Underwater pipe detector program - Version 1

Leopoldo Cadavid Piñero

February 27, 2023

Contents

1 Intro **3**

2 Preprocessing **3**

3 Image Processing **3**
3.1 First approach: K-means algorithm 3
3.2 Second approach: detection based on lines 4

4 Post Processing **7**
4.1 Cluster management 7
4.2 Line filtering 7

5 Subsection **8**
5.1 Used Dataset 8

6 Performed tests **8**
6.1 Methrics and testing methodology 8

7 Results **9**
7.1 K-means approach 9
7.2 Lines approach 9
7.3 Change PyrMeanShift parameters 9

8 Results **10**

9 Referencias **10**

1 Intro

The main goal for this algorithm is to make possible human-made structures detection at underwater environments, making use of classic computer vision techniques.

Especifically, this first version of the algorithm is focused on pipes or similar structures.

2 Preprocessing

First of all, is necessary to apply some preprocessing methods to get our images *cleaner* before trying to get features from them. Preprocessing pipeline consist on 2 parts:

- Apply a mask in order to increase contrast, decreasing L value in CIELAB channel space in our picture. This is done because we want to compensate colour distorsion result from ligth propagation along the picture.
- Second, we use a CLAHE (*contrast-limited adaptive histogram equalization*) in order to **redistribute luminescence**.

La aplicación de los métodos da como resultado una imagen mán nítida y contrastada donde es más fácil aplicar los algoritmos para la segmentación y detección de las estructuras.

3 Image Processing

3.1 First approach: K-means algorithm

Based on referenced papers, the use of Sobel filter to get gradient response on different directions. Then this response would be used to create a **features vector**. The final step would be to classiffy objects using K-means algorithm with this feature array as entrance.

At a first stage in the studies of the algorithm, no significative resutls were provided by this methon.

However, due to no progression on our second approach, it was decided to retake K-means idea.

Currently, big advances have been made, as it has been correctly implmenetated in a python script. In summary, the methodology has been the following:

- Apply CLAHE + blur L channel for improving quality in our image.
- Apply white balancing in order to achieve a more accurate color space (based on Gray world theory)
- Apply Sobel filtering to each hsv channel, for achiving gradients on each channel
- Create a features vector, each pixel has the following features: **F(H, S, V, GradR, GradG, GradB)**
- Apply k-means algorithm to our features vector, so we are getting N clusters of pixels.
- Reagrupate our clusters and process information. (this is described on image post processing section)

Above, in section 8 more specific details of our algorithm results will be explained. Some of the clusters detected are the following.

3.2 Second approach: detection based on lines

In this case, based on certain ideas discussed in the previous approximation, we're trying get a different pipeline from the studied one. The steps followed in this case they have been the following:

- Perform preprocessing, as described in 2.

Figure 1: Before-After preprocessing

- Apply a bilateral filter to remove some background noise from images.

Figure 2: bilateral filter

- Apply `PyrMeanShift()`, so that the image is segmented by colors.

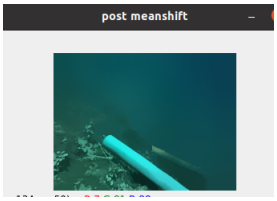

Figure 3: Meanshift segmentation

- Apply the algorithm **Sobel()**, to each of the channels of theprepro-cessed image RGB. This is done to get the gradient changes and is applied equally to the X and Y axes, performing a weighted sum of both in order to obtain the change in different directions.

Figure 4: Sobel gradients splitted

- Obtain, from the gradients, edges in the images applying the algorithm `Canny()`.

Figure 5: Edges with Canny

- We use morphological filters to erode and dilate, trying to eliminate, circular elements that may disturb and then we amplify rectangular elements.

Figure 6: Edges after morphological transformations

- Having the edges filtered, the Hough transform is used to find lines in inside the image.

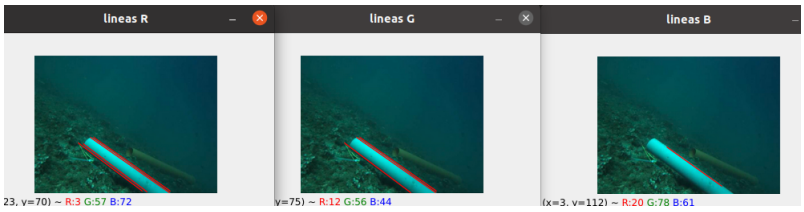

Figure 7: Lines from Hough transform

- From the 3 line vectors we have, we create a new one where we will join **all lines in the image**.

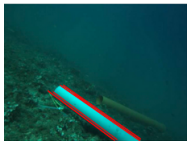

Figure 8: Lines from Hough transform

4 Post Processing

4.1 Cluster management

After obtaning the clusters from the image, its necessary to get the mask from this information.

4.2 Line filtering

From the extracted lines, it seeks to dete lines that do not correspond to those belonging to the real pipe. For this, it will be analyzed both the degree of parallelism and the distance between them.

The idea is to analyze in the vector of lines each pair of these in the following way:

- The slope and the angle of each of these with respect to the front are calculated
- The angles are compared and, if they are below a certain threshold, the lines will be considered parallel
- The distances in X and Y of the initial points of each line are compared, and we are left with those that are below a threshold away
- Lines that meet all conditions will be saved in a vector of parallel lines

Update: One of the updates implemented has been a new algorithm for parallel lines. First algorithm worked well when recognising parallelism between any kind of lines, desafortunatedly, it was a problem when many noise lines appear, because it turns out in many parallel non-significative lines.

Because of this, another criteria was tried. I could notice that, in most of the cases, the biggest line founded was always on our pipe so, instead of studying parallelism between all pairs of lines, you could only compare lines with the bigger one , this wouls help eliminating problems with noisy lines. The biggest issue with this new algorithm is that it relays on the assumption that the biggest line will be alway on the path.

After filtering the lines, these are highlighted on the original image, applying certain morphological filters to improve their shape, and we finally obtain the image with the mask applied on the structure.

5 Subsection

5.1 Used Dataset

Before showing the results, the images and video used to study the operation of the algorithm will be shown:

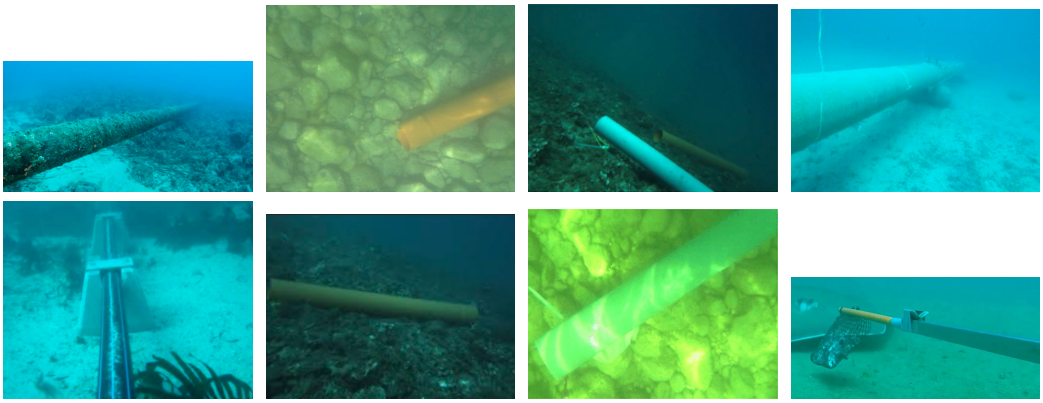

Figure 9: Some images

This youtube video has been used for video detection: https://youtu.be/L0Ev5R3fBEc

6 Performed tests

6.1 Methrics and testing methodology

There have been 2 different kind of tests applied to each pipeline, accuracy and time-performance tests.

For each one, we are going to explain which methrics have been used:

- **Intersection of Unions:** is a way to measure the similarity between two sets of data. To calculate this metric, we first take the union of both sets, which represents all the elements that are present in either set. We then take the intersection of the two sets, which represents the elements that are common to both sets. The intersection of unions metric is then calculated by dividing the size of the intersection by the size of the union.
- **Execution time in miliseconds:** provided by *Time* library in Python.

Going deep inside testing methodology, for our accuracy tests, what has been done is the next:

- Paint, manually, an ideal mask on each image of our dataset.
- Calculate intersection of unions for each image, betweem the mask produced by our algorithm and the ideal mask.
- Show each accuracy one by one and, finally, a mean accuracy in our algorithm.

On the other hand, when testing time performance, what has been done is the following:

- Split the code in separated functions, each one corresponding to one of the processes in our pipeline.
- Take the execution moment at the beggining and in the end of every function. The difference between this moments corresponds with the time execution taken by the process.

7 Results

7.1 K-means approach

7.2 Lines approach

7.3 Change PyrMeanShift parameters

As was described in section 3.2, `PyrMeanShift()` algorithm is used for decreasing noise after preprocessing our image. Specifically, its aim is to do colout segmentation based on 2 types of neighborhoodness: by colour and by space.

Testing this function, the parameters that have been tuned are the following ones:

- Spacial neighborhood radius was increased. This would help the algorithm find more same-colour pixels in a bigger range.
- Colour neighborhood radius was decreased. Because of this, less pixels will be considered as *same-coloured*. This produces a smaller segmentations.
- Max level of segementation has been elevated, meaning that we will get less contours as segmentation has been bigger.

This test have shown that, scalating coulour neighborhood can reduce noise significatively but, as a takeoff, it can erase part of our structure, destying performance.

8 Results

9 Referencias

https://journals.sagepub.com/doi/10.5772/60526 http://www.ce.unipr.it/ rizzini/papers/kallasi1