Computer vision - Homework 1

Nicolas Six

February 5, 2018

Question 1 Edges detection

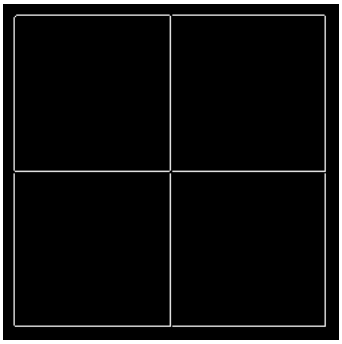


Figure 1: Edge image

Question 2 Hough line detection

The accumulator is a square array of size d, with d the source image's diagonal length. One of the dimension of the accumulator must be of size d as the line are described in a polar way. We chose to code the angle on the same number of value to keep a consistent precision between them.

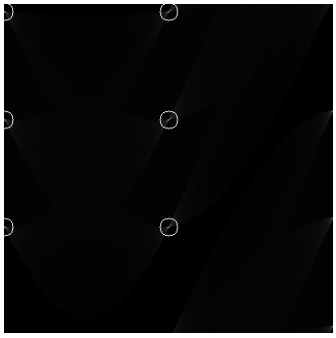
Question 3 Line on noisy image

3.a Smoothing

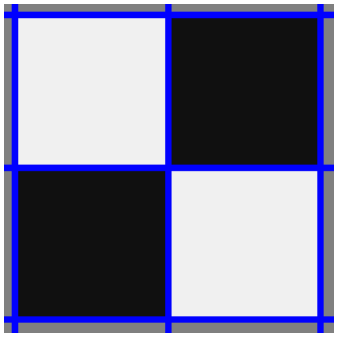
3.b Edges detection

3.c Hough lines

To have this result we had to adjust the threshold of the line detection as well as smoothing to prevent the Hough method to find unreal lines.



(a) Accumulator



(b) Line detected

Figure 2: question 2 results

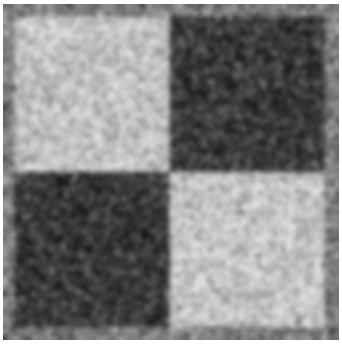
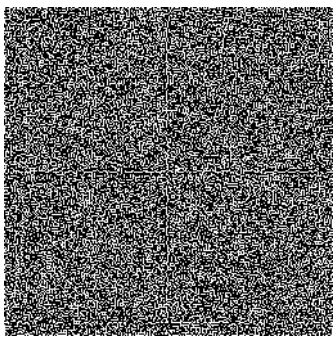
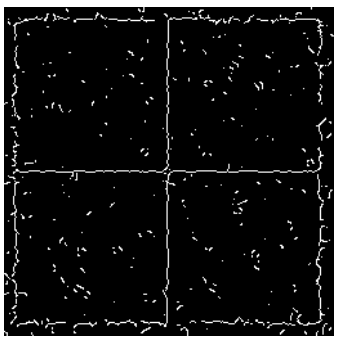


Figure 3: Smoothed image



(a) Edges on noisy image



(b) Edges on smoothed image

Figure 4: Edges detected

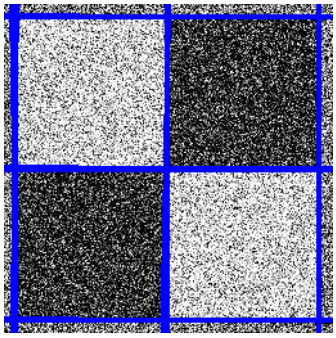
Question 4 Easy real image - lines

4.a Smoothing

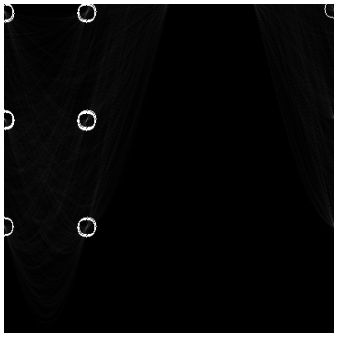
4.b Edges

4.c Hough line detection

This question needed very little adjustment of the the threshold mostly.



(a) Accumulator



(b) Line detected

Figure 5: question 3.c results

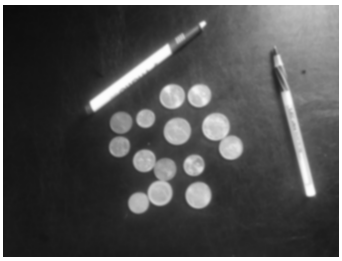


Figure 6: Smoothed image

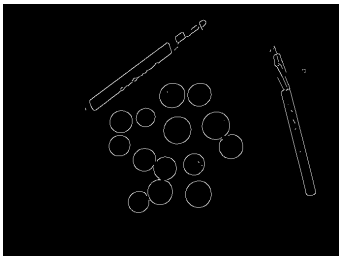
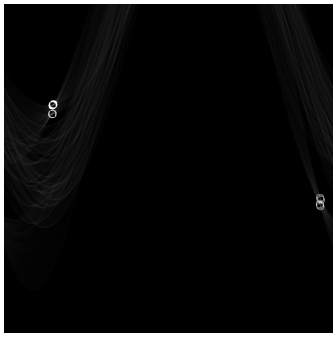


Figure 7: Edges image

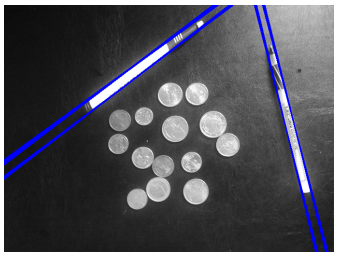
Question 5 Circles - easy

The Hough circle detection is very similar to the line detection, but with a 3D accumulator. To remove a nest for loop, we used the openCV function to draw a circle. This trick worsen the theoretical complexity but is more effective in python as both the circle draw and the plan addition are implemented in C.

```
1 for x, y in zip(edges_points_x, edges_points_y):
```

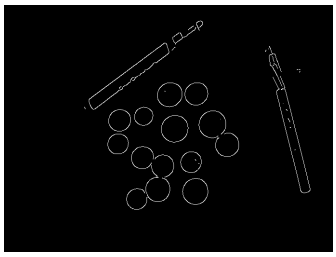


(a) Accumulator

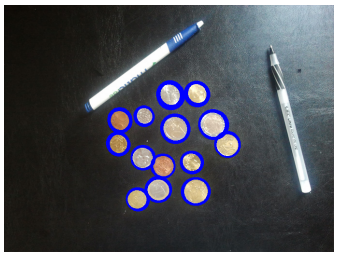


(b) Line detected

Figure 8: question 4.c results



(a) Edges image



(b) Circle detected

Figure 9: question 5 results

```
    for r in range(min_radius, max_radius):
        plane = np.zeros((h, w))
        cv2.circle(plane, (x, y), r, (1, 0, 0), 1)
        acc[r - min_radius] += plane
```

Question 6 More realistic images - Lines

6.a Line finder

6.b Problems present

Our main problem here, is that even if all the pen's lines are found other lines from the background are too. But, those background lines are sharper and longer than the one of the pens and thus thresholding is not enough to select only the pens lines.

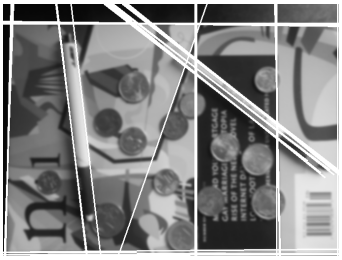


Figure 10: Line found

6.c Line finder - Bis

Figure 11: Line found

Question 7 More realistic images - Circles

7.a Circle finder

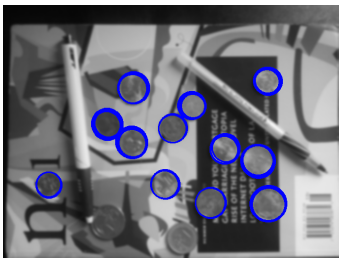


Figure 12: Circles found

7.b Face alarm

As it can be seen an Figure 12, their are no false alarm, but only twelve of the fourteen circles are found. To achieve that result, we mainly fine tuned the existing parameters such as the edges detection threshold and circle detection threshold as well as the maximum and minimum size of the circle. Another approach that we tried is to divide the number of vote for a circle by the radius of the given circle. The goal here is transform the threshold from the number of vote for a center, to the proportion of point of the circle voting for this center, and so have a threshold that manage both big and small radius.

Question 8 Distortion

8.a Line and Circles

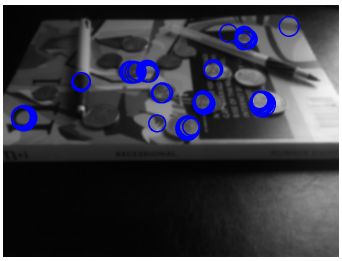


Figure 13: Circles and lines found

8.b How to fix circle problem

The most reliable way to fix the circle problem is to transform our circle finder into an ellipse finder. Doing so, we will still be able to find the wanted circles even when not exactly taken from above.

One way to do so is to add two others parameters, for example the second radius of the ellipsis and its orientation and have an 5D accumulator. If this solution probably work, it's far from an optimal solution in term of computation complexity. This complexity can probably be reduced using the local gradient, but it may still be time consuming. Another simplification can be that every circles are distorted in the same way, such as the angle of the ellipsis and the rapport between the two radius are constant. By fixing those parameters we came back to the same complexity than the classic circle finder.

8.c Fixing circle problem

But we do not chose to apply such a strategy to fund the circles. Instead we chose a more lazy way in term of implementation. As we know that the only problem we have to solve is that the image do not came from the right angle, but the plan on the image is still a plane. We can take advantage of the OpenCV function to transform the image such as circle look like circle again, in other word, transform the picture to make it look (a bit) like it was taken from above. The result is displayed on Figure 14.

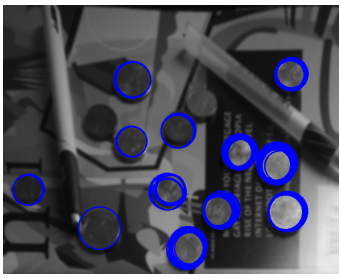
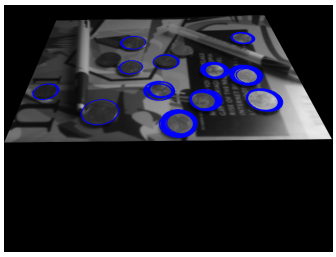
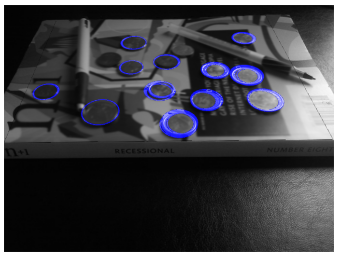


Figure 14: Circles found

It is easy to argument that the picture doesn't look like the original one, which can by corrected by applying the inverse transform (Figure 15a), and by adding back the previously discarded pixels (Figure 15b).



(a) Back in the original perspective



(b) Add original background

Figure 15: Distortion detection Final result