Waterline and Obstacle Detection in Images from Low-cost Autonomous Boats for Environmental Monitoring

L. Steccanella[a], D. D. Bloisi[b], A. Castellini[c], A. Farinelli[c]

[a]*DTIC, Universitat Pompeu Fabra C/ Roc Boronat 138 - 08018 Barcelona, Spain*
[b]*Dept. of Mathematics, Computer Science, and Economics, University of Basilicata, Viale dell'Ateneo Lucano, 10 - 85100 Potenza, Italy*
[c]*Dept. of Computer Science, University of Verona, Strada le Grazie 15 - 37134 Verona, Italy*

Abstract

Waterline detection from images taken by cameras mounted on low-cost autonomous surface vehicles (ASVs) is a key process for obtaining a fast obstacle detection. Achieving an accurate waterline prediction is difficult due to the instability of the ASV on which the camera is mounted and the presence of reflections, illumination changes, and waves. In this work, we present a method for waterline and obstacle detection designed for low-cost ASVs employed in environmental monitoring. The proposed approach is made of two steps: 1) a pixel-wise segmentation of the current image is used to generate a binary mask separating water and non-water regions, 2) the mask is analysed to infer the position of the waterline, which in turn is used for detecting obstacles. Experiments were carried out on two publicly available datasets containing floating obstacles such as buoys, sailing and motor boats, and swans moving near the ASV. Quantitative results show the effectiveness of the proposed approach with 98.8% pixel-wise segmentation accuracy running at 10 frames per second on an embedded GPU board.

Keywords: Water detection, Autonomous surface vessels, Robotic boats,

*Corresponding author
Email addresses: `lorenzo.steccanella@upf.edu` (L. Steccanella), `domenico.bloisi@unibas.it` (D. D. Bloisi), `alberto.castellini@univr.it` (A. Castellini), `alessandro.farinelli@univr.it` (A. Farinelli)

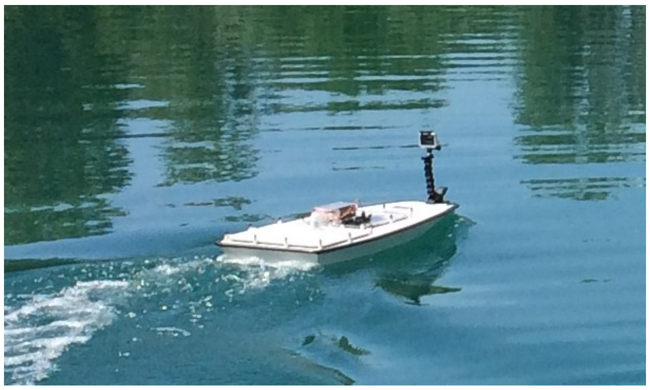


Figure 1: INTCATCH project uses Platypus Lutra boats, about $1m$ long and $0.5m$ wide. A camera has been mounted on the front of the ASV.

Robot vision, Water quality monitoring

1. Introduction

The water quality monitoring process costs every year more than 1 billion EUR at the European Union level [1]. In particular, analyses of pollution in large rivers or lakes supplying drinking water are in the range of 150,000 to 400,000 EUR. The current approach, based on field sample and laboratory analysis, is unable to assess temporal and spatial variation in the contaminants of concern. This means that, in case of accident, there is the possibility of taking inadequate and late decisions for mitigating the pollution impacts.

The use of autonomous surface vehicles (ASVs) for persistent large-scale monitoring of aquatic environments is a valid and efficient alternative to the traditional manual sampling approach [2]. ASVs are capable of undertaking long-endurance missions and carrying multiple sensors (e.g., for measuring electrical conductivity and dissolved oxygen) to obtain water quality indicators [3]. There exist commercial ASVs specifically developed for water quality monitoring. An example is the Lutra mono hull boat produced by Platypus[1], shown

[1]`http://senseplatypus.com`

in Fig. 1. Lutra boats are used in the EU-funded Horizon 2020 project INTCATCH[2], which aims to develop a new paradigm in the monitoring of river and lake water quality, by bringing together, validating, and exploiting a range of innovative tools into a single efficient and user-friendly model.

To achieve true autonomous navigation, an ASV must sense its environment and avoid possible collisions. In this paper, we propose to use vision-based sensing to reach this goal, focusing on the domain of small, low-cost ASVs where sensors commonly utilized for localization purposes (e.g., LiDARs) and powerful processing units cannot be used due to price caps.

The contribution of this work is two-fold: *i)* we present a pixel-wise deep learning based method able to segment images captured by cameras on low-cost ASV into water/non-water regions, and *ii)* we describe an algorithm for extracting the horizon line (called *waterline* in this context) from the segmented images and for detecting potential floating obstacles. It is worth noticing that, the relationship between waterline and obstacles is heavily dependent on the pitch and roll angles of the ASV, which are only roughly measured by cheap gyroscope and accelerometer sensors available in low-cost boats. Our goal is to use the waterline to reduce the search space for the detection of potential obstacles and to deal with the pitch and roll movements, which can cause misdetections, by dynamically tracking over-time the obstacle positions. In particular, we use Convolutional Neural Networks (CNNs) for image segmentation, edge detection and RANSAC for waterline extraction, and a multi-object tracker based on cost computation.

Although CNNs were previously introduced in other mobile robotics domains (e.g., indoor wheeled and quadrotors), and ASV obstacle detection techniques were employed with classical computer vision methods (e.g., optical flow), the two approaches have never been combined yet.

We provide the complete source code of our method (see Section 6) and the dataset of images and videos created for testing (see the INTCATCH Vision

[2]`http://www.intcatch.eu`

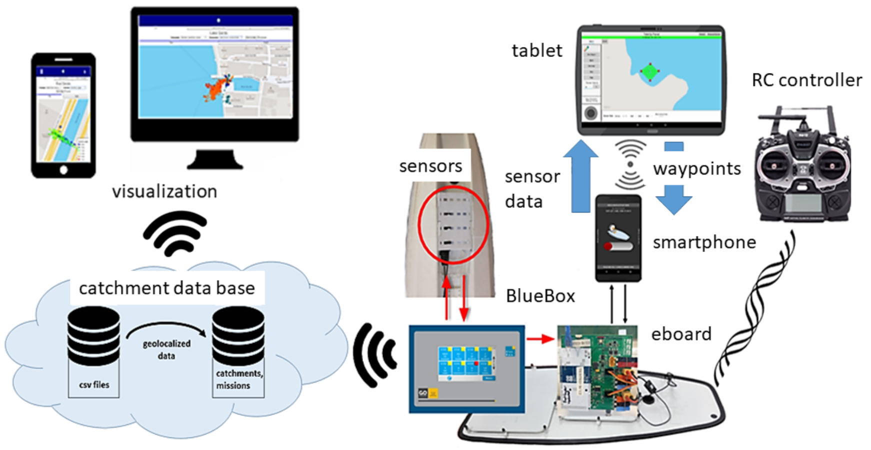


Figure 2: The INTCATCH autonomous boat system. The boat uses GPS data to navigate autonomously and can mount different types of sensors on the underside of the boat. Data are sent to the cloud and can be accessed through smartphones and PCs.

Dataset[3]). This dataset contains annotated data that can be used for developing supervised approaches for object detection.

The remainder of the paper is structured as follows. An overview of the INTCATCH system is given in Section 2 and related work is discussed in Section 3. The proposed segmentation algorithm is presented in Section 4, while the waterline and obstacle detection process is described in Section 5. Experimental results are shown in Sections 6. Finally, conclusions are drawn in Section 7.

2. System Overview

The realization of an easy-to-use system that can be used by non-expert users is a key aspect of the INTCATCH project. A general overview of the INTCATCH autonomous boat system is shown in Fig. 2. Its main components are briefly described below.

Boat. Two different models of the Lutra boats were acquired by the INTCATCH consortium to cover different deployment scenarios: 1) Prop boat, hav-

[3]`goo.gl/Kxt6HP`

ing in-water propellers to be deployed in as little as 25 cm of water; 2) Airboat, with a covered fan deployable in as little as 15 cm of water.

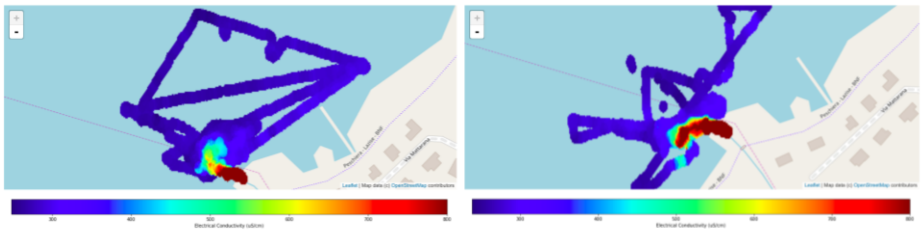
Controls. The user can interact with the boat using a tablet. A specific Android Control app has been developed for allowing non-expert users to create mission plans for the boat using a GUI. For example, it is possible to create a mission path by selecting a set of waypoints on a map with few clicks.

Proprioceptive sensors. A smartphone is on-board in order to provide the information needed for autonomous navigation (i.e., GPS, compass, and gyroscope data). The phone is placed inside the boat, providing location and orientation information to the autonomous navigation system.

Water quality sensors. A sensor control unit (Go-Sys BlueBox) is connected to an Arduino e-board and transmits information coming from the sensors measuring the water quality indicators including temperature, electrical conductivity, dissolved oxygen, and pH.

GPS based autonomous navigation. The current position of the boat, measured through the GPS data, is constantly compared with the desired position defined in the mission path. If the boat diverges from the expected route (e.g., due to water flow), the software running on the smartphone sends commands to the motors in order to correct the route. This behaviour is defined as "line following". Planning strategies using prior knowledge for battery management during autonomous navigation missions are described in [4, 5].

Homing. A core requirement for a safe deployment is recovering from a "runaway" condition, where the boat has navigated out of wireless communication range and is navigating further away from the operator. The boat periodically (every 0.1 seconds) "drops a breadcrumb", saving its current location in a hashmap indexed by the GPS location of the boat that is rounded to the meter. This essentially creates a regular grid where breadcrumbs are spaced by 1 meter in the four directions. The A* path planning algorithm [6] is then run on this structure considering every breadcrumb connected to its eight neighbors. The A* algorithm finds a sequence of waypoints that starts from the breadcrumb that is closer to the boat's current location and ends at the breadcrumb that is



Leaflet | Map data (c) OpenStreetMap contributors
300
400
500
600
700
800
Electrical Conductivity (uS/cm)

Figure 3: Electrical conductivity values measured on two different days in Lake Garda. For increased visual clarity at low levels, all values above 800 $\mu S/cm$ are the same shade of red.

closer to the home location.

Data storage and visualization. The BlueBox sends geolocalized data over the Internet to a catchment database called WAIS (Water Information System). The INTCATCH web app retrieves data about catchments from the WAIS and visualizes them. The web app can be accessed by different devices, like smartphones or desktop PC. An example of data visualization is shown in Fig. 3, where an interesting situation has been registered thanks to the sensors mounted on the boat. Two deployments were carried out on two different days, showing a change in the distribution of the electrical conductivity values. The two plumes are different, due to the changes flow of the water. A YouTube video showing monitoring activities carried out on Lake Garda can be seen at `https://youtu.be/yAy8Bl3UrO0`. Activity recognition on sensor data acquired by the drones has been performed using clustering and subspace clustering techniques [7, 8].

3. Related Work

Monocular vision-based obstacle detection for ASVs has previously received some interest. The approaches presented in the literature are mainly unsupervised ones.

3.1. Unsupervised Methods

El-Gaaly et al. [9] utilize sparse optical flow, reflection rejection, and an occupancy grid overlaid on the image. This process produces a significant number of false positives when the water surface is disturbed. Sadhu et al. [10] use grayscale histograms of pixel neighborhoods as a descriptor of texture and saliency to detect logs floating on the surface of water. However, their method ignores the shoreline. Wei and Zhang [11] present a waterline detection method based on structure extraction and texture analysis. Their algorithm combines local binary patterns (LBPs) and the gray level co-occurrence matrix (GLCM) to obtain a measure of the intensity contrast of neighboring pixels as well as the distance and orientation between them. However, their approach fails when both the trees and the shade of the trees on the bank are dark.

Paccaud and Barry [12] describe a waterline based obstacle-detection system for lake-deployed ASVs mounting a camera with a low viewing angle to the water. For detecting the horizon, they use a statistical method (i.e., the horizon is defined as the line that best separates the sky and water and minimizes their relative intra-class variance) and an edge analysis (the horizon is assumed to be the most salient line in the frame). Data from an Inertial Navigation System are used to track the cameras attitude in real time. This information is used then to improve horizon detection. Possible obstacles are detected using a gradient-based image processing algorithm. The system runs at 4 frames per second (FPS) and it has been tested on images from a publicly available dataset recorded in Lake Geneva (Switzerland).

3.2. Supervised Approaches

The dynamic nature of water is capable of producing mirror-accurate reflections of the environment and a turbulent, erratically textured surface, which are difficult to handle with classical unsupervised approaches. Recent advancements in deep learning methods for computer vision, based in particular on CNNs, show promise for the segmentation of images captured by ASV into semantic elements such as obstacles and the water that surrounds them. In this work, we use CNNs to separate pixels related to water from pixels related to non-water (e.g., sky, coast, and floating objects) obtaining a mapping between image pixels and the two classes *water/non-water*.

CNNs are a type of deep, feed-forward Artificial Neural Networks (ANNs). Since ANNs are able to approximate any continuous functional mapping, they can be employed when the form of the required function is unknown [13, 14]. CNNs have shown impressive performance on image classification [15] and object detection [16]. Segnet [17] is an example of deep fully convolutional neural network architecture for semantic *pixel-wise* segmentation. Its segmentation engine relies on an encoding-decoding scheme and it can be employed for scene understanding applications. Multiple mobile robotics domains have started to incorporate CNNs into obstacle detection and navigation. Giusti et al. [18] detect forest paths for a quadrotor to navigate. Chakravarty et al. [19] trained a CNN to produce depth maps from a single image in indoor environments.

To the best of our knowledge, CNNs have never been used for ASV applications and, in particular, for waterline detection. Our image segmentation algorithm is based on the U-net architecture. U-net [20] is an encoder-decoder type of network for pixel-wise segmentation. In U-net, the receptive fields after convolution are concatenated with the receptive fields in an up-convolution process, allowing the network to use original features in addition to features after transpose convolution. Every step in the expansive path consists of *i)* an upsampling of the feature map followed by a 2×2 transpose convolution that halves the number of feature channels, *ii)* a concatenation with the corresponding cropped feature map from the contracting path, and *iii)* two 3×3 convolutions, each followed by a rectified linear unit (ReLU). This results in overall better performance than a network that has access to only features after up-convolution.

4. Water/non-water Classification

Raw images coming from the camera mounted on the boat are used as input for a Fully Convolutional Neural Network that classifies pixels as water or non-water ones. In the following we describe the network architecture, the dataset used for training, and provide details on the learning process.

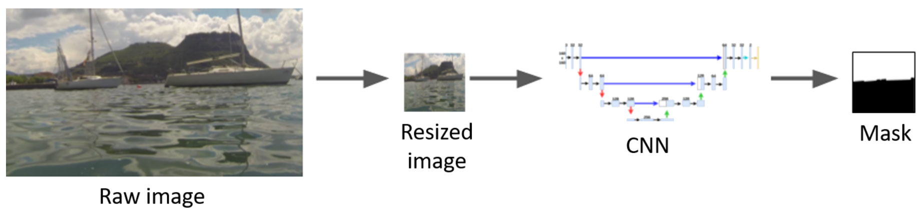


Figure 4: CNN feedforward prediction. Raw captured image is resized and fed to trained neural network, producing the class mask.

4.1. CNN Architecture

A U-net based architecture is used for the segmentation process [20]. The decision of choosing the U-net architecture has been taken for three main reasons:

1. U-net does not have any dense layer. This means that there is no restriction on the size of the input image. Convolutional layers can extract kernel features from any input shape adapting padding automatically. On the other hand, the dense layers need to have fixed size/length input by their definition.
2. The training stage can be carried out even if only a limited amount of training data is available.
3. The receptive fields in the encoding and decoding stages are concatenated. This allows the network to consider the feature after upconvolution together with the original ones.

Fig. 4 shows the pipeline for generating the class mask. The input image is downsampled to obtain a 160×160 resized image. The encoding stage is needed to create a 256 feature vector, while the decoding stage is needed to obtain the predicted mask at 160×160 pixels. The encoding stage is made of eight 3×3 convolutional layers, and by three 2×2 max pooling operations with stride 2. In particular, there is a repeated application of two unpadded convolutions, each followed by a batch normalization (BN) layer, a ReLU, and a max pooling operation.

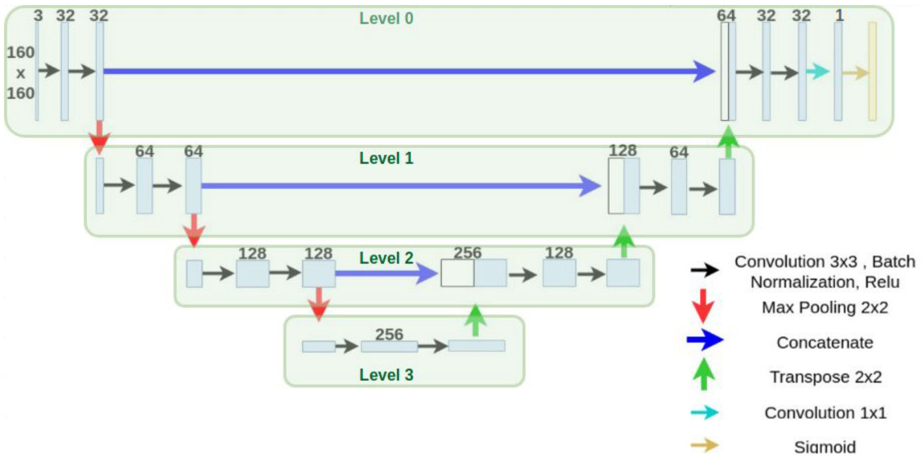


Figure 5: Architecture of the Full BN 160×160 network.

As a difference with respect to the original U-net architecture, our encoding stage has four levels instead of five and we extract at each level a number of features that is the half of the original U-net. The complete architecture of the proposed segmentation net, denoted as *Full BN 160×160*, is shown in Fig. 5. The expansive path (see the right side of Fig. 5) is made of six 3×3 convolutional layers and by three 2×2 transpose layers. There is a repeated application of two padded convolutions, each followed by a BN layer and a ReLU. The 32 channels map extracted in the last layer of the network are then projected in a single channel space with a sigmoid activation function that outputs the probability of belonging to the class "water".

Regularization. To improve the performance of the neural network, we explored the use of BN [21] as a regularizer for the training signal. BN is applied over a mini-batch gradient descent optimization technique (see Section 4.3) where small batches of data are sampled from the whole dataset D and the network parameters are updated on the basis of the loss values computed on these batches. For every batch B containing m values, sampled from D, BN performs a zero mean and unit variance regularization to each dimension of each hidden layer. The mini-batch mean is $\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i^{(k)}$, the mini-batch variance is $\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}\left(x_i^{(k)} - \mu_B\right)^2$, the normalization of a general activation $x_i^{(k)}$ is computed as $\overline{x}_i^{(k)} = \frac{x_i^{(k)} - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$. However, the simple standardization of each input of a layer may limit the expressiveness of the network. To address this problem, we also learn, for each activation $x^{(k)}$, two parameters $\gamma^{(k)}$ and $\beta^{(k)}$ by which the normalized value is scaled and shifted as $y_i^{(k)} = \gamma^{(k)} \overline{x}_i^{(k)} + \beta^{(k)}$. BN is performed on each hidden layer, thus enabling the use of higher learning rates that greatly accelerate the learning process.

Increasing the computational speed. When detecting obstacles while moving at full speed, processing time is a crucial aspect. This motivates the pursuit of scalable methods that can be deployed with limited processing power. With the above described Full BN 160×160 architecture, we are able to obtain a computational speed of about 9 FPS on an embedded board (see Section 6 for the details about the used hardware). In order to explore the trade-off between accuracy and processing speed, we trained and tested other network configurations which are denoted as Half-Conv BN 160×160, Full mobile-net-v1-layer 160×160, and Full mobile-net-v2-layer 160×160.

Half-Conv BN 160×160 is an architecture where the number of convolutional layers applied at each encoding/decoding level is reduced by a factor of 2 with respect to Full BN 160×160. By reducing the convolutional layers to a single layer per encoding/decoding level, it is possible to obtain a faster computation in terms of FPS at the cost of decreased accuracy. Furthermore, it is possible to skip BN to obtain an increase in speed, with a decrease in accuracy also in this case. We refer to the two versions of the net without BN as *Full 160×160* and *Half-Conv 160×160*, respectively.

The architecture denoted as *Full mobile-net-v1-layer 160×160* derives from MobileNet [22], where traditional convolutional layers from level 1 to 4 are replaced by depthwise separable convolutions (DSCs), i.e., depthwise convolution followed by pointwise convolution. Fig. 6 shows a comparison between the traditional convolutional layer with BN and ReLU and the MobileNet_v1 approach using DSCs with depthwise and pointwise layers followed by BN and ReLU.

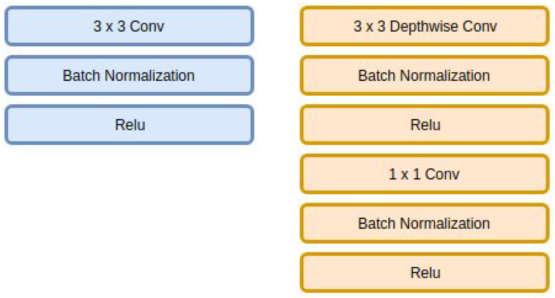


Figure 6: Left: Traditional convolutional layer with BN and ReLU. Right: MobileNet_v1 Depthwise Separable convolutions with depthwise and pointwise layers followed by BN and ReLU. Image adapted from [22].

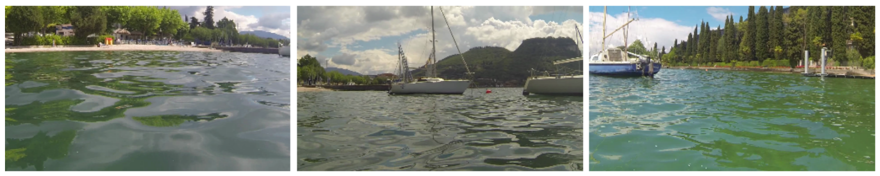


Figure 7: Three frames captured at Lake Garda in Italy, which is one of the sites of the INTCATCH project. We used images from 8 different videos for training and testing.

4.2. Dataset

In this study, we consider images coming from the INTCATCH Vision Dataset and the Obstacle Detection Data Sets published by Paccaud and Barry[4]. INTCATCH Vision Dataset is a repository that stores visual and sensor data of the INTCATCH project. The data set currently contains 22 videos collected on lake and river scenarios. All the image sequences were collected using a GoPro Hero 3 Black camera mounted on the bow of the boat. Some sequences have been collected at 60 FPS with 1920×1080 wide angle resolution and other at 25 FPS with 320×240 narrow angle resolution. This has been done to create a dataset containing a wider range of data. Here, we consider 8 videos of Lake Garda (Italy), captured at different times of the day and in different locations of the lake (see Fig. 7). In addition to the images coming from Lake Garda, we

[4]`https://osf.io/edq4b/`

Figure 8: a) Frame from a video captured on River Ter. b) Frame from the sequence DS8

use also images coming from River Ter in Spain (see Fig. 8a).

Obstacle Detection Data Sets is a collection of image sequences captured on Lake Geneva (Switzerland). It is made of 14 different datasets and each dataset includes a file with measured attitude (roll and pitch data). The sequences are useful for testing obstacle detection algorithms. Fig. 8b shows a frame from sequence DS8.

Annotation. Images were annotated using a custom tool[5] created for the task. The tool takes advantage of super-pixel segmentation to give hints to the user that needs to select the areas belonging to the water class. To perform super-pixel segmentation, we used the SLIC algorithm [23]. This algorithm performs K-means in the 5D space of color information and image location. The mask generated using the super-pixel segmentation have been further refined manually to obtain the final labelled images.

For training, we labeled 400 images from:

- 6 videos collected on Lake Garda, namely files *lakegarda-may-9-prop-X* in the dataset, where $X \in \{1, 2, 3, 4, 5, 6\}$ and the term "prop" refers to the use of the boat with submerged propellers.
- 1 videos captured on River Ter, i.e., *ter1.avi*.
- 1 image sequence from the Obstacle Detection Data Sets, i.e., DS4.

[5]`https://github.com/lorenzosteccanella/PixelWiseLabelTool`

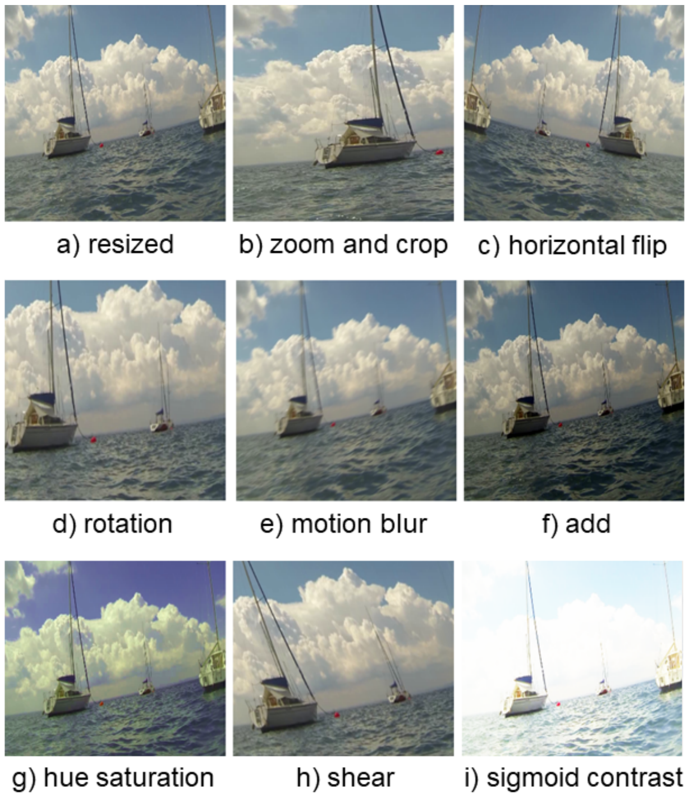


Figure 9: Data augmentation.

For testing, we labeled 115 images taken from the sequences *lakegarda-may-9-prop-7*, *lakegarda-may-9-prop-8*, *DS7*, and *DS8* from the Obstacle Detection Data Sets (see Section 6 for a discussion about the segmentation results).

Data augmentation. Working with a data set of limited size presents a problem related to overfitting: Models trained with a small amount of data can have a limited generalization capacity. Data augmentation is a common practice to handle training on small data sets [24]. In this work, we have performed augmentation on the training data to create a larger data set by using zooming and cropping, flipping, blurring, rotation, and image enhancing. Fig. 9 shows some results of the data augmentation process.

Water currents produce significant rolling (i.e., rotation) effects on the ASV.

Since rolling motion is captured in the raw video footage, this effect must be taken into account to maximize the segmentation accuracy, so we augment the data set with rotations within -20 to 20 degrees (see Fig. 9d). Other key transformations are horizontal flipping (Fig. 9c), shearing (Fig. 9h), and zooming (Fig. 9b). Motion blur (Fig. 9e) was used to simulate the effect of rapid movements of the boat.

The color range of images was augmented by considering variations in the hue, saturation, and value (brightness) channels. It is important to note that it is extremely difficult to predict the hue of water since multiple factors contribute to the surface color of water bodies, including the depth of the water, the amount and type of sediment in the water, the color of the sky reflecting on the water, and also the water movement. In particular, pixel affected by sky reflections can cover the full hue spectrum and present low saturation values and high brightness values [25]. To simulate different light conditions, we augmented the images using two techniques:

1. Add a value to each channel (Fig. 9g). A simple way to reproduce different light condition is to add of a random value to each channel of the image using the formula below.

   $$O(i,j) = I(i,j) + c \tag{1}$$

   where $O(i,j)$ is the output image, $I(i,j)$ is the input image, c is the additive factor.

2. Sigmoid contrast (Fig. 9i): this transformation applies a sigmoid function as a mask to the input image [26] and can be formalized as:

   $$O(i,j) = 1/(1 + e^{(g*(c-I(i,j)))}) \tag{2}$$

   where $O(i,j)$ is the output image, $I(i,j)$ is the input image, g is the gain factor, c is the cutoff factor. We randomly sampled g in the range $[9, 11]$ and c in the range $[0, 0, 70]$ by assessing the effects on the images.

The final augmented training dataset becomes 10 times bigger than the original training set. All the images obtained through the data augmentation process were resized to 160×160 pixels.

4.3. Training

The training step has been performed taking advantage of mini-batch gradient descent. Mini-batch gradient descent performs an update of weights θ for each mini-batch of m samples, as in [27]:

$$\theta = \theta - \alpha \cdot \nabla J(\theta; x_{(z:z+m)}; y_{(z:z+m)}). \tag{3}$$

where α is the learning rate, x is the input image, y the labelled mask, z indicates the mini-batch subset sampled from the training set, ∇ is the gradient function and J is the cost function computed as:

$$J(\theta, x_{(z:z+m)}, y_{(z:z+m)}) = \frac{1}{m} \cdot \sum_{z=0}^{m} J(\theta, x_{(z)}, y_{(z)}) \tag{4}$$

We use an Adam optimizer [28] with batch size of 32 images. The Adam optimizer performs gradient descent with momentum, involving a weighted average of gradients. The hyper parameters chosen for training our net are: learning rate $\alpha = 10^{-4}$, decay rate for first and second moment estimates $B_1 = 0.9$ and $B_2 = 0.999$, respectively. To prevent overfitting, the training was performed over 100 epochs using early stopping with patience 20 based on the loss over the validation set [29].

We compared different loss functions, i.e., binary cross entropy (used in the original U-net formulation [20]), mean absolute error, mean squared error, and Dice (see Fig. 10). The comparison, performed using a Dice/F1 metrics, showed that the Dice loss (see Eq. 5) is the better choice in our application since it reached the highest rate. The Dice Similarity Coefficient (DSC) loss function is defined as [30]:

$$J = \mathcal{L}_{DSC} = 1 - \frac{2\sum_i^N p_i g_i}{\sum_i^N p_i + \sum_i^N g_i} \tag{5}$$

where $p_i, g_i \in [0, 1]$ represent the continuous values of the sigmoid prediction map and the ground truth, respectively, at each pixel i. In our application,

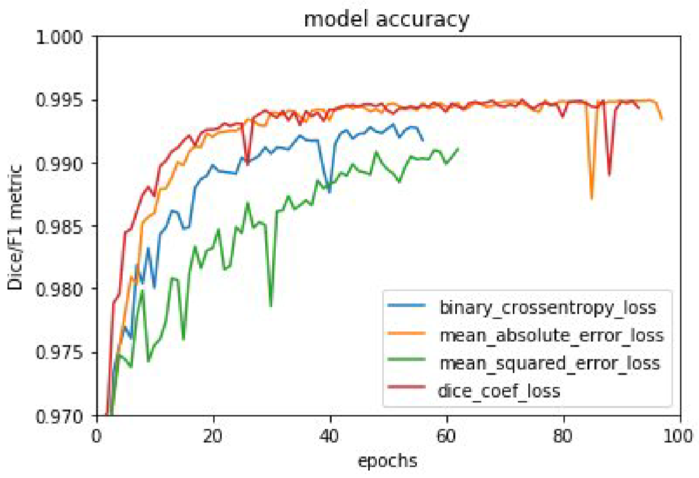


Figure 10: Comparison of different losses for training the network Full BN 160x160. The results are performed on validation set at each epoch. The training is performed with an early stopping patience of 20 epochs

DSC loss function obtains better results thanks to its capability of mitigating the class imbalance problem, i.e., pixel belonging to class water are usually more than other pixels.

5. Waterline Computation and Obstacle Detection

Once the segmentation mask for water/non-water is ready, two further steps are performed, namely, waterline computation and obstacle detection. Details about these two steps are provided below.

5.1. Waterline Computation

The binary mask produced by the CNN has value 1 for pixels belonging to water and 0 for other pixels. A median filter is applied to it for reducing narrow protrusions (such as sailboat masts). Edge detection is then used to isolate the pixels on the contour between the two classes. To do it, we looked at the points where the pixels change from 1 to 0. The probabilistic Hough transform with line primitive is applied to filter out pixels that may be part

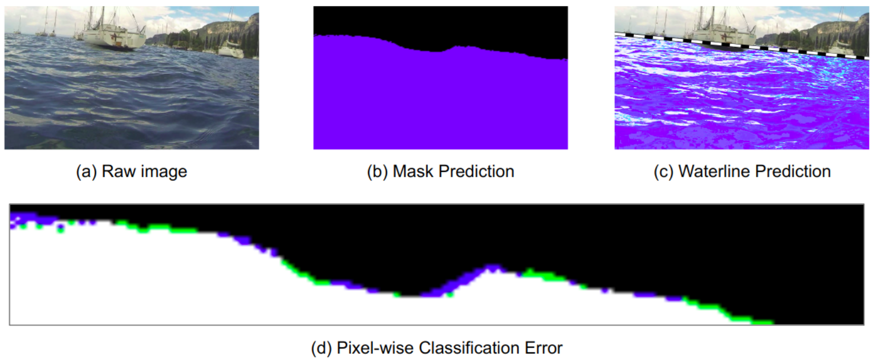


Figure 11: Example classification and waterline result. In (d), black is true negative, white is true positive, green is false positive, and purple is false negative. The image is zoomed to the region around the waterline.

of shorter, closed contours that appear below the horizon line (a small buoy below the horizon line, for example). Only the pixels that compose a long edge are retained. Specifically, we used a threshold equal to 20 for intersections in Hough grid cell, the minimum number of pixel making up a line equal to 20, and a maximum gap in pixel between connectable line segments equal to 5.

Finally, linear regression is performed utilizing RANSAC [31] to further reduce the influence of sharp changes in the contour. Fig. 11 shows an example of waterline detection achieved by the proposed algorithm. The raw image (Fig. 11a) is segmented by the CNN to generate the mask of water/non-water pixels (Fig. 11b). The waterline is then inferred from the mask using a linear regression model (Fig. 11c). In the bottom (Fig. 11d) the pixel-wise classification error is displayed.

5.2. Obstacle Detection

Fig. 12 shows the four steps needed to detect the objects laying under the waterline. The information about the current position of the waterline (see Fig. 12a) is used to modify the current frame by erasing all the pixels located over the waterline (Fig. 12b). Then, the blobs formed by non-water pixels located below the waterline are extracted (Fig. 12c) to obtain the final bounding

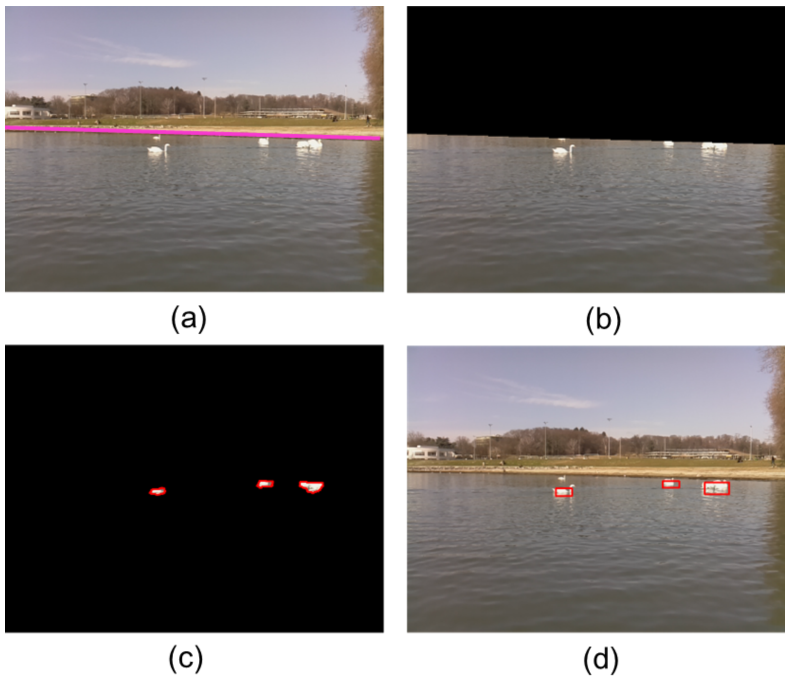


Figure 12: Obstacle detection in the current frame. a) Waterline prediction. b) Filtered image. c) Blob extraction. d) Bounding boxes around detected objects.

boxes of the objects in front of the boat (Fig. 12d). Details on each step are provided below.

Image filtering. The first two steps require to filter the original frame from the camera to reduce the search space where finding potential obstacles in front of the boat. The original color frame is transformed to grayscale and pixels are removed starting from the top location in each column down to the waterline. Then, the pixels labelled as water are also removed obtaining a filtered grayscale image with few non-zero pixels.

Blob extraction. The filtered image obtained with the two steps above is used to extract the blobs with area greater than 20 pixels. This threshold has been set experimentally and can be adapted to the specific application scenario. The blobs are computed using the OpenCV function *findContours*, while the bounding boxes are created using the OpenCV function

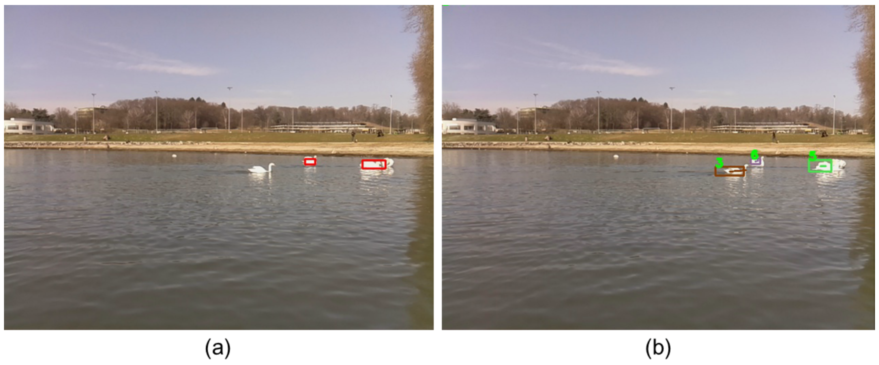


Figure 13: Object tracking example. a) Presence of a false negative in the detection step. b) The tracking algorithm keep track of the object using previous correct detections.

boundingRect. The source code for object detection is available at `https://github.com/dbloisi/asv_obstacle_detection.git` and a video with detection results on the sequence DS8 of the Obstacle Detection Data Sets is available at `https://youtu.be/OAUtOyMkRu4`

5.3. Obstacle Tracking

Detecting possible objects in single images is not enough to provide a reliable method for avoiding obstacles. False detection and misdetection must be filtered overtime to avoid sending false alarms to the navigation system. To this end, we use a tracking algorithm to keep track of the presence of potential obstacles in front of the boat. Fig. 13 shows an example where, even if the swan in the middle of the image is not detected in that specific frame (see Fig. 13a), the tracking algorithm keeps track of it because it was correctly detected in previous frames (see Fig. 13b).

The tracking algorithm is derived from the one available at `github.com/pennisi/multitargettracking`, which has been tested in the maritime environment [32]. It is a multi-object tracker based on cost computation. Costs are calculated using the Mahalanobis distance. Association between tracks and observations is carried out using a linear assignment approach together with a Munkres algorithm. The tracker considers the tracks having their bounding

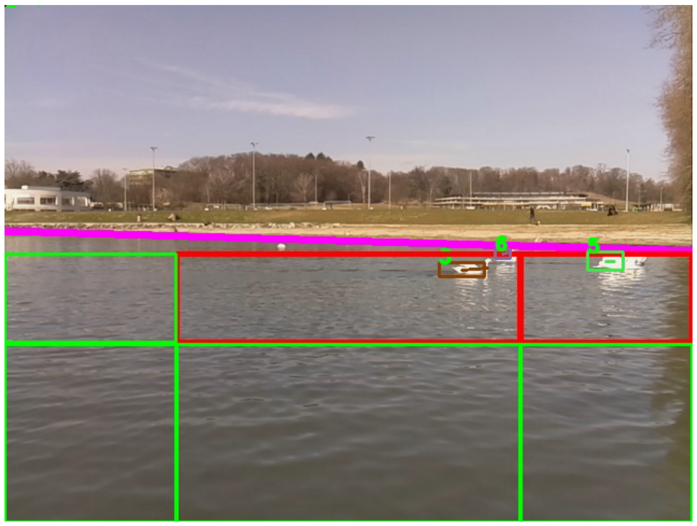


Figure 14: Alarm grid example.

boxes moving closer to each other tracks to form a group [33]. This means that collapsing tracks are represented by a single identification number, instead of tracking them separately. A group evolves considering both the estimated trajectory and the observations coming from the detector.

We use the value 5 as the number of observations after which a track is considered valid and the value 10 as the number of frames after which a track is deleted if no associations are found. The source code of the obstacle tracker is available at `https://github.com/dbloisi/asv_obstacle_tracking.git` and a video with the tracking results on the sequence DS8 of the Obstacle Detection Data Sets is available at `https://youtu.be/-CMP0nRjKJY`. It is worth noticing that, as a difference with respect to common visual tracking application (e.g., people tracking), re-identification errors can be accepted, since every obstacle should be avoided independently from its identity.

5.4. Alarm Grid

Object tracks are used to create alarms in specific locations of the image plane captured by the camera mounted on the boat. Fig. 14 shows an example. A grid of six cell is considered as the occupancy grid map in the area in front of the boat. It gives an indication of the traversability for each cell. If an object track is located inside a cell, than the cell is considered as non traversable. For example, in Fig. 14 the two grids located at the top right of the grid are labeled as non-traversable (red color), while the other cells are traversable (green color). A video with the alarm grid generated on the sequence DS8 of the Obstacle Detection Data Sets is available at `https://youtu.be/KzcSooRyhE8`.

6. Experimental Results

We tested both the capacity of our approach to segment images taken on the ASV and the accuracy of the predicted waterline calculated from the segmented image. Five different network configurations were considered.

6.1. Pixel-wise Segmentation

Experiments were carried out using a Jetson TX2 embedded GPU board with the following specifications: ARM Cortex-A57 (quad-core) @ 2GHz + NVIDIA Denver2 (dual-core) @ 2GHz, 8GB of LPDDR4 memory with a 128-bit interface, and an integrated 256-core NVIDIA Pascal GPU. Training was performed taking advantage of the Google Colab service that offers a NVIDIA Tesla K80. The training time varies according to the complexity of the network and the number of features extracted. In particular, it takes from a minimum of 4 hours for the Half-Conv 160x160 network to a maximum of 6 hours for Full BN 160x160.

The performance of the pixel-wise classification is evaluated by using precision (P), recall (R), accuracy (A), and F1-score ($F1$) metrics [34]. These measures are computed from true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) by the following formulas:

$$P = \frac{TP}{TP + FP} \tag{6}$$

$$R = \frac{TP}{TP + FN} \tag{7}$$

Table 1: Segmentation results using five different network configurations on a region of -200 to +200 pixels around the waterline.

Network	N param	Precision	Recall	Accuracy	F_1 Score	FPS - CPU	FPS - GPU
Half-Conv 160x160	948001	0.974	0.983	0.978	0.978	**5.23**	**13.27**
Half-Conv BN 160x160	950817	0.974	0.982	0.977	0.977	4.88	12.18
Full 160x160	1925601	0.994	0.980	0.988	0.986	3.51	10.53
Full BN 160x160	1931233	0.993	**0.983**	**0.989**	**0.988**	3.06	9.90
Full mobile-net-v1-layer 160x160	**422465**	**0.996**	0.972	0.986	0.983	4.25	9.85

$$A = \frac{TP + TN}{TP + FP + TN + FN} \tag{8}$$

$$F_1 = 2\frac{P \times R}{P + R} \tag{9}$$

Table 1 shows the segmentation results for the five different network configurations. To obtain a finer evaluation, results were generated considering an area of interest around the waterline, which is more prone to errors. The area consists of a slice of 200 pixel above and 200 pixel below the center point C of the waterline (see Fig. 11d). To define this area, we computed the C_y coordinate of the center of the horizontal line and we extracted an image with original width and height of 400 pixels, i.e., 200 pixel above and below C_y.

All the FPS values shown in Table 1 are calculated on average over 10 runs. Full mobile-net-v1-layer 160x160 achieves better FPS performance on the CPU than both Full 160x160 and Full BN 160x160. It obtains similar performance with respect to Half-Conv BN 160x160. We believe that this is due to the fact that DSCs are not fully supported in CudaNN.

The networks that we tested are written in Python, using functions included in the libraries OpenCV, TensorFlow, and Keras. The complete source code is available in a GitHub repository[6]. The test set used to compute the quality

[6]`https://github.com/lorenzosteccanella/Intcatch_pixelwise_segmentation`

Table 2: Results on pixel vertical distance between ground truth and predicted waterlines.

Network	**Mean Max Distance**	**Median Max Distance**	**Std. Dev.,**	**Quantile 0.25-0.75**
Half-Conv 160x160	25.03	8.0	55.15	5.0 - 14.5
Half-Conv BN 160x160	24.84	9.0	44.85	5.0 - 17.0
Full 160x160	**13.24**	9.0	**15.80**	5.0 - 15.0
Full BN 160x160	13.53	**8.0**	17.08	**6.0 - 12.0**
Full mobile-net-v1-layer 160x160	15.28	9.0	18.77	6.0 - 13.0

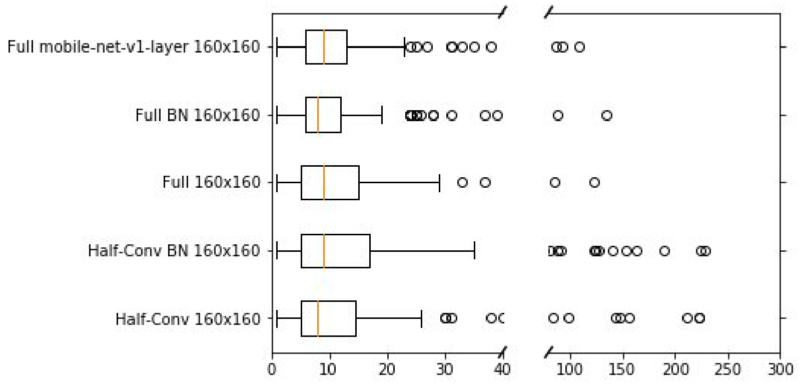


Figure 15: Boxplot of the pixel vertical distance between ground truth and predicted waterlines.

metrics for water segmentation and waterline extraction can be downloaded at `https://sites.google.com/unibas.it/waterline`

6.2. Waterline Detection

A pixel-wise evaluation of classification error does not translate directly into an error in the waterline. To evaluate the waterline detection capabilities of our approach, we calculated the maximum vertical distance in pixels between the ground truth waterline (a RANSAC fit over the ground truth mask) and the predicted waterline for each original size test image to provide a geometric measure of the error. The median and the mean values of this distance for the entire test set ar summarized in Table 2. Figure 15 shows the boxplot of the pixel vertical distance between ground truth and predicted waterlines.

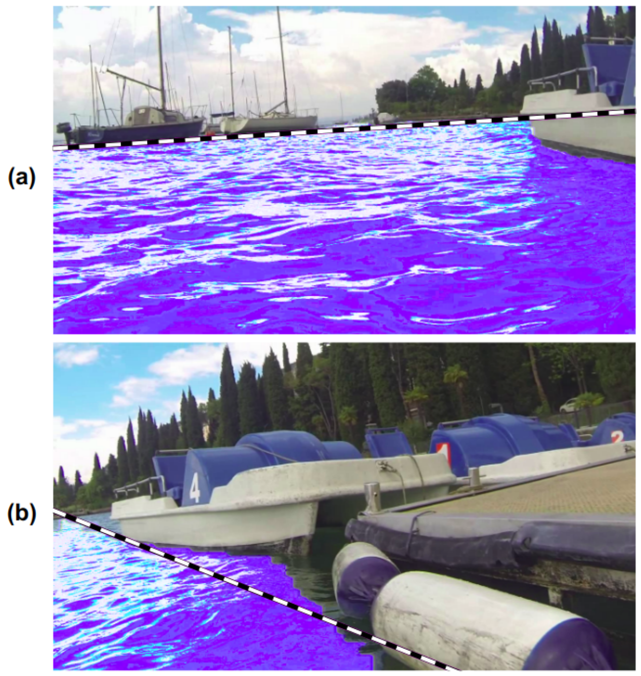


Figure 16: Two examples demonstrating the limitations of a water line. (a) The contour of a boat begins to appear and is classified correctly. RANSAC line sticks to dominant horizon line. (b) Waterline construct breaks down completely, motivating the use of a water contour.

When the boundary between water and non-water pixels is dominated by the horizon line, assuming the boundary between water and obstacles is linear is reasonable. But as the distance to obstacles decreases, this assumption begins to break down. Fig. 16 shows two examples of this issue. At a certain point, a water boundary "contour" becomes more useful.

7. Conclusions

In this paper, we have shown the use of a deep learning based method for waterline and object detection on a low-cost ASV. All the pixels in the images captured by the ASV are labelled into water and not-water classes using a CNN. A line is fit to the edges in this binary class mask to create the waterline prediction. Object laying under the waterline are detected and tracked over-time to generate alarms that can be sent to the navigation system.

A quantitative experimental evaluation has been carried out on different test sequences captured at Lake Garda in Italy, River Ter in Spain and Lake Geneva in Switzerland. The chosen application scenarios are challenging, due to the presence of large waves (compared to the dimension of the considered ASV) and a number of floating objects (e.g., buoys, swans, sailing and motorboats).

We have demonstrated the effectiveness of the proposed approach using five different segmentation network configurations, two with batch normalization, two without batch normalization, and one with Depthwise Separable convolutional layers.

The pixel-wise classification achieves good performance in all the five cases, with an accuracy ranging from 0.977 to 0.988. The use of batch normalization allows to improve the training signal and the generalization of the network on the test set. The use of Depthwise Separable convolutional layer improves the computational performance on CPU without producing a reduction in the accuracy.

The waterline detection approach provides a precision that is acceptable for small, low-speed ASV (i.e., a mean max distance ranging from 25.03 to 13.24 pixel and median max distance ranging from 8.0 to 9.0 pixels for an image of 128x720 pixels).

Future directions. This work represents a first proof of the feasibility of a deep learning approach to waterline detection and water segmentation on images coming from a small ASV. It paves the way for investigating the use of different types of network architectures. A Recurrent Neural Network over the latent space on our network can be used to track the feature over-time and this could provide improvements in accuracy and smoothness of transitions between frame detection. The network itself could provide directly the waterline attaching dense layers over the encoded latent space, and having a multiple output network.

Acknowledgment

We are grateful to Stefano Aldegheri for helping in implementing the waterline detection algorithm on the Jetson board and in generating the experimental results.

This work is partially funded by the European Union's Horizon 2020 research and innovation programme under grant agreement No 689341.

References

[1] E. P. E. C. (EPEC), Detailed assessment of the market potential, and demand for, an eu etv scheme (2013).
URL `https://publications.europa.eu/s/mNoC`

[2] M. Dunbabin, A. Grinham, Quantifying spatiotemporal greenhouse gas emissions using autonomous surface vehicles, Journal of Field Robotics 34 (1) (2017) 151–169.

[3] D. L. Codiga, A marine autonomous surface craft for long-duration, spatially explicit, multidisciplinary water column sampling in coastal and estuarine systems, Journal of Atmospheric and Oceanic Technology 32 (3) (2015) 627–641.

[4] A. Castellini, J. Blum, D. Bloisi, A. Farinelli, Intelligent battery management for aquatic drones based on task difficulty driven pomdps, in: Proceedings of the 5th Italian Workshop on Artificial Intelligence and Robotics - XVII International Conference of the Italian Association for Artificial Intelligence, AIRO@AI*IA 2018, Trento, Italy, November 22-23, 2018., 2018, pp. 24–28.

[5] A. Castellini, G. Chalkiadakis, A. Farinelli, Influence of state-variable constraints on partially observable monte carlo planning, in: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019, 2019, pp. 5540–5546.

[6] P. E. Hart, N. J. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE Transactions on Systems Science and Cybernetics 4 (2) (1968) 100–107. `doi:10.1109/TSSC.1968.300136`.

[7] A. Castellini, G. Beltrame, M. Bicego, J. Blum, M. Denitto, A. Farinelli, Unsupervised activity recognition for autonomous water drones, in: Proceedings of the 33rd Annual ACM Symposium on Applied Computing, SAC '18, ACM, New York, NY, USA, 2018, pp. 840–842.

[8] A. Castellini, F. Masillo, M. Bicego, D. Bloisi, J. Blum, A. Farinelli, S. Peigner, Subspace clustering for situation assessment in aquatic drones, in: Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC '19, ACM, New York, NY, USA, 2019, pp. 930–937.

[9] T. El-Gaaly, C. Tomaszewski, A. Valada, P. Velagapudi, B. Kannan, P. Scerri, Visual obstacle avoidance for autonomous watercraft using smartphones, in: Autonomous Robots and Multirobot Systems workshop, 2013, pp. 1–15.

[10] T. Sadhu, A. B. Albu, M. Hoeberechts, E. Wisernig, B. Wyvill, Obstacle detection for image-guided surface water navigation, in: 13th Conference on Computer and Robot Vision (CRV), 2016, pp. 45–52.

[11] Y. Wei, Z. Y., Effective waterline detection of unmanned surface vehicles based on optical images, Sensors 16 (10) (2016) 1–18.

[12] P. Paccaud, D. A. Barry, Obstacle detection for lake-deployed autonomous surface vehicles using RGB imagery, PloS one 13 (10) (2018) 1–24.

[13] K.-I. Funahashi, On the approximate realization of continuous mappings by neural networks, Neural Networks 2 (3) (1989) 183–192.

[14] A. Castellini, D. Paltrinieri, V. Manca, MP-GeneticSynth: inferring biological network regulations from time series, Bioinformatics 31 (5) (2015) 785–787.

[15] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: Proceedings of the 25th International Conference on Neural Information Processing Systems, Vol. 1 of Advances in neural information processing systems (NIPS), 2012, pp. 1097–1105.

[16] D. Erhan, C. Szegedy, A. Toshev, D. Anguelov, Scalable object detection using deep neural networks, in: Computer Vision and Pattern Recognition (CVPR), 2014, pp. 2155–2162.

[17] V. Badrinarayanan, A. Kendall, R. Cipolla, Segnet: A deep convolutional encoder-decoder architecture for image segmentation, IEEE Transactions on Pattern Analysis and Machine Intelligence.

[18] A. Giusti, J. Guzzi, D. C. Cirean, F. L. He, J. P. Rodrguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. D. Caro, D. Scaramuzza, L. M. Gambardella, A machine learning approach to visual perception of forest trails for mobile robots, IEEE Robotics and Automation Letters.

[19] P. Chakravarty, K. Kelchtermans, T. Roussel, S. Wellens, T. Tuytelaars, L. Van Eycken, Cnn-based single image obstacle avoidance on a quadrotor, in: IEEE International Conference on Robotics and Automation (ICRA), 2017, pp. 6369–6374.

[20] O. Ronneberger, P.Fischer, T. Brox, U-net: Convolutional networks for biomedical image segmentation, in: International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI), 2015, pp. 234–241.

[21] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, arXiv preprint arXiv:1502.03167.

[22] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, Mobilenets: Efficient convolutional neural networks for mobile vision applications, arXiv preprint arXiv:1704.04861.

[23] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, S. Süsstrunk, Slic superpixels compared to state-of-the-art superpixel methods, IEEE transactions on pattern analysis and machine intelligence 34 (11) (2012) 2274–2282.

[24] L. Perez, J. Wang, The effectiveness of data augmentation in image classification using deep learning, arXiv preprint arXiv:1712.04621.

[25] A. Rankin, L. Matthies, A. Huertas, Daytime water detection by fusing multiple cues for autonomous off-road navigation, Transformational Science and Technology for the Current and Future Force (2006) 177–184.

[26] N. Hassan, N. Akamatsu, A new approach for contrast enhancement using sigmoid function, The International Arab Journal of Information Technology 1 (2).

[27] X. Peng, L. Li, F. Wang, Accelerating minibatch stochastic gradient descent using typicality sampling, ArXiv abs/1903.04192.

[28] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980.

[29] R. Caruana, S. Lawrence, C. L. Giles, Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping, in: Advances in neural information processing systems (NIPS), 2001, pp. 402–408.

[30] F. Milletari, N. Navab, S.-A. Ahmadi, V-net: Fully convolutional neural networks for volumetric medical image segmentation, in: 2016 Fourth International Conference on 3D Vision (3DV), IEEE, 2016, pp. 565–571.

[31] M. A. Fischler, R. C. Bolles, Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography, Commun. ACM.

[32] M. Fiorini, A. Pennisi, D. Bloisi, Optical target recognition for drone ships, in: Proc. of the 12th International Conference on Marine Navigation and Safety of Sea Transportation (TransNav), 2017, pp. 371–375.

[33] F. Previtali, D. D. Bloisi, L. Iocchi, A distributed approach for real-time multi-camera multiple object tracking, Machine Vision and Applications 28 (2017) 421430.

[34] A. Pennisi, D. D. Bloisi, D. Nardi, A. R. Giampetruzzi, C. Mondino, A. Facchiano, Skin lesion image segmentation using delaunay triangulation for melanoma detection, Computerized Medical Imaging and Graphics 52 (2016) 89 – 103.