Image processing for extracting features

Clément Chenevas-Paule, Alexandre Audibert,
Daria Senshina, Roman Kozhevnykov

2020/2021

Contents

1 Presentation of the problem

1.1 Context

Over the last fifteen years, thousands of people were killed by rockfalls incidents all around the world. In January 1962, a total of 2,000 people died as a result of a landslide in Peru, making this natural phenomenon an international disaster that needed to be overcome. Rockfalls are quantities of rock falling freely from a cliff face causing rock cuts for highways and railways in mountainous areas. They result in disrupting road traffic in the mountainous regions creating an unsafe environment for people. Climatic and biological events are behind rockfalls: an increase in pore pressure due to rainfall infiltration, erosion of surrounding material during heavy rain storms, chemical degradation, root growth or leverage by roots moving in high winds are all in the panoply of environmental events that results in rockfalls. There are many factors that affect the severity of a rockfall such as the slope geometry, the surface material and its retarding capacity, the size and shape of the rock, the coefficients of friction of the rock surfaces and many others. Their study is therefore important as it allows us to define secure areas for building houses and roads

1.2 Study from experinental outcomes and acquisition of images

One of the way to study this rock falls is based one experimental trials done over a simple scaled model that is supposed to mimic some possible geographical topography. All the trials were realized in laboratory and in various conditions (a combination of the environmental conditions defines each setup of

the experience). The experimental process is the following : a collection of n identical objects are throw from the top of the slope and a photo is taken of the deposit area. The scaled model is built as follows :

Figure 1: Experimental setup from different angles.

1.3 Image processing

Up to now each photo was treated partially manually to produce a file where were collected the positions of thrown objects in a .csv file, and some other information describing the setup of the trial. Each photo is treated with logiciel ImageJ and when clicking over each barycenter of the objects the coordinates of each click are given. Up to now 20 different setups where tested and for each of them 50 launches were done so that there are 50 photos to be treated in each setup. An example of a photo (file.jpg) and R-plot of the points given a file.csv (where are collected coordinates of the brick's centers) are given below :

The scaled model permits quite a lot of variations of the setup of the experimental trials hence it becomes necessary to develop a completely automatic process for the treatment of each photo.

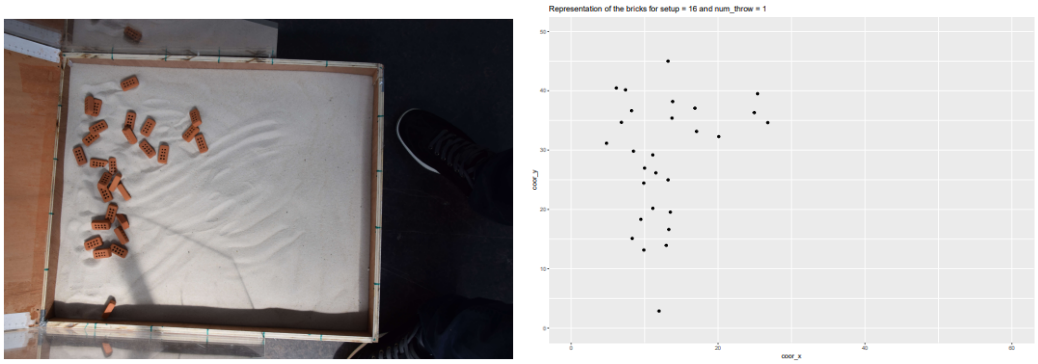


Figure 2: Example of (from left to right): initial photo and plotted brick centers.

1.4 Goal of this work and expected results

Hence the main goal here is to built a tool that given a photo in format jpeg or JPG will automatically produce a .csv file with coordinates of the objects stopped in the deposit area. In the following figures are shown the photo of one of the trials and the representation of the the barycenter's positions of the thrown objects that have arrived in the deposit and photographed array. Moreover it would be necessary to resize and adjust all the photos on the same rectangular pattern and possibly apply transformations to correct the parallax effects. Some particular focus will be done over the situation where one brick is overponed on another one. Even if some objects are mixed up it is also important to know how many of them could be superimposed.

1.5 Presentation of the report

First of all, this report will present the main algorithms of image segmentation, edge detection, thresholding, ...

Secondly, this report will present how we use these famous algorithms to solve our problem.

2 Canny Edge Detection

The Canny edge detector is an edge detection operator The Process of Canny edge detection algorithm can be broken down to 5 different steps:

1. Apply Gaussian filter to smooth the image in order to remove the noise
2. Find the intensity gradients of the image
3. Apply non-maximum suppression to get rid of spurious response to edge detection
4. Apply double threshold to determine potential edges
5. Track edge by hysteresis: Finalize the detection of edges by suppressing all the other edges that are weak and not connected to strong edges.

2.1 Noise reduction by Gaussian filter

Mathematical operation like gradient are really sensitive to noise on the image. Then the goal of this step is to delete noise in order to avoid false detection. To reach this goal, the algorithm use a Gaussian filter. For that, we use a convolution between the image I and a gaussian which takes in argument x, y and σ :

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

where

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} exp(\frac{-(x^2 + y^2)}{2\sigma^2})$$

with :

- L is the smooth image.

- The operator * is the convolution operator.
- G is the gaussian smooth operator.
- x and y are the coordinates.
- σ is the quantity of smooth. More σ is big, more the smooth is big.
- I is the picture.

The convolution operator between an image and a kernel can be computed thanks to the next formula :

$$g(x,y) = \omega * f(x,y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} \omega(dx,dy) f(x+dx, y+dy)$$

where $g(x,y)$ is the filtered image, $f(x,y)$ is the original image,ω is the filter kernel. Every element of the filter kernel is considered by $-a \leq dx \leq a$ and $-b \leq dy \leq b$.

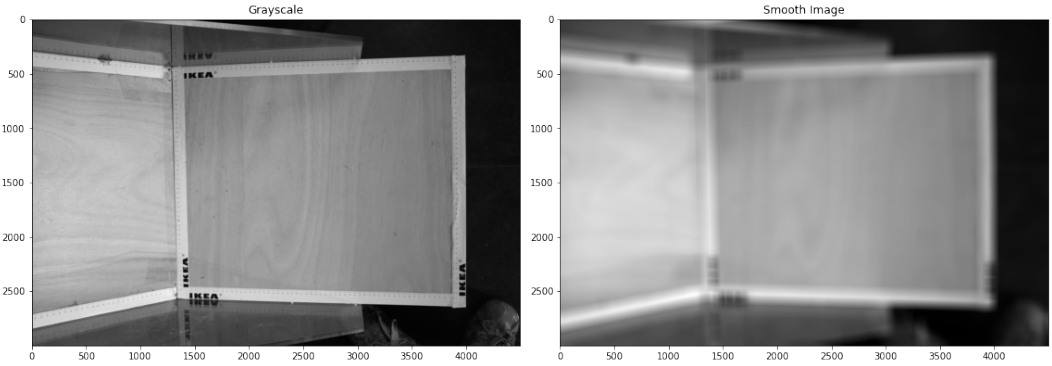


Figure 3: Example of (from left to right): photo until gaussian noise applying; photo after applying of gaussian filter of size 100 with $\sigma = 1000$.

We applied a gaussian filter of size 100 with $\sigma = 1000$ on the left picture, the result appear on the right.

We voluntarily chose tremendous parameters to well visualize the effect of a gaussian filter but in reality we choose more reasonable parameters.

2.2 Finding the intensity gradient of the image

Let's use a Sobel filter to compute edge intensity and direction.

$$K_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, K_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

$$I_x = K_x * I$$

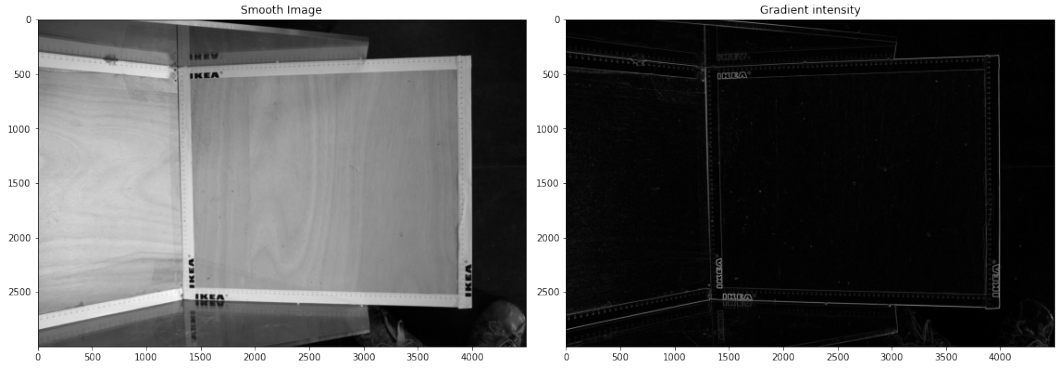
$$I_y = K_y * I$$

The magnitude and the direction of the gradient is given by the formulas :

$$|G| = \sqrt{I_x^2 + I_y^2}$$

$$\theta(x, y) = atan2(I_y, I_x)$$

These values are saved in 2 matrices of the same size of the image.
Let's print the matrix of intensity as an image.

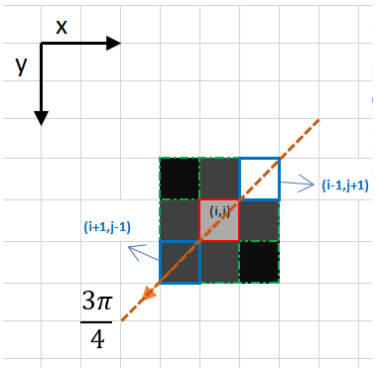


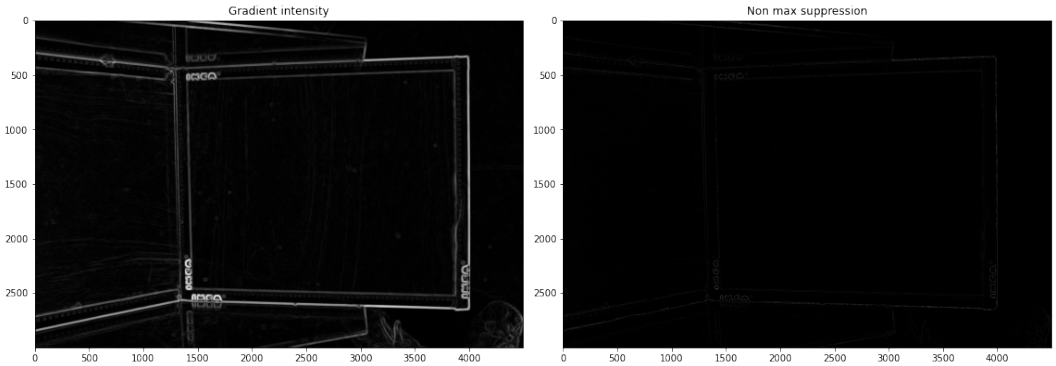
2.3 Finding the intensity gradient of the image

The gradient map obtained previously provides an intensity at each point of the image. A high intensity indicates a high probability of the presence of an outline. However, this intensity is not sufficient to decide whether a point corresponds to an outline or not. Only points corresponding to local maxima are considered to correspond to contours, and are retained for the next stage of detection.

The algorithm for each pixel in the gradient image is:
- Compare the edge strength of the current pixel with the edge strength of the pixel in the positive and negative gradient directions.
- If the edge strength of the current pixel is the largest compared to the other pixels in the mask with the same direction (e.g., a pixel that is pointing in the y-direction will be compared to the pixel above and below it in the vertical axis), the value will be preserved. Otherwise, the value will be suppressed.

In this example, we consider the pixel (i, j).
Darker is the color of a pixel, greater is the intensity gradient.
The broken line represent the direction of the gradient of the considered pixel.
Toward this line, we can see that the pixel (i, j) hasn't the biggest intensity, but the pixel $(i + 1, j - 1)$ has it.
As a result, we suppress the pixel (i, j) because it is a non maximum.



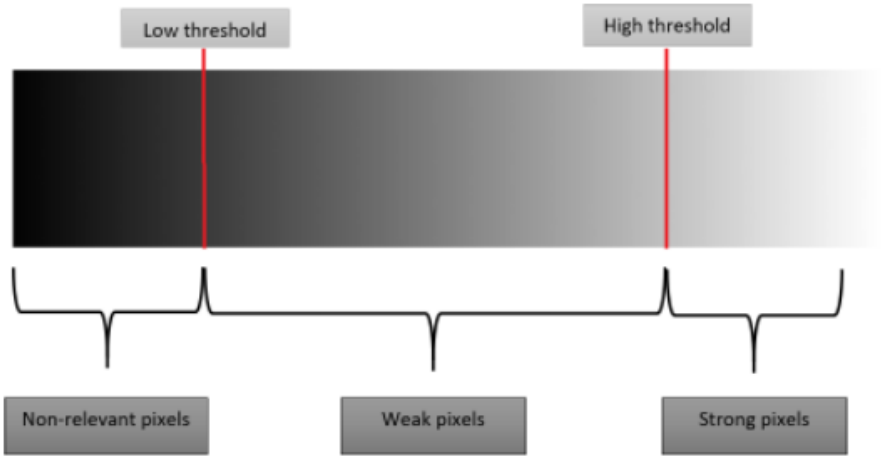


Here an application of the non maximum suppression on our image.

2.4 Double threshold

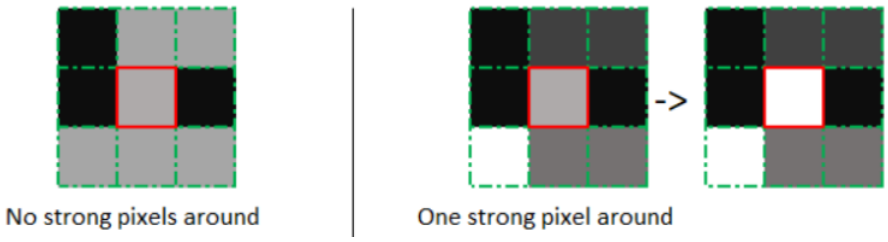
After application of non-maximum suppression, remaining edge pixels provide a more accurate representation of real edges in an image. However, some edge pixels remain that are caused by noise and color variation. In order to account for these spurious responses, it is essential to filter out edge pixels with a weak gradient value and preserve edge pixels with a high gradient value. This is accomplished by selecting high and low threshold values. If an edge pixel's gradient value is higher than the high threshold value, it is marked as a strong edge pixel and it is conserved. If an edge pixel's gradient value is smaller than

the high threshold value and larger than the low threshold value, it is marked as a weak edge pixel. If an edge pixel's gradient value is smaller than the low threshold value, it will be suppressed. The two threshold values are empirically determined and their definition will depend on the content of a given input image.



2.5 Edge tracking by hysteresis

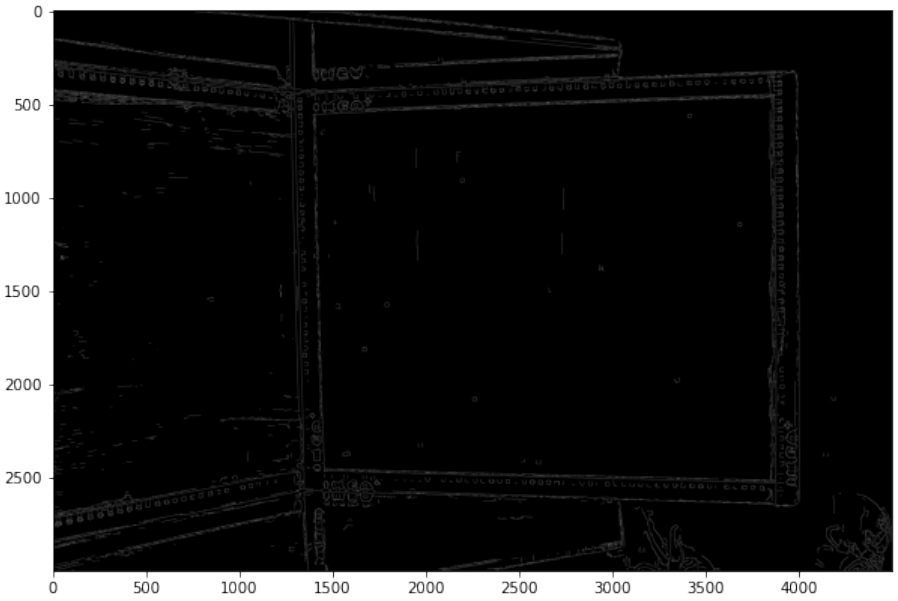
Based on the threshold results, the hysteresis consists of transforming weak pixels into strong ones, if and only if at least one of the pixels around the one being processed is a strong one.



Here appear 2 possible examples.

White represents strong pixels, black represents non relevant pixels and gray represents weak pixels.
On the left, the considered pixel (which is of course a weak pixel) has no strong pixel in his neighbourhood. As a result it turns to a non relevant pixel.
At the opposite, on the right example, a strong pixel belongs to the neighbourhood of the considered one so it turns to a strong pixel.

In our image this gives us :



This picture is a binary image in which white pixels are considered to belong to a contour whereas black pixels don't belong to a contour.

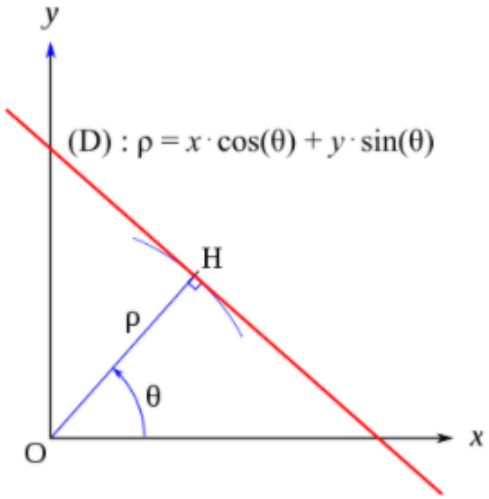
3 Hough transform

The aim of the algorithm of Hough is to detect the lines of a picture, but it can also to detect circles or even other forms.

3.1 Line detection

3.1.1 Hough space

Classically, a line is represented by its equation $y = ax + b$ but if the line tends to the vertical, a will tend to $+\infty$. To avoid this problem, we represent a line thanks to its polar coordinates ρ and θ.



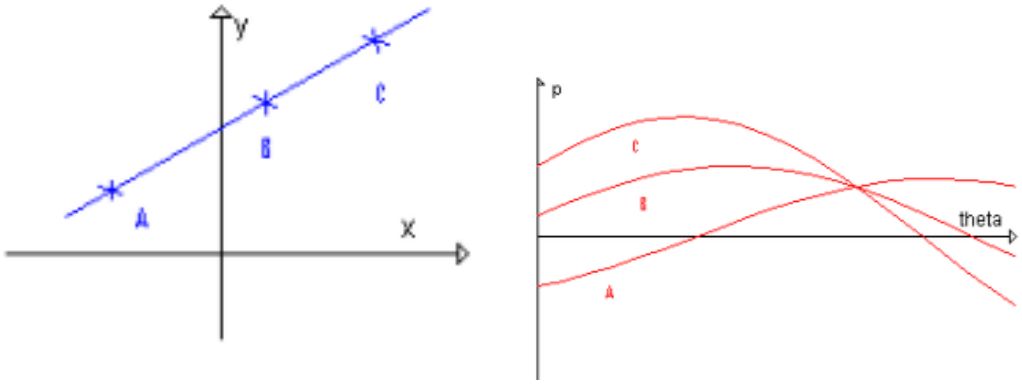
source : `https://fr.wikipedia.org/wiki/Transform%C3%A9e_de_Hough`

$\{(\rho, \theta)\}$ is then call Hough space.
In the Hough space, a line is obviously represented by a point and the set of all lines going through a given point is a sinusoidal curve.

Demonstration :
We fix (x_0, y_0) a point in the cartesian plan.
We are searching for the collection of all the (ρ, θ) representing a line going through (x_0, y_0).
$\rho = x_0.cos(\theta) + y_0.cos(\theta) \Rightarrow \rho = \sqrt{x_0^2 + y_0^2}(\frac{x_0}{\sqrt{x_0^2 + y_0^2}}.cos(\theta) + \frac{y_0}{\sqrt{x_0^2 + y_0^2}}.cos(\theta)) \Rightarrow$
$\rho = \sqrt{x_0^2 + y_0^2}(cos(\phi)cos(\theta) + sin(\phi)sin(\theta)) \Rightarrow \rho = \sqrt{x_0^2 + y_0^2}cos(\theta - \phi)$

As a consequence, the collection of all the (ρ, θ) representing a line going through (x_0, y_0) is :
$\{(\rho, \theta) \in \mathbb{R} \times [0, \pi] | \rho = x_0.cos(\theta) + y_0.cos(\theta)\} = \{(\rho, \theta) \in \mathbb{R} \times [0, \pi] | \rho = \sqrt{x_0^2 + y_0^2} cos(\theta - \phi)\}$ which is a sinusoidal curve in the Hough space.

In the next example, red curve A represents the set of all lines going through the blue point A. It is the same for B and C.



source : `https://fr.wikipedia.org/wiki/Transform%C3%A9e_de_Hough`

The 3 red lines cross in the same point, this point represents the unique line which going through A, B and C.
We will use this principle in the Hough's algorithm.

3.1.2 Hough alorithm

We assume that a line of the image make part of a contour so the first step is to find a binary image with contours in white and other pixels in black. The Canny algorithm is well adapted for achieve this task, but other methods like gradient can be used.

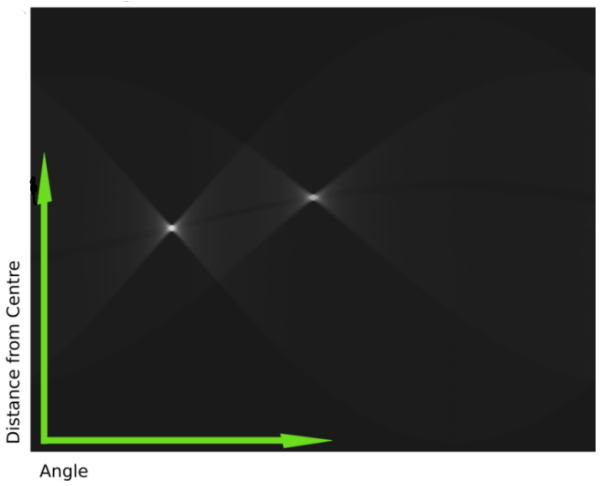
The Hough's transform algorithm uses a 2-D matrix called accumulator. This accumalator is a discretisation of the Hough space.

Algorithm 1 Hough's algorithm

$I \leftarrow$ image
$J \leftarrow$ contours of I (by using Canny for instance)
$\delta \leftarrow$ discretisation step
$M \leftarrow$ accumulation matrix (discretisation of Hough space with step δ)
for $(x, y) \in J$ **do**
  for $\theta \in [0, \pi]$ with step δ **do**
    $\rho = x.cos(\theta) + y.sin(\theta)$
    $M[\rho, \theta] += 1$
  end for
end for
return M

Then we can represent the accumulator thanks to the next figure, more a point is white, bigger is its value in the accumulator.

In the next example, easily understand than this accumulator represents 2 points in Hough space so the original image contains 2 lines.



source : `https://en.wikipedia.org/wiki/Hough_transform`

In true images, the accumulator matrix is more complex than the previous one and we have to define hyperparameter which designs the threshold above

which a point in the accumulator can be consider as a line in the original space.

3.2 Circle detection

We sow how we can use Hough algorithm to detect lines in an image. But in reality, we can use the same process for each form which can be represented by a finite number of parameters. This number of parameters is the dimension of our accumulator.

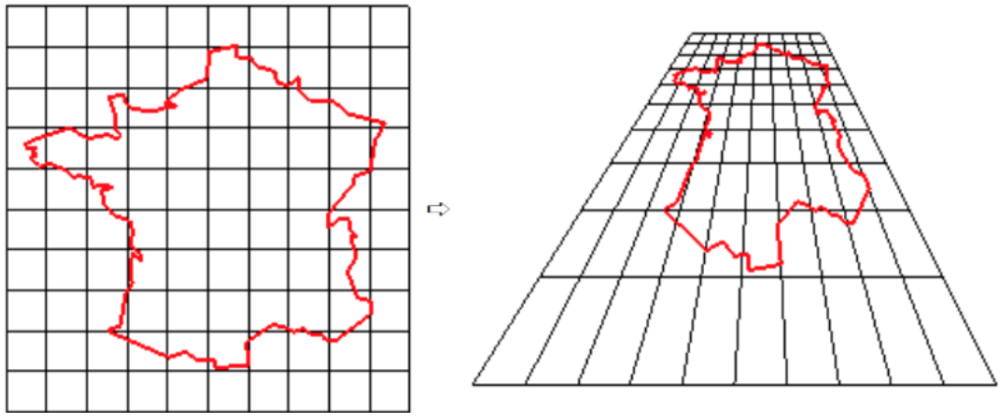
Let's take the example of circles.

We know that a circle can be represented by 3 parameters. a and b such that (a, b) is the center our the circle, and r its radius. (Recall that the circle equation is $(x–a)^2 + (y–b)^2 = r^2$).

Then the process is the same but this time the accumulator is a discretisation of all the possible a, b and r.

4 Homography

In our case, we have many parallaxes problems. To solve this issue, Homography is used. Example of parallax issue:



As reminder any two images of the same planar surface in space are related by a homography.We want to use this theory to return the tray to its original shape, i.e. a rectangle. Before that It?s necessary to introduce an important mathematics notion, which is Homogenous coordinates.

This notion make it possible to characterise transformations in space with matrix. Basically A point in cartesian coordinates, (X,Y) becomes (X,Y,W) in homogeneous coordinates.

$$(x, y) \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

homogeneous image coordinates

Converting *from* homogeneous coordinates

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} \Rightarrow (x/w, y/w)$$

In our case, transforming something like a parallelogram into a rectangle is called an affine transformation. This transformation is obtained by applying the following matrix:

$$H = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix}$$

.

We want to find the matrix H such that for all pixels (x,y) in our picture where our tray is rectangular and (x_0,y_0) corresponding pixel to (x,y) in our original picture:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}$$

We can see that or equation depends on a new a paramter z_0. This parameter has a scaling role. The first step to find all pwith

$$x' = x_0/z_0$$

$$y' = y_0/z_0$$

We can also write :

$$z_0 \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

By eliminating c we can formulate the above equation in the form.

$$\begin{cases} z_0 = gx + hy + 1 \\ x' \times z_0 = ax + by + c \\ y' \times z_0 = dx + ey + f \end{cases} \tag{1}$$

$$\begin{cases} x' \times (gx + hy + 1) = ax + by + c \\ y' \times (gx + hy + 1) = dx + ey + f \end{cases} \tag{2}$$

$$\begin{cases} ax + by + c - x'gx - x'hy - x' = 0 \\ dx + ey + f - y'gx - y'hy - y' = 0 \end{cases} \tag{3}$$

We get :

$$Ah = 0$$

with

$$A = \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -x'x & -x'y & -x' \\ 0 & 0 & 0 & x & y & 1 & -y'x & -y'y & -y' \end{bmatrix}$$

and

$$h = \begin{bmatrix} a & b & c & d & e & f & g & h & 1 \end{bmatrix}^T$$

We can easily conclude that we have 8 unknown variables and if we know the correspondence between two pixels of the two images it gives us two equations. Basically we will use the 4 angles of the plate which are the easiest pixels to recognize.

To summary if you want to obtain our rectangular tray :

Note the function I(x,y) which takes as input the coordinates of a pixel of the target image and returns its colour.

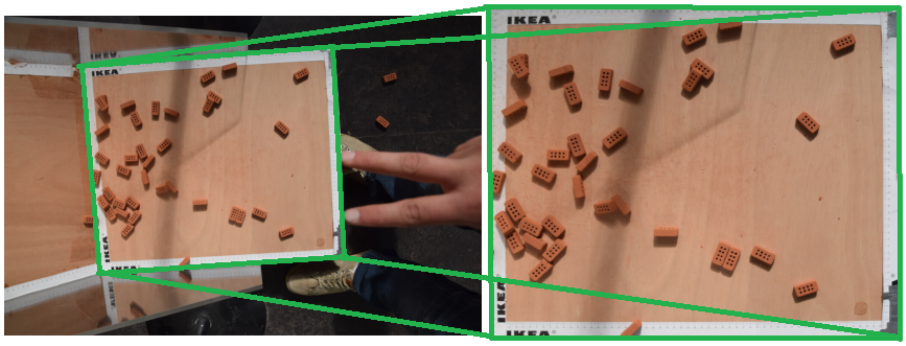
Note the function I'(x,y) which takes as input the coordinates of a pixel of the initial image and returns its colour.

To calculate I(x,y): We use the previous equation :

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}$$

Finally we define I(x,y) like : I(x,y) = I'($\frac{x_0}{z_0}$, $\frac{y_0}{z_0}$)

Some results



5 Thresholding

5.1 Otsu method

The OTSU method is used to perform automatic thresholding based on the shape of the image histogram. This method applies to greyscale images and assumes that these images are composed of two classes which are the background and the objects. This method has for goal to minimize the intra-class variance:

First step is to create an Histogram function. The goal of hist(k) is to count

the number of pixel equal to k.

$$Hist(k) = \sum_{x,y} \mathbb{1}_{I(x,y)=k}$$

Where I(x,y) is the intensity of pixel (x,y)

The goal of our algorithm is to minimize the intra-class variance between our two classes. Note the threshold : t

$$\sigma_w^2(t) = w_1(t)\sigma_1(t) + w_2(t)\sigma_2(t)$$

where:

- $w_1(t) = \sum_{k=0}^{t} \frac{Hist(k)}{N}$ probability to be in class 1 and $w_2(t) = 1 - w1(t)$probability to be in class 2
- σ_i variance of class i.

Algorithm :

1. Calculate histogram function
2. for all possible t $\in \{0..255\}$ compute $\sigma_w^2(t)$
3. Find t which minimizes $\sigma_w^2(t)$
4. Transform or picture in binary image

6 Morphological Transformations

The first step of the morphological transformations is to define a kernel.
As an example, we will take a 5×5 sized kernel.

$$K = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

6.1 Erosion

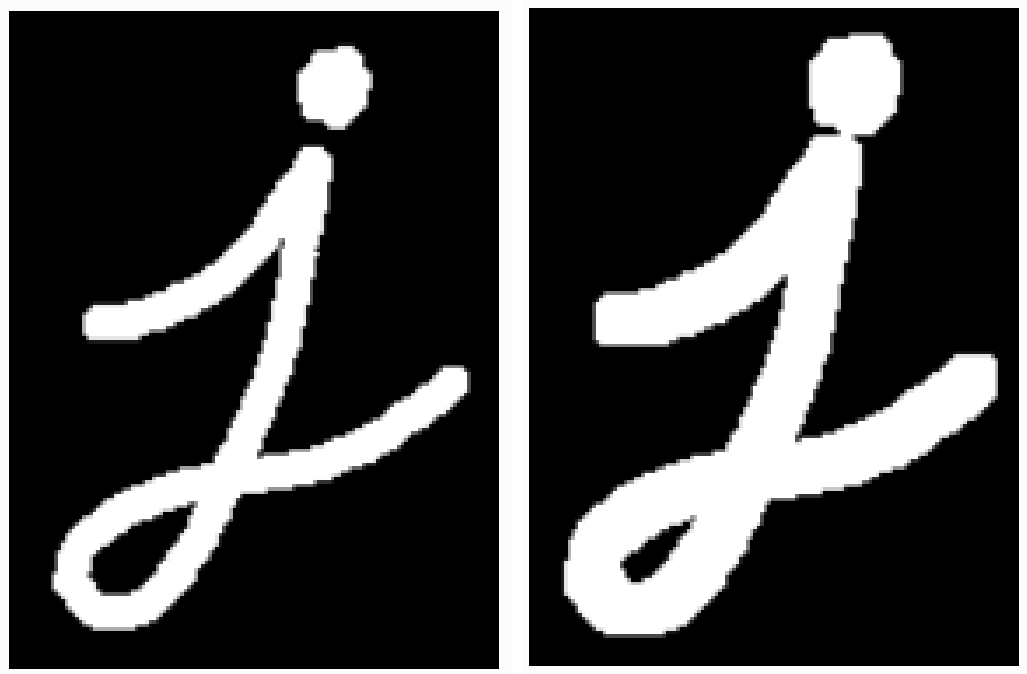
The erosion is used, as its name suggests, to erode an image like in the next example :

source : https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html

The kernel slides through the image (as in 2D convolution). A pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel is 1, otherwise it is eroded (made to zero).

6.2 Dilatation

At the opposite, dilatation dilate a image like in the next example :



source : https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html

It is just opposite of erosion. Here, a pixel element is '1' if at least one pixel under the kernel is '1'. So it increases the white region in the image or size of foreground object increases.

6.3 Examples of use

6.3.1 Opening

This operation consists in respectively execute an erosion, followed by a dilation. To goal of such an operation is to remove noise, as it appears on the next example :

source : `https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html`

7 FindContours

8 Return to the problem

Our problem is complex, because all the photos are really different. In fact, the point of view is never the same, the orientation of the trail can change, the luminosity also.

This is for instance visible in the next photos :


Thus we decide to cut the project in 2 main parts :

- The first part consists in searching for the 4 sides of the trail (or the 4 angles, it is the same) and then apply an homography to extract the trail and suppress the projective effect of the photo.
- The second part consists in searching for the position of the blocks in the extracted trail. Many strategies are tested and described in the next.

8.1 Trail extraction

In this section we are going to present the part of the algorithm dedicated to trail extraction.

8.1.1 Trail corners extraction

The overview of proposed method looking for the corners of trail is following:

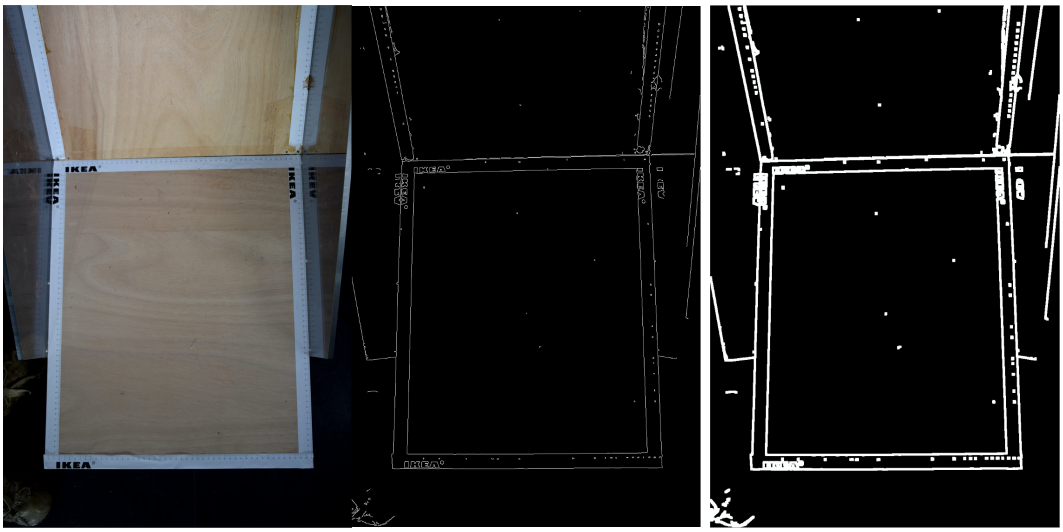
1. Canny edge detection
2. Dilatation
3. Probabilistic Hough transform: search for the most probable line segments
4. Convert image to binary: 0 for resulting lines from previous step, 1 otherwise
5. Delete all connected regions crossed by the edge of the image but the biggest one
6. Skeletonizing
7. Hough transform: search for lines
8. Search for points of intersection
9. Search for the convex hull of these points

Results of each step are presented on the fig. 4. And lets consider the sequence of chosen steps of the algorithm more deeply step-by-step.

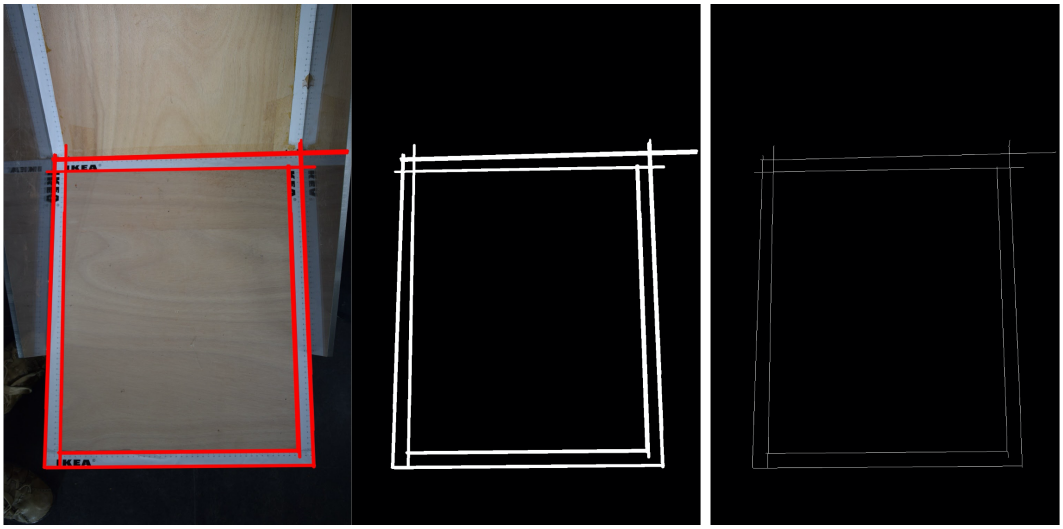
First of all we use Canny edge detection in order to extract all edges from the photo, since some of them are candidates to be edges of the trail. The example of this step can be seen on the fig. 4b.

Then, as well as we obtained all possible candidates for the trail edges, we also obtained some small edges relating to the structures and noise. To suppress at least a little influence of it we apply dilatation with two iterations using the 4×4 kernel.

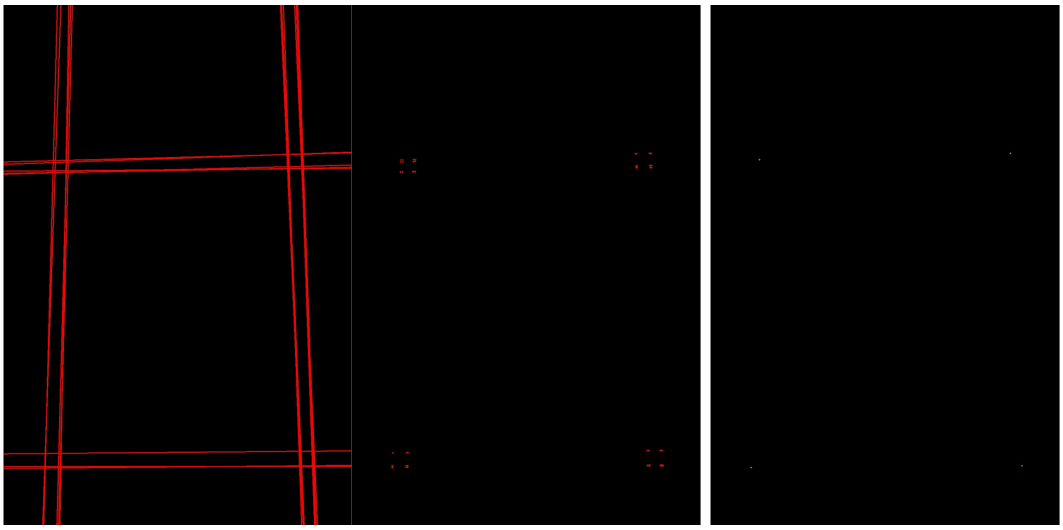
The next step ...



(a) (b) (c)



(d) (e) (f)



(g) (h) (i)

Figure 4: Step-by-step examples of using trail corners extraction algorithm: (a) — initial photo; (b) — after applying Canny algorithm; (c) — after dilatation step; (d) — after probabilistic Hough transform; (e) —- ; (f) — ; (g) — ; (h) — ; (i) — ;.

8.1.2 Homography

Since we for a given photo, we know the 4 angles of the trail the last step remained consists in doing an homography to extract the trail and suppress the projective effect. An example is given on the fig. 5.

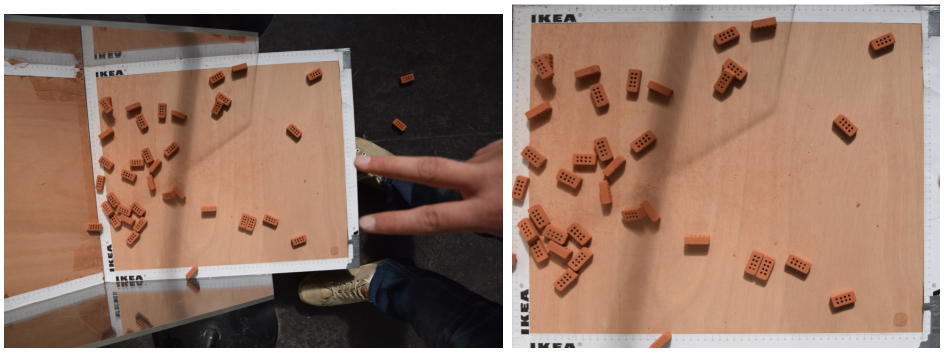


Figure 5: Example of homography algorithm performance (from left to right): before applying homography; after applying homography.

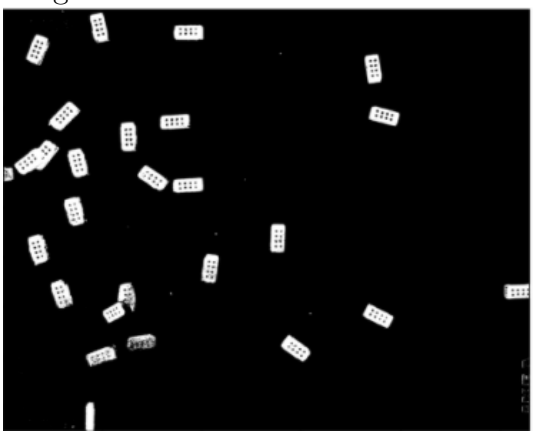
8.2 Blocks detection

The work is now to use the right picture to find and extract the blocks.
Many strategies can be considered, we will have a look on the mains ones we implemented.

8.2.1 Strategy : Threshold and Contours

First mask

The first step of this strategy is simply a thresholding. The goal is to get an image similar to the next one :



To reach susch a threshold, we can simlpy use a color threshold. This threshold is particularly efficient when the trail is well illuminated, whitout shadow and with blocks of color different from the floor.
Another mean to have such a threhold is to use the Otsu algorithm described earlier in this report.

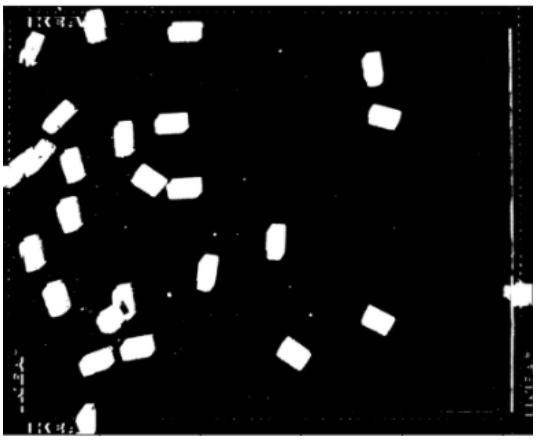
Important remark.

In some case, we dispose of a photo of the trail before the rocks fall. This image can exploted in the same manner than the previous photos. That is to say that we can extract the trail, this will be particularly usefull as we will see soon.

One idea that we can exploit is to substract the empty trail with the full one. This give us the next result :

This operation is particularly interesting because it delete the white band, and other default of the image like the 'IKEA' written on the angular.
The threshold method are now more efficient because with have in the final image 2 main colors.

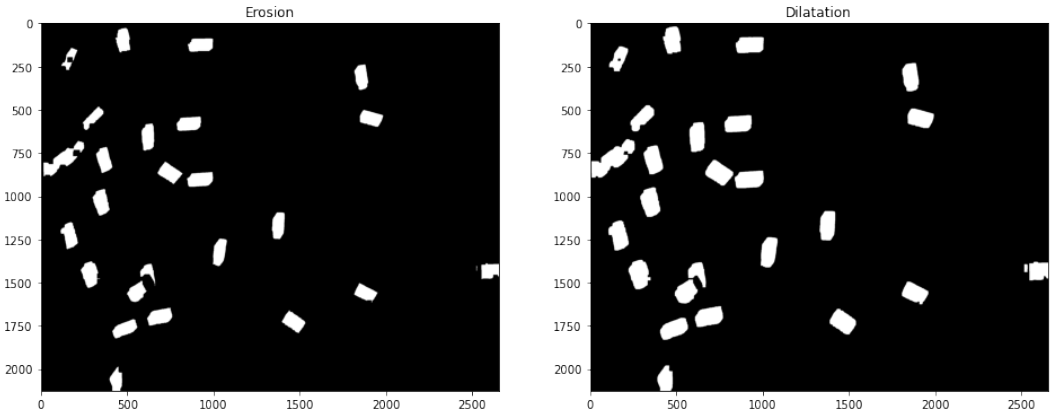
For instance, an Otsu threshold on the previous image will give :



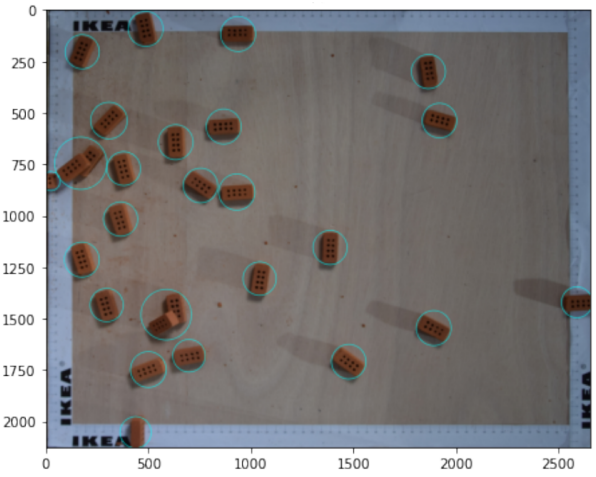
However, you can constat that some defaults exist on the image. These defaults are due to little movements of the camera (in the case of the photo substraction) or threshold errors otherwise.

This issue is easily solved thanks to an opening operation (see morphological

transformations before). If we apply succesively an erosion and a dilation on our image, we get :



Now we have got a mask, the first idea is to extract contours of this mask and to take the more little circle which contain each contours. Such a operation allows us to get contours around the white parts. As an exmple we get :

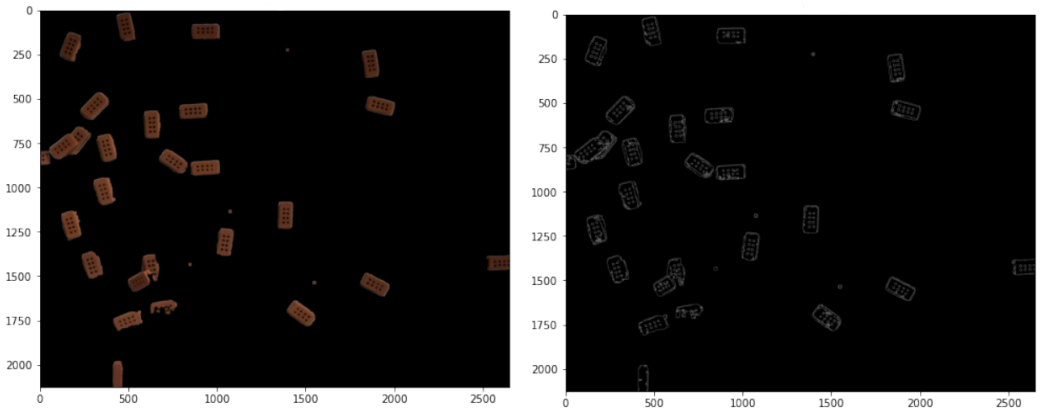


This result is interesting because we are already able to find the isolated blocks. However, when several blocks are regrouped, the circle covers all the blocks of the group. This is not the good result but some peaces of information can be extracted from it. Indeed the radius of the circle is bigger when the number of

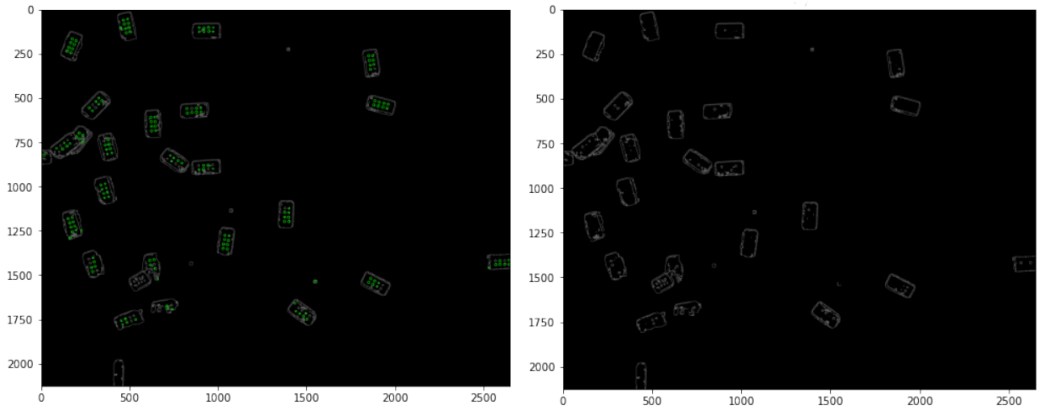
blocks in the group increases. But this is insufficient and we have to continue the process on the image.

Second mask

To separate blocks in a group, the idea is to exploit the fact that the blocks are separated by an edge. Then we will use the previous mask, apply it on the originial image, and then use a Canny edge detector to find the separation between the blocks. Here an example of application of this procedure :

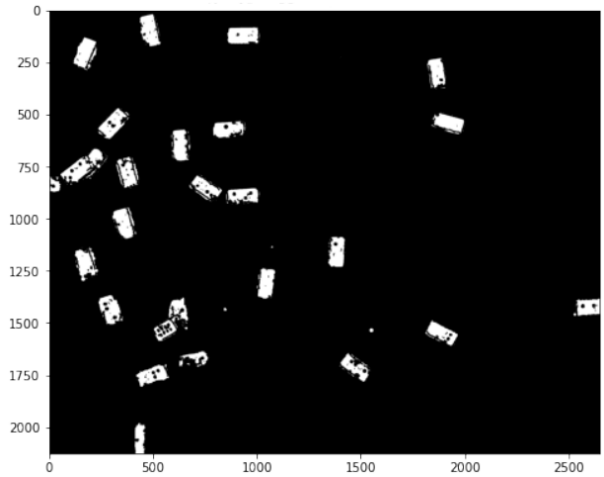


As we can see, the Canny filter extract all the edges ; the edges that we want and also the edges we don't want. The main example of bad detected edges is the circles corresponding to the black hole of the brocks. Nevertheless, we know a method to localize circles on an image, this is the Hough transform for circles that we already talk about. In addition, we can see that all the circles we want to remove have the same size. So let's remove all the detected circles of this size.

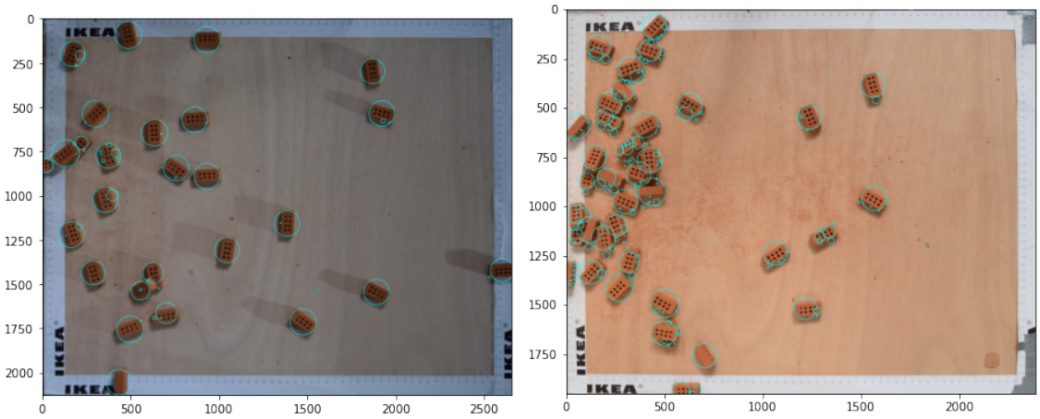


In green appears all the detected circles with the good size, on the right their is the image without the circles.

The next is quite simple, we will invert the last image (the white becomes black and the black becomes white) and add it with the previous mask. In additon, all the detected circles are colored in white. We get a new mask, as we can see below.



If we draw the circles around each contours the new result will be really different from the previous one. This is due to the new mask which is more complexe than the first one. As a example, we get for two different images :

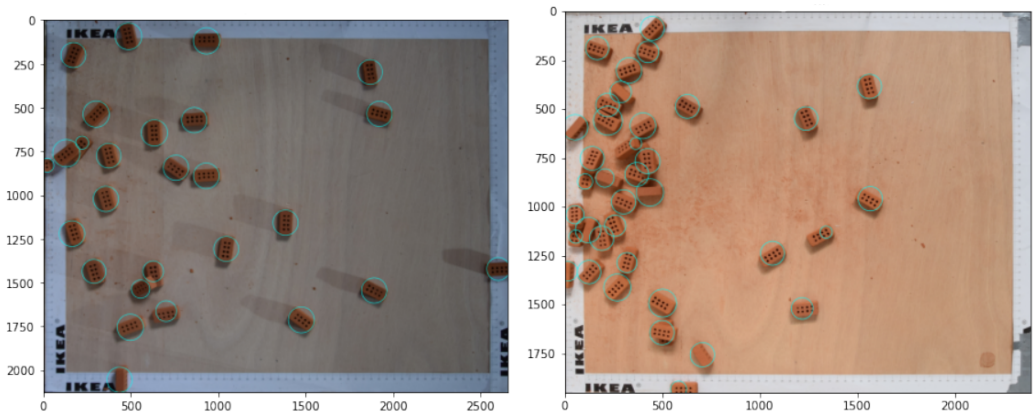


Circles filtering

The problem is now clear, we have too many circles. To solve this issue we have applyied different circles filtering.

- Little circles filtering : we suppress all the circles under a radius threshold.
- Interlocked circles filtering : we can see that a lot of circles are inside a bigger circle, in such a case we suppres it.
- Medium circles reunion : if two circles with medium size are really near, we suppress them and create a new circle which is bigger and centered in the average of the two previous centers.
- Big circles processing : if a circle exceed a threshold size, we consider that it corresonds to two blocks. As a result we suppres it we add the the two biggest circles which were inside the big circle.

If we apply all this procedure to the previous examples, we get the new result :

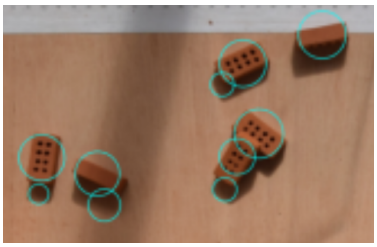


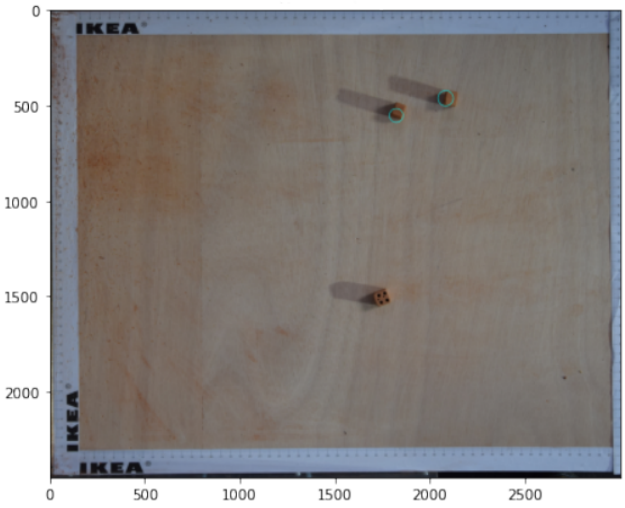
Now we have just to extract the center of each circle.

8.3 Limitations

The procedure described before is only available for the red blocks on a wood tray, but it can be easily updated for other kind of blocks and trays.

In addition, this procedure is not really resistant to shadows as we can see the the next picture.





In the same way, the procedure is not really resistant to low brigtness.

9 Python Application

9.1 Instructions manual

The application contains several repository :

- ./CSV the folder in which the csv files will be saved.
- ./tests the folder contains some photos in order to test the app.
- ./results the folder in which the result images are saved.

The application can be easily launch with the command :
python ./main.py

to do

9.2 Recommendation for the shooting

The methods used in our application are very sensitive to the quality of the photos provided. This remark is true for all image processing methods,that's why we are going to provide a maximum of advice in order to optimise the efficiency of the algorithms used.These remarks apply to three parts of the photo generation: the way the photos are taken, the outdoor environment and the equipment used.

First of all the way the photo is taken is very important. Indeed, the plate must be centred and must not be cut out. In addition, each folder should contain an image without blocks at the beginning, for that the edges of the tray can be identified more easily. Finally, the camera and the tray should not move between photos taken from the same folder.

Contrary to what one might think, the external environment plays an essential role in image processing. The outside of the stage must be uniform in colour and must not have textured objects. Furthermore, the question of lighting is of paramount importance, in fact it must be positioned in such a way that the presence of shadow on the set is minimal, which also implies that the person taking the photos must not create unnecessary shadows. And finally, the brightness must be constant during the printing of the photos.

The last part is not the simplest as it probably requires investment in new equipment. Indeed a uniform colour is strongly desired for the tray and also to have a single uniform colour for the blocks in order to facilitate the different locations. Furthermore, it is strongly recommended not to use objects that are the same colour as the board because they cannot be located. Finally, it is also important to find a way to significantly separate the board from the slope.