

Image Processing

Assignment 4 – Object detection – Marogna Lorenzo

In this last assignment of the course, the aim is to design and implement a pipeline which allows the user to detect a class of objects. The object must be realistically detectable as a composition of other elements such as lines, squares, rectangles, shapes, ellipses and so on.

After a careful examination, I decided that an electrical socket has the characteristics that I am looking for:

1. There is a square or a rectangle of different areas and proportions.
2. There could be a circle inside, but this may vary depending on the type of socket.
3. There are some holes inside, usually very dark, and they can appear with different orientations and alignment.

The project has been very challenging, and this is what I've been able to come up with after these 2 weeks of work.

The pipeline

Preprocessing

First things first, the image gets converted in a grayscale version and the contrast gets adjusted to have a better usage of the range of intensities.

Line detection

The first step is to detect lines in the image, to compose some of them to create squares and rectangles.

At first, I used the same technique from assignment 3:

1. Calculate the edge magnitude of the grayscale image with a Sobel Filter
2. Use Hough Transform on the edge magnitude image.
3. Adjust contrast on the result (linearly normalization in the range 0 – 255)
4. Threshold the image (60% of maximum intensity)
5. Close the image (11x11, Square-shaped structural element)
6. Apply flood fill to label the regions.
7. Find centroids of labelled regions ((r, θ) pair representing a line)

I noticed that some lines which would have been useful to build the rectangle are not detected. So, I decided to implement the Canny edge detection to maybe find some new lines to compose shapes with.

In the Canny edge detection, these steps are executed:

1. Smooth the grayscale image with a gaussian filter through a convolution.
2. Compute horizontal and vertical gradient with a Sobel filter.
3. Calculate the magnitude of the gradient (different result from previous step 1 because of the smothering)
4. Rotate the gradient of 22.5 degrees to facilitate the process of finding the direction of the border (horizontal, vertical, 45 degrees diagonal, 135 degrees diagonal). With the 22.5 rotation we can avoid computing the $\tan^{-1}(\frac{\Delta y}{\Delta x})$, which is not efficient, and focus on directly comparisons of the signs of the components and their respective length.
5. Apply “not maximum suppression” in the detected edge direction for each pixel.
6. Set a higher (100/255) and a lower (50/255) threshold. Every pixel which has survived and has a greater value than 100 is automatically a border pixel.

7. Apply hysteresis process to find any pixel over the lower threshold which is reachable from a border pixel and add it as a border pixel.
8. The edge map is now complete, from here we can go back at the previous step 2 and detect the lines.

Canny edge detection is allowing the program to detect some new lines, which get added to those we already knew.

Grid constructor

Now that we have a series of lines, we need to find a group of lines which are either parallel or perpendicular to each other, to then identify in this grid which are the rectangles representing a socket. The algorithm looks at the lines that have at least one parallel line (no rectangle can be built otherwise) and find the biggest grid available (e.g., a vertical horizontal grid with a total of 20 line is more valuable than a diagonal diagonal grid with 4 lines, even if it's possible to leave a socket back this way).

Given a grid, we now have a list of rectangular regions to investigate to find a socket.

Special Note

It would now be the moment to look if there is a circle inside of the region, but I decided to skip this step for 2 reasons:

1. The Hough transform for circle was only partially ready, it worked with one radius at a time to make the problem 2 dimensional so I thought I had more urgent things to implement before.
2. Most sockets I could find on the internet don't have a circle to look for. This would have probably increased the number of false negatives. I could have kept some probability variable to make it still possible to find sockets without a big circle inside the rectangle, but I decided to stick with a binary decision.

Blob detection

One of the main things that can help distinguish a socket from just a rectangle is the presence of dark small blobs in the image, which are the little black holes to insert the plug. To detect these blobs, I could have used Normalized Laplacian of Gaussian (NLoG), threshold the result and find the centroids of the labelled regions, but I thought of a different approach that can work well enough, given that the blobs I'm looking for are very dark with a very bright region around it (sockets are usually white). This is the pipeline I used for every rectangle of the grid:

1. Crop the image on the rectangle to find only the blobs inside of the rectangle.
2. Threshold the image (20% of 255) – dark blobs are very likely to be mapped to 0 and the rest of the rectangle (if white) is going to be mapped to 255.
3. Invert the image.
4. Open the image (5x5, plus shaped structural element) – to get rid of single pixels which are not big enough to be considered as blobs.
5. Apply flood fill to label the blobs and find their centroids.

Socket detection

It is now time to decide if there's a socket in the rectangle.

The first thing I check is whether the number of blobs is less than 2, in that case it is not a socket.

Since the holes of the socket can have different shapes and dispositions, my idea is to calculate the barycentre of all the blobs that have been detected and check if it is close enough to the barycentre of the rectangle.

Socket recognition

Now that the sockets are detected, in this final step the program tries to recognize whether the socket is an Italian socket, a British and so on. To do so, it looks at the blobs position and orientation and a different colour is assigned to each type of socket, so that the result is distinguishable in the final Image.

Side note:

Every parameter in the project has been chosen a-priori and empirically, to get the best result possible.

Results


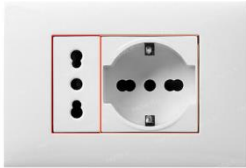
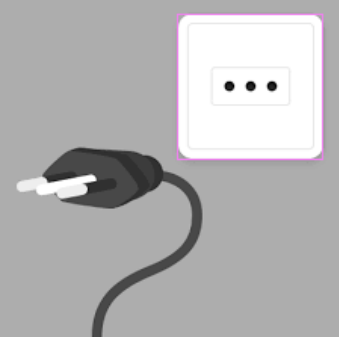
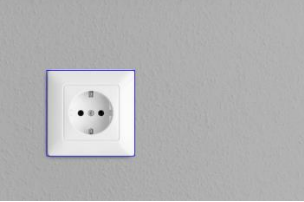
For the simpler images the program is working fine, with some resistance to noise and a good ability in distinguishing different types of sockets.



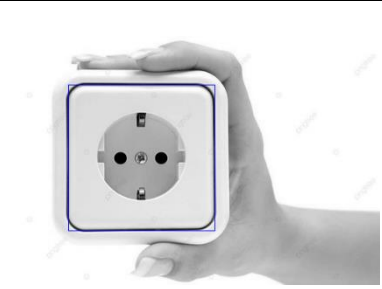
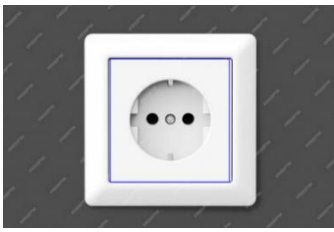
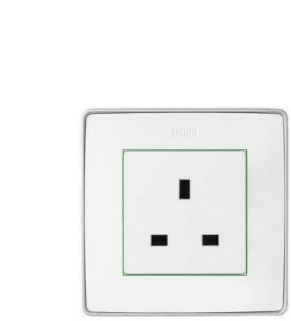
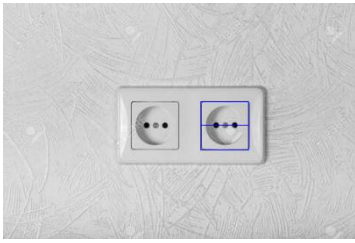
Here is the legend for colours:

- British (equilateral triangle) => Green
- German / French (very hard to distinguish, both have 2 horizontal blobs) => Blue
- Italian (3 vertical blobs) => Red
- Horizontal Italian (3 horizontal blobs) => Violet
- Unknown => Coral

True positive

Here are some of the best results.

	The socket is detected twice. In both cases it's a German/French but it could be more precise than this. Upper bounds are not perfect, there's room of improvements for the line detection		Both sockets are detected correctly. The one on the left is an 'Italian', while the one on the right is an 'Unknown'. The right one has some blobs detected on the top and on the bottom which the program is not able to manage.
	Socket correctly detected. The colour indicates it as an 'Horizontal Italian'.		The wall in the background now has some texture which could affect the line detection, but the socket is correctly detected as a 'German/French'

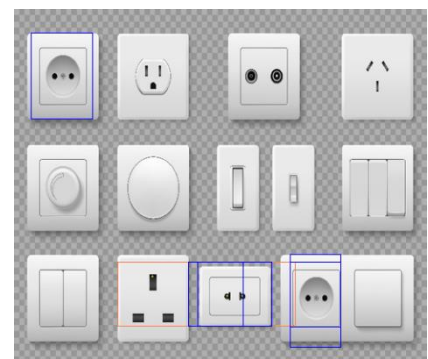
	Socket correctly detected as German/French.		The socket is detected with as 'Unknown'. It is like the British but the opening step in the blob detection is separating the dark circle from the rest, creating a fourth blob and allowing the program to differentiate it from a British
	Socket correctly detected, with some noise in the background which is not affecting the result		Socket correctly detected as German/ French. The background has diagonal line which are correctly ignored.
	British socket correctly detected, but the image was very simple.		The wall is now creating a lot of noise for the line detector. The quality of the result is now much lower. Moreover, the socket on the left is not detected because the vertical line on the left has a low contrast with the background.




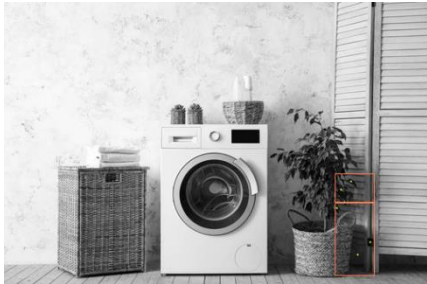

The program is also capable of managing grids of sockets, but the precision gets affected a lot if they're not precisely aligned.

Some German sockets are detected correctly, but the big amount of horizontal and vertical lines are creating meaningless results. It is worth to notice that the blob analysis is preventing from the detection of a lot of possible false positive (the switches).

False positive

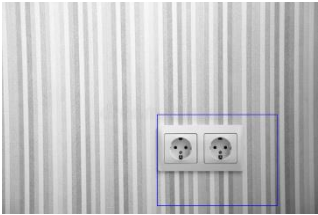
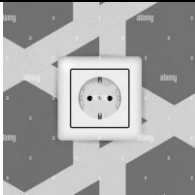


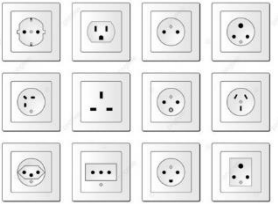
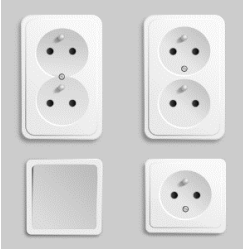


Testing the program on some different objects, it looks like the program may have some confusion but usually the number of false positives is limited. The image on the right is a clear example, but I wanted to try with a couple of different objects and see the results:



Washing machine, there's a rectangle and some dark regions inside. No socket is detected		Unfortunately, there is some confusion on the top, but the washing machine is not detected as a socket.	
A digital camera has a similar shape, but some colours might be different. No socket is detected		The washing machine is again not detected, but there's some confusion on the right. Darker areas seem to create some problems if any rectangle is detected around them, because it is easy to have the barycentre of the blobs and of the rectangle to be close enough. A properly executed blob detection with NLoG would also solve the problem.	
In this case the algorithm completely failed. The green dots are the blobs, and their barycentre is close enough to centre of the rectangle to get a false positive. To prevent it, I should probably have stricter requirements on the blob disposition, looking for some sort of symmetry before allowing the detection of the object to be complete. Again, using NLoG to detect blobs would probably solve the problem			

False negative

The weakest point of the program is probably the line detection, which is causing a lot of sockets not to be detected. This is more frequent when there's a lot of noise in the background and when there's a low contrast between the socket edge and the background. The program is also weak in detecting power stripes because the short side of the stripe is usually difficult to locate, and no socket can be detected without the presence of a rectangle.

	A lot of noise leads to strange result. 2 sockets should be detected. (This is not really a false negative but it helps to show line detection problem)		Lines in the background are easier to detect than the socket.
	Short edges are not detected, long edges are detected although there's a partial occlusion		Vertical edges are not detected
	Low contrast image and line detection is not working properly		Some edges are again not detected
	A lot of noise in the background and the rectangle is not located		Background leads to a diagonal grid => socket is now impossible to find

Based on the results I obtained, this is the object table:

<i>Criterion</i>	<i>Possible Values</i>
Minimum/ Maximum size	Only input image restriction (512 x 512), the biggest the socket the biggest the probability to detect the edges
Lighting variations	Can also work with sub-optimal lighting, contrast with the background is more important. Indoor is usually preferred, but it is not mandatory.
Rotation Variation	Socket detection should be stable on 360 variations (but I found no image to prove it), socket recognition only supports upright rotation.
Occlusion	Is possible if it doesn't affect the detection of the line (Partial). Rectangles are built on top of lines, not on segments, so a partial occlusion is allowed
Other	Partially robust to noise, colour resistance, object must be view frontally.

Conclusions

To make the pipeline better, here are some ideas that could be developed in the future:

- use NLoG for the blob detection.
- Improve line detection with automatized parameters for functions.
- Use a local thresholding in Hough Transform to better identify peaks (right now higher peaks push lower peaks under the threshold with the normalization process)
- Normalize the Hough Transform result on the length of the line to detect power stripes.
- Using 2 parallel lines with gradient in opposite directions for deeper evaluation of what's between the lines (long lines are often detected but the shorter lines aren't and it's therefore impossible to find a rectangle)