Emergency exit recognizer

Course: Multimedia systems

David Savev, ID: 14057

January, 2021

1 Introduction

The purpose of the developed tool is to recognize the presence or the absence of emergency exit shields inside a picture and, in case of presence, to correctly identify the arrow within the shield and acoustically signal the direction to follow to leave the building. An example of emergency exit is the following:

Figure 1: Emergency exit example

We can easily spot the main recurrent patterns within similar images, which

are fundamental in order to correctly apply the processing steps:

- Within the image we have a closed area which is predominantly green (the green shield);
- The green area is rectangular and a rectangle can be viewed as convex polygon with 4 vertices;
- Within the green region, we shall have a region whose color is significantly different from the background one (green vs. white);
- An arrow, similarly to the analysis applied to the rectangle, is a concave polygon (some diagonals of a concave polygon lie partly or wholly outside the polygon) which can have either 7 or 9 vertices.
  7 or 9 vertices since we may encounter two different kind of arrows, one simpler and the other more structured:

Figure 2: 9 vertices arrow

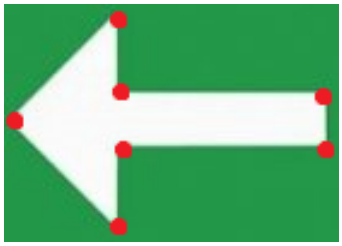

Figure 3: 7 vertices arrow

In order to process the images, I opted for the usage of the high programming language Python and the support of OpenCV library; OpenCV was selected since it is well documented (`https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_tutorials.html`) and supported among several programming language (Python, C++, Java), hence is widely adopted for computer vision applications.

2 Processing

The first part of the processing permits the recognition of the emergency shield within the image and is composed by the following blocks:

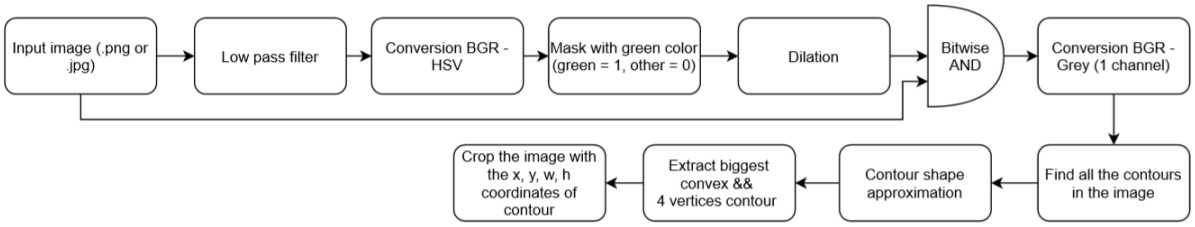

Figure 4: First part of the image processing

For what concerns the second part of the processing, it is applied to the output of the previous block; its purpose is to recognize the arrow parameters out from the cropped image and compute its direction:

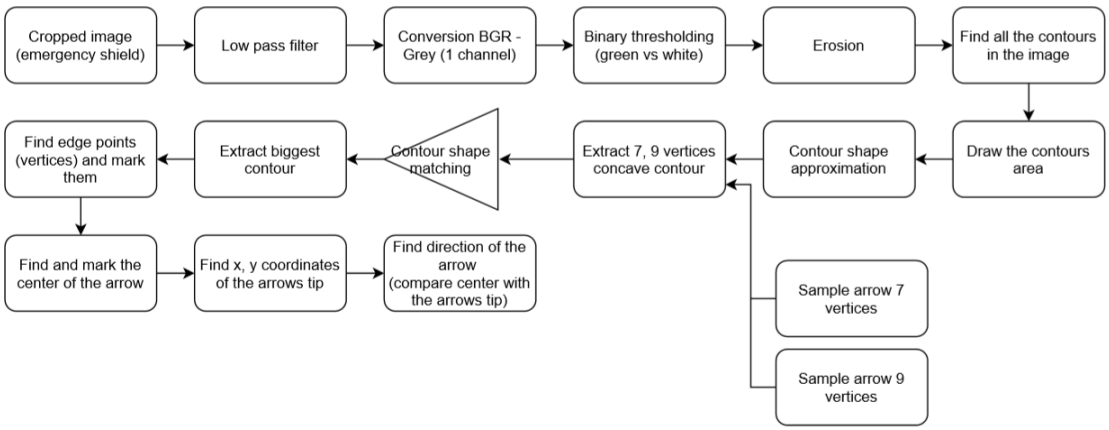

Figure 5: Second part of the processing

More in detail, the implemented steps within the source code are as follows:

1. Read the image (.png or .jpg);
2. Filter the image with a 5x5 L.P. Gaussian smoothing filter in order to reduce noise;
3. Conversion of the image color space from BGR to HSV (provides linearity in selecting color ranges);

4. Apply a mask on the HSV image to maintain only green pixels (Green: [31, 40, 40] – [86, 255, 255]);
5. Dilate the green mask with a 3x3 filter (neon lights on top of the emergency shield may create reflection problems on the top edges);
6. Perform a bitwise AND on the original image with the green mask;
7. Convert previous image in gray and find the contours (findContours() does not accept 3-channel images);
8. Contour approximation through the Douglas–Peucker algorithm.
   In OpenCV, the method is called approxPolyDP(contour, epsilon) and its purpose is to reduce the number of segments composing the contour polygon (hence the number of edges).
   This step is applied since in the image the blurring or the rotation may affect the correct extraction of the contours.
   Within the method, the epsilon argument indicates the accuracy of the algorithm (the distance from the original to the approximated contour).
   An accuracy value of 0.05 was empirically selected as tuning parameter.
9. Extract the biggest contour which has rectangular shape (polygon with 4 edges), is convex and has an internal area $\geq 1 / 900$ of the image (small contours are not taken into account due to low resolution);
10. Crop the shield contour from the original image by taking x, y, w, h values (clearly if exist);
11. Filter the cropped image with a 3x3 L.P. Gaussian smoothing filter in order to reduce noise;
12. Apply binary thresholding on the grey version of the cropped image (val > 180); we need to recognize the white arrow in the shield, which has completely different color from the main background (green);
13. Erode the image by a 3x3 mask; since the "white man" in the shield is thinner than the arrow, it will usually cut it off;
14. Find all the contours and draw them with red color;
15. Contour approximation through the Douglas–Peucker algorithm; ϵ value $= 0.02$ was empirically selected.
16. Extract the biggest contour which has 7 or 9 edges (simple or complex arrow), is concave and the shape matches with one of the two sample images (9 edges arrow and 7 edges arrow).
    The comparison is performed through the method matchShapes() which returns a float within 0 and 1 (the nearer to 0 the similar shape the contour has);
17. Mark the edge points of the arrow (if exists);
18. Estimate the center point of the polygon composing the contour and mark it;

19. Find out which one out of the 7 or 9 edges is the tip of the arrow. According to my approach, the tip is the point which has the biggest distance to the closest neighbor;
20. Now, check the direction of the arrow by comparing x, y coordinates of the center and the tip point and acoustically signal it; the approach implemented to verify the direction is:

    - Find whether the center and tip differ at most in the x or in the y coordinate;
    - If the maximum difference occurs in the y coordinate, it means that the arrow indicates either up or down; it is now sufficient to find whether the y coordinate of the tip is smaller (arrow pointing up), or bigger (arrow pointing down);
    - If the maximum difference occurs in the x coordinate, it means that the arrow indicates either left or right; also here, if the x coordinate of the tip is smaller it means the arrow is pointing left, if higher is pointing right.

3 User interaction

The tool implements also a simple GUI (developed with the tkinter library) which simplifies the interaction with users; the main features of the GUI are:

- The interface permits the upload of .jpg or .png images;
- The uploaded image is printed on the left hand side of the frame;
- In case an emergency exit shield is recognized, its processed image and all its features (contour, vertices, center) is printed on the right hand side of the frame;
- We have the possibility also to plot the intermediate steps of the processing by clicking the "show info" button; it will output the green mask, the green mask applied on the original image, the cropped area (emergency shield) and the emergency shield after the thresholding operation.
- In case the arrow is correctly recognized, an acoustic indication with the direction to follow is emitted.

4 Conclusion

Since no Machine Learning techniques were used, the reconnaissance value is not optimal; the tool is able to recognize correctly the presence or absence in 70-75% of the images that were collected as samples.
In the other 20-25% of sample images, false positive and false negative results may be returned.
This error is given due to the big number of different conditions within the

images (image colors, image rotation, blur) and tuning parameters (green color range, dimension of filtering matrixes, threshold values, approximation of polygon vertices, upper bound of the shape matching between contours).
Further investigation, analysis, refinement and tuning is so needed in order to increase this percentage.

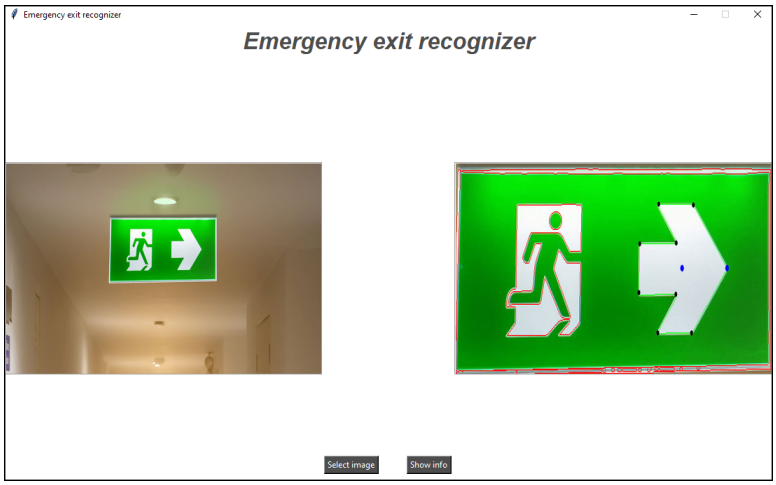

Figure 6: Application GUI 1

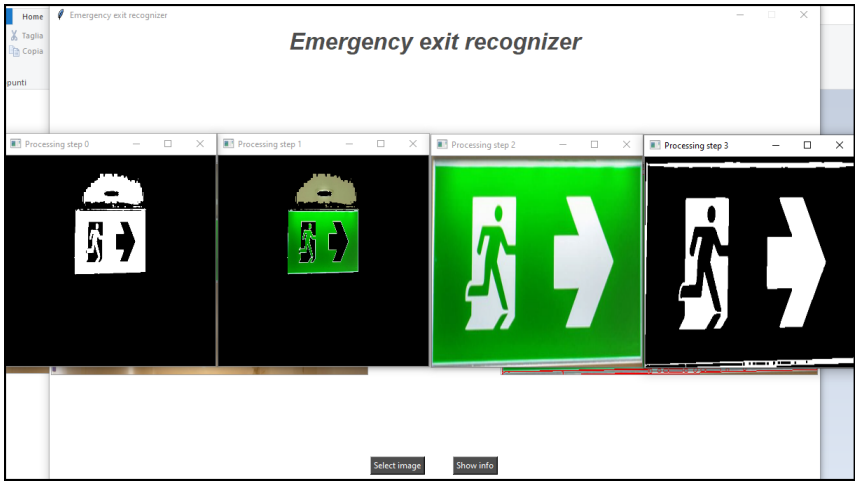

Figure 7: Application GUI 2