Evaluate the contribution of introducing AI in feature detection to improve the performance of navigation and reconstruction algorithms, such as SLAM.

2. Background

Semantic segmentation is a very important technique in the field of computer vision, being used for problems like object detection and classification, development of self driving vehicles or virtual reality. In the context of this project, semantic segmentation is the approach that makes the most sense, since we want to split images in regions containing obstacles, like sea bed, and background, such as areas containing only water.

2.1. Fully Convolutional Networks

Fully convolutional networks [4] are neural network architectures widely used for semantic segmentation tasks.

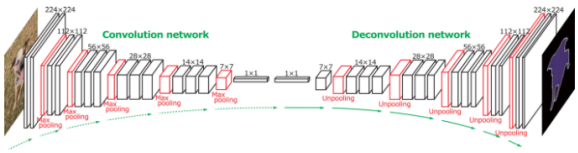


Figure 1: Fully convolutional network. Source: [8]

Today, a good amount of the best performing state-of-the-art semantic segmentation methods relly on fully convolutional networks. The FCNs are able to yield dense pixel-wise predictions on arbitrary sized inputs and can be trained end-to-end. Another advantage is that these networks are built on top of CNNs which are able to produce rich feature representations. These CNNs can be pretrained models on general purpose datasets, further reducing the time required to train the network.

These networks are denominated 'fully convolutional' because they do not have fully connected layers like CNNs and most artificial neural network architectures. Fully connected layers were removed for two reasons: first, they entail a single input size, not allowing the model to perform on arbitrary sized inputs; second, they only yield a single feature vector for the entire image. To replace the fully connected layer, fully convolutional layers were introduced, which are able to deal with the arbitrary size input.

2.1.1 From classifier to segmentation

The work of Long et. al [4] takes networks used for image classification problems, like GoogleNet, AlexNet or VGG16. In all the previous networks, the classifier is discarded and all the fully connected layers are converted to convolutions with 1×1 kernels. Convolutional layers with size 1 kernels are similar to fully connected layers but are capable of dealing with arbitrary size inputs. In a normal CNN, with fully connected layers, at the end of the classifier, the output would be a single predicted label. But since the fully connected layers are replaced by convolutional layers, what the network outputs is a feature map where features are assigned to a certain class.

The output of the fully convolutional layers has lower resolution than the input image, due to the pooling layers used to downsample. The method used by [4] uses transposed convolutions, or deconvolutions, to upsample the predictions. By doing this, the FCN's output has the same resolution as the original image.

From training and validating this architecture on the PASCAL dataset, Long et al [4] found the FCN with VGG16 was the one that yielded the best results.

2.1.2 Convolution

At the output of the convolutional layer new images, called feature maps, are generated. The feature map is a version of the original image where unique features are emphasised. This layer also has an important difference compared to layers in other neural networks, since weights are not present in connections between neurons, instead there are filters (*kernels*) responsible for the convolution where weights are stored.

Since the convolution operation occurs in a 2D plane, it is better explained graphically. Therefore, figure 2 is a good representation of the operation. First, we perform the dot product between the kernel and a submatrix of the input matrix, the value is stored in a new entrance of a new matrix (feature map). Then we do the same for the next submatrix of the input image. It should be noted that kernels usually come in the form of 3×3 or 5×5 matrices and the weights are not selected by the network designer, instead they are usually randomly initialized and iteratively updated through the training process.

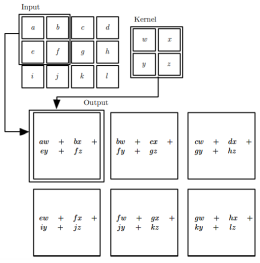


Figure 2: Graphic representation of a convolution. The kernel slides through the input image, performing dot products between the kernel values and a submatrix of the input matrix. Source: [1]

2.1.3 Deconvolution

Deconvolution, or transposed convolution as can be seen in some literature, is a mathematical operation that reverses the effect of a convolution. The input feature map is upsampled to a desired output feature map. In the case of a FCN, it is usually upsampled until the dimensions of the input image are reached.

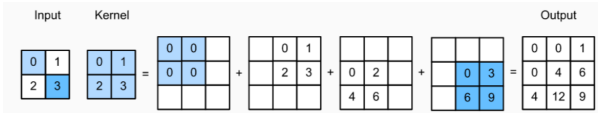


Figure 3: Deconvolution process schematic. Deconvolutions are used to upsample the input feature map to a desired output feature map using learnable parameters. Source: [6]

Figure 3 portrays the deconvolution process. An input matrix and a kernel both of size 2×2 are considered for the example. The first step is to take every element of the kernel and multiply for every element of the input image, creating the 4 matrices displayed in the image. Next, all the matrices are added giving place to a 3×3 matrix, which has larger dimensions than the input.

3. Simultaneous Localization and Mapping (SLAM)

Simultaneous Localization and Mapping (SLAM) is a computational problem of reconstructing and updating a vehicle's trajectory by mapping while, simultaneously, knowing the location of the vehicle in the same map. The SLAM problem is widely regarded as one of the major problems in the field of fully autonomous vehicles [10] and although the field has seen major progresses, it is still a great challenge, particularly when a vehicle is intended to navigate through dynamic large scale environments, as is the case of deep ocean or space exploration. SLAM algorithms are commonly used in self-driving cars, autonomous underwater vehicles, unmanned air vehicles and robotics.

3.1. Definition of the SLAM Problem

The SLAM problem can be described as a robot, capable of sensing its surroundings, roaming a previously unknown environment with known starting coordinates. The way the robot moves is not certain, which increases the difficulty of determining its global coordinates. SLAM algorithms should deal with the task of mapping the environment while simultaneously knowing the robot's position in this map.

The trajectory followed by the robot is given by

$$X_T = \{x_0, x_1, x_2, ..., x_T\} \tag{1}$$

where each x_i is a known location of the robot and T denotes a terminal time period.

Odometry is a robotics technique where data from motion sensors is used to estimate changes in position over time. With u_t being the odometry that describes the changes in position between two intervals, the sequence

$$U_T = \{u_0, u_1, u_2, ..., u_T\} \tag{2}$$

can describe a series of steps in the robot's motion. This data can be retrieved from controls given to the vehicle's motor, sensors or cameras. If the measurements of U_T were noise free, then it would be sufficient to retrieve the trajectory X_T.

Let m denote the ground-truth map of the environment. The measurements from the robot's sensor provides information that relates the map m with the robot's position x_t. If the robot measures exactly once in every time instant, the set of measurements is described by

$$Z_T = \{z_0, z_1, z_2, ..., z_T\}. \tag{3}$$

Now the SLAM problem can be resumed to generating a map m of the location and recover the robot's trajectory X_T from the sensor's measurements Z_T and the odometry data U_T. Current literature distinguishes between two major SLAM problems: online SLAM and offline SLAM.

Online SLAM focuses on recovering the current robot position x_t and it can be mathematically described as

$$p(x_t, m|Z_T, U_T). \tag{4}$$

Offline SLAM, sometimes called full SLAM, seeks to determine the full path of the vehicle, as well as the map, after observation. Mathematically the problem is defined as

$$p(X_T, m|Z_T, U_T).$$

Both SLAM problems are very important. Offline SLAM can be used for generating a map and recover textures and structures after exploring a certain area with a robot, while online SLAM is used for real time map and position estimation. In terms of algorithms, the two differ in one major point: for online SLAM, algorithms must be able to process one data item at a time for real time accuracy; for offline SLAM, since the algorithm is ran post-exploration with the acquired data, the methods usually work in batches, being able to process all the data at the same time.

Furthermore, to tackle both problems, two more models should be considered. One that sets a relation between odometry measurements and the robot's positions

$$p(x_t|x_{t-1}, u_t), \tag{5}$$

and another that relates the sensor's measurments of the surroundings with the map and the robot's location

$$p(z_t|x_t, m). \tag{6}$$

In the last we have the problem of not knowing neither the robot's position x_t nor the environment's map m. The Bayes rule allows us to transform the mathematics of equation 6 into an equation where we can retrieve probability distributions over the map and position.

$$p(z_t|x_t, m) = \frac{p(x_t, m|z_t) \cdot p(z_t)}{p(x_t, m)}. \quad (7)$$

3.2. ORB-SLAM

ORB-SLAM [7] is a real time feature-based monocular visual SLAM algorithm. Visual SLAM aims at reconstructing the camera's trajectory while mapping the environment.

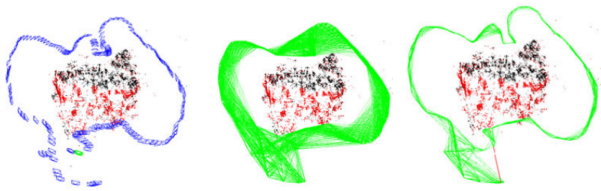
ORB-SLAM uses bundle adjustment (BA) to perform its optimization tasks. BA is a non-linear least-squares optimization problem that amounts to refining a series of initial camera and trajectory parameters to estimate the set of values that describes the locations of observed features more accurately.

In terms of feature detection and choice, ORB-SLAM uses the ORB feature detector, which allows for fast computation while maintaining accurate results. One of the core ideas of ORB-SLAM is that the same features are used for very different tasks, such as mapping, tracking, place recognition and loop closing, contributing to overall efficiency of the algorithm.

This SLAM algorithm exploits the use of three threads running simultaneously in parallel: one for tracking, another for local mapping and the last for loop closing. The tracking is responsible for estimating the camera position and infer when a new frame is a key-frame. The local mapping thread takes the new keyframes and performs bundle adjustment to attain a reconstruction of the area surrounding the camera. Also, local mapping searches for correspondence between features in the new keyframe among the other connected keyframes. The last thread, loop closing, searches for loops everytime a new keyframe is inserted. To ensure consistency, pose graph optimization is performed. This thread is also responsible for finding and fusing duplicated points to avoid redundancy.

ORB-SLAM uses covisibility graphs to represent keyframes. In these graphs each keyframe is a node and edges connecting different keyframes mean there is shared observation of map points. Each edge has weight that is equal to number of shared map points.

Whenever a loop is detected and closed, it shall be corrected using pose graph optimization. As complexity increases with the number of keyframes, the covisibility graph can become very dense, therefore the pose graph optimization is performed on a smaller *Essential* graph that keeps all the keyframes but uses less edges, which keeps accurate results at lower complexity levels. A graphical representation of a scene reconstruction with the observed map points, keyframes and the generated covisibility and essential graphs can be seen in figure 4.



(a) (b) (c)

Figure 4: **(a)** Keyframes (blue), map points (red and black), current camera (green); **(b)** covisibility graph, with all the edges connecting keyframes; **(c)** Essential graph, only the edges with high weight are present and a loop closing edge in red. Source:[7]

4. Implementation

4.1. Dataset

The process of gathering visual data for the dataset consisted of three steps: first, going through all the videos and selecting periods with images suited for the task; second, sampling these video periods; last, from all the images collected from the sampling, clear the ones that were not good candidates, like images with too much noise or full of bubbles and sand.

The methodology followed to create the segmentation ground truths, after having all the inputs gathered, consists of three stages: applying contrast enhancement; produce an estimate segmentation mask, using edge detection tools; applying the necessary corrections, to yield ground truths as accurate as possible (Fig. 5).

4.2. Contrast Enhancement

When light travels in turbid mediums, like water or atmosphere, it suffers from absorption and scattering, which result in degraded images, with low contrast and poor colour quality. This scattering is not homogeneous across the scene, since this effect depends on the distance of scene points to the camera. Furthermore, in the conditions that this footage was produced, where natural lighting is non-existent, properly illuminating the entire captured scene is impossible, resulting in dark areas that can compromise the data analysis.

The Single Scattering Atmospheric Model (SSAM), models a hazy image, **I(x)**, as

$$I(x) = J(x)t(x) + A(1 - t(x)), \quad (8)$$

where **x** is a coordinate vector of a given pixel, **J** is the hazeless image, **A** is the atmospheric light and **t** is the light that

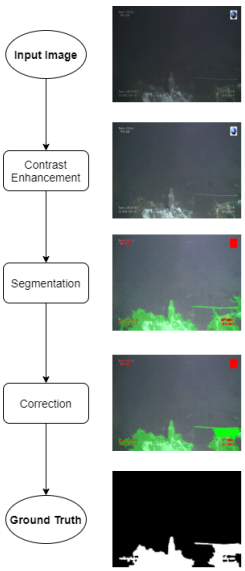


Figure 5: Dataset creation flow chart.

reaches the camera without scattering. If we calculate $\mathbf{t}$ and $\mathbf{A}$, given the input $\mathbf{I}$ we can recover $\mathbf{J}$ and have the enhanced haze free image.

The first thing to be done is calculate the *dark channel*. The dark channel is formed by the pixels with the lowest intensity in one of the three RGB channels, in a patch of given size. The dark channel, $\mathbf{J^{dark}}$, is given by

$$J^{dark}(x) = \min_{y \in \Omega}(\min_{c \in r,g,b} J^c(y)) \tag{9}$$

where $\mathbf{\Omega}$ is a patch centered at $\mathbf{x}$.

Considering what we first introduced about the DCP, we can say that if $\mathbf{J}$ is a haze-free image, with recovered contrasts, then

$$J^{Dark} \rightarrow 0. \tag{10}$$

The atmospheric light $\mathbf{A}$, or the scene light, is calculated by choosing the 0.1% brightest pixels of the dark channel. This corresponds to the area with the most haze and lower contrasts, as the scattering is highly dependent on the distance travelled by light. The same pixels are retrieved from the original image $\mathbf{I}$. Then, the mean of this group of pixels is calculated for each RGB channel.

Another necessary step for the improvement of image quality is to determine the transmission map, from the hazy image equation (eq. 8), which can be written as

$$t(x) = 1 - w \min_{y \in \Omega(x)} \left(\min_{c} \frac{I^c(y)}{A^c} \right) \tag{11}$$

where $\mathbf{A^c}$ is the atmospheric light in each color channel, considering pixel intensities ranging from 0 to 1. Our understanding of human vision considers that haze allows the perception of distance and depth, an occurrence denominated aerial perspective.

If the transmission map was to be used just like it is yielded from equation 11, we would obtain an output image with undesired artifacts (halos or pixelated blocks) around objects present in the scene. In order to prevent this, the transmission map must be refined (filtered). The approach proposed by Tunai et al. [5] consists of using a guided filter. According to which, a filtered image $\mathbf{q}$ can be recovered from a guidance image $\mathbf{I}$ and an input image $\mathbf{p}$ using

$$q_i = a_k I_i + b_k, \forall i \in w_k \tag{12}$$

with $\mathbf{i}$ being the pixel's index and and k the index of a local square window $\mathbf{w}$ of radius $\mathbf{r}$. The values of $\mathbf{a_k}$ and $\mathbf{b_k}$ can be determined using:

$$a_k = \frac{\frac{1}{|w|} \sum_{i \in w_k} I_i p_i - \mu_k \overline{p}_k}{\sigma_k^2 + \epsilon} \tag{13}$$

$$b_k = \overline{p}_k - a_k \mu_k \tag{14}$$

where $\mu_\mathbf{k}$ and $\sigma_\mathbf{k}^\mathbf{2}$ are the mean and the covariance of $\mathbf{I}$ in $\mathbf{w_k}$, $|\mathbf{w}|$ is the number of pixels in $\mathbf{w_k}$ and $\mathbf{p_k}$ is the mean of $\mathbf{p}$ in $\mathbf{w_k}$. The equations above were provided by He et. al at [2].

After determining the scene light $\mathbf{A}$ and the filtered transmission map $\mathbf{t(x)}$ we have the necessary information to recover the the enhanced image $\mathbf{J(x)}$, from equation 8 comes

$$J(x) = \frac{I(x) - A}{\max(t(x), t_0)} + A \tag{15}$$

where $\mathbf{t_0}$ is a constant introduced to limit the value of the denominator. For certain denominator values, the recovered image can be prone to noise in the most hazy regions, so it is good practice to introduce a constant ($\mathbf{t0}$) and limit the denominator value.

The parameter values chosen for this implementation were $w = 0.5$ (Eq. 11), $t_0 = 0.6$ (eq. 15), using patches of dimensions 15×15.

4.3. Operator Selection

Annotating an entire dataset by hand would be extremely time consuming and prone to human error. Thus, developing a methodology to help create the dataset is advantageous in what concerns efficiency and accuracy. Analysing the dataset, a characteristic becomes evident: most of the images are very rich in edges and corners, due to the presence of shellfish and algae on the seabed. Therefore, edge detection operators were chosen to extract those features from the images. Besides the edge detection approach, the k-means clustering algorithm for segmentation was also tested.

4.3.1 Sobel

Since the method that yielded the best results, even compared with K-means, was the Sobel approach, this document only covers this one. For this method, there are filters (kernel) that are convoluted accross the image. These kernels are an estimate of the derivative of pixel intensity, estimating the direction of the highest variation of pixel intensity (from bright to dark), which provides insights on how the the pixel intensities changes in each given point, and allows intensity gradient estimation. In regions where intensity variation is significant, we can say that it is a separation between objects and, thus, we are in the presence of an edge. All these filters perform 2-D spatial gradient measurement, which means that a gradient is estimated across the horizontal (G_x) and vertical (G_y) dimensions and are then used to calculate the absolute gradient magnitude

$$G = \sqrt{G_x^2 + G_y^2}. \tag{16}$$

In theory, Sobel operator typically uses 3×3 kernels, however for improved segmentation results 5×5 kernels are also used. These kernels usually take the following values

$$G_x = \begin{bmatrix} +2 & +2 & +4 & +2 & +2 \\ +1 & +1 & +2 & +1 & +1 \\ 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & -2 & -1 & -1 \\ -2 & -2 & -4 & -2 & -2 \end{bmatrix} \tag{17}$$

$$G_y = \begin{bmatrix} +2 & +1 & 0 & -1 & -2 \\ +2 & +1 & 0 & -1 & -2 \\ +4 & +2 & 0 & -2 & -4 \\ +2 & +1 & 0 & -1 & -2 \\ +2 & +1 & 0 & -1 & -2 \end{bmatrix}. \tag{18}$$

To prevent the detection of edges that appear in the image due to noise, the images must be filtered. The median filter from OpenCV [2] was used with kernels of size $ksize = 5$. After filtering, the edge maps are retrieved by convoluting the kernels across the image. The edges alone are not a segmentation mask, because pixels are not labeled individually, instead only the pixels that are part of edges are labeled. However, if we perform morphological operations, such as closing and dilation [3], we are able to create a mask that, if overlayed in the original image, covers the entire object instead of just the edges. To remove unwanted areas, morphological transformations such as opening and eroding can be implemented. Furthermore, a function was developed to erase areas that are bellow a certain size threshold, since most objects that would be segmented are large and unwanted areas of dust and other particles are small in comparison.

[2] https://docs.opencv.org/3.4/d4/d86/group__imgproc__filter.html#ga564869aa33e58769b4469101aac458f9

[3] https://docs.opencv.org/master/d9/d61/tutorial_py_morphological_ops.html

4.4. Fully Convolutional Network

The chosen framework for the implementation of the deep learning solution for semantic segmentation was Pytorch, developed by Facebook AI Research Lab and is seeing its usage increase at good pace. Comparing to other frameworks, such as Tensorflow or Keras, Pytorch is very flexible, offers good debugging capabilities and runs faster, meaning shorter training duration.

4.4.1 Data Loading and Preprocessing

The first step of the data loading process consists of cropping the images. The implemented model only accepts input images whose dimensions are multiples of 16. Since we have 1440 pixels of width, which already is a multiple of 16, the only dimension that should be cropped is the height. The original image's height is 1080 and it was converted to 1056.

Also, images should be converted from RGB format to BGR and pixel intensities for the three channels are normalized, so that they stay in the range $[-1; 1]$ instead of $[0; 255]$.

4.4.2 Architecture

The research by Long et al. [4] evaluates the performance of three different fully convolutional networks (FCNs), the FCN32s, FCN16s and FCN8s. The one that achieved the best results in [4] was the FCN8s, since it combines the output of deeper layers with outputs of shallower ones, preventing the loss of spatial information that is crucial to capture all the details. This operation of combining results of different layers will be further explained ahead. The implementation of the FCN8s can be split in two stages. First, a transfer learning approach was used where a pretrained VGG16 network imported from Pytorch is integrated for the convolutional part of the network. The traditional VGG16 consists of 5 convolutional and pooling layers followed by fully connected layers. However, since we want the network to accept inputs of arbitrary size, the fully connected layers are replaced with another two convolutional layers,

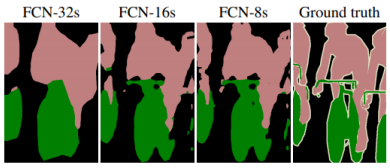


Figure 6: Comparison between the results achieved by the FCN32s, FCN16s and FCN8s. Source:[4]

performing the convolutions with kernels of size 1×1. Second, the transpose convolution (deconvolution) layers are added at the output of the last convolutional layer, to upsample results to the size of the input image, providing an image where each pixel is assigned to a class.

This architecture yields good results because it combines results from different layers, making local pixel predictions while respecting global structure. The outputs of the 3^{rd}, 4^{th} and 5^{th} convolutional layers from VGG16 are combined with results from the transpose convolution layers(figure 7). When we go deeper in the network, we lose spatial information, due to all the convolution and pooling layers. So, if we combine the output of deeper layers with the output of shallower ones, we are adding location information, it is expected that the quality of the results increases, due to increased detail. This combination is an elementwise sum. The output of the 5^{th} convolutional layer is upsampled by a factor of 2 and summed with the output of the 4^{th} convoluional layer. The output of this operation is again upsampled by a factor of 2 and summed with the output of the 3^{rd} convolutional layer, which is later upsampled by a factor of 8 to yield the final result.

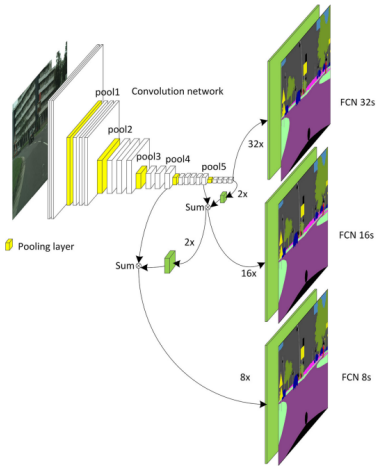


Figure 7: Graphic representation of the combination of results from different layers. This process is used to improve the quality of results, by adding feature maps from deeper layers. Source: [3]

The chosen activation function used at the end of each layer was the rectified linear unit (ReLU), as it is the most commonly used for CNNs and FCNs. The selected loss function was binary cross entropy (BCE) and the optimization process is done through stochastic gradient descent, using the RMSProp optimization algorithm.

4.4.3 Network Parameters

Regarding the parameters chosen for the training phase, for computational capacity reasons the batch size was set to 1, as images have a very large number of pixels and consume a great ammount of memory, even training on GPUs using the CUDA toobox from Nvidia. Learning rate was set to $1 \cdot 10{-}5$, weight decay $1 \cdot 10^{-5}$, step size 50 and momentum 0. The training dataset was iterated across 130 epochs.

To arrive to this configuration of parameters several tests were executed, by training the model with different parameters. The values chosen for the first training session were the ones provided by the paper of Long et. al [4] and different sets of parameters were experimented from there.

4.4.4 Dataset Management

The dataset consists of 1198 images, of which 798 were used for training and 150 for validation throughout each training epoch. After training, the model was tested with a dataset of 150 images, with examples that were not used in the training phase. Furthermore, since the model was trained several times with different parameters, a dataset with 100 unique examples from a different underwater site was stored to evaluate the model performance after the configuration of training parameters was settled. The goal of the last is to assess how robust the model is. No data augmentation was performed, since the work of [4] states that performing such task for does not perceptibly improve network performance and increases the time required to train.

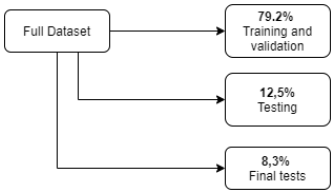


Figure 8: Schematic of the dataset management. A dataset with images from a different underwater site was stored to evaluate the model after all the network hyperparameters were settled.

4.4.5 Evaluation Metrics

To evaluate the network's performance during training and testing sessions the selected performance metrics were mean pixel accuracy and intersections over union, as these are the ones considered most relevant in the work of Long et al. [4].

The mean pixel accuracy (MPA) can be defined as

$$MPA = \frac{CP}{TP} \tag{19}$$

where CP is the number of correct pixel classifications and TP is the total number of predictions. The number of correct pixels is a sum of the number of true positives and true negatives.

The intersections over union (IoU), sometimes also called Jaccard index, quantifies the overlap between the ground truth mask and our prediction mask. In other words, it measures the number of pixels common between the target and the prediction and divides by the total number of pixels present on both masks. [9]

$$IoU = \frac{target \cap prediction}{target \cup prediction} \tag{20}$$

The intersections over union are determined for each class separately. Then the average of the IoUs is determined to provide a global metric of the segmentation predictions.

5. Results

5.1. Fully Convolutional Network

This section's objective is to evaluate the results attained by the Fully Convolutional Network. The model was trained a total of 8 times, with different configurations of hyperparameters and correction of the dataset between each training phase. After determining the configuration of parameters that achieves the most accurate results, the model is tested on a set of images, collected from a different site. The results of these last test are also exposed in this section.

5.1.1 Training an FCN with Different Parameters

These interim tests are the set of tests that aim at reaching the optimal configuration of network training parameters for this problem. All the tests were performed on the same dataset, as seen on figure 9. After each epoch of training, a validation set is ran through the network to evaluate performance evolution throughout the epochs of training. For each training phase, the validation dataset is always a random subset of 150 images from the overall training set.

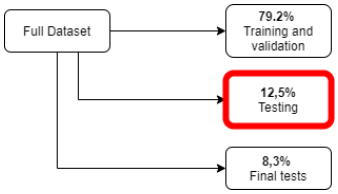
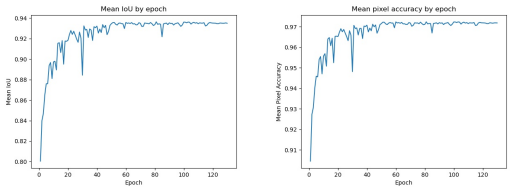


Figure 9: Dataset management chart. The red box denotes the part of the dataset used for testing throughout the tests exposed in this section. The training and validation sets are the same throughout all tests.

Throughout this testing phase, the network parameters, like the number of epochs, learning rate or weight decay have been tuned to evaluate the impact of changing these values on the model's performance. In total, 7 tests were conducted and there was no expressive changes in performance across them. The chosen configuration of network parameters for the final test, to be performed with a different dataset, was the one used on test number 7, with 130 epochs, batch size 1, learning rate 5×10^{-5}, weight decay 1×10^{-5}, step size 50, gamma 0.5 and momentum 0. This configuration was selected since the accuracy and IoU evolution performed well on the chosen metrics and presented good stability at the end of the training phase, indicating the algorithm had converged towards an optimal solution (figure 10). The achieved results on the validation and test sets are present in tables 1 and 2.



(a) Mean IoU evolution for 7^{th} test.

(b) Mean pixel accuracy evolution for 7^{th} test.

Figure 10: Evolution of the performance metrics, IoU and pixel accuracy, across epochs for the 7^{th} test.

Pixel Accuracy	Mean IU
96.70%	92.53%

Table 1: Results achieved on the validation set.

Pixel Accuracy	Mean IU
97.9%	94.61%

Table 2: Results achieved on the test set.

5.1.2 Final Test

For the last test, the model used was the one reached when training for test number 7. The dataset used for this test was significantly different from the one used for training, since the images were collected in a different underwater cite, with different conditions. The goal was to evaluate the model's capacity to generalize when the data being used is different.

Looking at table 3, the network achieved very accurate results, with nearly 93% mean pixel accuracy and 85.21% IoU. The Sobel-based method achieved 91.5% mean pixel

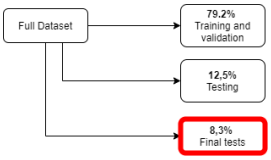


Figure 11: Dataset management chart. The red box denotes the part of the dataset used for the test explained in this section.

Pixel Accuracy	Mean IU
92.80%	85.21%

Table 3: Results achieved on the test set highlighted in figure 11

accuracy over the same samples. Although the FCN's accuracy decreased, comparing to previous tests, we can affirm that it is still a very accurate result.

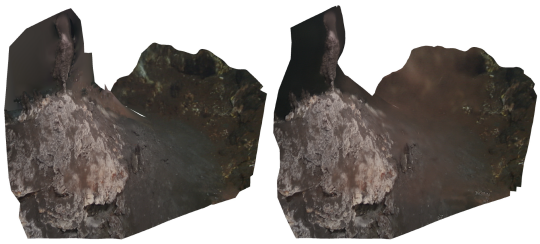
Another interesting metric, the FCN can yield segmentation predictions in around 0.01 seconds, which translates to 100 Hz. In detail, this approach is able to generate 100 segmentation masks per second. The video footage used to create the dataset for this thesis had a frame rate of 25 frames per second. Meaning that our approach is 4 times faster than the frame rate of the camera.

5.2. Simultaneous Localization and Mapping

To evaluate the impact of the proposed methodology on real time navigation algorithms we have ran offline SLAM and online SLAM tools on sequences of images with no masks and with masks generated by the trained FCN model.

5.2.1 Offline SLAM

Agisoft Metashape Standard (Version 1.7.2)(Software) is a photogrammetry software, used for 3-D reconstruction of scenes. To evaluate the impact of semantic segmentation on offline SLAM, the program ran a sequence of 46 images. Figure 12 displays the software's output with and without using masks. Although there is no huge difference between the two cases, we can see that when no masks are used (12 **(b)**) the software builds the texture for a dust cloud that goes in front of the camera. The dust cloud has a brownish color. Furthermore, in the case without masks, the program also puts alpha-numerical characters, that appear on the corners of the input images, on the scene. When (12 **(a)**) masks are used, neither the dust cloud, nor the characters are built on the reconstructed scene.

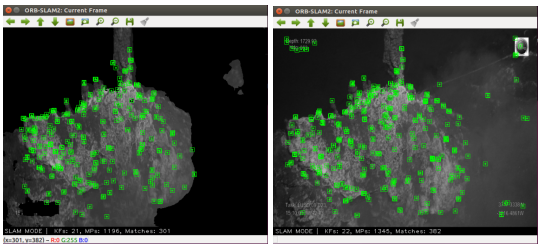


(a) (b)

Figure 12: **(a)** Output of Agisoft Metashape after processing a sequence of images with masks; **(b)**Output of Agisoft Metashape after processing a sequence of images without masks, in this case we can see the algorithm reconstructs a dust cloud.

5.2.2 Online SLAM: ORB-SLAM

On the scope of online SLAM, ORB-SLAM was ran on a sequence of 1000 images. To evaluate the impact of using masks, the images and their respective masks were overlayed so that only the foreground was visible and everything else was 'deleted', by making every non-foreground pixel black, see figure 13 **(a)**.



(a) (b)

Figure 13: Frame with features extracted by ORB-SLAM **(a)** with mask; **(b)** without mask.

On a more quantitative analysis, let us consider the trajectory's jerk, which is a physics concept that amounts to the rate at which an object's acceleration changes over time. It is denoted by j and its units are m/s^3. It is usually expressed as a vector, being the first derivative of acceleration. It can also be expressed as a third derivative of position or a second derivative of velocity.

$$j(t) = \frac{d^3r(t)}{dt^3} = \frac{d^2v(t)}{dt^2} = \frac{da(t)}{dt} \tag{21}$$

Since ORB-SLAM outputs, for every keyframe, the coordinates of the camera's optical center, in respect to the world's coordinate system, jerk will be a 3-dimensional vector at

any given point in time.

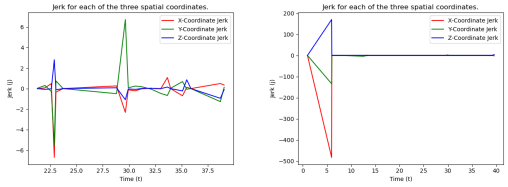
$$j(t) = \left(\frac{d^3x(t)}{dt^3}; \frac{d^3y(t)}{dt^3}; \frac{d^3z(t)}{dt^3} \right) \quad (22)$$

Besides the coordinates output, ORB-SLAM also provides the timestamps of those keyframes. This allows us to compute these derivatives through finite differences.

$$r'''(t_0) = \frac{-\frac{1}{2}r(t_{-2}) + r(t_{-1}) - r(t_{+1}) + \frac{1}{2}r(t_{+2})}{h_t^3} \quad (23)$$

where h_t is the time difference between each finite difference interval.

In robotics, a low jerk means the trajectory is smooth. In terms of the robot's control system, a smoother trajectory allows for less complex and more robust performance. Figure 14 and table 4 establish a comparison between the trajectory's jerk using masks and without masks. For a more accurate analysis, it should be noted that when running with segmentation masks, ORB-SLAM takes more time to initialize, since it detects less features per frame. However, it is able to find the same amount of keyframes (21) in a much smaller period. The results on table 4 were computed by calculating the average of all jerks' norms. When running with masks, the jerk is approximately 12 times better than without masks.



(a) (b)

Figure 14: Trajectory's jerk **(a)** with mask; **(b)** without mask.

With Mask	Without Mask
0.021	0.258

Table 4: Average of jerk norms.

6. Conclusions

The purpose of this work was to study the impact of introducing AI-based semantic segmentation tools on the performance of SLAM algorithms, for posterior integration in autonomous underwater vehicles. An annotated dataset, with 1200 pairs of images and masks, was created to train a neural network to perform semantic segmentation and the results were tested on both online and offline SLAM algorithms.

To develop the dataset, images were enhanced to improve feature detection. Also, a tool was developed to yield an initial estimate of segmentation mask, using Sobel-based edge detection, which were hand corrected afterwards to create the segmentation ground truth for each image.

The chosen deep learning model for semantic segmentation was a Fully Convolutional Network, that was trained several times with different hyperparameters. With the network that yielded the best results, at around 93% accuracy, a sequence of frames was ran through the model to have masks to test on the SLAM algorithms.

The results were tested on offline SLAM, using Agisoft Metashape Standard (Version 1.7.2)(Software) and on online SLAM, with ORB-SLAM. On both cases the proposed methodology improved the quality of the results.

Future work should include the development of a larger and more diverse dataset and using it to train a Fully Convolutional Network, so the model can cope with conditions different from those of the proposed dataset. Also, for further validation, this methodology should be tested using an underwater autonomous vehicle on the context of real time ocean exploration.

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org. 2

[2] Kaiming He, Jian Sun, and Xiaoou Tang. Guided image filtering. *IEEE transactions on pattern analysis and machine intelligence*, 35(6), Dec. 2013. 5

[3] Xiaolong Liu, Zhidong Deng, and Yuhan Yang. Recent progress in semantic image segmentation. In *Artificial Intelligence Review*, volume 52, pages 1089–1106. Springer International Publishing, 2019. 7

[4] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 2, 6, 7

[5] Tunai Porto Marques, Alexandra Branzan Albu, and Maia Hoeberechts. Enhancement of low lighting underwater images using dark channel prior and fast guided filters. *Lecture Notes in Computer Science*, 11188, 2018. 5

[6] Divyanshu Mishra. Transposed convolution demystified. https://towardsdatascience.com/transposed%2convolution%2demystified%284ca81b4baba, 2020. 3

[7] Raúl Mur-Artal, J. M. M. Montiel, and Juan D. Tardós. Orb-slam: A versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015. 4

[8] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *Computer Vision (ICCV), 2015 IEEE International Conference on*, 2015. 2

[9] Md Atiqur Rahman and Yang Wang. Optimizing intersection-over-union in deep neural networks for image segmentation. In George Bebis, Richard Boyle, Bahram Parvin, Darko Koracin, Fatih Porikli, Sandra Skaff, Alireza Entezari, Jianyuan Min, Daisuke Iwai, Amela Sadagic, Carlos Scheidegger, and Tobias Isenberg, editors, *Advances in Visual Computing*, pages 234–244, Cham, 2016. Springer International Publishing. 8

[10] Sebastian Thrun. *Simultaneous Localization and Mapping*, pages 13–41. Springer Berlin Heidelberg, 2008. 3