Underwater pipe detector program - Version 1

Leopoldo Cadavid Piñero

February 17, 2023

Contents

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 Line detector

3.1 First approach

Siguiendo con los métodos referenciados, se propone el uso de un filtro sobel para obtener las respuestas de gradiente, y añadir esta respuesta de gradiente a un vector de características, con el objetivo de dividir la imagen según estas características en diferentes clusters. Para ello se pretende usar el algoritmo k-means.

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.

The results received from those transformations applied to our picture weren't successful. One of problems found, was an information loss, that appeared after grayscale conversion from RGB when we apply Sobel. Grayscale meshes all channels in one, ommitting part of the information.

The solution implemmented for this issue has been

As a result of this, the sobel filter has been applied to each of the image channels separately. Subsequently, the average gradient of each channel and each is added to the k-means algorithm. In this way, clustering has been substantially improved. Still, it's not for making an accurate segmentation.

At the moment, this approach has been abandoned, as long as it has not reported results of any kind. However, it could be interesting to come back to its study sometime in the future.

3.2 Second approach

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.

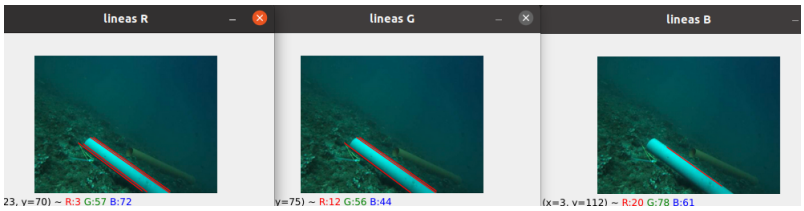

Figure 7: Lines from Hough transform

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

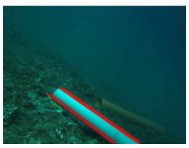

Figure 8: Lines from Hough transform

4 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

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:

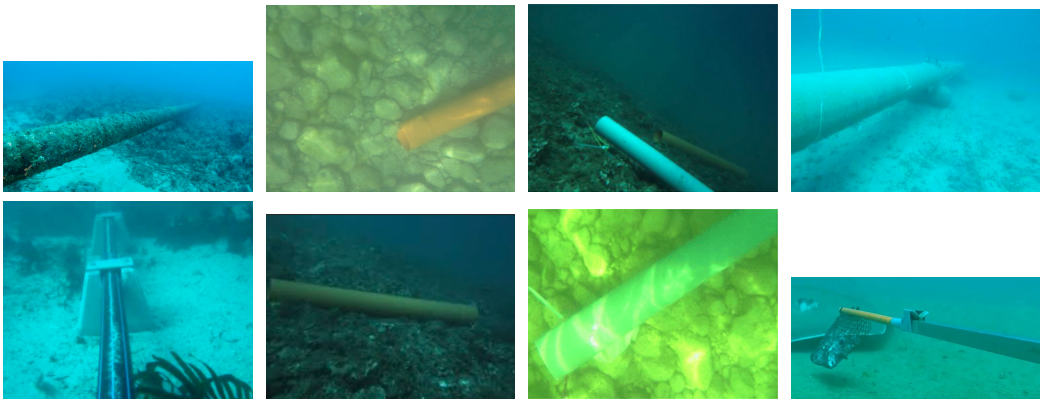

Figure 9: Some images

El video utlizado ha sido el siguiente https://youtu.be/L0Ev5R3fBEc

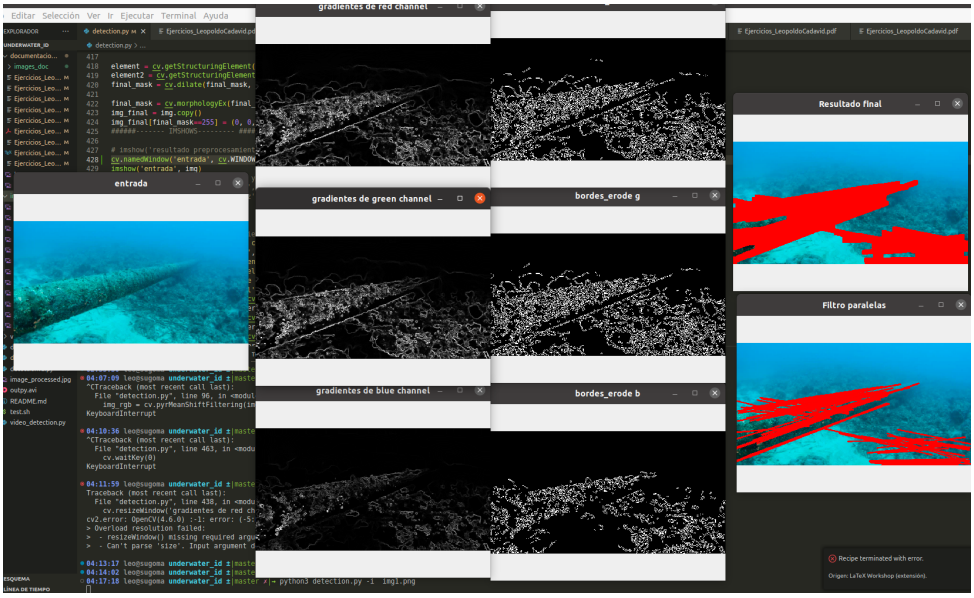

Figure 10: res1

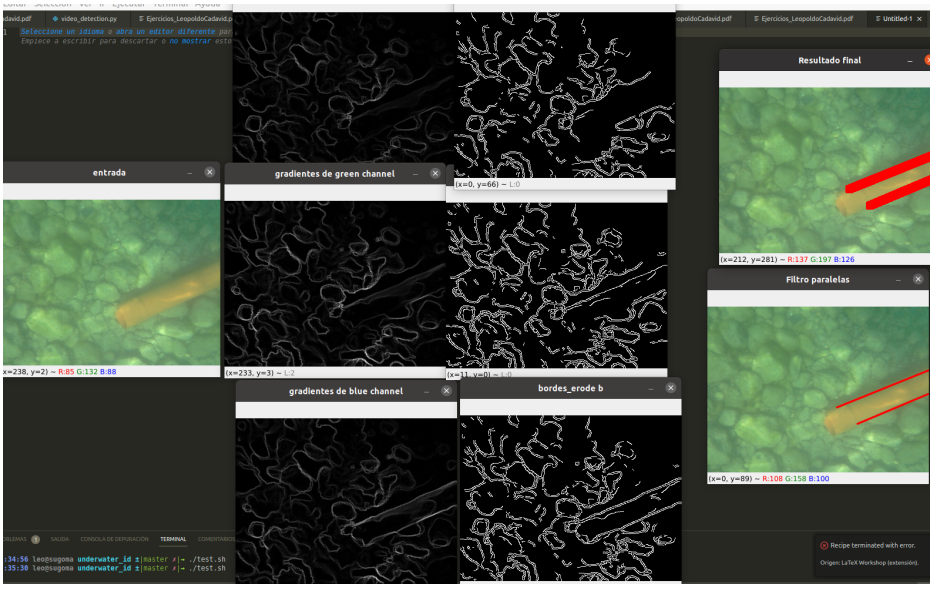

Figure 11: res1

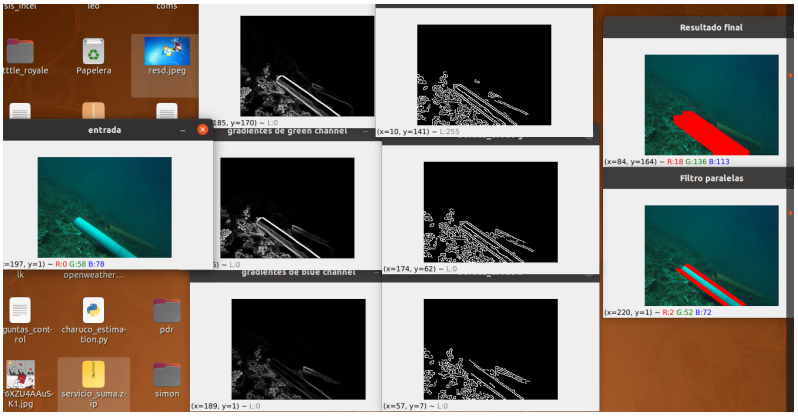

Figure 12: res1

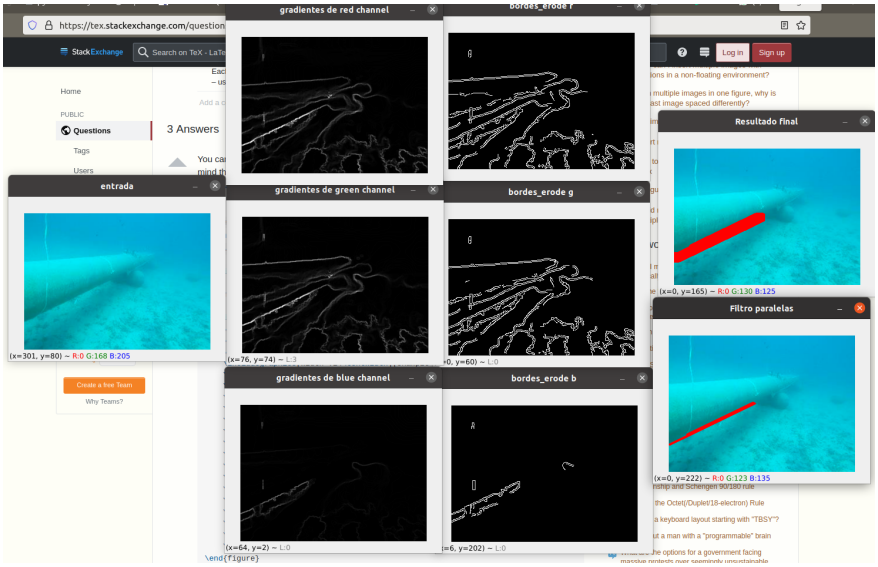

Figure 13: res1

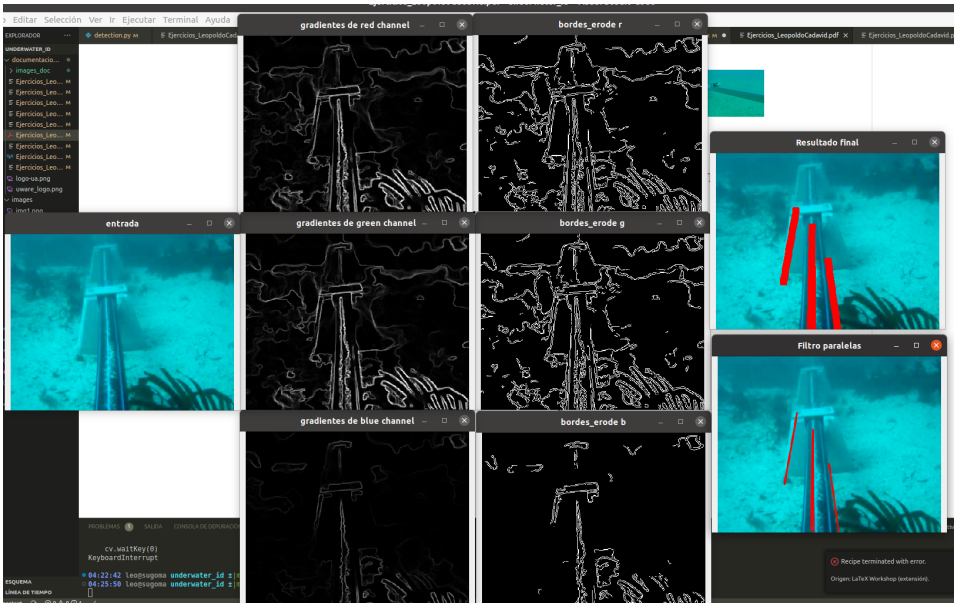

Figure 14: res1

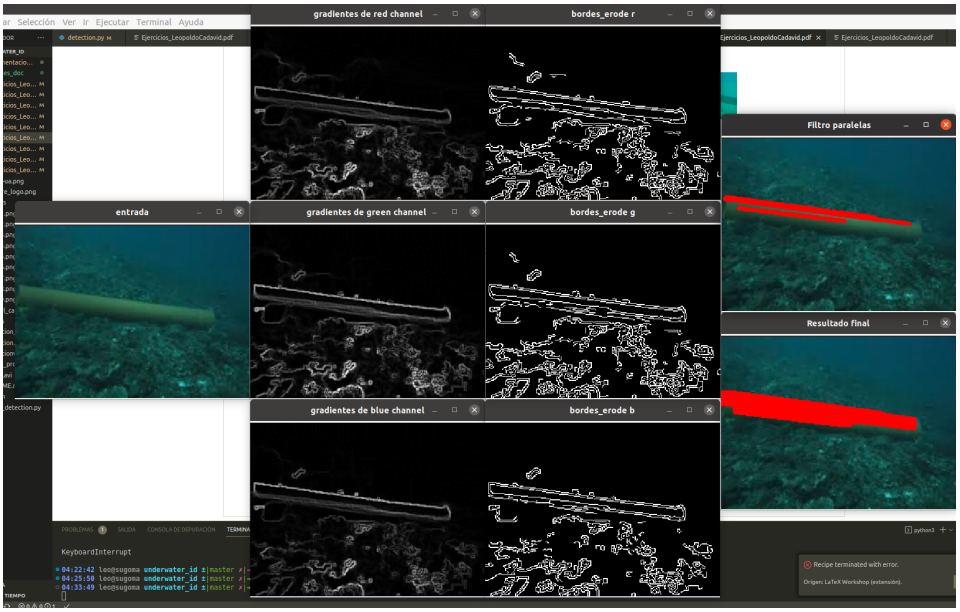

Figure 15: res1

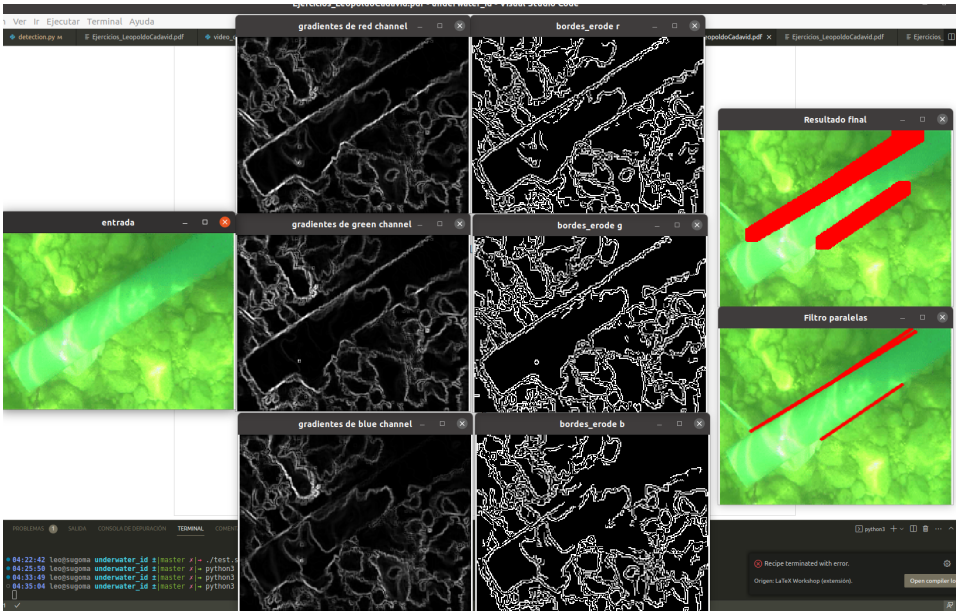

Figure 16: res1

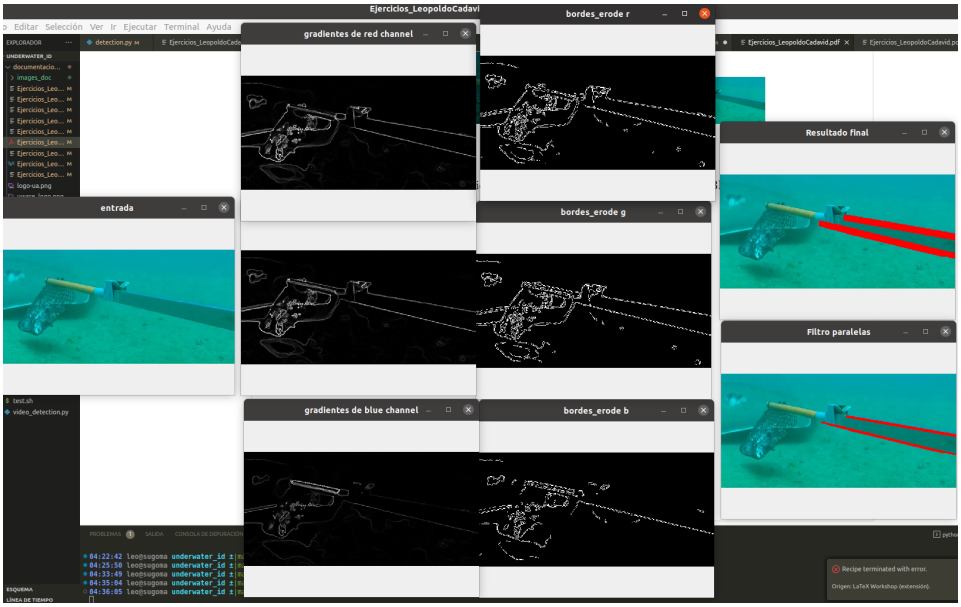

Figure 17: res1

5.2 Possible improvements

- Tune or delete some esteps in order to increasing computing velocity. Analyze each algorithm time-cost vs its ultility inside our pipeline.
- Make line detection more sophisticated in order to avoid information losses or fake lines. Study new algorithms or methrics for the used ones.
- Implement as a ROS node and make performance tests with more images.

6 Performed test

6.1 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.

7 findparallel2()

One of the actualizatons 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.

8 Referencias

https://journals.sagepub.com/doi/10.5772/60526