Nuno André da Silva Marques

AUTONOMOUS SURFACE VEHICLE NAVIGATION AND MAPPING IN RIVER BASINS

Dissertação no âmbito do Mestrado em Engenharia Eletrotécnica e de Computadores, do ramo de especialização em Robótica, Controlo e Inteligência Artificial, orientada pelo Professor Doutor Lino José Forte Marques e Mestre Hugo Magalhães e apresentada ao Departamento de Engenharia Eletrotécnica e de Computadores da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Setembro de 2022

· U C ·

FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Autonomous Surface Vehicle Navigation and Mapping in River Basins

Nuno André da Silva Marques

Coimbra, September 2022

· U C ·

Autonomous Surface Vehicle Navigation and Mapping in River Basins

Supervisor:

Prof. Doutor Lino José Forte Marques

Co-Supervisor:

Mestre Hugo Magalhães

Jury:

Prof. Doutor Rui Paulo Pinto Rocha

Prof. Doutor Manuel Marques Crisóstomo

Prof. Doutor Lino José Forte Marques

Dissertation submitted in partial fulfillment for the degree of Master of Science in Electrical and Computer Engineering.

Coimbra, September 2022

Acknowledgements

First, I would like to express my gratitude to my Supervisor, Professor Lino Marques, for all the support and suggestions given during the development of this dissertation and for the opportunity to work in a great lab.

To my Co-Supervisor, Hugo Magalhães, a big thank you for all the help and work done that made this dissertation possible and a little less intimidating.

To all my colleagues, João, Rui and Sedat in the Field Robotics Lab, who made the adaptation really easy and helped me when I had any problems.

To all my friends, especially André, JP and Joca who were always by my side during this big journey that was this course.

Lastly, to my family, in particular to my parents and grandmother, who gave me all the support and motivation that I needed during these five years.

Resumo

Os veículos autónomos de superfície (VAS) são veículos que operam na superfície da água sem tripulação ou operador. Podem ser utilizados em tarefas de vigilância, monitorização ambiental e cartografia do solo marinho/fluvial. Estas tarefas levam tempo e precisam de ser automatizadas, pelo que o VAS deve navegar de forma autónoma. Para tal, podem ser utilizados muitos tipos de sensores (LiDARs, Sonares, Pares de Câmaras Estéreo), mas um dos mais versáteis é a câmara monocular.

Esta dissertação analisa o estado da arte de métodos de navegação e implementa uma estratégia para mapear e evitar obstáculos. Para o conseguir, é implementada uma rede de segmentação da água que atinge um dice score de $0,9896$ num dataset misto com imagens do rio Mondego e imagens do dataset ROSEBUD ([18]). Foi desenvolvido um algoritmo de reconstrução 3D que estima as distâncias com precisão e elimina o problema da escala, utilizando a distância percorrida entre imagens. O algoritmo toma partido de uma câmera calibrada para simplificar a computação. O desvio de obstáculos é conseguido através da utilização de informação da máscara prevista pela rede neuronal convolucional. Dois algoritmos foram desenvolvidos. Um baseado em controlo de posição e o outro usa controlo de velocidade.

Como trabalho futuro, outros tipos de redes neuronais podem ser desenvolvidos e uma estratégia the desvio de obstaculos mais robusta e eficiente pode ser implementada, tirando partido da máscara prevista pela rede.

Palavras Chave: Veículos Autónomos de Superfície, Rio, Desvio Obstáculos, Câmara Monocular, Estrutura a Partir de Movimento, Redes Neuronais Convolucionais

Abstract

Autonomous Surface Vehicles are vehicles that operate on the surface of the water without a crew or operator. They can be used in surveillance, environmental monitoring, and sea/river floor mapping tasks. These tasks take time and need to be automated, so the ASV must navigate autonomously. To achieve this, many types of sensors can be used(LiDAR, Sonar, Stereo Camera Setup), but one of the most versatile is a single monocular camera.

This dissertation takes a look at the state-of-the-art methods for navigation and implements a strategy to map and avoid obstacles. To achieve this, a water segmentation network is implemented that achieves a dice score of 0.9896 in a mixed dataset with images from the Mondego River and the ROSEBUD dataset ([18]). A 3 Dimensional reconstruction algorithm has been developed that estimates distances accurately and eliminates the problem of scale by using the distance travelled between frames. This algorithm takes advantage of a calibrated camera to simplify the distance estimations. Obstacle avoidance is achieved by using information from the CNN prediction mask. Two algorithms were developed. One based on position control and the other using velocity control

For future work, other types of CNN could be developed and a more robust and efficient obstacle avoidance strategy could be implemented, taking advantage of the segmented water mask. The 3D reconstruction needs to be optimized to run in real time in the Single Board Computer (Nvidia Jetson Nano).

Keywords: Autonomous Surface Vehicle, Unmanned Surface Vehicle, River, Obstacle Avoidance, Monocular Camera, Structure From Motion, Convolutional Neural Networks

"Ponho o carro, tiro o carro, à hora que eu quiser"

— Joaquim Barreiros, *A Garagem da Vizinha*

Contents

List of Acronyms

ASV	Autonomous Surface Vehicle
BLDC	Brushless Direct Current
CNN	Convolutional Neural Network
FPS	Frames Per Second
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
HSV	Hue Saturation Value
IMU	Inertial Measurement Unity
ReLU	Rectified Linear Unit
RGB	Red Green Blue
ROI	Region of Interest
ROS	Robot Operating System
RTK	Real-time kinematic positioning
SBC	Single Board Computer
SFM	Structure From Motion
SURF	Speeded-Up Robust Features
USV	Unmanned Surface Vehicle
VFF	Virtual Force Field
VFH	Vector Field Histogram

List of Figures

List of Tables

1 Introduction

All life on Earth depends on water for survival. It is the world's most vital resource. Water is a very plentiful resource, but the majority of it is salt water from the oceans. Less than 3% of the world's total water resources are made up of freshwater, and of that, less than 1% is found in the liquid freshwater that covers the Earth's surface (the rest is in the planet's subsurface or is frozen in the ice caps) [12].

The amount of fresh water that is present on Earth supports a high level of biodiversity, which includes both aquatic life and also land species that depends on inland waters.

The continuously use of chemical fertilizers and pesticides, the occurrence of chemical spills and heavy rains that wash out hazard substances from land to water streams make it important to make sure the water present in rivers is not polluted so that it can keep providing support to all the biodiversity.

1.1 Context and Motivation

To protect water quality and avoid health related risks, natural waters streams require monitoring systems that are able to detect the presence of pollutants. This can be done with static sensor networks or mobile sensors [26]. The use of mobile sensors, such as underwater or surface vehicles presents some advantages over sensors networks: dynamic coverage area due to the ability of the vehicles to move in the environment and the possibility of deploying these vehicles in places that suffered an environmental catastrophe to help find the sources of pollution.

The task of monitoring water resources using mobile sensors is very time consuming and has the need of several operators for the system to work. There is also health risks to the operators if the type of pollution is toxic for humans. This way, Autonomous Surface Vehicles (ASV) have the potential to provide cost and time savings by diminishing the number of operators and also provide enhanced safety by reducing the probability of humans interacting with hazard chemicals.

ASV, also known as Unmanned Surface Vehicles (USV), are vehicles that operate on the surface of the water without a crew. They have a wide range of applications, including: environmental and climate monitoring [7], sea/river floor mapping [1] and surveillance [24].

Unmanned vessels are not a new concept. Nikola Tesla envisioned aquatic drones in his 1898 patent for "Method of and apparatus for controlling mechanism of moving vessels or vehicles" [30].

To achieve autonoumous navigation the ASV needs to have the capability to detect and avoid obstacles. An obstacle can be any surface object that obstructs the navigation path of the ASV, such as surface sandbanks, floating tree logs or canoes used for recreational purposes. Just recently new advancements in autonomously navigating water environments are being made. Many works focus on maritime environments, which present different challenges when autonomous navigation is the goal, when compared to river scenes. Some of the challenges that may arise in this kind of environment in comparison of maritime ones are: fast changing unpredictable water currents, shallow floor that may get the ASVs stuck, a variety of obstacles including small islands and more frequent derbies (wood logs) and a narrower navigation space with uneven margins.

To detect obstacles a variety of sensors can be used (sonars, radars, LiDARs, stereo vision systems) that can be quite expensive and difficult to calibrate. Sonars can only sense underwater and LIDARs are difficult to use due to water turbulence. Monocular cameras on the other hand are relatively cheap, but detecting obstacles using vision can present some challenges because of the complex river environment including: complex water-shore-line, lots of reflections, a broad spectrum of water colors like blue, green, brown, and yellow [10]. Figure 1.1 illustrates some of these challenges. The sensor used in this work is a monocular camera because of the previous stated advantages over other sensors, and for the fact that few works are developed using only a camera as the source of perception in fluvial environments.

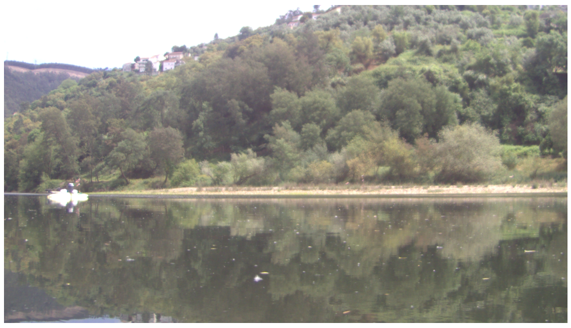


Figure 1.1: Mondego river during May 2022.

1.2 Goals

The goals of this dissertation can be divided in two major goals, and each major goal can be further divided in other goals that need to be accomplished using a monocular camera as the perception mean:

1. Make an ASV navigate in inland waters
    - Water Segmentation
    - Perceive if an obstacle is close
    - Avoid obstacles
2. Map the obstacles that may appear during navigation
    - Collect the necessary information to identify and map obstacles

1.3 Document Overview

The current dissertation is divided in 5 chapters. The first chapter is the Introduction, where the uses of ASV are presented and an introduction to the problem of autonomous navigation is made.

The second chapter is the "Obstacle Detection and Mapping", in which a theoretical background about the techniques used will be presented, as well as a review of the state of the art methods.

The third chapter is Method, in this chapter the solutions found to achieve the goals proposed in 1.2 are presented. This Chapter is divided in 3 main sections. Each section presents the solution for the problems of Water Segmentation, Obstacle Avoidance and Obstacle Mapping, respectively.

The fourth chapter is "Experimental Results" and will contain the results of the conducted experiments in simulation and real life. The experimental setup is also described in this section

The fifth and last chapter is the Conclusion and Future Work were an overview of all the work developed is made as well as suggestions for further improvement of the current state of the work.

2 Obstacle Detection and Mapping

This section aims to provide a theoretical background to better understand the method used to solve the problem of autonomous river navigation and mapping. A review of previous developed works is conducted to identify the limitations that justify this dissertation.

2.1 Related Work

Monocular camera navigation and mapping is a challenge that was tackled in several studies. Some of these studies implement Visual SLAM algorithms capable of using only a monocular camera. Some examples of tese types of algorithms are the ORB-SLAM3 [3] and the Open VSlam [29]. Merzlyakov and Macenski tested different Visual Slam algorithms in [20] that include the ones previously refereed. The conclusion was that, outdoors, Open VSLAM and ORB- SLAM3 out-performed all the others algorithms but using only a monocular camera the algorithms were very dependent of light conditions and not very reliable.

One challenge to overcome to achieve autonomous navigation is to determine the region of interest (ROI), which in the case of autonomous navigation in a river is the area under the riverbank line since it reduces the area to search for obstacles in. A real-time riverbank line detection algorithm is proposed in [9] that uses watershed algorithm and classical Sobel operator for extracting the riverbank line. This work shows promising results but was not tested in complex river environments and lacks evaluation metrics, so the method needs further testing. In [21] a watershed algorithm is combined with the Vector Field Histogram (VFH) algorithm to avoid obstacles. This article also lacks evaluation metrics and it is unclear how the obstacle detection stage is implemented.

Another type of strategy to identify the ROI is based on Convolutional Neural Networks (CNN). The work described in [28] uses several CNN based on the well-known U-Net [23] to segment the water in an image and uses the resulting mask to compute the waterline and detect obstacles. One of the networks used achieves an accuracy of 98.9% in the INTCATCH

Vision Dataset[1] and can run at almost 10 frames per second (FPS) using a Jetson TX2.[2] Ling et al [19] also use an image segmentation network based on U-Net and achieves 96.5% accuracy on a dataset sampled from a ASV on several different rivers. In Chen et al work [5] an encoder network is also used to segment water, although in a maritime environment that does not present many reflections. The main drawback of using networks is the need to have powerful hardware capable of performing the great amount of calculations in a small time period.

Detecting and avoiding the obstacles is explored in [8] using an optical-flow based system that runs on smartphones with a single source of perceptual sensing, via a monocular camera.

The challenge of estimating distances using a monocular camera was solved in [4] using motion stereo where the camera position can be estimated using RTK GNSS and an IMU (Inertial Measurement Unity). The 3D environment can be treated as a 2D space and feature matching can be applied to two different images. The obstacle depth can be calculated using motion parallax method. The method is tested in maritime environment and with a wide baseline between frames that is not possible in most inland waters.

2.2 Water Segmentation

To segment the water and get the ROI two methods were implemented and tested. The first one is based on the computer vision algorithm called Watershed [27], and the second is an image segmentation network based on the U-Net. The following two sections provide a theoretical background that allows the reader to understand the two methods.

2.2.1 Watershed

Before applying the watershed algorithm to an image it is necessary to apply pre-processing techniques to create an image that is easier to segment. Below some of these processes are explained.

Gaussian filtering is used to blur images and remove noise and detail. When working with images, the two dimensional Gaussian function (2.1) must be used, where σ is the standard deviation and x and y are the distances from the kernel center pixel $(0, 0)$.

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.1}$$

[1]`http://profs.scienze.univr.it/farinelli/intcatchvisiondb/intcatchvisiondb.html`

[2]`https://developer.nvidia.com/embedded/jetson-tx2`

To reduce computational costs and because the pixels of an image have discrete coordinates, the 2D Gaussian function is discretized in the form of a kernel. The kernel coefficients are sampled from 2.1. Equation 2.2 shows a 3x3 convolution kernel that approximates a Gaussian with a σ of 1.

$$\omega = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{2.2}$$

The kernel ω is then convoluted with the original image $f(x,y)$ to create the blurred image $g(x,y)$ (2.3).

$$g(x,y) = \omega * f(x,y) \tag{2.3}$$

Hue Saturation Value (HSV) is a color space that separates color from intensity and is often used in computer vision and image processing. It is composed by the hue channel(H), saturation channel(S) and brightness channel(V). The conversion from RGB to HSV is explained and documented in the OpenCV library documentation.[3] Figure 2.1 exemplifies the RGB and HSV color spaces.

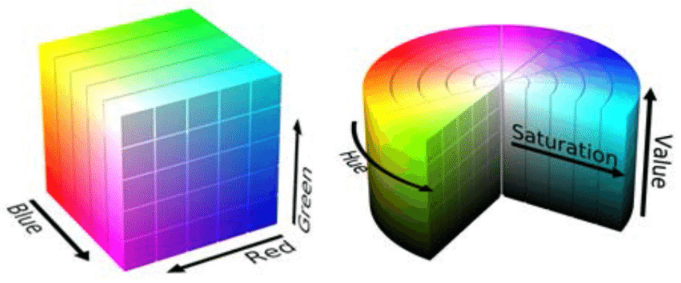


Figure 2.1: RGB Color Space (left) and HSV Color Space (right).[25]

In the field of mathematical morphology, dilation is one of the two fundamental operators. In order to dilate an image, each pixel must be given the highest value found over the neighborhood of the structuring element (kernel). Dilation is represented in 2.4, in which $f(i,j)$ is a gray scale image, $b(s,t)$ is the kernel and $Dilate(i,j)$ is the dilated image.

$$\text{Dilate}(i,j) = \max_{(s,t)\in b}\{f(i+s, j+t)\} \tag{2.4}$$

Erosion is the other basic operator of mathematical morphology. The erosion of an image involves assigning to each pixel, the minimum value found over the neighborhood

[3]`https://docs.opencv.org/3.4/de/d25/imgproc_color_conversions.html`

of the structuring element (kernel). Erosion is represented in 2.5, where $Erode(i, j)$ is the eroded image.

$$\text{Erode}(i, j) = \min_{(s,t)\in b}\{f(i + s, j + t)\} \tag{2.5}$$

The classical watershed algorithm uses flooding simulation in 2D images (where the values of the pixels are seen as heights), as described in [27]. The idea consists in viewing the image as a height map that is going to be flooded with water sources starting from the lowest points and dams are placed where different water sources meet. This dams are the lines that segment the image. Figure 2.2 shows the segmentation process of the watershed algorithm.

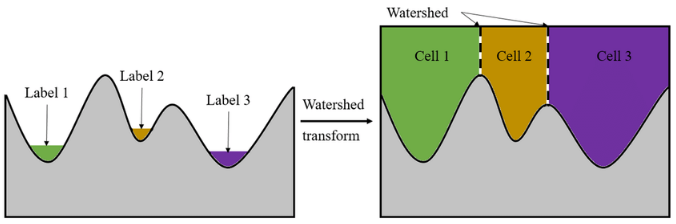


Figure 2.2: Watershed Algorithm segmentation process.[32]

2.2.2 Neural Network

Since this work will use an encoder-decoder type of network for pixel-wise prediction of water, some core concepts need to be explained to fully understand its architecture. An encoder-decoder architecture can be divided in two networks. The first one receives an image and outputs a sequence of numbers. The second network receives a sequence of numbers and outputs an image (mask). These networks use convolutional and transpose layers that will be explained below.

Convolutional Layer

A convolutional layer applies sliding convolutional filters to an input. This type of layer convolves the input by moving the kernel vertically and horizontally over the input and computing the dot product of the weights and the input, and then adding a bias value. The output value of the layer with input size (C_{in}, H, W) and output $(C_{out}, H_{out}, W_{out})$ described as:

$$out(C_{outj}) = bias(C_{outj}) + \sum_{k=0}^{C_{in}-1} weight(C_{outj}, k) \star input(k) \tag{2.6}$$

Where C is the number of channels (for example an RGB image has 3 channels), H is the height of the input plane and W is the width of the input plane, and $\star$ is the 2D cross-correlation operator, also known as the sliding dot product. The stride, the padding and the kernel size are important parameters that need to be tuned to achieve the desired result. The kernel slides from left to right and top to bottom during convolution until it has passed through the entire input image. Stride (S) is defined as the kernel's step. Padding is used to increase the contribution of pixels at the corners and edges in the learning procedure. If a padding of P is applied to an image of size $W \times H$, the output has dimensions $(W + 2P) \times (H + 2P)$. A convolution with the inputs:

- An image of dimensions $W_{in} \times H_{in}$;
- A filter of dimensions $K \times K$;
- Stride S;
- Padding P;

originates an output with dimensions:

- $W_{out} = \frac{W_{in} - K + 2P}{S} + 1$
- $H_{out} = \frac{H_{in} - K + 2P}{S} + 1$

For a better understanding of the procedure, Figure 2.3 represents the first 3 steps of a convolution, where the input has size 4×4, the kernel size is 2×2, the padding is 0 and the stride is 1.

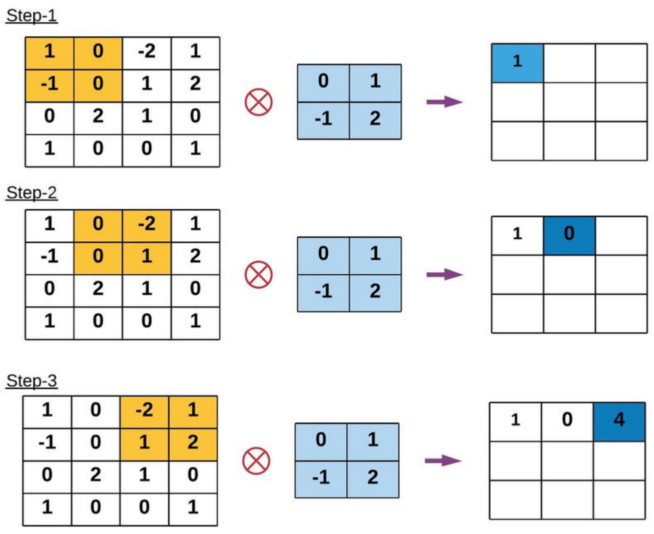


Figure 2.3: First 3 steps of a convolution.[11]

Batch normalization

Batch normalization improves the performance of neural networks by allowing the use of higher learning rates, which significantly accelerates the learning process and also allows the removal of Dropouts [16]. The dataset D is divided in batches B containing m values. Batch Normalization executes a zero mean and unit variance regularization to each dimension of the hidden layers. The batch mean, batch variance and normalization of a general activation are defined in Equations 2.7, 2.8, 2.9 respectively, where x is a layer of the network. A small constant ϵ is added for numerical stability.

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{2.7}$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \tag{2.8}$$

$$\overline{x_i}^{(k)} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{2.9}$$

Normalizing the mean and standard deviation of a unit can reduce the expressive power of the neural network containing that unit [13, Section 8.7.1]. To maintain the expressive power of the network scale and shifting learnable parameters ($\gamma^{(k)}$ and $\beta^{(k)}$) are used. In other words, this operation allows the model to choose the optimum distribution for each hidden layers. The final hidden unit activation becomes:

$$y_i = \gamma\overline{x_i} + \beta \tag{2.10}$$

Max pooling

Max pooling layers are used to sub-sample the feature maps generated by the convolutional layers. It maintains the majority of the dominant information (features) and similarly to convolutional layers both the stride and the kernel size can be tuned. The kernel (pooling window), slides through the plane and outputs the maximum of the values of the input plane inside the pooling window. The max pooling output shape can be computed using equation 2.11, where I is the input shape (width or height), P is the pooling window size and S is the stride. Figure 2.4 exemplifies the process with a stride of 2 and a polling window size of 2×2.

$$Out = \lfloor \frac{I - P}{S} \rfloor + 1 \quad (2.11)$$

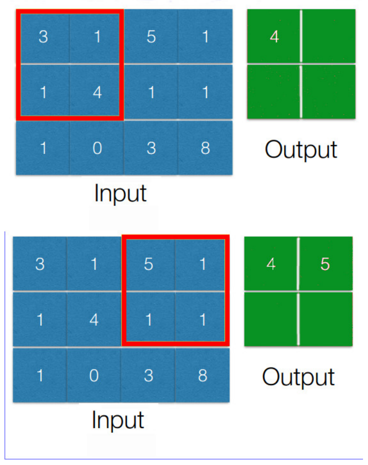


Figure 2.4: First 2 steps of Max Pooling

Activation functions

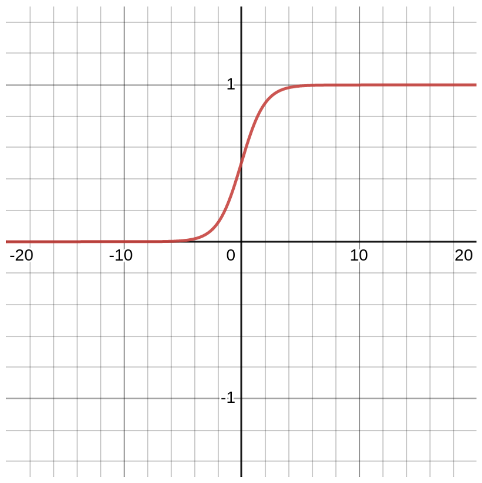
The primary job of an activation function in a neural network-based model, is to translate the input to the output, where the input value is determined by summing the weighted inputs of all the neurons and then adding bias to it. In other words, by generating the corresponding output, the activation function determines whether or not a neuron will activate for a given input. An activation function must be differentiable so that the optimizer can train the model using error back propagation. The two used activation functions in this work are the Sigmoid and the ReLU.

The sigmoid activation function takes real numbers as its input and bind the output between 0 and 1. The graph of the sigmoid function is shown in Figure 2.5a. The mathematical representation of the sigmoid is:

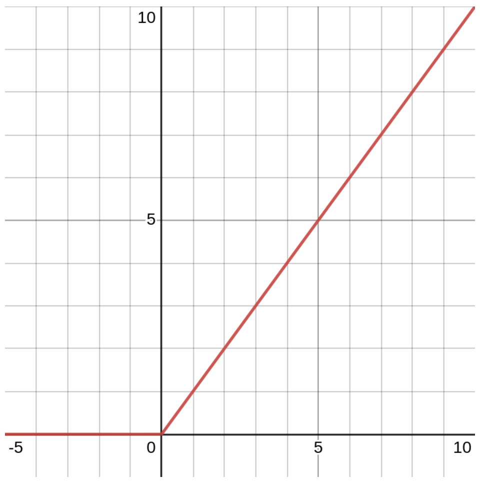
$$S(x) = \frac{1}{1 + e^{-x}} \quad (2.12)$$

The ReLU is used to convert all the input values to positive numbers. The graph of the ReLU function is shown in Figure 2.5b. The mathematical representation of the ReLU is:

$$R(x) = \max(0, x) \quad (2.13)$$



(a) Sigmoid function graph



(b) ReLU function graph

Figure 2.5: Activation Functions

Transposed Convolutions

Transposed Convolutions are upscalers used in Encoder-Decoder architectures as decompressors. They reconstruct the spacial resolution from before a convolution. Imagine a given input as the outcome of a direct convolution applied to a starting feature map. One way to conceptualise a transposed convolution on that input is to imagine the process that enables the recovery of the shape of the initial feature map. To better understand the concept, Figure 2.6 presents the transpose convolution of an 3×3 kernel with an 2×2 input padded with 2×2 border of zeros and a stride of 1.

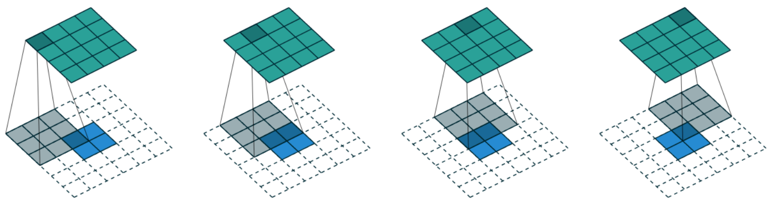


Figure 2.6: Transposed Convolution. Blue represents the input, grey the kernel and cyan the output. Image retrieved from [6].

Skip Connections

Skip connections concatenate two feature maps along the channels dimension. This allows the gradients to more freely flow through the model, mitigating the issue of vanishing gradients [15] and allow features from the encoder side of the network to pass to the decoder side of the network, providing information that could otherwise be lost due to the downsampling on the encoder side.

Loss Function

The Loss Function is used to calculate the prediction error generated by the model over the training instances. The error tells how off is the prediction from the ground truth and is optimized during the training process of the model. The Loss Function uses the prediction of the network and the actual output (label or ground truth) to calculate the error. In this work 3 different Loss Functions were used and are described below, assuming that x is the prediction, y is the label and N is the size of the prediction vector.

Mean Absolute Error is a loss function where the loss is the mean of the absolute difference between the prediction and the label. It can be written as:

$$MAE(x, y) = \frac{1}{N} \sum_{i=0}^{N} |y_i - x_i| \tag{2.14}$$

Mean Squared Error is similar to Mean Absolute Error but uses the mean of the squared difference between the prediction and the label. It can be written as:

$$MSE(x, y) = \frac{1}{N} \sum_{i=0}^{N} (y_i - x_i)^2 \tag{2.15}$$

Binary Cross Entropy Loss is used in binary classification problems. Since the purpose of the network is to predict is a pixel is water or not, it makes sense to use this Loss Function. It can be written as:

$$BCE(x, y) = \frac{1}{N} \sum_{i=0}^{N} y_i \cdot log(x_i) + (1 - y_i) \cdot log(1 - x_i) \tag{2.16}$$

2.3 Distance Estimation

The reconstruction of a 3D scene from a single monocular camera is a complex process that needs a theoretical background to be understood. The process of estimating the 3-D structure of a scene from a set of 2-D images is know as Structure from Motion. Below some of the key concepts to understand SFM are explained.

Pinhole Camera Model

The pinhole model explains the mathematical relationship of the projection of points in three-dimensional space onto an image plane and is the simplest camera model. Figure 2.7 represents the pinhole model in which the virtual image plane is in front of the center of projection.

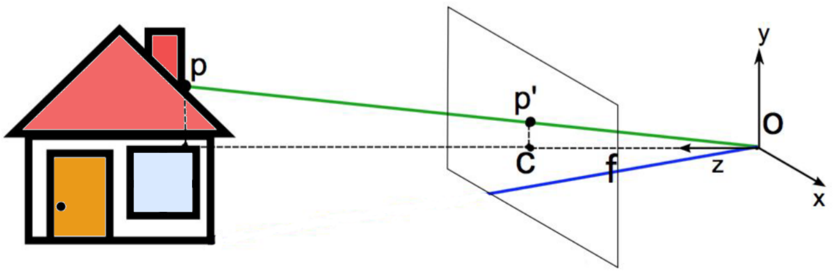


Figure 2.7: Pinhole Camera Model

Considering that $p = [x_p \; y_p \; z_p]'$ and using triangle similarity, it can be derived that:

$$p' = \begin{bmatrix} \frac{x_p}{z_p} f \\ \frac{y_p}{z_p} f \end{bmatrix} \tag{2.17}$$

To express the projection as a linear process, homogeneous coordinates can be used:

$$P' = \begin{bmatrix} fx_p \\ fy_p \\ z \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} \tag{2.18}$$

In the majority of cameras the pixel (0,0), known as the principal point (c_x, c_y), is often located in the top left corner of the image grid, rather than the middle. Hence the previous equation becomes:

$$P' = \begin{bmatrix} fx_p + c_x z_p \\ fy_p + c_y z_p \\ z \end{bmatrix} = \begin{bmatrix} f & 0 & c_x & 0 \\ 0 & f & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} \tag{2.19}$$

The camera calibration matrix (intrinsic parameters) can be represented as:

$$K = \begin{bmatrix} \alpha_x & s & c_x \\ 0 & \alpha_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.20}$$

Where s is the skew coefficient, which is non-zero if the image axes are not perpendicular and α_x and α_y are the focal lengths in pixels.

The ideal pinhole camera has no lens, hence the camera calibration matrix does not take lens distortion into consideration. The camera model must take radial and tangential lens distortion into account in order to effectively depict a genuine camera. Radial distortion occurs when light rays bend more near the edges of a lens than they do at its optical center and tangential distortion occurs when the lens and the image plane are not parallel [4].

To remove distortion from images and estimate the intrinsic parameters, camera calibration is performed [31].

SURF

Feature detection and matching is what allows a program to find the same point in two separate images. Many feature detectors and describers are available to use, but in this work the chosen one was SURF (Speeded-Up Robust Features) [2]. SURF is a scale and rotation invariant feature detector and descriptor that does not use colour information (is applied to gray-scale images).

The use of Integral images reduces the computation time by allowing fast calculation of box type convolution filters. The integral image $I_{\sum}$ at a location x is defined as:

$$I_{\sum}(x) = \sum_{i=0}^{i<x} \sum_{i=0}^{i<y} I(i,j) \tag{2.21}$$

The detector is based on the Hessian matrix. Blob-like structures are detected where the determinant is maximum. The Hessian matrix in a point x at a scale σ is defined as:

$$H(x,\sigma) = \begin{bmatrix} L_{xx}(x,\sigma) & L_{xy}(x,\sigma) \\ L_{xy}(x,\sigma) & L_{yy}(x,\sigma) \end{bmatrix} \tag{2.22}$$

Where $L_{xx}(x,\sigma)$ is the convolution of the Gaussian second order derivative $\frac{\partial^2}{\partial x^2} g(\sigma)$ with the image I in point x and similarly for $L_{xy}(x,\sigma)$ and $L_{yy}(x,\sigma)$. The approximation of the Hessian matrix is made using box filters exemplified in Figure 2.8.

[4]https://www.mathworks.com/help/vision/ug/camera-calibration.html

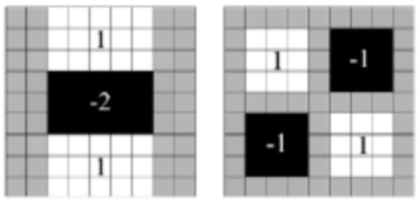


Figure 2.8: Approximation of the second order Gaussian partial derivative in y (D_{yy}) and xy (D_{xy})

These approximations can be computed very fast using integral images. The determinant of the approximated Hessian can be calculated as in Equation 2.23

$$det(H) = D_{xx}D_{yy} - (wD_{xy})^2 \tag{2.23}$$

Where w is a weight used to balance the expression so that the determinant for the simplified kernel roughly matches that of the continuous version. The determinant of the Hessian represents the blob response in the image at point x. Several responses are stored over different scales and local maxima are detected. Scale is implemented by up-scaling the box filters size. The Figure 2.8 represents a 9x9 filter that is considered the initial scale (aproximates the Gaussian derivatives with $\sigma = 1.2$).

To be invariant to rotations SURF needs to assign orientation to the interest points. This is achieved using wavelet responses in horizontal and vertical direction using the filters in Figure 2.9 inside a circular region around the interest point. Once again integral images can be used to speed up the process.

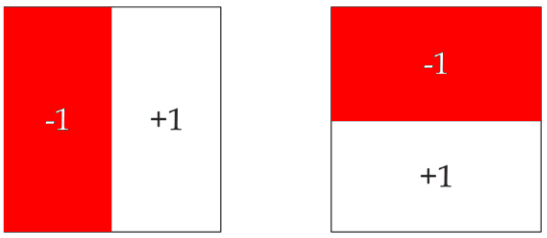


Figure 2.9: Haar wavelet filters used to get the responses in x (left) and y (right) directions

As soon as the responses are computed, they are represented in a space as points. The orientation is computed by summing all the points inside a rotating window of size $\frac{\pi}{3}$. The longest vector across all windows is defined as the orientation of the interest point. Figure 2.10 exemplifies the process.

The descriptor is another part of SURF. To extract descriptors, first a square region is

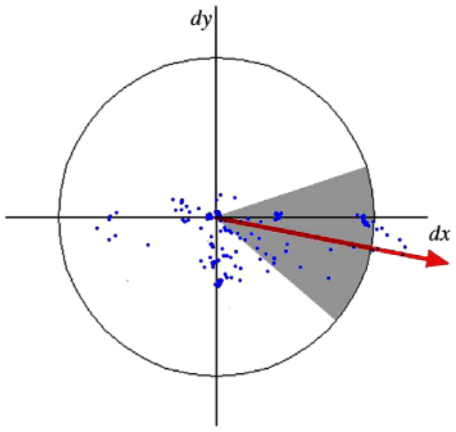


Figure 2.10: Assignment of orientation to a interest point. Image retrieved from [2]

constructed around the interest point and with the same orientation computed previously. The region is split into 4x4 square sub-regions. At each region, the Haar wavelet responses are computed at 5x5 spaced sample points. d_x is the Haar wavelet response in horizontal direction and d_y in the vertical direction. A 4D descriptor vector v (Equation 2.24) is assembled for each region. Because there is a total of 16 sub-regions the feature vector for an interest point becomes 64D.

$$v = (\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|) \tag{2.24}$$

Essential Matrix

The essential matrix E is used when the description of the geometric relations in a image pair is desired. This can be in images taken by a stereo camera setup or in the case of this work, a monocular camera moving in the environment. It contains all the information about the relative orientation from corresponding points. It is a 3x3 homogeneous matrix with rank 2. This rank deficiency is used to formulate the coplanarity constrain:

$$p_1^T * E * p_2 = 0 \tag{2.25}$$

Where p1 and p2 are matching points in two images. To estimate the essential matrix a minimum of 5 points is necessary. A efficient method to estimate the essential matrix was proposed in [22]. After estimating the essential matrix, the rotation R and translation t (up to a scale) of one camera relative to another can be computed. In [14, Chapter 9.6.2] a method to obtain 4 solutions for R and t is presented using singular value decomposition. To choose the correct solution it is only necessary to test with a single point to determine if it lies in front of both cameras.

Camera Matrix

The camera matrix (P) is a 3x4 matrix which maps points in 3D world coordinates to 2D image points:

$$P = K[R|t] \tag{2.26}$$

Where K is the camera calibration matrix (Eq. 2.20) and R and t are the rotation and translation matrix respectively. To map a 3D point X in homogeneous coordinates to 2D image coordinates x it is only necessary to pre-multiply the camera matrix by it:

$$x = PX \tag{2.27}$$

To compute the 3D world coordinates of a pair of matched points in two different images taken by a calibrated camera in two different positions, the triangulation method can be applied. This method employs Direct Linear Transformation and is described in [14, Chapter 12.2].

3 Method

In this chapter the method used to solve the problem of autonomous navigation using a monocular camera is explained.

The solution implemented in this dissertation follows the workflow of Figure 3.1. First an image is acquired through a RGB camera. After that the image is segmented in water and not-water. If a obstacle is sensed to be close by, the ASV must avoid the obstacle and also map the its.

Although two methods of water segmentation are implemented, the one based on CNN is explored more thoroughly because the watershed failed when tested in real images.

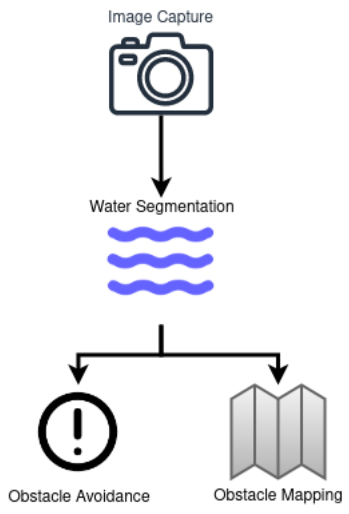


Figure 3.1: Workflow of the implemented work.

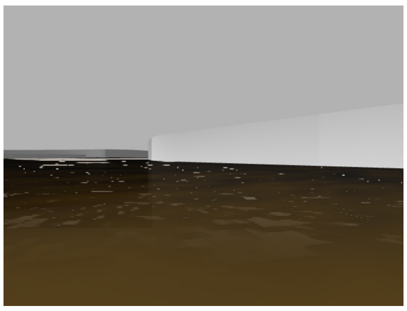
3.1 Water Segmentation

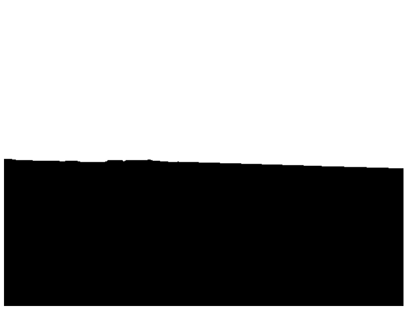
Two methods of water segmentation were implemented. One based on classical computer vision algorithms and the other using Convolutional Neural Networks (CNN).

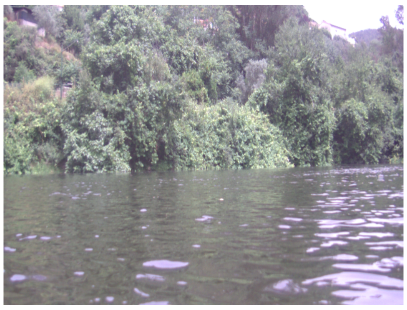
The watershed based segmentation algorithm was chosen because, as presented in 2.1, two articles use this method for water segmentation and both lack evaluation metrics. Convolutional Neural Networks were chosen because some works discussed in 2.1 achieved good results of water segmentation using this method. Another advantage of using CNN is the adaptability to different type of data reducing the need for human supervision to detect important features.

3.1.1 Morphological Watershed

The first method is based on the the Morphological Watershed algorithm as proposed in [9]. This approach takes a RGB image and blurs it using a 7x7 Gaussian filter and converts the blurred image to HSV color space. The Value channel of the image is processed using morphological dilation and erosion, to obtain two images (D and E). The morphological gradient G is computed by subtracting E to D. The last step is to apply the watershed algorithm to the image G. Figure 3.3 shows the evolution of an image through the algorithm and Figure 3.2 shows the results of that algorithm both for simulated and real images.


(a) Simulation Original


(b) Simulation Segmented


(c) Real Original

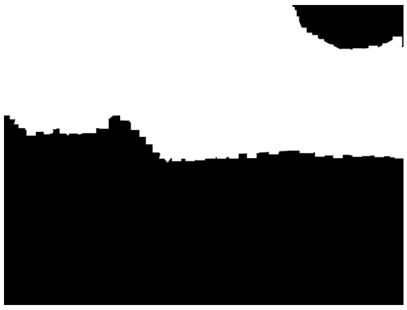
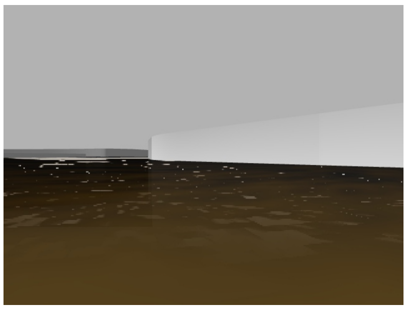
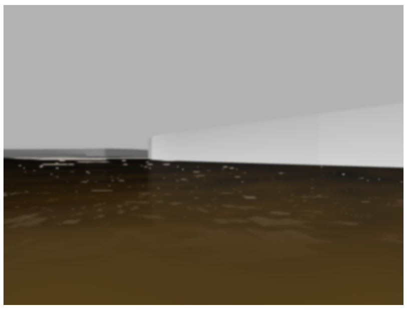
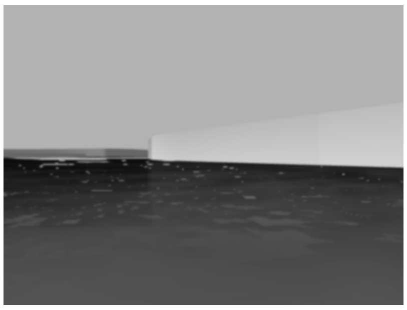
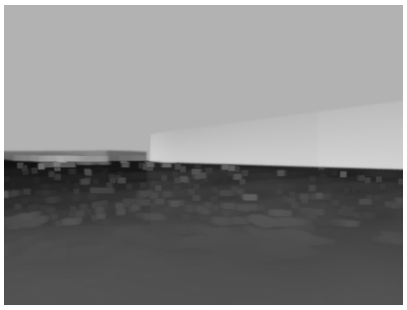

(d) Real Segmented

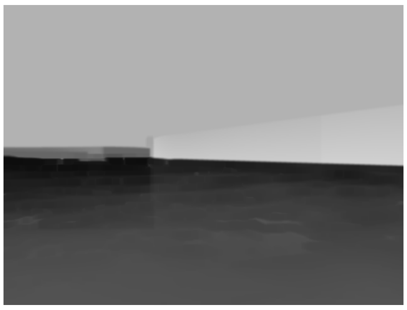
Figure 3.2: Results of the watershed algorithm in simulated and real images


(a) Original


(b) Blurred


(c) Value Channel


(d) Dilation


(e) Erosion

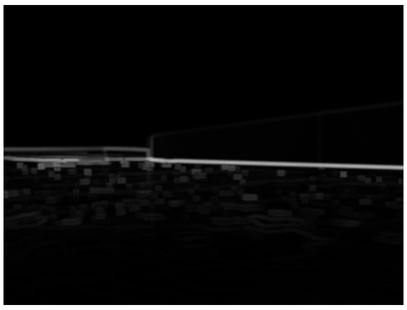

(f) Morphological Gradient

Figure 3.3: The steps of the Morphological Watershed Algorithm

As can be seen from Figure 3.2, the algorithm performs the correct segmentation with images from simulation, but when the images are collected from the real world, the algorithm fails. The lack of success with field data can be attributed to the complex environment. The contrast between water and land is smaller than in the simulation images. Also some parts of the sky are overexposed which leads to a detection as being water. For this reason the method of using the morphological watershed algorithm for water segmentation was not further explored.

3.1.2 Convolutional Neural Network

The second method of water segmentation is based on an Encoder-Decoder type of neural network inspired in the U-Net [23] and the work done by Steccanella et al [28]. This type of architecture has some advantages:

1. The concatenation of channels adds information that would have been lost during the downsampling stage.
2. The training does not require a big amount of data.
3. This architecture doesn't have dense layers. Therefore, the size of the incoming image is not constrained. Convolutional layers have the ability to extract kernel features from any input form while automatically adjusting padding. In contrast, dense layers need to have fixed input size in their definition.
4. By adjusting the number of channels and encoding/decoding stages its possible to easily create models that can run faster or slower obtaining different levels of accuracy.

The pipeline for getting a mask of what is water is shown in Figure 3.4. The image is captured in high resolution (1920x1080) and resized to a smaller size (240x160). The resized image is the input to the CNN which outputs a mask with the probability of a given pixel being water. This mask is then transformed in a binary mask (probability bigger than 0.5 is considered water).

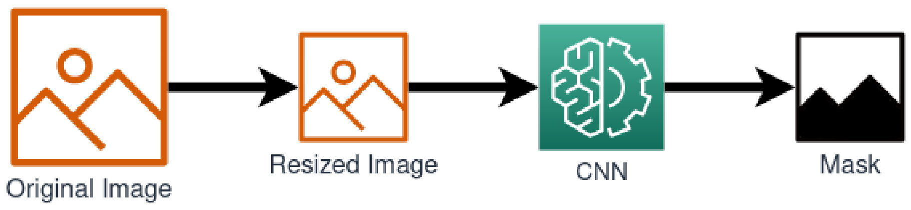


Figure 3.4: Pipeline to get the segmentation mask.

Several variations of architectures were tested, but the main one is presented in Figure 3.5.

The network is composed of an encoding stage that transforms the input image in a 256 feature vector and a decoding stage that transforms the feature vector in the output mask.

The encoding stage is composed of 3 sub-stages, each one made of two sets of 3x3 convolutions, batch normalization and a Relu activation function, followed by a 2x2 max pooling.

The decoding stage is also composed of 3 sub-stages, each one made up of a 2x2 transpose layer, followed by two sets of 3x3 convolutions, batch normalization and a Relu. In each sub-stage of the decoding, a feature map coming from the encoding stage is concatenated with the feature map coming from the transpose layers. In the end the 32 channels map is projected in a single channel and passed through a Sigmoid function to get the probability mask of belonging to the class "water".

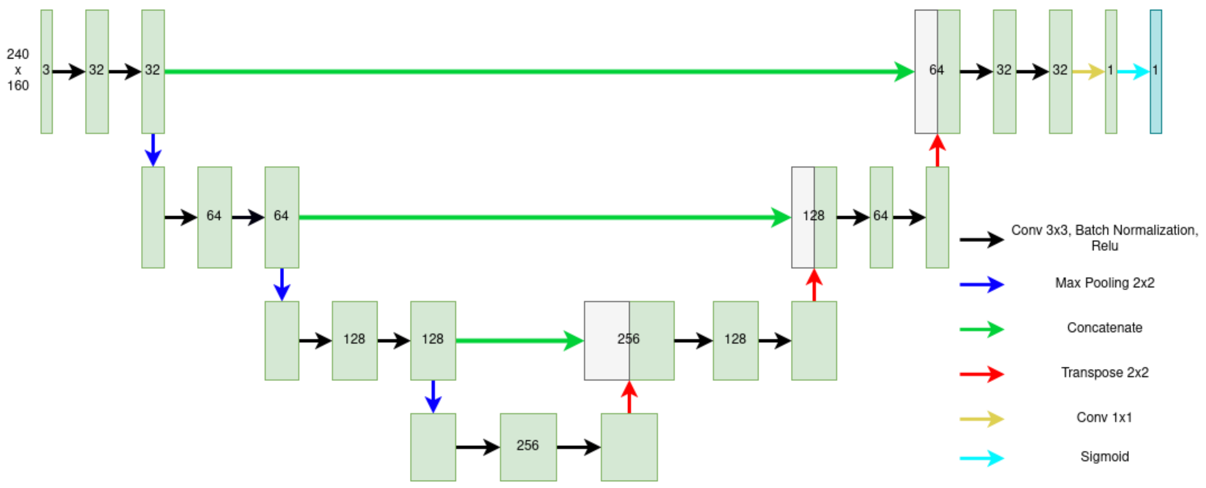
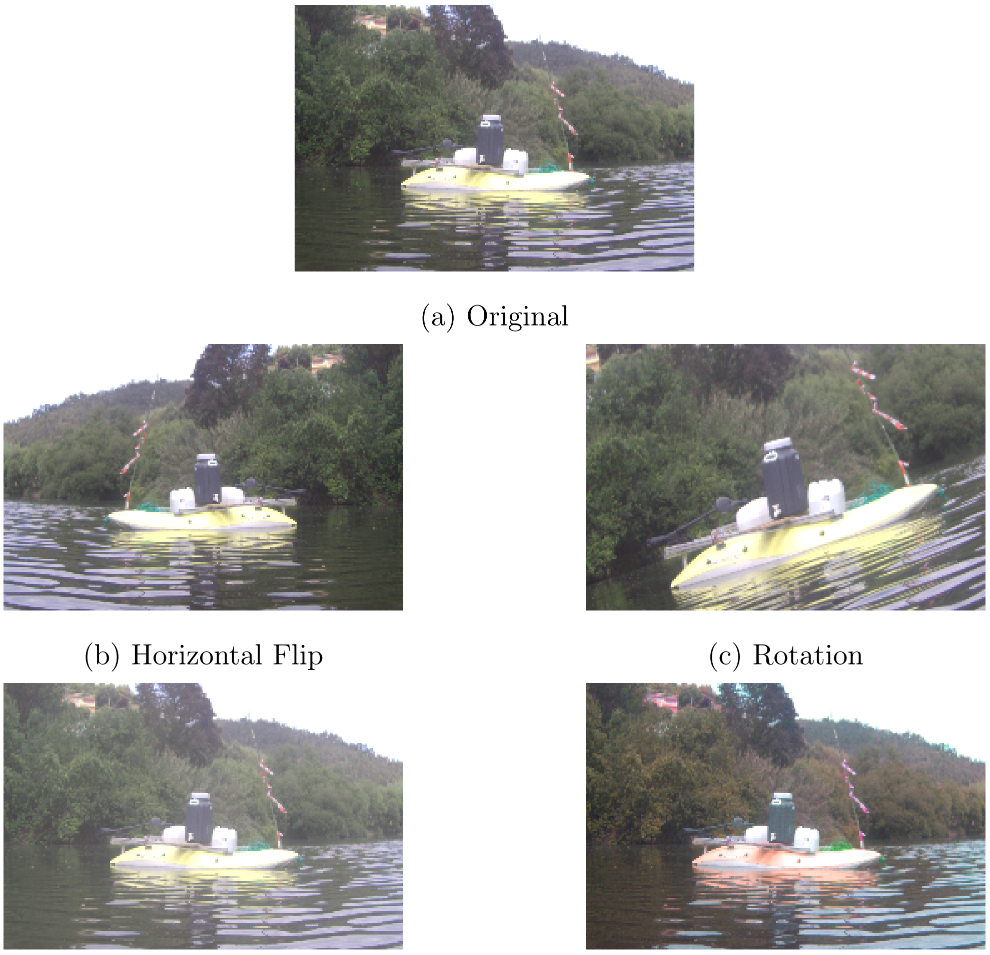


Figure 3.5: Architecture of the main CNN.

To train and evaluate the CNN a dataset is required. Only as recently as of July 2022 that a dataset of fluvial scenes was released to public. This dataset is called ROSEBUD [18] and is composed of 549 images taken from two rivers in the United States of America. Since that at the beginning of the development of this dissertation ROSEBUD was not available, a dataset from the Mondego river was put together, containing 156 images and the corresponding 156 binary masks. The final used dataset was a fusion of both datasets, 156 images from the Mondego and 90 images from the ROSEBUD dataset, totalling 246 images. This dataset was divided in 3 datasets, one for training (70% of the images), one for testing (20%) and one for validation (10%).

A method of expanding a dataset is making data augmentation. This process adds slightly different versions of the same image to the dataset. The data augmentation techniques used in this dissertation are listed bellow and depicted in Figure 3.6.

- Horizontal Flipping
- Image Rotation (the images collected through the ASV will suffer rotations due to small waves)
- Brightness Adjustment (the ASV can be exposed to various lighting conditions)
- Hue and Saturation Adjustment (the color of the river water can vary a lot)



(a) Original

(b) Horizontal Flip

(c) Rotation

(d) Brightness Adjustment

(e) Hue Saturation Adjustment

Figure 3.6: Data Augmentation Techniques

3.2 Obstacle Avoidance

Two very simple avoidance algorithms are proposed, one based in position control and other based in velocity control of the ASV.

In the first one, when an obstacle is detected to be in teh collision path of the ASV, it receives a waypoint to a safe place approximately at the side of the obstacle. After reaching that waypoint, the ASV returns to its original path. Knowing the position of the ASV in the world (x_W, y_W, θ) and the new waypoint coordinates in the local ASV frame (x_L, y_L), the new waypoint in world coordinates is:

$$\begin{cases} x_G = x_W + (x_L \cdot \cos(\theta) - y_L \cdot \sin(\theta)) \\ y_G = y_W + (x_L \cdot \sin(\theta) + y_L \cdot \cos(\theta)) \end{cases} \tag{3.1}$$

The second one uses velocity control of the ASV. When an obstacle is detected to be close by, the algorithm checks the mask to assign an angular and linear velocity so that the ASV moves away from the obstacle. The angular velocity direction is determined by looking to the segmented water mask and choosing left or right based on the section (left or right) of the mask that contains less obstacle belonging to the class "not water".

3.2.1 Close Obstacle Detection

To detect a close obstacle in front of the ASV a simple method was implemented.

First the image of the camera is segmented into a mask containing pixels that are either water or not water. Next, a limit in the y axis of the mask is defined. The number of pixels bellow that limit, belonging to the not water class are counted. An obstacle is considered to be close if the identified pixels are above a certain threshold.

3.3 Obstacle Mapping

To Reconstruct and Map the obstacles present in the environment, an algorithm that performs structure from motion (SFM) from two views was implemented. This algorithm was coded in MATLAB an was not tested in the field since is computational heavy and not very optimized. It was tested using a recorded ROS Bag taken from the field, that contains the images captured by the camera and the positions of the ASV at each moment. The segmentation of the Bag images was done in real time in a laptop. The algorithm is described in 1.

The algorithm seeks to obtain two images of the same obstacle taken from different positions, in order to perform feature matching. The output is the localization of the detected features in the camera frame.

Since the reconstruction of the Obstacle Points can only be done up to a scale, the traveled distance between images is used to calculate the right scale for the points 3D reconstruction. This distance is obtained by the data fusion process running internally in the PixHawk. This process combines information from GNSS RTK signal with the internal IMU data.

This reconstruction is made in the camera frame at the moment of acquisition of the first image. The frame can be easily changed to world frame by using Equations 3.1.

Algorithm 1 SFM From 2 Views

```
1: $t = 500ms$
2: $Mapping = True$
3: while Mapping do
4:     Get Original Image, Prediction Mask, ASV Position
5:     if Obstacle Close then
6:         Saves Image, Mask, Position as Old
7:         waits($t$)
8:         Get New Image, Mask, Position
9:         if Obstacle Close then
10:            Convert Old Image and New Image to Gray scale
11:            Computes Obstacle Points Positions and Camera Poses
12:            $Mapping = False$
13:        end if
14:        waits($t$)
15:    end if
16: end while
17: $\Delta d = NewPosition - OldPosition$
18: $scale = norm(\Delta d)$
19: Scale Obstacle Points Positions by $scale$
```

3.3.1 Camera Calibration

An important step in the SFM algorithm is the estimation of the relative camera poses. The camera poses can be estimated using the Essential Matrix or the Fundamental Matrix, but using the process of estimating the Fundamental Matrix is computationally heavier and requires 8 matching points between images instead of 5 for the Essential Matrix [14, Chapter 11.2]. For this reason a calibrated camera is useful because it allows the estimation of the Essential Matrix. To calibrate the camera used in this dissertation the ROS package *camera_ calibration* [1] in conjunction with the chessboard pattern in Figure 3.7 were used.

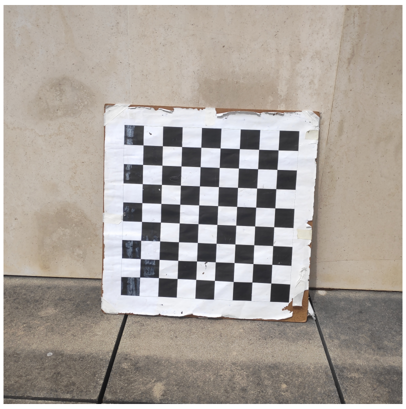


Figure 3.7: Pattern used in camera calibration.

[1]`http://wiki.ros.org/camera_calibration`

3.3.2 Obstacle Position Estimation

To reconstruct the obstacle in 3D, and algorithm that uses feature matching, essential matrix estimation, relative camera pose estimation and triangulation is used. This algorithm is described in Algorithm 2, where image1 and image2 are a pair of images of the obstacle retrieved from two different positions.

Algorithm 2 Obstacle Reconstruction

```
1: Get $image1$ and $image2$
2: Detect and extract features in $image1$ and $image2$ using SURF
3: Match Features
4: Estimate Essential Matrix using matched points and camera calibration matrix $K$
5: Compute the relative camera poses using the Essential Matrix
6: Triangulate the matched points using the estimated camera poses
7: Remove points with a big re-projection error
```

4 Experimental Results

To validate and test the implemented algorithms, it is necessary to conduct tests. Two types of tests were performed: (1) simulations (2) field tests. Simulation allow the evaluation of the developed algorithms without all the trouble of moving the ASV to the field an be dependent on batteries, and also enables testing the algorithms with different combinations of environmental parameters. Field tests validate the results obtained in simulation. This occurs if the simulation is a good representation of the real life scenario.

4.1 Experimental Setup

4.1.1 Hardware

The main piece of hardware used in this work is the ASV developed by Hugo Magalhães and Rui Baptista. Its components architecture is presented in Figure 4.1 and a photograph in Figure 4.2a.

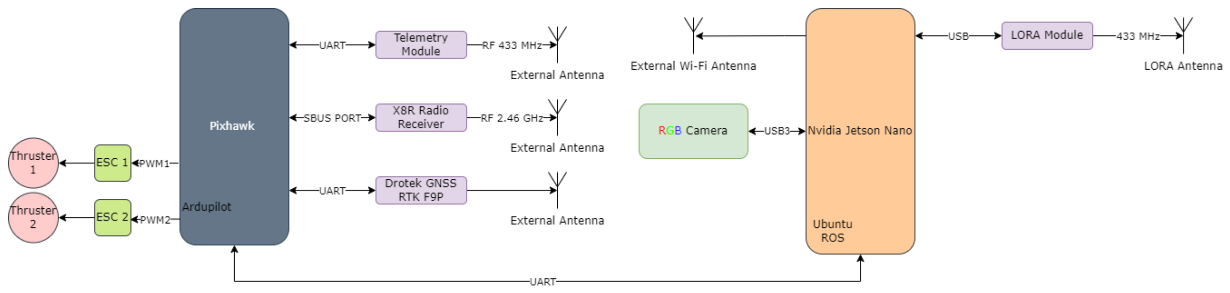


Figure 4.1: ASV Architecture

For sensing the environment a RGB camera is used, more specifically a Basler puA1920-30uc[1] that is able to deliver 30 FPS at a resolution of 1920x1080 . It is paired with a Computar lens with adjustable focal length (3.5-8mm) and an aperture of f1.4.

The Nvidia Jetson Nano[2] was used to run the developed algorithms and all the supporting

[1]https://www.baslerweb.com/en/products/cameras/area-scan-cameras/pulse/pua1920-30uc/

[2]https://developer.nvidia.com/embedded/jetson-nano-developer-kit

software. It is a powerful SBC with a 1.43 GHz Quad-core ARM A57 processor that can run neural networks taking advantage of its own 128-core Maxwell GPU.

The Pixhawk 4[3] is an open-hardware platform that supports many different sensors and can run autopilot software. It comes with integrated accelerometer, gyroscope, barometer and compass.

The ASV needs to know its location in the world with precision to be able to navigate to waypoints. For this reason Global Navigation Satellite System (GNSS) must be used. In this case the Drotek GNSS RTK F9P[4] module is used. GNSS Real-time Kinematic (RTK) allows to correct the position of the ASV by using two receivers (one in the ASV and other in a base station described later) and achieve centimeter level accuracy.

The ASV uses differential thrust for moving along the river, thus allowing for in place rotations and very sharp turns. This is achieved by using two T200 Thrusters from Bluerobotics[5] that use BLDC motors.

During the development of this work the thrusters supports where re-design as well as the floater tips. A camera mount was design from scratch. This parts were then 3D printed using Polyethylene Terephthalate Glycol (PETG). Figures of the designs are available in Appendix A.

To allow the communication between a laptop and the ASV and to use RTK GNSS a base station containing a WiFi Router and antenna was used (Figure 4.2b). This base station was first developed for the project TIRAMISU [6].

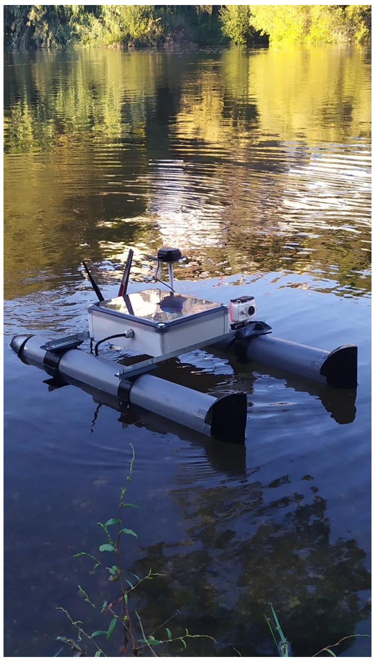
The training process of the neural network and 3D reconstruction algorithms were run in a laptop with a 2.4 GHz Intel Core i5-9300HF and a 4 GB NVIDIA GTX 1650.

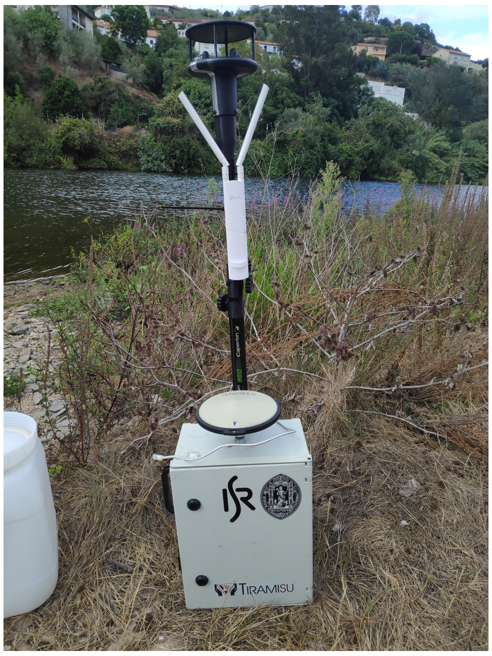
[3]https://docs.px4.io/main/en/flight_controller/pixhawk4.html

[4]https://store-drotek.com/891-rtk-zed-f9p-gnss.html

[5]https://bluerobotics.com/store/thrusters/t100-t200-thrusters/t200-thruster-r2-rp/

[6]https://cordis.europa.eu/project/id/284747


(a) ASV



(b) Base Station

Figure 4.2: Photographs of the ASV and Base Station.

4.1.2 Software

For the implementation of this work several software tools were used, either being open source like ROS[7] or commercial tools like MATLAB[8].

ROS is an open-source system for robots. It provides the services similarly to an operating system like hardware abstraction, low-level control, message-passing between processes and package management. Through the use of tools and libraries it allows to obtain, write and run code across multiple computers. The ASV is running a ROS node called MAVROS that converts ROS messages and topics to MAVLink messages allowing the Jetson to communicate with the Pixhawk running Ardupilot[9] (an open source auto pilot framework).

MATLAB is a programming language based on matrix operations. Many different packages are available that add functionalities and simplify the coding of complex algorithms.

PyTorch[10] is an open source deep learning library for tensor operations with support for GPU and other hardware acceleration and efficient tools. It simplifies the process of coding and running neural networks.

All the simulations where run in the Gazebo simulator which is fully integrated with

[7]`https://wiki.ros.org/ROS/Introduction`

[8]`https://www.mathworks.com/products/matlab.html`

[9]`https://ardupilot.org/`

[10]`https://pytorch.org/`

ROS.[11] The worlds and models for the ASV used were taken from the repository Iq_Sim[12] and from river reconstructions performed by the team in the Field Robotics Lab.

4.2 Water Segmentation

The performance of CNN classification was evaluated by using 3 metrics: accuracy (A), dice score (DS) and worst dice score. These metrics are calculated from true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) using the formulas below:

$$A = \frac{TP + TN}{TP + FP + TN + FN} \tag{4.1}$$

$$DS = \frac{2 \cdot TP}{2 \cdot TP + FP + FN} \tag{4.2}$$

The worst dice score is the smallest dice score achieved during the testing phase.

The training was executed during 40 Epoch and the optimizer Adam [17] was used in all the experiments. The model of the epoch with the best Dice Score during training was saved and the test dataset was run through it.

4.2.1 Simulation

To train and evaluate the performance of the CNN in the simulation environment, a small dataset was collected and labelled from the Gazebo simulation (100 images). This dataset was divided in training (70 images), test (20 images) and validation (10 images). The architecture of the CNN is the same as the one presented in Figure 3.5. To facilitate representations of alternative architectures the parameter $features$ will be used. Each value of $features$ represents an encoding stage and the numbers of channels in that stage. For the architecture in Figure 3.5, $features$ is equal to $[32, 64, 128]$. The Hyper-Parameters used during training are represented in Table 4.1.

Table 4.1: Simulation Dataset Training Hyper-Parameters

Model Name	Learning Rate	Batch Size	Loss Function	Features
Sim	0.0001	16	BCE	[32, 64, 128]

[11]`https://gazebosim.org/home`

[12]`https://github.com/Intelligent-Quads/iq_sim`

The dice score of the validation dataset during training is shown in Figure 4.3 and more results are present in Table 4.2.

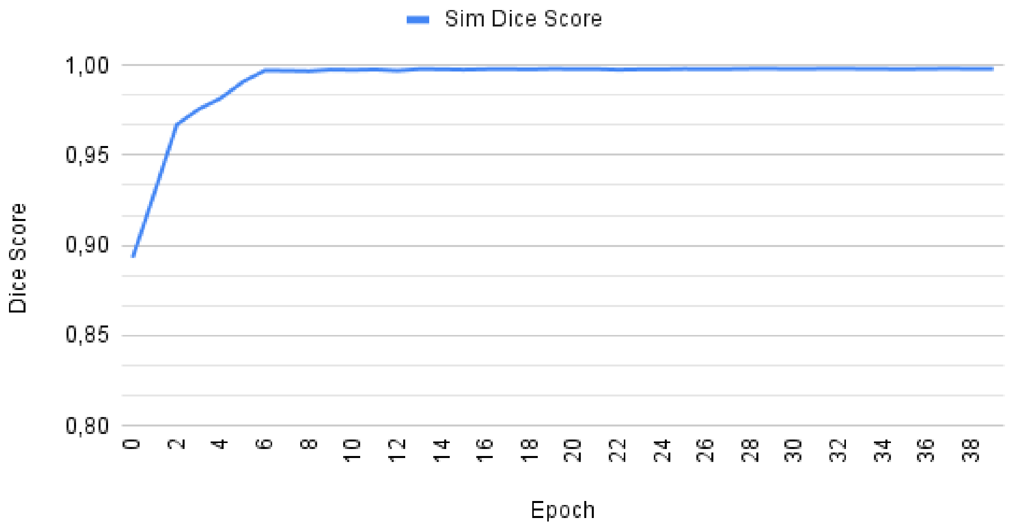


Figure 4.3: Evolution of the Dice Scores during training in the Simulation Dataset

Table 4.2: Results for Simulation Dataset

Model Name	Validation Max Dice	Test Accuracy	Test Dice	Test Worst Dice
Sim	0.9980	0.9978	0.9971	0.9950

As can be seen by these results, the CNN has very high accuracy and dice scores when run using simulated images.

4.2.2 Field Experiment

To evaluate the CNN capabilities in real life data, several Hyper-Parameters combinations were tested. These combinations are represented in Table 4.3.

Table 4.3: Real Life Dataset Hyper-Parameters Combination #1

Model Name	Learning Rate	Batch Size	Loss Function	Features
LR1 (Standard)	0.0001	16	BCE	[32, 64, 128]
LR2	**0.0002**	16	BCE	[32, 64, 128]
LR3	**0.00005**	16	BCE	[32, 64, 128]
MAE	0.0001	16	**MAE**	[32, 64, 128]
MSE	0.0001	16	**MSE**	[32, 64, 128]

The dice score of the validation dataset during training is shown in Figure 4.4 and more results are present in Table 4.4.

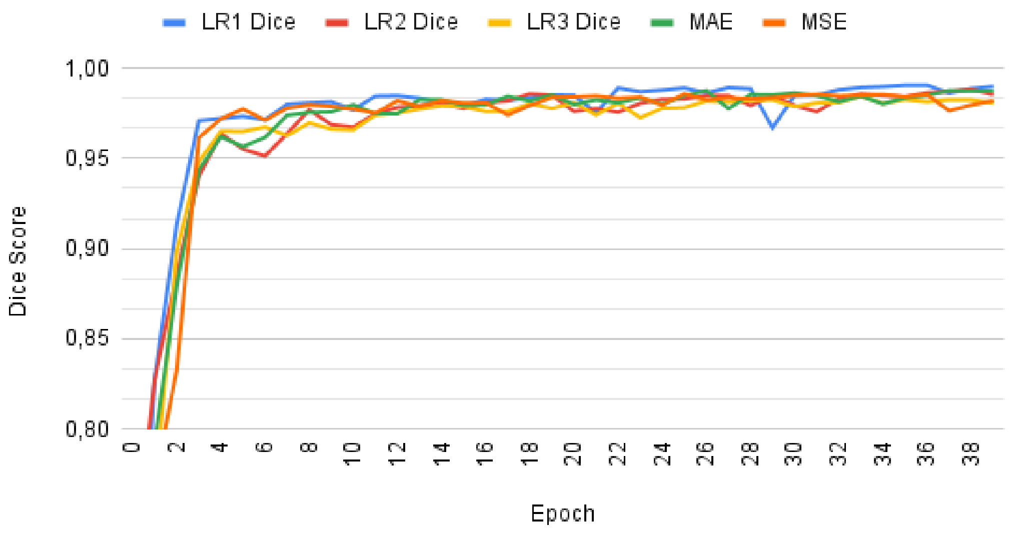


Figure 4.4: Evolution of the Dice Scores during training in the Real Life Dataset #1

Table 4.4: Results for Real Life Dataset #1

Model Name	Validation Max Dice	Test Accuracy	Test Dice Score	Test Worst Dice
LR1	0,9905	0,9909	0,9896	0,9347
LR2	0,9882	0,9901	0,9886	0,9329
LR3	0,9844	0,9871	0,9853	0,9340
MAE	0,9874	0,9896	0,9881	0,9332
MSE	0,9857	0,9848	0,9825	0,9266

As can be seen in 4.4, the best performing combination in all the metrics is LR1. For this reason it is this combination of Hyper-Parameters that will be used. As expected the results are not as good as the simulation due to a bigger complexity and variety of images in the Real Life dataset.

Another experiment that was carried out, was to try two more complex models of this type of architecture. One with an extra encoding/decoding stage and another with more channels per stage. The hyper-parameters of these more complex networks, as well as the parameters of the $LR1$ combination (for comparison purposes) are represented in Table 4.5.

The dice score of the validation dataset during training is shown in Figure 4.4 and more results are present in Table 4.6.

Table 4.5: Real Life Dataset Hyper-Parameters Combination #2

Model Name	Learning Rate	Batch Size	Loss Function	Features
LR1 (Standard)	0,0001	16	BCE	[32, 64, 128]
FEA1	0,0001	16	BCE	[32, 64, 128, 256]
FEA2	0,0001	8	BCE	[64, 128, 256]

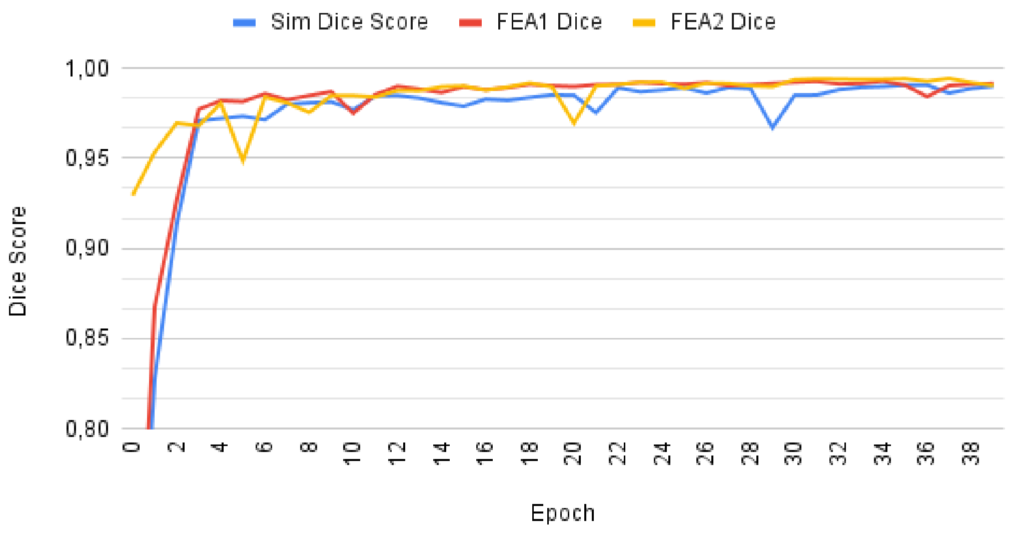


Figure 4.5: Evolution of the Dice Scores during training in the Real Life Dataset #2

Table 4.6: Results for Real Life Dataset #2

Model Name	Validation Max Dice	Test Accuracy	Test Dice	Test Worst Dice
LR1 (Standard)	0,9905	0,9909	0,9896	0,9347
FEA1	0,9926	0,9913	0,9901	0,9505
FEA2	0,9877	0,9889	0,9874	0,9023

For training the combination $FEA2$ the batch size was reduced because the GPU didn't have sufficient memory to carry out the training with a batch size of 16. As can be seen from Table 4.6, $FEA1$ outperformed every other combination in all metrics. The big disadvantage of this model is the size and complexity that does not allow it to run in real-time in the Nvidia Jetson Nano.

4.3 Obstacle Mapping

The goal of obstacle mapping is not only to estimate the distance of an obstacle to the ASV, but also do some 3D reconstruction of the obstacle in question. In reality, no map was constructed; instead, a plot of points in real-world scale representing the position of the obstacles features was created.

4.3.1 Simulation

The simulated world is built in Gazebo and contains two different obstacles that need to be mapped. The first one is a truck that is partially submerged, and the other is a big tree in the middle of the water. For the ground-truth distance a LiDAR was added to the ASV model.

Figures 4.6 and 4.7 show the result of feature matching using SURF features. As can be seen, SURF does a good job of matching features in both cases, with very few mismatches.

Figure 4.8 presents the result of distance estimation to the obstacle. It is a plot of a top view of the 3D reconstruction, in which the red dots are the estimated points and the blue dots are the ground-truth achieved using the LiDAR. The ASV position is at the origin. Figure 4.9 represents the 3D reconstruction of the tree. The two figures show good results in both tasks (the distance error between the estimated and the ground-truth is less than 30 cm).

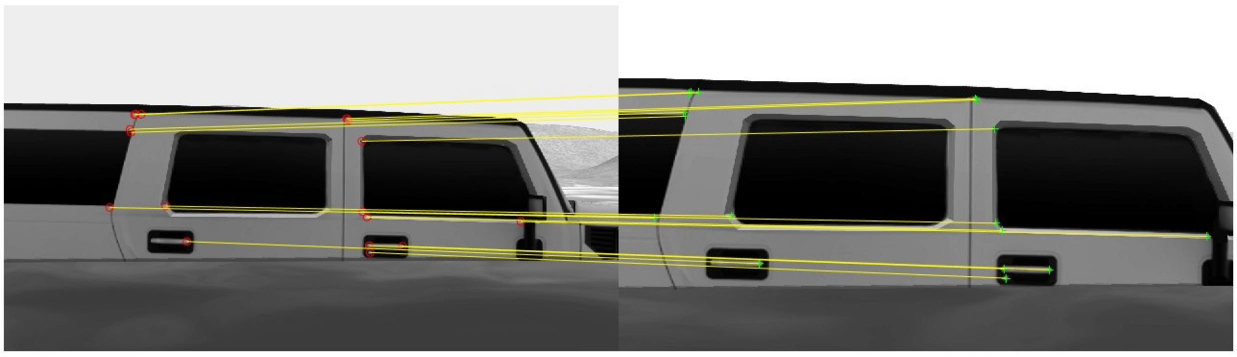


Figure 4.6: SURF Features Matches of Truck Obstacle

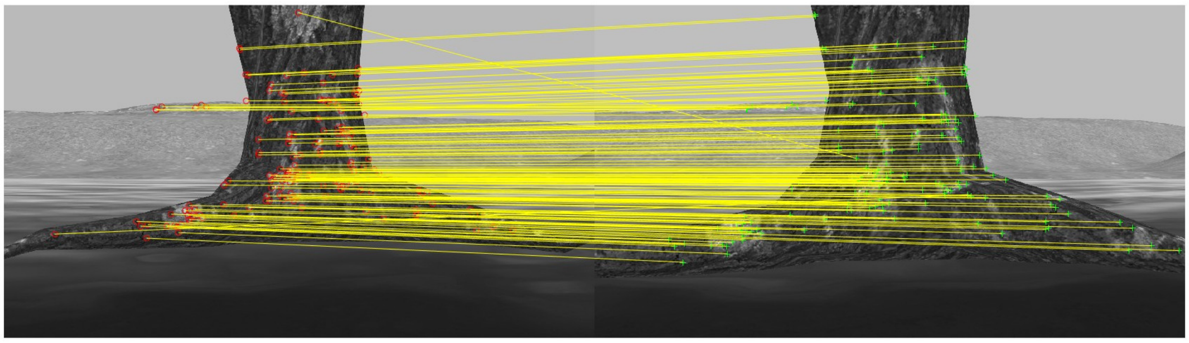
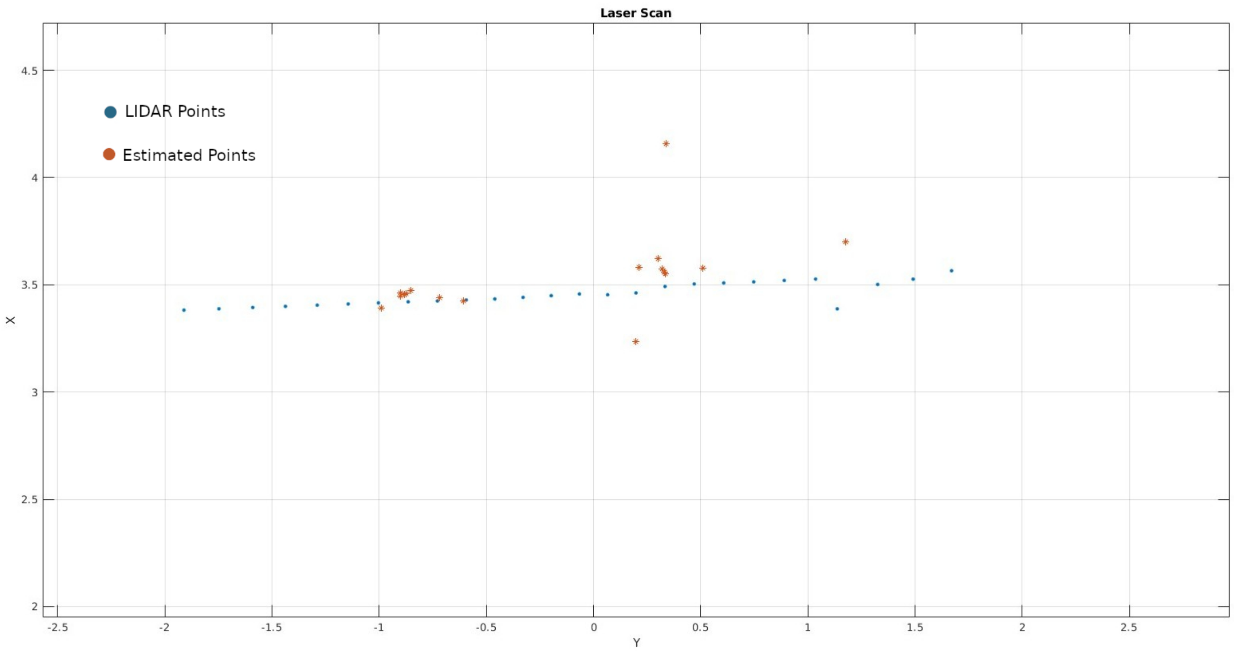
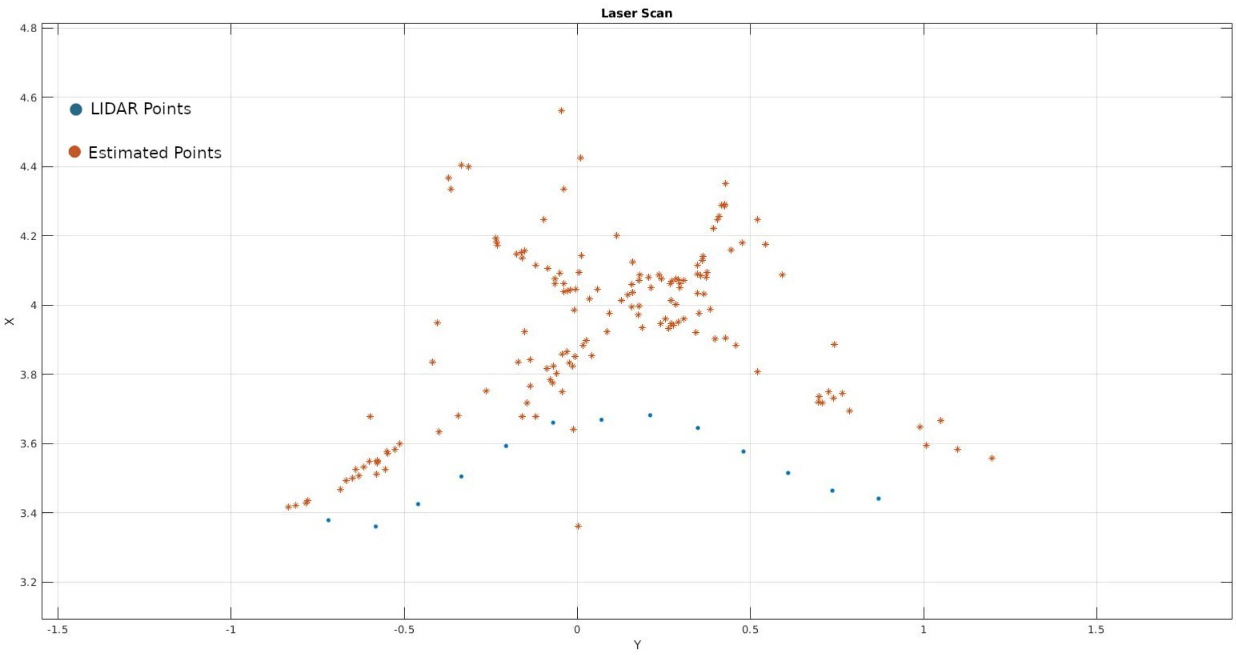


Figure 4.7: SURF Features Matches of Tree Obstacle



(a) Truck Obstacle



(b) Tree Obstacle

Figure 4.8: Distance to Obstacles Plot

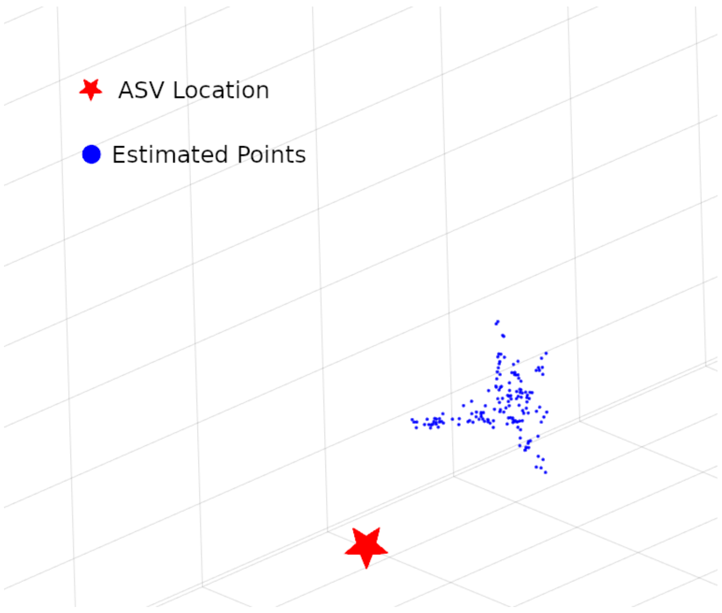


Figure 4.9: 3D reconstruction of the tree

4.3.2 Field Experiments

The data for the real life tests was collected in Mondego river and a canoe was used as an obstacle. For the ground-truth the position of the two extremes of the canoe was extracted using GNSS RTK.

Figure 4.10 shows the result of feature matching using SURF features. SURF was able to extract and match features bellowing to the canoe with success, but also extracted some features of the background hills and trees.

Figure 4.11 presents the result of the estimation of distance to the obstacle. It is a plot of a top view of the 3D reconstruction, in which the blue dots are the estimated points and the red line is the ground-truth for the position of the canoe. Figure 4.12 represents the 3D reconstruction of the canoe. The two figures show good results at both tasks but the number of points in the reconstruction of the canoe is a little low and some outliers are present in the distance plot.

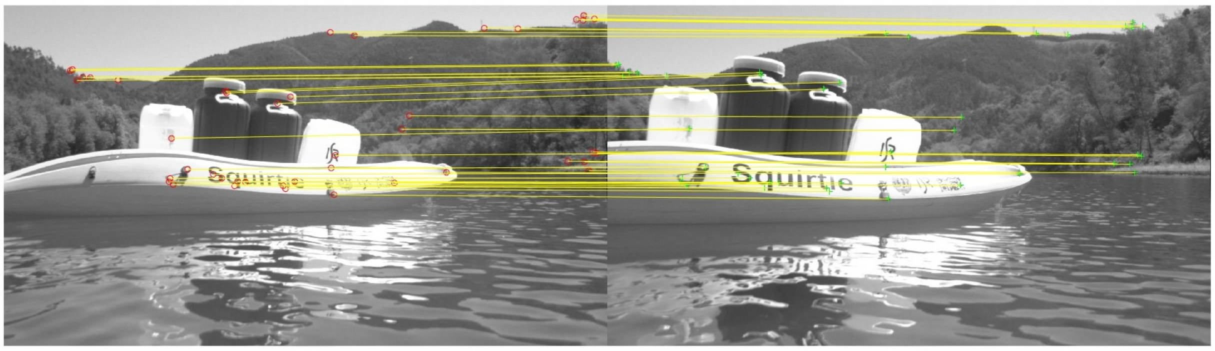


Figure 4.10: SURF Features Matches of Boat Obstacle

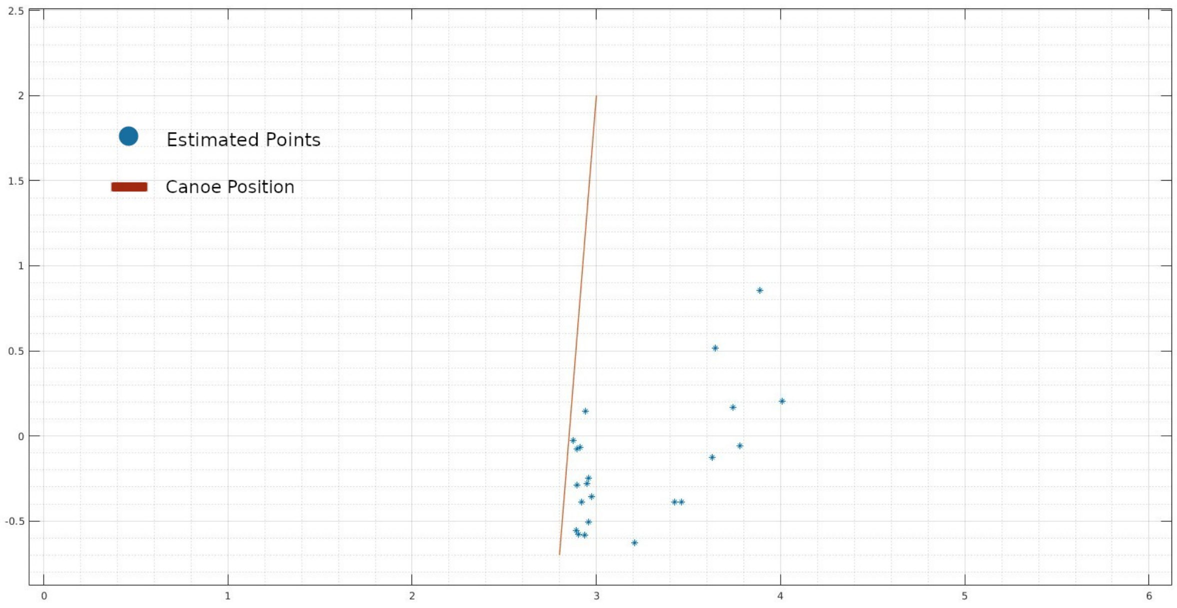


Figure 4.11: Distance to Canoe Plot

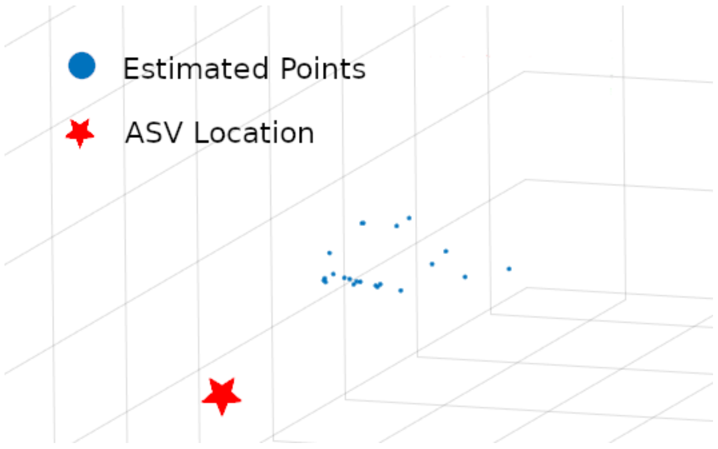


Figure 4.12: 3D reconstruction of the canoe

4.4 Obstacle Avoidance

4.4.1 Simulation

The simulated world is built in Gazebo and a partially submerged truck was used as an obstacle (Figure 4.13). The mission of the ASV was to move twenty meters forward from its starting position without crashing with the truck. The waypoints used to avoid the obstacle when detected were $(1; 3)$ and $(4; 3.5)$. The path of the ASV is presented in Figure 4.14, where the red rectangle represents the truck.

As can be seen from Figure 4.14, the ASV was able to detect the obstacle within a safe distance that allowed for an in place rotation to move to a safe place and finish the starting mission. The velocity avoidance algorithm was not successful in simulation. This was because the big dimensions of the simulated ASV that caused it to crash with the truck when turning.

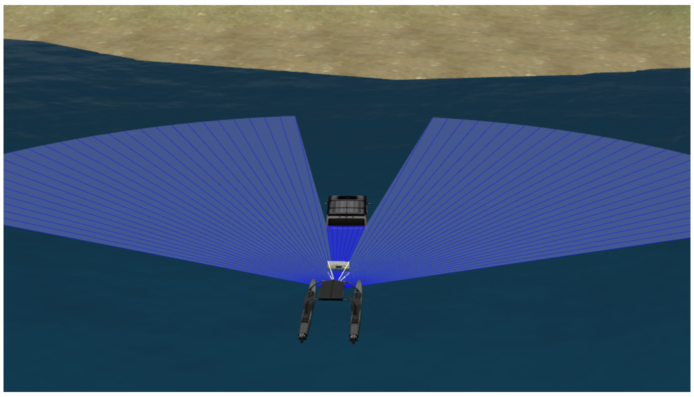


Figure 4.13: Scenario for Obstacle Avoidance Test

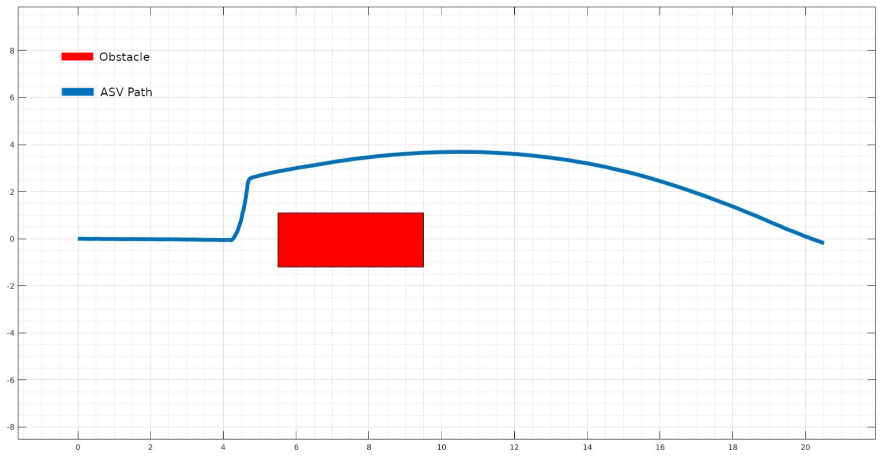


Figure 4.14: Path described by the ASV during the test

4.4.2 Field Experiment

To test the obstacle avoidance algorithm in the field, a floating obstacle consisting of 3 empty water jugs tied together was used (Figure 4.16a). The mission of the ASV was to move to a pre-programmed location in the river. The obstacle avoidance algorithm tested was the velocity control since the CNN presented a big number of false positives detections of obstacles. This was due to the water being particularly choppy and reflections that overexposed the image in some areas (Figure 4.15). Figure 4.17 depicts the path of the ASV during one successfully obstacle avoidance test and Figure 4.16 shows the image and mask of the obstacle close by.

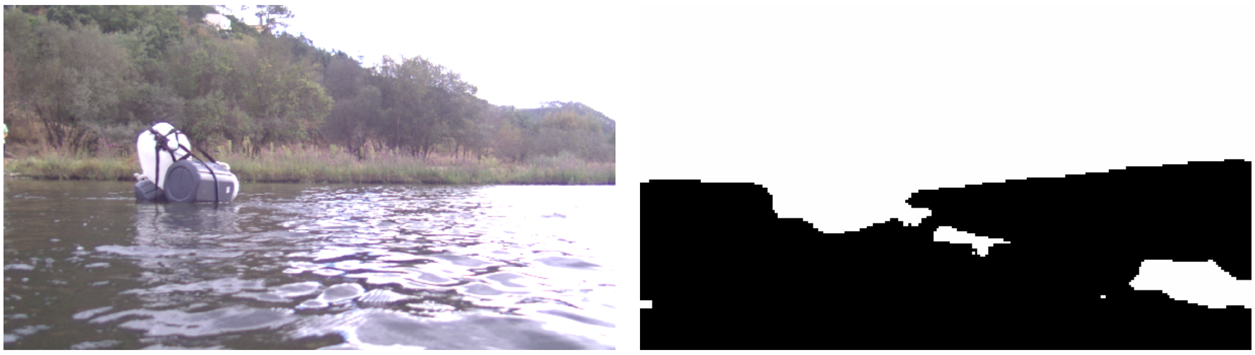

(a) Image of the obstacle far away (b) Mask containing a false positive obstacle

Figure 4.15: False Positive Obstacle Detection

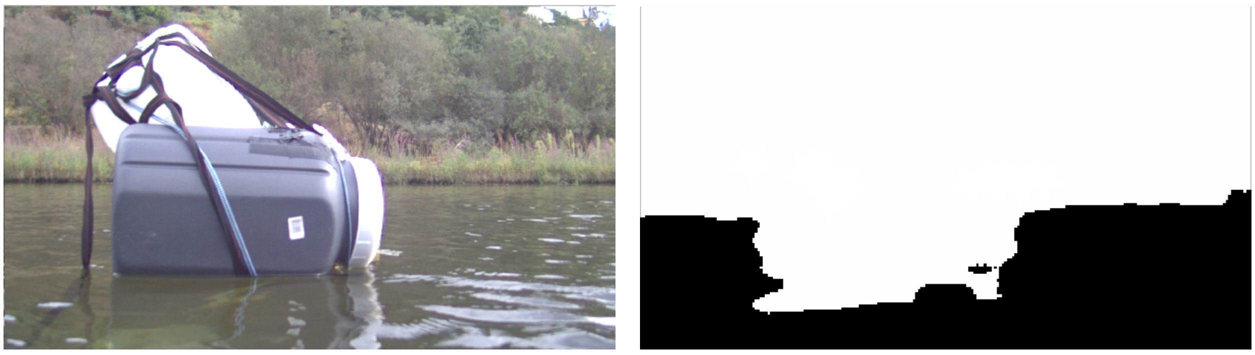

(a) Image of the obstacle close by (b) Mask when the obstacle is close by

Figure 4.16: Close Obstacle Detection

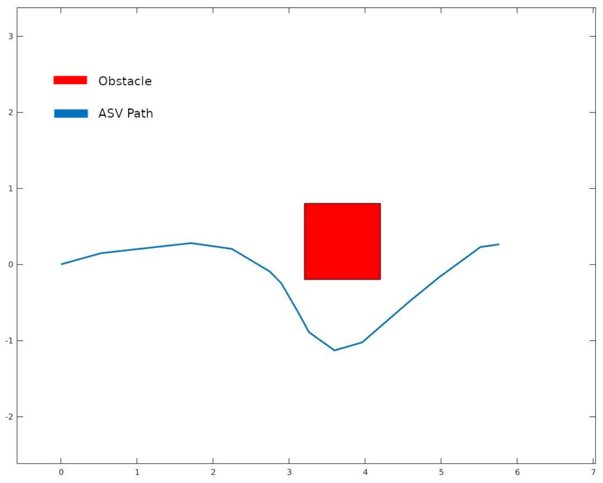


Figure 4.17: Path described by the during the test

During the field tests many runs were not successful because of false detections of the obstacle and some failled due to the GNSS RTK not working and depending only in the GNSS, which didn't allow a accurate velocity control. The RTK did not work properly due to some sort of electromagnetic interference that occurred when the Jetson was running at full power (segmenting the images).

5 Conclusion

This dissertation aimed to develop a method that would allow an ASV to autonomously safely navigate and map obstacles in river basins using only a monocular camera as the source of perception and different types of algorithms were studied.

The developed algorithms can be divided into three categories: Water Segmentation, 3D Reconstruction, and Obstacle Avoidance. For water segmentation, an approach based on Neural Networks was used since this task proves to be too complex and unpredictable for classical computer vision algorithms like the watershed. The developed CNN was computationally light enough to run in real time in a SBC (Nvidia Jetson Nano) and accurate enough to detect and avoid obstacles safely in simulation. In real life some false positives limited the success of the obstacle avoidance task. The Obstacle Mapping algorithm takes advantage of a calibrated camera to simplify the distance estimations and allow more accurate results. This algorithm uses many computer vision concepts such as: calibration matrix, essential matrix, triangulation and feature detection and matching that when combined result in a 3D mapping/reconstruction of the obstacles. For the Obstacle Avoidance algorithm, a simple waypoint reactive avoidance strategy has been implemented, as well as a velocity control strategy. The waypoint strategy worked in the simulation but failed in the real world scenario due to a high number of false positives obstacle detections. The velocity control strategy didn't work in simulation but in real life had moderate success, but with lots of false positives.

In short, the developed CNN was fast and accurate in simulation but it often failed in real life due to overexposed reflections being detected as obstacles and used almost all the processing power of the SBC. The 3D reconstruction worked well for obstacles with texture, but when the obstacles presented a more plain texture, the number of reconstructed points was low, although the distance estimation was accurate for both cases. Since the algorithm was sub-optimized, it needed to be run on a separate computer. The obstacle avoidance algorithm has lots of room for improvement.

5.1 Future Work

In spite of the fact that the ASV navigates well in simulation and is able to map obstacles in both natural environments and simulation, some improvements can be made in future work.

New CNN architectures are being developed each day and can be tested in the task of water segmentation. A lighter/more optimised network could be developed that would allow to free some of the processing power of the SBC and solve the problem of false positives obstacle detections. Few datasets of complex water environments are publicly available. The ones available present some limitations: the majority are from maritime environments and the ROSEBUD dataset lacks variety of weather conditions and obstacles. A creation of an opensource database of field data collected using ASVs could solve the problem of data scarcity and propel the use of deep learning in aquatic scenarios.

A number of optimizations can be made to the mapping algorithm to be able to run in real-time in the SBC. Moving from MATLAB to C++ and using well optimised libraries could reduce the computational time. Testing other feature descriptors and matching algorithms can further improve computational time. The fact that the plot of features belonging to the obstacles is already in real world scale and the distances predictions are accurate allows an implementation of a mapping algorithm in the future.

The obstacle avoidance can be severely enhanced. An algorithm based on VFH or VFF could be combined with the mask outputed by the network to decide the best path to take in order to avoid the obstacle. Finally, tests in different rivers and seasons could be conducted to assess the robustness of the algorithms.

6 References

[1] Muhammad Faiz Abu Bakar and M.R. Arshad. ASV data logger for bathymetry mapping system. In *2017 IEEE 7th International Conference on Underwater System Technology: Theory and Applications (USYS)*, pages 1–5, 2017.

[2] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (SURF). *Computer Vision and Image Understanding*, 110(3):346–359, 2008.

[3] Carlos Campos, Richard Elvira, Juan J. Gómez Rodríguez, José M. M. Montiel, and Juan D. Tardós. ORB-SLAM3: an accurate open-source library for visual, visual-inertial and multi-map SLAM. 2020.

[4] Jiaying Chen and Han Wang. An obstacle detection method for USV by fusing of radar and motion stereo. In *2020 IEEE 16th International Conference on Control & Automation (ICCA)*, pages 159–164, 2020.

[5] Xiang Chen, Yuanchang Liu, and Kamalasudhan Achuthan. WODIS: Water obstacle detection network based on image segmentation for autonomous surface vehicles in maritime environments. *IEEE Transactions on Instrumentation and Measurement*, pages 1–13, 2021.

[6] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *ArXiv e-prints*, 2016.

[7] Matthew Dunbabin and Lino Marques. Robots for environmental monitoring: Significant advancements and applications. *IEEE Robotics & Automation Magazine*, 19(1):24–39, 2012.

[8] Tarek El-Gaaly, Christopher Tomaszewski, Abhinav Valada, Prasanna Velagapudi, Balajee Kannan, and Paul Scerri. Visual obstacle avoidance for autonomous watercraft using smartphones. In *Proceedings of AAMAS '13 Workshop on Autonomous Robots and Multirobot Systems (ARMS '13)*, 2013.

[9] Tianwei Feng, Junfeng Xiong, Jinchao Xiao, Jinqing Liu, and Yuqing He. Real-time riverbank line detection for USV system. In *2019 IEEE International Conference on Mechatronics and Automation (ICMA)*, pages 2546–2551, 2019.

[10] John R. Gardner, Xiao Yang, Simon N. Topp, Matthew R. V. Ross, Elizabeth H. Altenau, and Tamlin M. Pavelsky. The color of rivers. *Geophysical Research Letters*, 48(1), 2021.

[11] Anirudha Ghosh, A. Sufian, Farhana Sultana, Amlan Chakrabarti, and Debashis De. *Fundamental Concepts of Convolutional Neural Network*, pages 519–567. 2020.

[12] P.H. Gleick. *Water in Crisis: A Guide to the World's Fresh Water Resources*. Oxford University Press, 1993.

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

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

[15] Thien Huynh-The, Cam-Hao Hua, Jae-Woo Kim, Seung-Hwan Kim, and Dong-Seong Kim. Exploiting a low-cost CNN with skip connection for robust automatic modulation classification. pages 1–6, 2020.

[16] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, page 448–456. JMLR.org, 2015.

[17] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2014.

[18] Reeve Lambert, Jalil Chavez-Galaviz, Jianwen Li, and Nina Mahmoudian. ROSEBUD: A deep fluvial segmentation dataset for monocular vision-based river navigation and obstacle avoidance. *Sensors*, 22(13), 2022.

[19] Gui Ling, Feiyang Suo, Zhen Lin, Yanjun Li, and Ji Xiang. Real-time water area segmentation for USV using enhanced U-Net. In *2020 Chinese Automation Congress (CAC)*, pages 2533–2538, 2020.

[20] Alexey Merzlyakov and Steve Macenski. A comparison of modern general-purpose visual SLAM approaches. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9190–9197, 2021.

[21] Hua Ni, Wei Guan, and Chaoyang Wu. USV obstacle avoidance based on improved watershed and VFH method. In *2020 11th International Conference on Prognostics and System Health Management (PHM-2020 Jinan)*, pages 543–546, 2020.

[22] D. Nister. An efficient solution to the five-point relative pose problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2004.

[23] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, 2015.

[24] Jack Rowley. Autonomous unmanned surface vehicles (USV): A paradigm shift for harbor security and underwater bathymetric imaging. In *OCEANS 2018 MTS/IEEE Charleston*, 2018.

[25] Felippe Roza, Vinicius Silva, Patrick Pereira, and Douglas Bertol. Modular robot used as a beach cleaner. *Ingeniare. Revista chilena de ingeniería*, 24, 2016.

[26] Soledad Rubio and Dolores Pérez-Bendito. Recent advances in environmental analysis. *Analytical chemistry*, 81, 2009.

[27] Pierre Soille and Luc M. Vincent. Determining watersheds in digital pictures via flooding simulations. In Murat Kunt, editor, *Visual Communications and Image Processing '90: Fifth in a Series*, volume 1360, 1990.

[28] Lorenzo Steccanella, Domenico Bloisi, Alberto Castellini, and Alessando Farinelli. Waterline and obstacle detection in images from low-cost autonomous boats for environmental monitoring. *Robotics and Autonomous Systems*, 124, 2019.

[29] Shinya Sumikura, Mikiya Shibuya, and Ken Sakurada. OpenVSLAM: A versatile visual SLAM framework. 2019.

[30] Nikola Tesla. Method of and apparatus for controlling mechanism of moving vessels or vehicles, Nov 1898.

[31] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11), 2000.

[32] Tao Zheng, Zhizhao Duan, Jin Wang, Guodong Lu, Shengjie Li, and Zhiyong Yu. Research on distance transform and neural network Lidar information sampling classification-based semantic segmentation of 2D indoor room maps. *Sensors*, 2021.

Appendix A

3D Designs

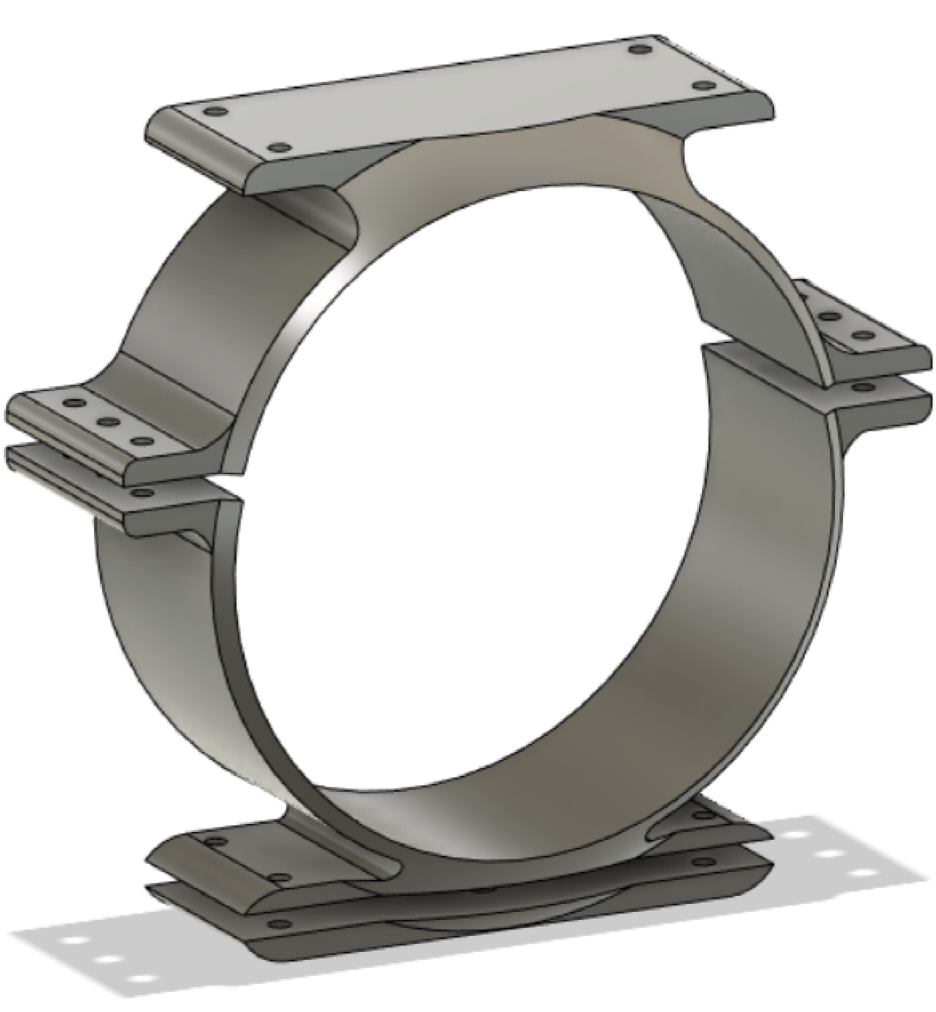


Figure A.1: 3D Design of the Thruster Support

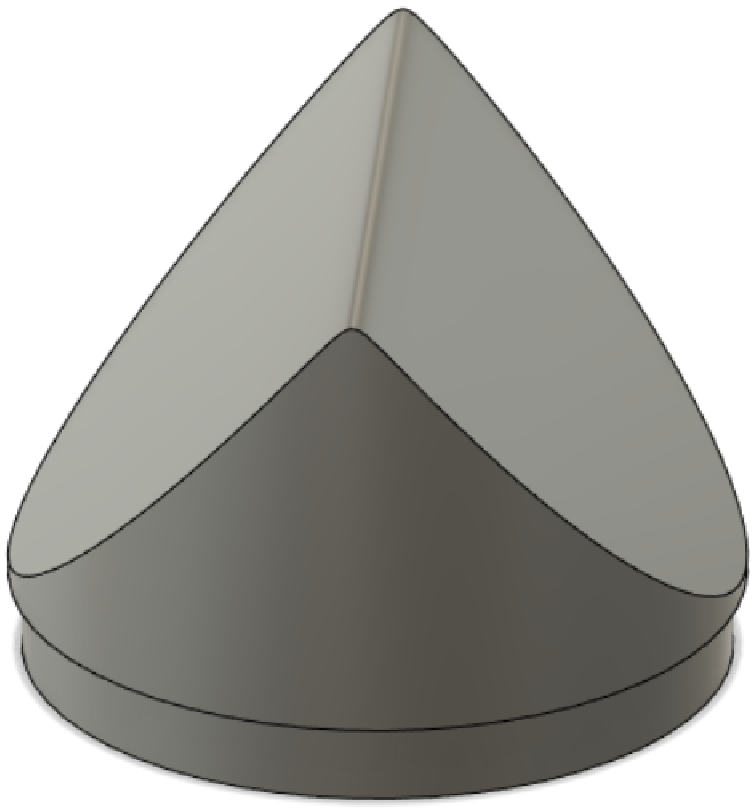


Figure A.2: 3D Design of the Floater Tip

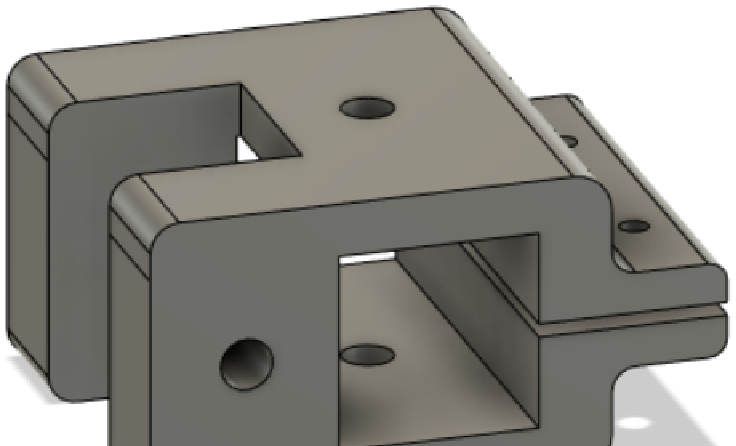


Figure A.3: 3D Design of the Camera Support