A Deterministic Approach to Boat and Sea Detection in Images

Alberto Zancanaro

I. Introduction

During the last years the use of machine learning and neural network has been gradually increase in the fields of computer vision and image elaboration. This is especially true for image segmentation and object searching but although the optimal result neural networks have some serious drawback, first of all the training time. Despite deterministic algorithms lack the capability to generalize like neural network, in specific case they can still provide semi-optimal solution without necessity of any training.

This work analyzed two deterministic algorithms developed to resolve the following two tasks:

- To detect boat in the photos.
- To detect sea in the photos.

The boat problem can be easily solved, if certain condition in the photos are respected. The principal techniques used for this task are edge detection and morphological operations.

The detection of the sea otherwise is more difficult, principally because the characteristics of the sea can vary a lot depending on the photo. Various approach based on pixel color and pixel similarity was used to try to overcome this obstacle.

In the first part of the report it will be illustrated the algorithm used to detect the boat, how it works, the condition to use it, its results and its problems. The second part of the reported has the same structure of the first but is is dedicated to the detection of the sea. The third part is a brief tutorial about the code, how it was structured and how it works. In the last part it will be discuss the future improvements and the knowledge acquired through the project.

The algorithms were developed in C++ language with the help of the OpenCV library. All the *function* mentioned in the following section are from OpenCV library, unless otherwise specified.

II. Boat Detection Algorithm

A. Introduction to the problem and requirement

The detection of a singular object in a photo it is not a difficult task. The main problem is infer if the object is present or not in the photo. The feasibility of this task depends on the category of the object.

If the object has a well determined structure or characteristic (i.e. plates) deterministic algorithms can understand if the object is present or no in the photo. Otherwise if the object has a great variability we must assume a priori that the object is in the photo because we have no clear rule to decide if what we detect is the object that we want.

Unfortunately the boat are in the second category of object. We can have various type of boat (rowing boat, speedboat, yacht ecc) and each one has its own characteristics. So for this task we assume that in all the photo there is a boat.

B. The algorithm

The basic idea behind the algorithm is to find the edge in the image, expand them through morphological operation and search contours in the image with the expanded edges. Once the contours are found the boat was select through a filtering of the possible candidate since each candidate it's represented by a contours.

The algorithm is divided in 5 phases:

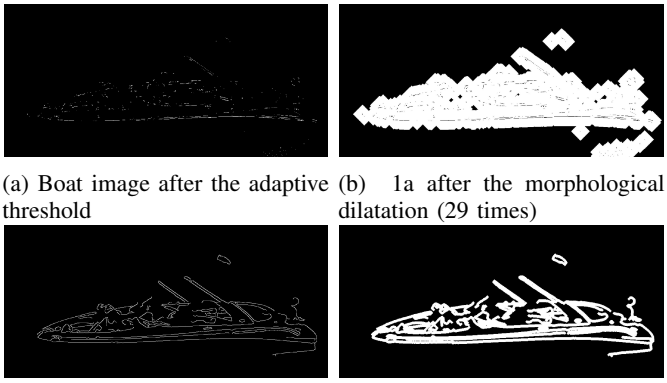
1) Filtering of the image
2) Edge detection
3) Contours detection
4) Search candidates
5) Choose best candidate

The filtering was performed with a Gaussian filter. Median filter was also tested but the Gaussian filter provides better results. The filtering was needed to remove general noise from the image. Two different kernels (3 x 3 and 11 x 11) were used, each one related to a different type of edge detection.

In the first method, related to the 3 x 3 kernel, an adaptive threshold was used and the resulting binary image has been dilated. The thresholding was performed thorough the *adaptiveThreshold()* function with an adaptive Gaussian threshold and a constant c = -9. The second approach use the famous Canny algorithm always followed by a dilation. The purpose of the dilation it is to merge the various edge in order to facilitate the detection of a single border for the boat. The morphological operation was performed through the function *morphologyEx()*. The structure used was a 3 x 3 cross. The first type of thresholding require an elevate number of dilatations (e.g. around 30) to merge the edge together. Instead for the edge image provide by the canny algorithm only a few dilation is generally sufficient. The number of dilatation for a precisely detection may vary depending of the image; good result were generally obtain with 29 dilatation for the image obtained by the thresholding and 4 dilatation for the image obtained by the Canny algorithm. Results can be seen in fig 1.

The detection of the contours was performed through the function *findContours()* that use the algorithm illustrated in **??**. Result of this operation can be seen in 2.

Subsequently, in phase 4, each contours is inscribed in a rectangle. The list of rectangle is analyzed and the contours



(a) Boat image after the adaptive threshold

(b) 1a after the morphological dilatation (29 times)

(c) Boat image after the application of the Canny algorithm

(d) 1c after the morphological dilatation (4 times)

Fig. 1: Edge of the boat detected with the two methods

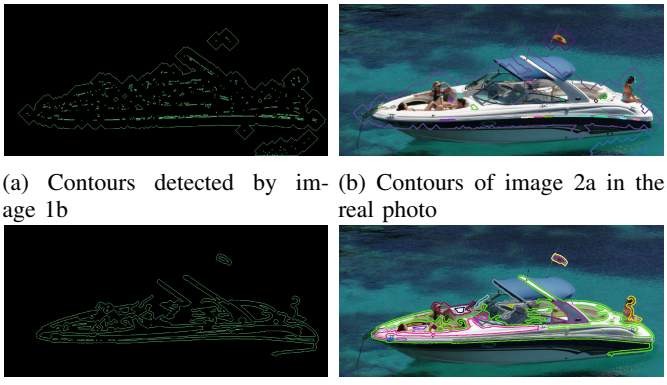
that cannot inscribe a boat are discarded. The selection is based on the geometric property of the rectangle, more precisely its area.

After this, in phase 5, the biggest contours remain was choose as a boat. This choice derived from the fact that after the merging of the edge the boat will be probably have the biggest contour in all the photo. So the biggest contour was selected as a boat. Results can be seen in figure 3a and figure 3b.

C. Results and discussion

The results of the algorithm are generally good but don't always satisfactory. In image with a lot of detail the edge detection and the subsequent dilatation lead to a contour too large. Example of this can be seen in figure 4. A possible solution is to decrease the number of dilatation but in this case the boat detection performance generally decrease in other image.

The other principal problem is the fact that there can be very different type of image with boat. This lead to the impossibility to create a deterministic algorithm that work well for every image. Every choice of parameter leads to bad results in some image.



(a) Contours detected by image 1b

(b) Contours of image 2a in the real photo

(c) Contours detected by image 1d

(d) Contours of image 2c in the real photo

Fig. 2: Contours detected in the image

(a) Boat image obtained by the figure 2b

(b) Boat image obtained by figure 2d

(c) Another boat image obtained by the thresholding

(d) Another boat image obtained by the Canny algorithm

Fig. 3: Contours detected in the image

(a) Contours (Canny threshold) of an image with a lot of details

(b) Boat obtained by image 4a

Fig. 4: Contours detected in the image

The best results were obtained with the Canny algorithm and 4 dilatations. This can be traced to the fact that the image used by Canny algorithm to find the edge were filtered with a bigger kernel (11 x 11) and therefore more small details were removed. The use of *adaptiveThreshold()* allows the only the use of a small kernel (3 x 3) and subsequently more unnecessary details remains.

Despite its simplicity this algorithm can be used for every kind of object detection[1]. The research of candidate can be improved with check on the height and wide of the rectangle respect the entire image, with the ratio of height and wide and so on. The type of analysis on the rectangle can be decided based on the type of object to be searched.

Other images can be found in the folder *Image/boat*. The name refers to the type of threshold (*adapTH = adaptive threshold*) and the number is referred to the number of dilatation.

III. Sea Detection Algorithm

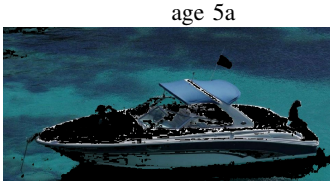
A. Introduction to the problem

Despite appearances the research of the sea in images can be a very difficult task, much more difficult that find a boat. This is caused by the fact that sea in the photo is part of the background (so it can be revealed by edge detection) and its features can have a very large variability (e.g. the sea is generally blue but can have the shades of blue can change a lot from image to image).

[1]In fact was initially develop for plate detection

(a) Original image

(b) Blue mask obtained by image 5a



(c) Sea image obtained by image 5c

Fig. 5: Segmentation based on blue pixels in HSV color space

The first approach involved the use of blue pixel to threshold the image but the results were not good enough. An evolution of this approach with the combination of results of thresholding in different color space led to some improvement but still not good enough. The last approach used an algorithm created ex-novo and based on the similarity between pixel to segmented the image in a variable number of zones.

B. The algorithm

The algorithm is divided in phases:

1) Filtering of the image
2) Detection of blue pixels
3) Sea detection

The filtering was performed with a 3 x 3 Gaussian filter. This step is necessary to remove noise. Also filtering the image improve the image segmentation.

The blue pixels map has been obtained with the function *inRange()* which return a binary image where the white pixels are the pixels that in the original image have value inside the parameter pass to the function. Since the function *inRange()* works in every color space the HSV color space was chosen for its property. Specifically HSV separates the pure color (Hue, H) from Saturation (S) and Value (V); this makes it easier segmentation based on color. Results can be seen in figure 5.

Once obtained the mask of blue pixel (Figure 5b) it possible to decide if the sea is present in the photo. This is done by calculating the percentage of blue pixel in the photo and if the percentage is greater than a threshold there will be probably sea. For this project the percentage was set to 35

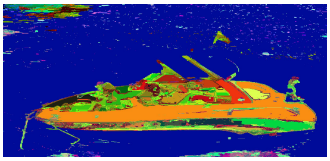
The detection of blue pixel in only 1 color space is not sufficient to create a mask of the sea. Image 5c show that despite all the sea was found large pieces of the boat were recognized as sea.

This approach can be improved with the overlap of various blue mask obtained by the thresholding in different color space. Image 6a was obtained by the overlap of thresholding in RGB color space, HSV color space and HSV color space; pixels present in all 3 mask are colored in white, pixels present in 2 mask are colored in grey and pixel present only in 1

(a) Mask of blue pixels with thresholding in various color spaces

(b) Sea mask obtained by image 6a

Fig. 6: Confront between sea mask



(a) Result of the clustering algorithm

(b) Sea mask obtained by image 7a

Fig. 7: Confront between sea mask

mask are eliminated. Image 6b is the sea mask obtained by image 6a. From the comparison of image 5c and image 6b some improvement can be seen, especially in the deck of the boat but the bottom of the photo is still not recognized as sea.

The last approach used is based on the similarity between pixels. An algorithm developed ex-novo and implemented with the function *averageMapZone()* divides the image in cluster (called family inside the code). The algorithm started in the top left corner of the image and assign the pixel to the first family. Then check the pixels around and if they are enough similar they are assigned to the family. When all the neighbor pixels are checked the algorithm pass to another member of family and checks its surrounding pixel. When no member of a family has pixel around him that can belong to that family the algorithm pick up a pixel that still has no family and start crating a new family. When all the pixel are assigned to a family the algorithm ends. A graphical examples of the results of the algorithm can be seen in image 7a. Once obtained the list with all the family the biggest one is selected and all the pixels of this family are used like a mask to extrapolate the sea from the original image. If the biggest family has too few pixel the two biggest family were used to create the mask.

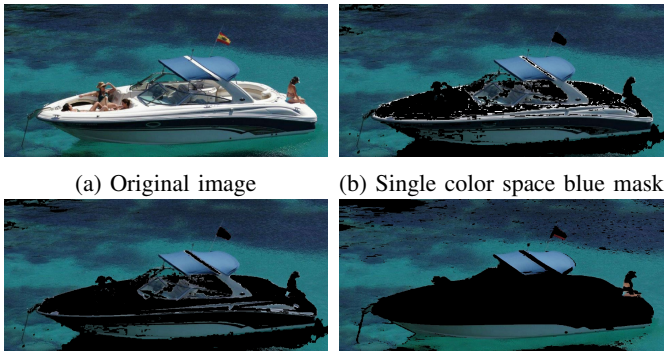
The criterion of similarity between pixels is based on the mean value of the pixels evaluated with this formula:

$$m(x,y) = \frac{B(x,y) + G(x,y) + R(x,y)}{3}$$

where m is the mean value of the pixel and B, G and R are the intensity of Blue, Green and Red component of the pixel. To be assigned to a family a pixel must respect the following condition:

$$(1-p) * m_f(x,y) \leq m_n(x \pm 1, y \pm 1) \leq (1+p) * m_f(x,y)$$

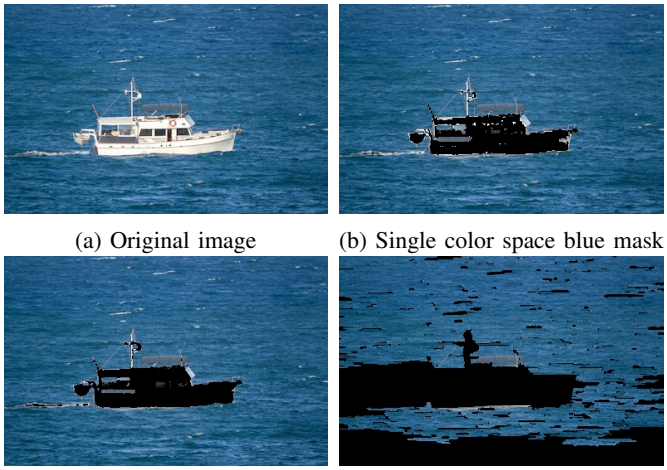
where m_f is the mean value of a pixel inside the family, m_n is the mean value of the possible candidate (the n stands for neighbor) and p is a factor of similarity that can range between



(a) Original image (b) Single color space blue mask

(c) Multi color space blue mask (d) Family mask

Fig. 8: Confront between the different approaches



(a) Original image (b) Single color space blue mask

(c) Multi color space blue mask (d) Family mask

Fig. 9: Confront between the different approaches

0 and 1. For example if $p = 0.15$ the value of m_n must be between the 85% of m_f and the 115% of m_f to be assigned same family of m_f. The results in this report are obtain for a value of $p = 0.0334$ (the value was choose by testing different p; $p > 0.1$ are generally useless).

C. Results and discussion

From the comparison of the images in figure 8 a good improvement in the results can be seen with the last approach; this is especially true for the bottom of the image and the deck of the boat. In some occasion like the one in figure 9 the blue thresholding works better but generally the family map has better performance. Other examples can be found in the folder *Image/sea*; the name of various folders inside specify the type of image (mask 1 = blue threshold in one color space, mask 2 = blue threshold in multi color space, mask 3 = see last paragraph of this section, mask 4 = family mask)

In the end the best approach were the multi color space thresholding and the family mask, with the latter that provided the better results.

The main problem related to the family mask is a bug linked with incorrect memory management that cause the program to stop suddenly. Unfortunately this bug seems to have no precise logic; the same code can be compiled and executed without change and one time be interrupted by the bug and the other time work perfectly.

Another initial idea to improve the performance was to combine the use of different color space with the family algorithm. It is implemented in the code although the memory bug makes it difficult to use it and there are no significant improve (examples of the results of this type of mask can be found in the folder *Image/sea/mask 3*).

IV. Code Tutorial

A. Why this section?

The code have a lots of comment to help the analysis and generally the name of the variables and function are heavily related to their purpose; this was done to be able to reconnect quickly the various function to the description of the two algorithms. Consequently in this section only a few info, to integrate the comments in the code, are reported.

Both classes are structured like tools to be used by external users. Each class has its set of parameters that can be adjusted with the various setters so alter the results of the final function. The classes has been built in this way for two reasons:

- The possibility to tune the parameters means that the code can be adapted to work with different kind of images. Hypothetical we can set the parameter for each image
- The final user don't need to directly modify the parameter into the code but can easily do with the various setter methods.

So the final users can be free to vary the parameter to see the different results. The parameter set in the main (called Zancanaro_CV_Project) provided with this report are the parameters that provide the best general solution with the samples of image provided for this task.

B. Guide to the classes - Common function and parameter

Despite the different tasks that the two classes perform they have some support functions that are called in the same way and work in the same way in both classes. Also the parameters related to this function are the same. List of common function:

- *showImage*: this function can be used to visualized the various image during the elaboration. Both *print* and *print˙image* must have value *true* for the visualization. The function require in input an integer, that specify the image to show, and a string, used as a window names. The string is optional.
- *setPrint* and *setPrintImage*: this function are used to set the respective parameters

List of common parameter:

- *print*: if set to *true*, during the execution of the program some info concerning the progress of the algorithm are printed in the terminal. Useful for debugging.
- *print_image*: if set to *true*, during the execution of the program various image related to the various step of the algorithms are visualized. Useful for debugging and parameters tuning.

C. Guide to boatDetection

Function:

- *boatEnanching*: the function that perform the morphological dilation on the binary edge image.
- *setMorphOpening*: see *morph_opening* below.
- *setBoundType*: see *bound_type* below.

Parameters (and variables):

- *contours_img*: original image but with the contours drawn above. An example of this image is the figure 2d.
- *morph_opening*: if set to *true* perform a morphological opening on the binary image of the contours.
- *bound_type*: if set to 1 the bound in image *contours˙img* will be rectangular. If set to 2 the bound will be like the ones in the image inside this report.

D. Guide to seaDetection

Function:

- *evaluateMapZoneV1* and *evaluateMapZoneV2*: function use to create the *sea_mask*. The V1 correspond to mask 1, mask 2 and mask 3. V2 is used to create mask 4. They have been divided to prevent the creation of a function too big.
- *findBlue*: create the blue mask.
- *averageMapZone*: function that implement the division of the image in family. Inside the code each family is represented with a integer value; the first family is the family 1, the second the family 2 and so on. The algorithm return an Mat containing the original photo divided in family. It also possible pass a Mat to the function and at the end of the algorithm each cell of the Mat (corresponding to a pixel in the original photo) will contain the value of the family of that pixel.
- *pixelMeanValue*: function use to evaluate the mean value of the pixel. If the parameter weighted_mean is set to true it will perform a weighted mean.
- *matNormalization*: this function normalize a Mat so all its values will range from 0 to 255. Used to improve the performance of *averageMapZone* if the image in input have a color space where its channels have different range of value (e.g. HSV).
- *evaluateMapZoneSupportFunction*: function used to help the implementation of mask 3 type.

V. Conclusion

A. Knowledge acquired

The main knowledge acquired with this project are:

- Use of edge in image and morphological operations for object localization.
- Segmentation of image (color space and families mask).
- Programming skills: organization of the code (keep the code clean and legible) and the importance of implements directly in the code the tools for debugging.

B. Future improvement - Boat detection

Improvement in boat detection can be done in the phase of candidate selection with deeper analysis on the contours find by the function *findContours()*.

Also another improvement could be the capability to recognize if there is or not a boat in the photo. Unfortunately this is very difficult with deterministic algorithm since the large variability of possible boat, viewpoint ecc. A possible solution is to create a neural network for this task.

C. Future improvement - Sea detection

Possible improvement in sea detection can be done, especially in the use of *family algorithm*. The implementation in the code make it easy change the criteria of confrontation between pixel. The weighted mean was implemented but no used so for example the correct tuning of the weight could improve the performance.

The evaluation of the family mask in different color spaces could also improve performance, especially if the various channel have the right weight in mean evaluation. Overlap of different family masks obtained in this way could improve the segmentation of image.