# 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 result 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 to define secure areas for building houses and roads

## 1.2 Study from experimental outcomes and acquisition of images

One of the way to study this rock falls is based on 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 :

## 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 logical

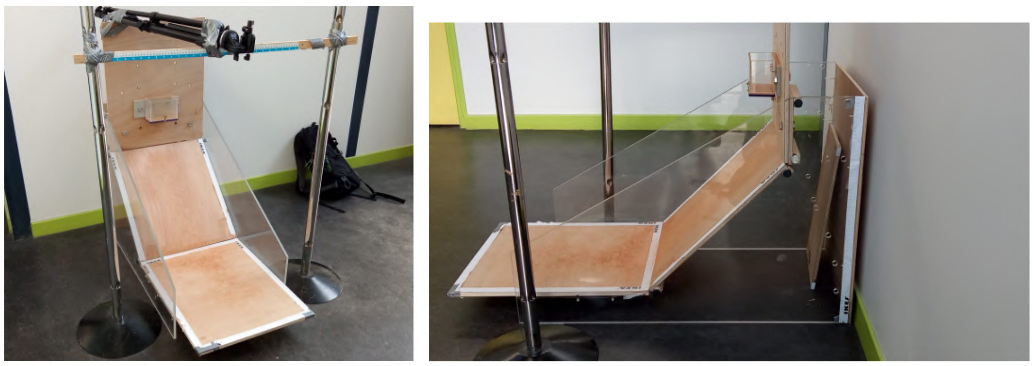


Figure 1: Experimental setup from different angles.

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 (Figure 2) and R-plot of the points given a file.csv (where are collected coordinates of the brick's centers) are given below :

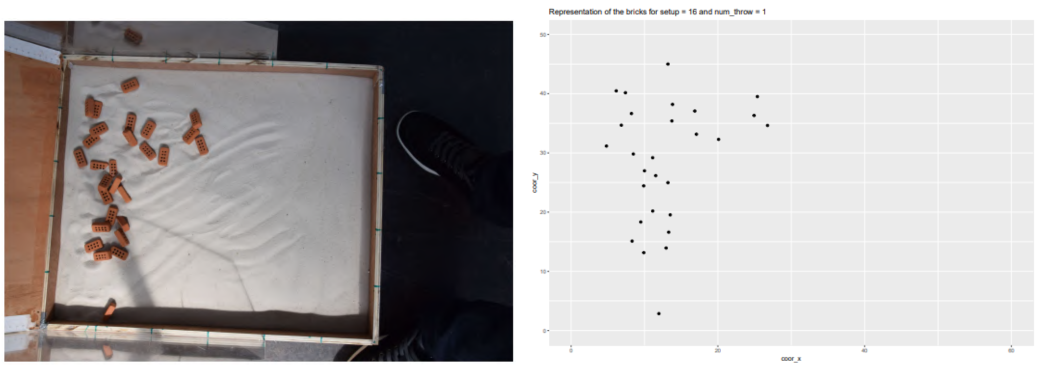


Figure 2: Example of (from left to right): initial photo and plotted brick centers.

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.

## 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 figure 2 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 also 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.

## 2 Organization of the report

- First of all, this report will present the main algorithms of image segmentation, edge detection, thresholding, ... All these methods will next be used to create an application which provides the expected results.
- The second part will consist in describing the principle of the developed application. It is divided in two main sub-algorithms, the first one will focus on the tray extraction and the deletion of parallax effects. To do so we consider 3 main tools which are the Canny edges detector, Hough transform the homography. These methods are described in the section 2. The second sub-algorithm consists in extracting the blocks from the tray. Here different techniques are used like thresholding, morphological transformations and again Canny edges detector which are also described in section 2.
- Then in section 4, we want to analyse the results of the application. A statistical study is lead on the 14 blocks data set.
- Next we will focus on the use of the application, this section looks like a manual of use.
- We finally provide some advises for future shooting in order to make the application work better.

## 3 The tools and methods

In this section, we describe all the tools and methods used in our application. You can refer to this section when methods are quoted in the main algorithm.

## 3.1 Canny Edge Detection

The Canny edge detector is an edge detection operator proposed by John Canny [1]. The Process of Canny edge detection algorithm can be divided into 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 edges, which are weak and not close with a strong edges.

### 3.1.1 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 uses a Gaussian filter. For that, we use a convolution between the image $I$ and a gaussian which takes in argument $x$, $y$ and $\sigma$ :

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

where

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

where :
- L is the smooth image.
- The operator * is the convolution operator.
- G is the Gaussian smooth operator.
- $x$ and $y$ are the coordinates.
- $\sigma$ allows to control the level of smoothing. Bigger is $\sigma$, more important is the smoothing.
- I is the original 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,$\omega$ is the filter kernel. Every element of the filter kernel is considered by $-a \leq dx \leq a$ and $-b \leq dy \leq b$.
Here the size of the filter is the $(2 \times a, 2 \times b)$.

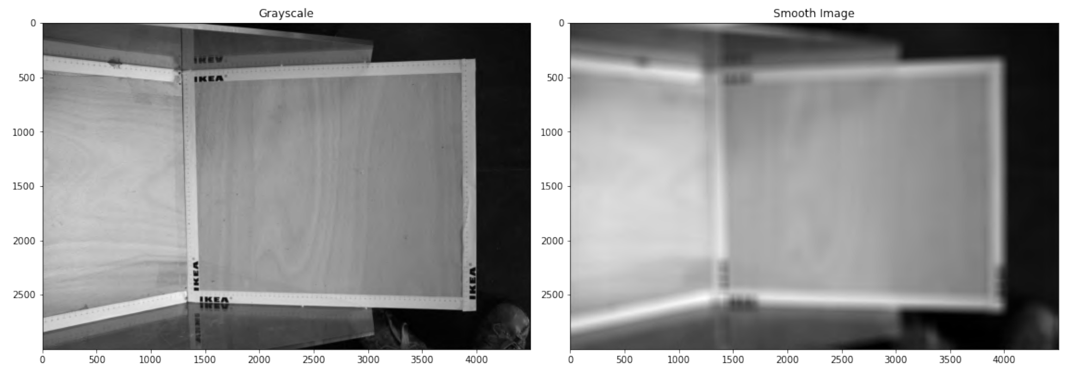


Figure 3: Example of (from left to right): photo until Gaussian noise applying; photo after applying of Gaussian filter of size 100 with $\sigma = 1000$.

We applied a Gaussian filter of size 100 (that is to say $a = 100$ and $b = 100$) 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 which are tested directly on the data set.

### 3.1.2 Finding the intensity gradient of the image

Let's use a Sobel filter [2] 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 are respectively given by the formulas :

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

$$\theta(x, y) = atan_2(I_y, I_x)$$

These values are saved in 2 matrices of the same size of the image.
Let's print the matrix of intensity as an image (fig. 4).

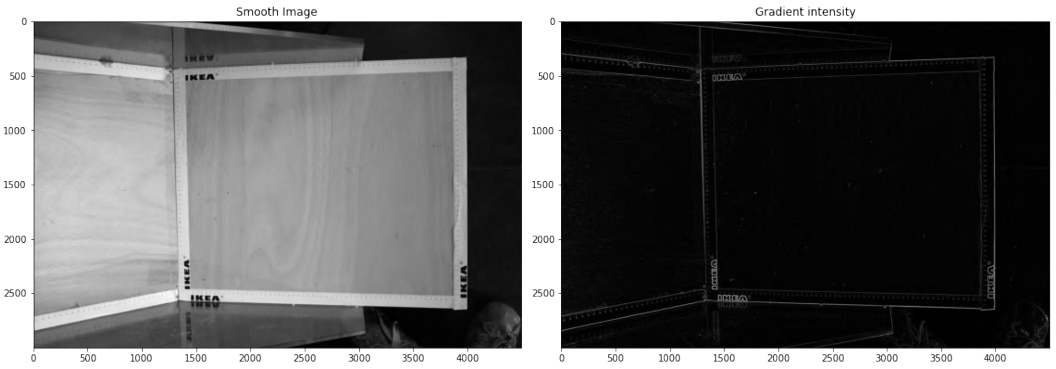


Figure 4: Example of using Sobel filters to edge intensity and direction computing (from left to right): smooth image; obtained gradients intensities.

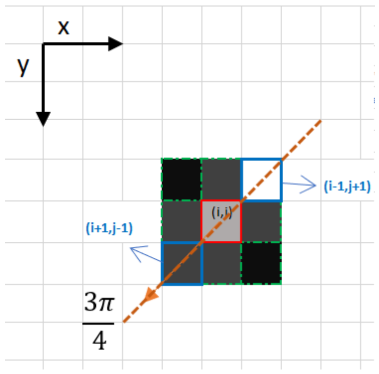
### 3.1.3 Apply non-maximum suppression to get rid of spurious response to edge detection

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

The algorithm for each pixel in the gradient image is:
- Compare the intensity 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 among 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, lower 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 local maximum.



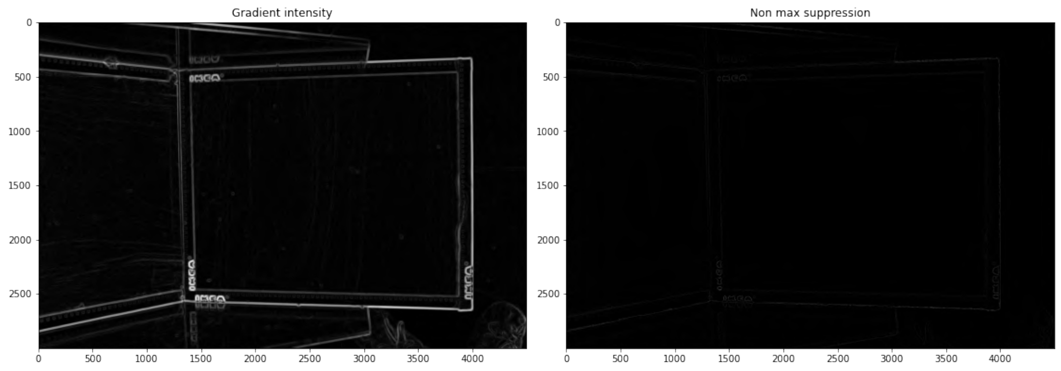


Figure 5: Example suppression of non local maximum (from left to right): before suppression; after suppression.

### 3.1.4 Double threshold

After application of local 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.When 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 (we test different values by hand) and their definition will depend on the content of a given input image.

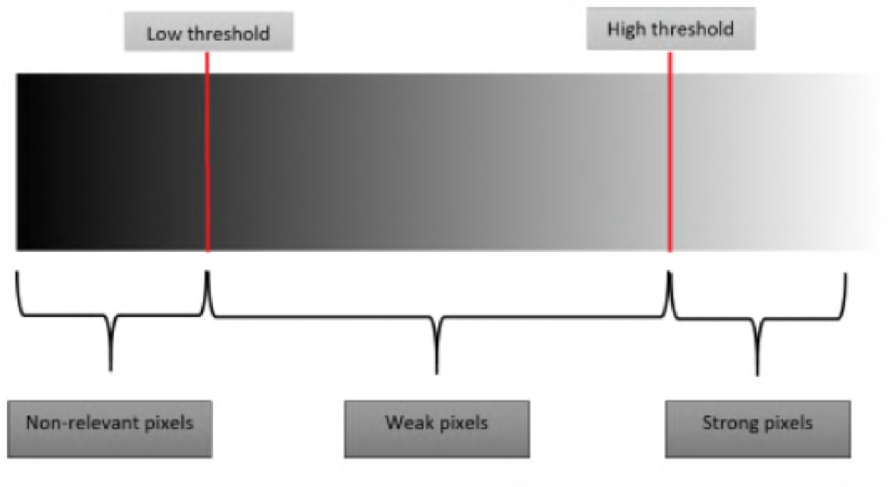


Figure 6: Double tresholding explanation.

### 3.1.5 Edge tracking by hysteresis

Based on the threshold results, the hysteresis consists of transforming weak pixels into strong ones. It is done as follow: if at least one of the pixels around the one being processed is a strong one then the pixel will be considered as a strong one.

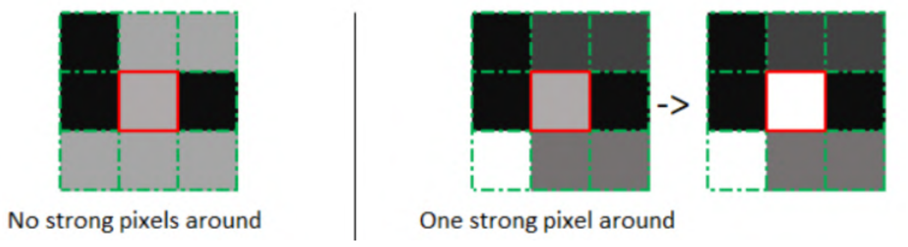


Figure 7: Examples of possible cases of hysteresis.

Here appear 2 possible examples (fig. 7).
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 be 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 the result showed on the fig. 8.

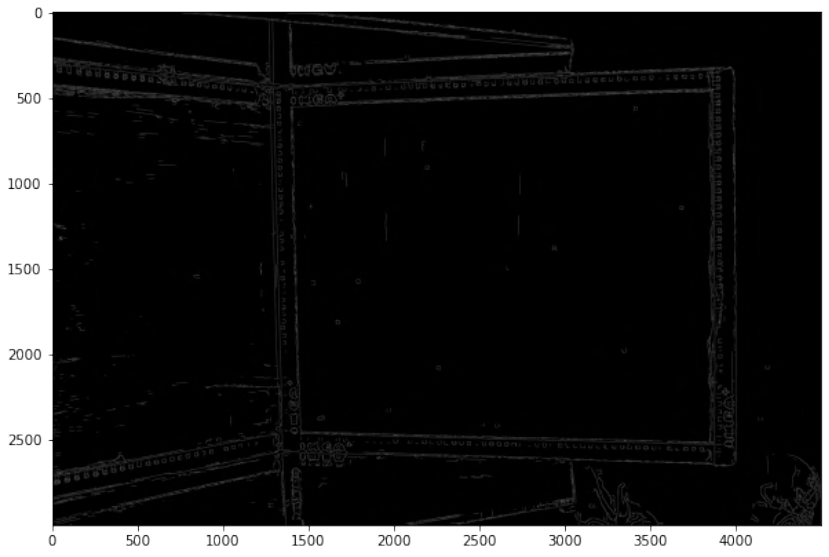


Figure 8: Result of using edge tracking by hysteresis on our image.

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.

## 3.2 Hough transform

Initially, Hough transform was patented by Paul Hough in 1962 [3]. As it is universally used today this method was invented by Richard Duda and Peter Hart in 1972 [4], who called it a "generalized Hough transform", based on the original patent of Paul Hough. 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.

### 3.2.1 Line detection: 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 $\rho$ and $\theta$.

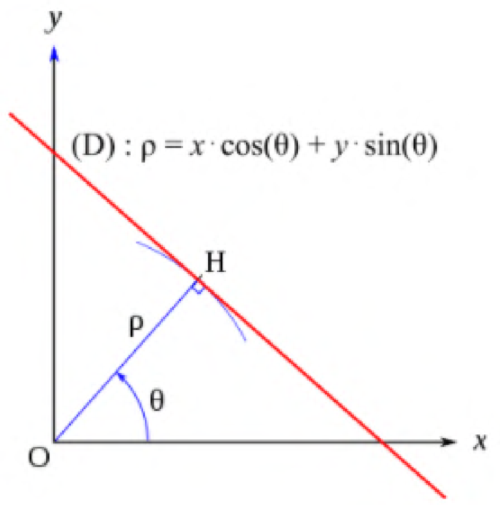


Figure 9: Polar coordinates representation of the line.

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 $(\rho, \theta)$ 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 $(\rho, \theta)$ 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 $C$.

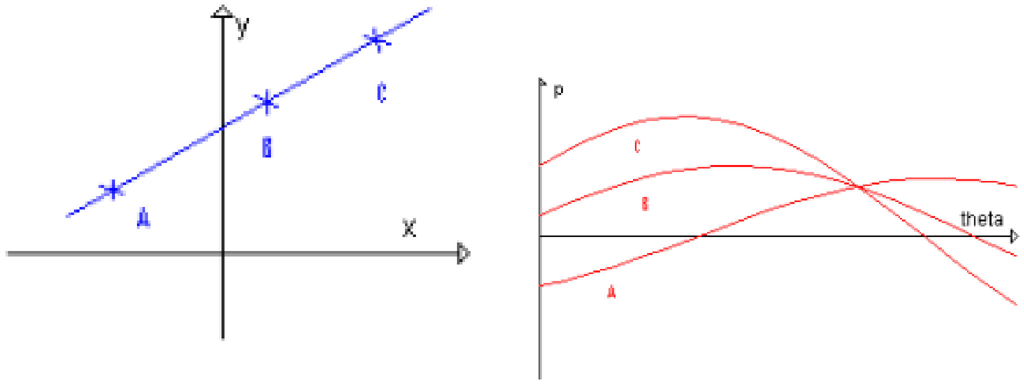


Figure 10: Example of representation of the lines going through the points $A$, $B$ and $C$ correspondingly.

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.

### 3.2.2 Hough algorithm

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 as discussed in the previous section , but other methods like gradient can be used.

The Hough's transform algorithm uses a 2-D matrix called accumulator. This accumulator 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 $\delta$)
**for** $(x, y) \in J$ **do**
    **for** $\theta \in [0, \pi]$ with step $\delta$ **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,whiter is a point, 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.

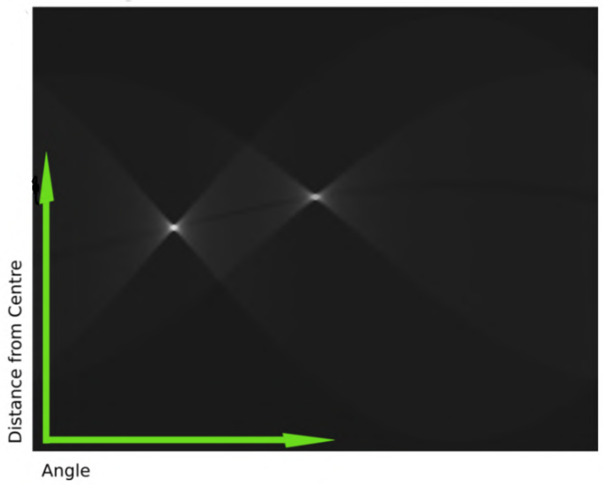


Figure 11: Example of accumulator.

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.

### 3.2.3 Circle detection

We saw 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. Circle Hough transform (CHT) is a specialization of Hough transform.
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 $r$.

## 3.3 Homography

In our case, we have many parallaxes problems. To solve this issue, Homography is used. Example of parallax issue can be seen on the fig. 12.

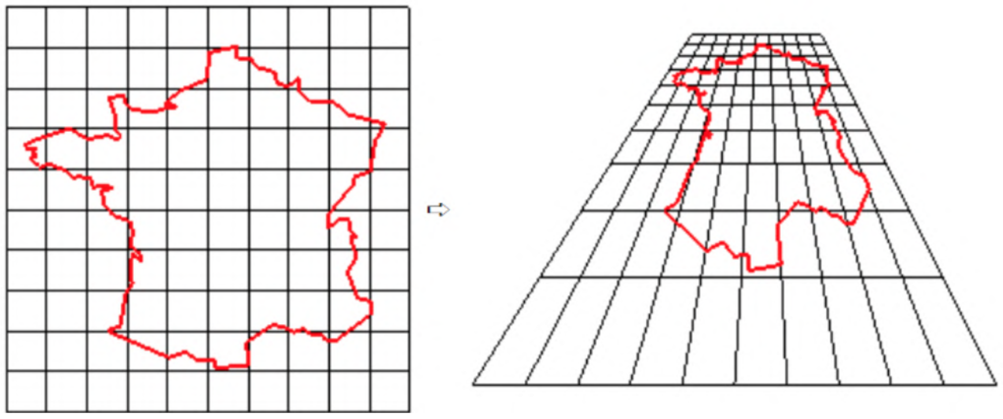


Figure 12: Example of parallax effect.

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 is 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:

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

.
Wdge 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 parameter $z_0$. 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'($\frac{x_0}{z_0}, \frac{y_0}{z_0}$)

Example of homography is shown on the fig. 13.

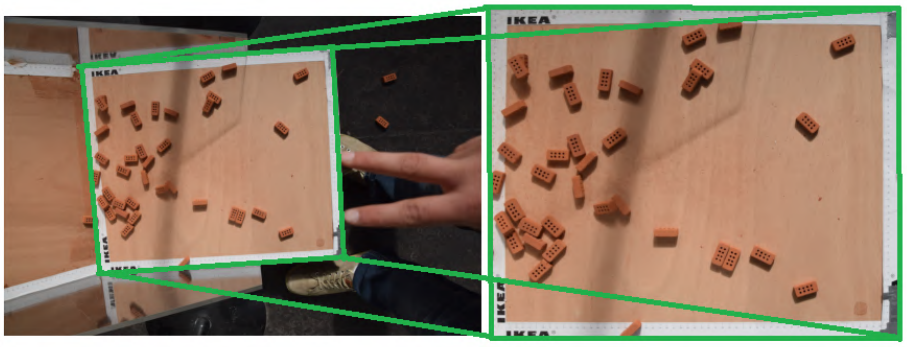


Figure 13: Example of performance our homography implementation.

## 3.4 Thresholding by Otsu method

The OTSU method [5] 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 and to produce a binary image:

First step is to create a frequency Histogram function. The goal of hist(k) is to count the number of pixel among 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.

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

where:

- t is the threshold
- N is the number of pixel
- $w_1(t) = \sum_{k=0}^{t} \frac{Hist(k)}{N}$ probability to be in class 1 and $w_2(t) = 1 - w_1(t)$probability to be in class 2

- $\sigma_i$ 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

## 3.5 Morphological Transformations

Morphological transformations are some simple operations based on the image shape. It is normally performed on binary images. It needs two inputs, one is our original image, second one is called structuring element or kernel which decides the nature of operation. Two basic morphological operators are Erosion and Dilation.

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

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

### 3.5.1 Erosion

The erosion is used, as its name suggests, to erode an image like in the fig. 14.

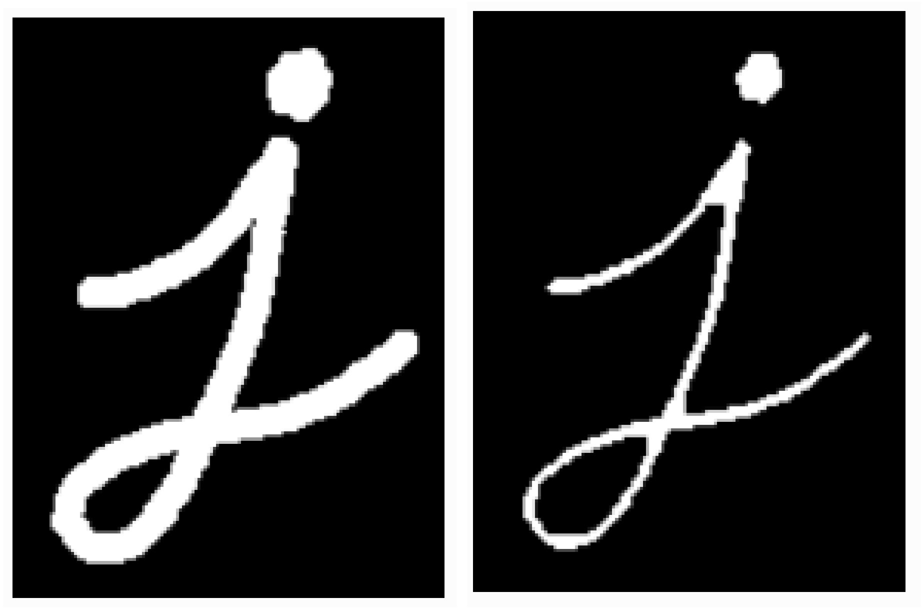


Figure 14: Example of erosion.

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).

### 3.5.2 Dilatation

At the opposite, dilatation dilate a image like in the fig. 15.

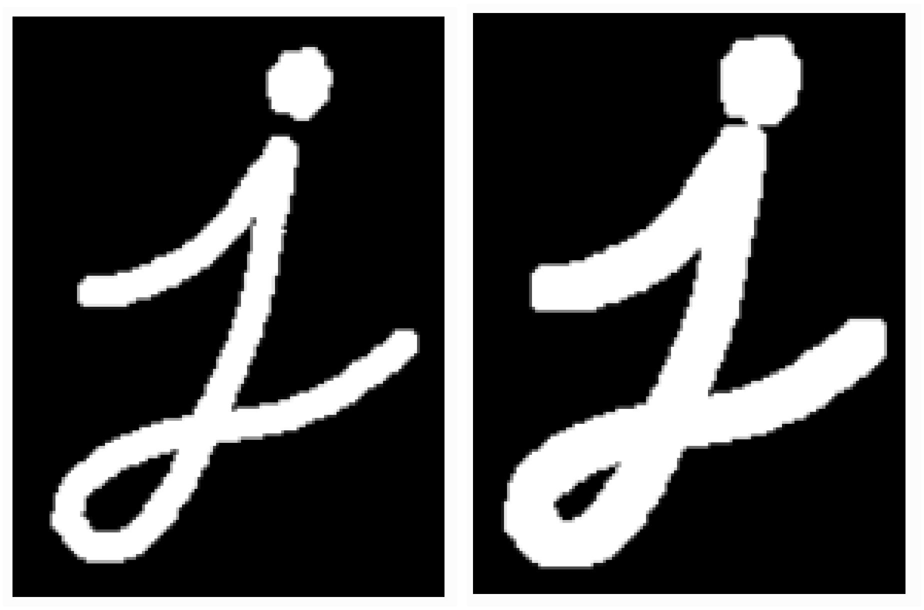


Figure 15: Example of dilatation.

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.

### 3.5.3 Examples of using

A famous operation called ’opening’ consists in respectively executing an erosion, followed by a dilation. The goal of such an operation is to remove noise, as it appears in the fig. 16.

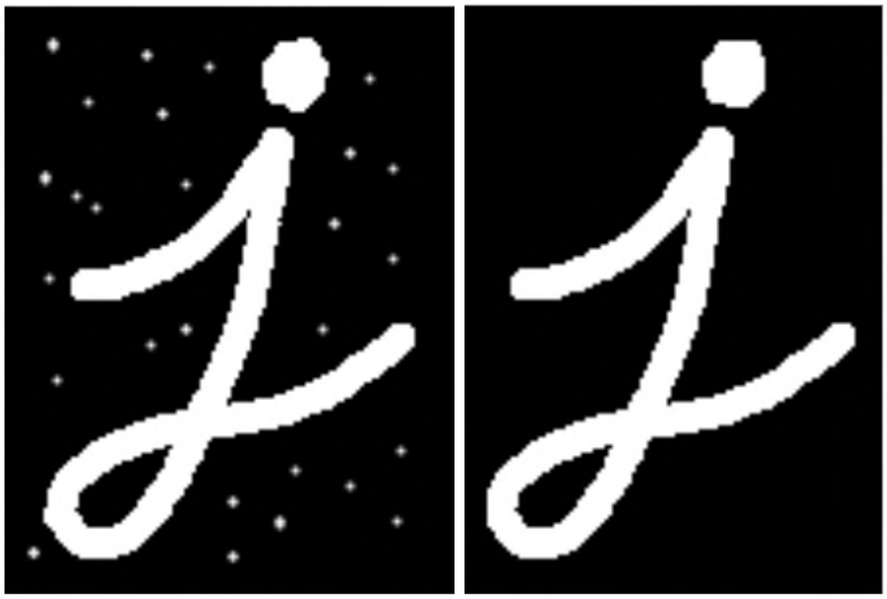


Figure 16: Example of using erosion and dilatation to remove noise.

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

## 3.6 FindContours

In this section we will quickly look at the function FindContours[**link**] from the OpenCV library [6], which we use in our algorithm. This function isbased on the paper [7]. Our description is written in an short way. FindContours is a very popular function, it works significantly well, that's why we have chosen it.

In previous section we obtain binary version of image without noise. FindContours can help us to extract contours of the block.
The algorithm that computes the result just scan each pixels with TV raster and interrupt the raster scan when a pixel (i, j) satisfies some conditions, for example the border following starting point of either an outer border (see fig. 17a) or a hole border (see fig. 17b). If the pixel is a hole, then in the table at the corresponding place 0, if the border, then in the table the number is $\geq 1$.

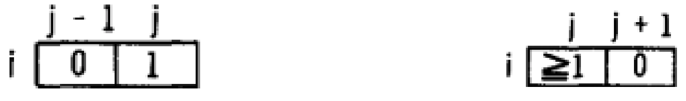


Figure 17: The conditions of the border following starting point (i, j) for an outer border (a) and a hole border (b)

If the pixel (i, j) satisfies both of the above conditions, (i, j) must be regarded as the starting point of the outer border. Assign a uniquely identifiable number to the newly found border. Adjacent pixels with the same labels are in the same set. So, after this scanning process image will be divided into the "border" and "holes" sets. It means that each pixel has one of two labels "boarder" or "hole" and by some properties, written in the original paper[7], we can distinguish unconnected boundaries as different objects. We will not describe all the properties presented in the original article, this was not the purpose of our work.

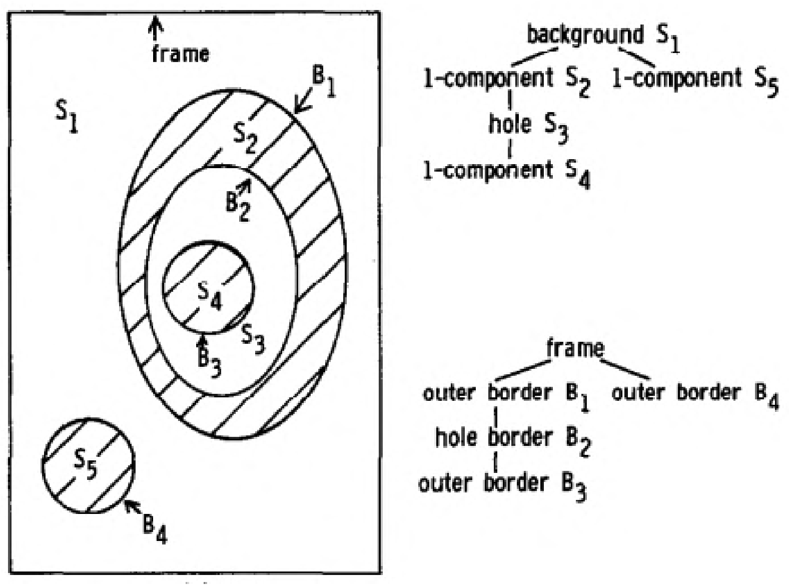
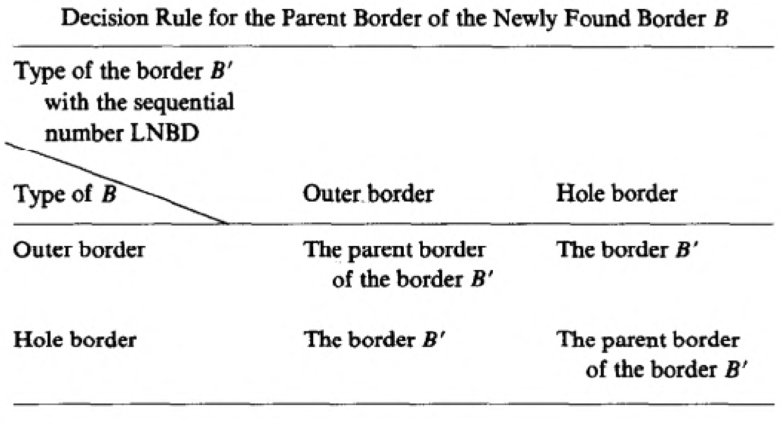


Figure 18: Surroundedness among connected components (up) and among borders (down)

The last step of the algorithm is to group founded contours into a multilevel hierarchy. Determine the parent border of the newly found border as follows.

During the raster scan we also keep the sequential number LNBD of the (outer or hole) border encountered most recently. This memorized border should be either the parent border of the newly found border or a border which shares the common parent with the newly found border. Therefore, we can decide the sequential number of the parent border of the newly found border with the types of the two borders according to fig. 19.



Decision Rule for the Parent Border of the Newly Found Border $B$

| Type of the border $B'$ with the sequential number LNBD / Type of $B$ | Outer border | Hole border |
|---|---|---|
| Outer border | The parent border of the border $B'$ | The border $B'$ |
| Hole border | The border $B'$ | The parent border of the border $B'$ |

Figure 19: Decision Rule for the Parent Border of the Newly Found Border B

## 4 Application 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 tray can change, the luminosity also.
This is for instance visible in the images shown in fig. 20. Thus we decide to cut the project in 2 main parts :
- The first part consists in searching for the 4 sides of the tray (or the 4 angles, it is the same) and then apply an homography to extract the tray and suppress the projective effect of the photo.
- The second part consists in searching for the position of the blocks in the extracted tray. Many strategies are tested and described in the next.

### 4.1 Tray extraction

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

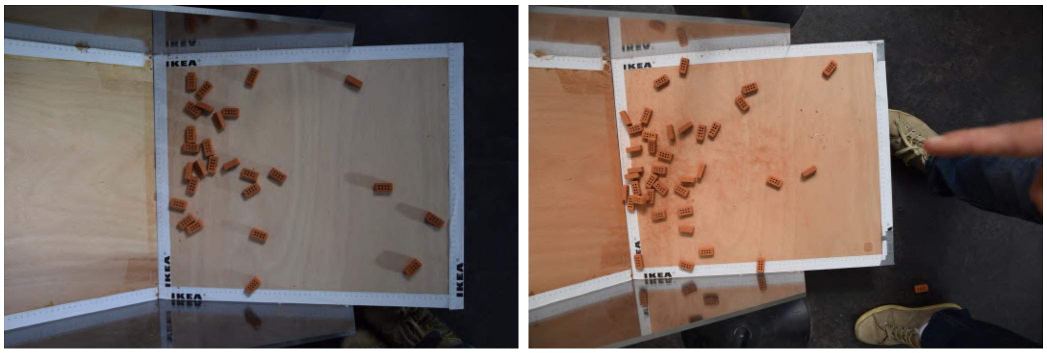


Figure 20: Examples of photos with different orientation.

### 4.1.1 Tray corners coordinates extraction

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

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

Results of each step are presented on the fig. 22. And lets consider the sequence of chosen steps of the algorithm more deeply step-by-step using as an example photo shown in the fig. 21. Most of steps were implemented using the OpenCV [6] library.

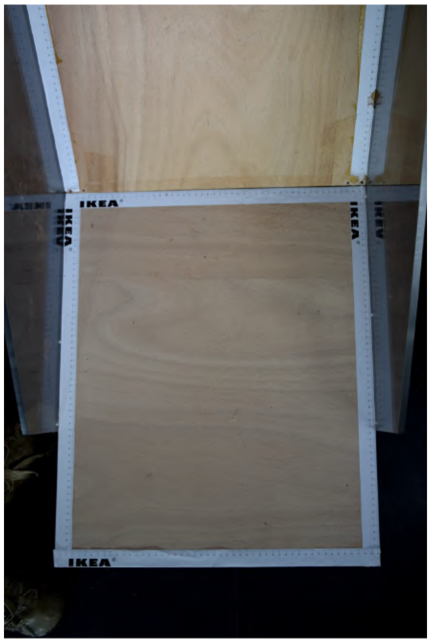


Figure 21: Example of the initial photo.

First of all we used Canny edge detection in order to extract all edges from the photo, since some of them could be candidates to be edges of the tray. The example of this step can be seen on the fig. 22a.

The next step was to find the most probable line segments with the length at least $0,65 \cdot l$, where $l$ is the length of the smaller side of the image. This length of the line segment was chosen manually after some tests on different images from the given set. Special type of Hough transform — probabilistic Hough transform [8] allows to do it. See an example on the fig. 22b.

After previous step we pulled out all desired information from the photo — all candidates to be the tray edges were extracted. Consequently, on this stage we want to get rid of the excess information. To do it we performed manual binarization, showed on the fig. 22c, such that each pixel corresponding to the line segments from the previous step become the value 1 and all the rest the value 0. In other words, we just drawn found line segments with 1-pixel width on the fully black image.

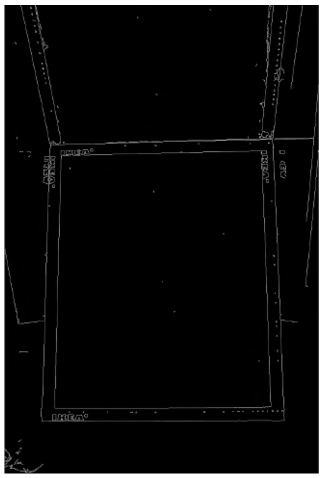
From the analysis of given photos, we found out, that there is possibility to extract some other long enough line segments, that actually belong to other parts of experimental setup (for example glass protective walls). In order to exclude such segments, we performed following selection: using the function *skimage.measure.label* (you can read about it by this [**link**]) we extracted each connected regions (the set of neighboring pixels labeled with 1) and then we deleted all of these regions except regions satisfying at least one of the following rules:

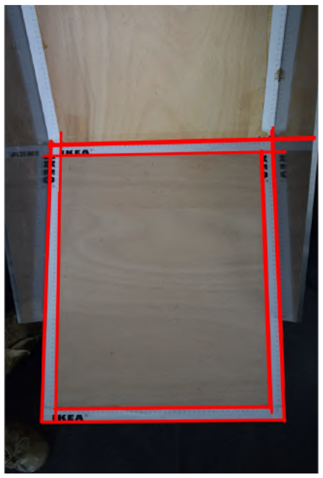
- The region does not cross some of image border;
- The region posses the largest square.

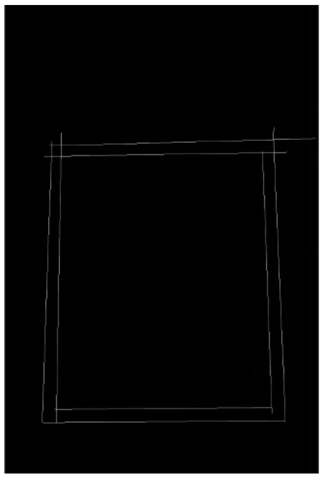
This rules were inspired by observation, that such excess detected segments commonly occurs near a side of the photo and also the largest region corresponds to the tray. But this step was created just in case, during our work from all considered examples were extracted only such line segments that some of them related to the edges of the tray and others related to the inner edges of white measure band.

In order to find corners of the tray we need to find points of intersection of obtained lines. To do that, we need to build corresponding straight lines and find its points of intersection. We applied standard Hough transform to build these straight lines and find the points of intersection. An example of these steps can be seen on the fig. 22d and 22e.

Now we have all candidates to be the corners of the tray. If we construct convex hull of these points, it will be exactly the tray and its corners will be our desired tray corners. On the fig. 22f you can see the result of this last step (obtained points are marked with tiny green circle).


(a) Canny algorithm


(b) Prob. Hough transform


(c) Binarization

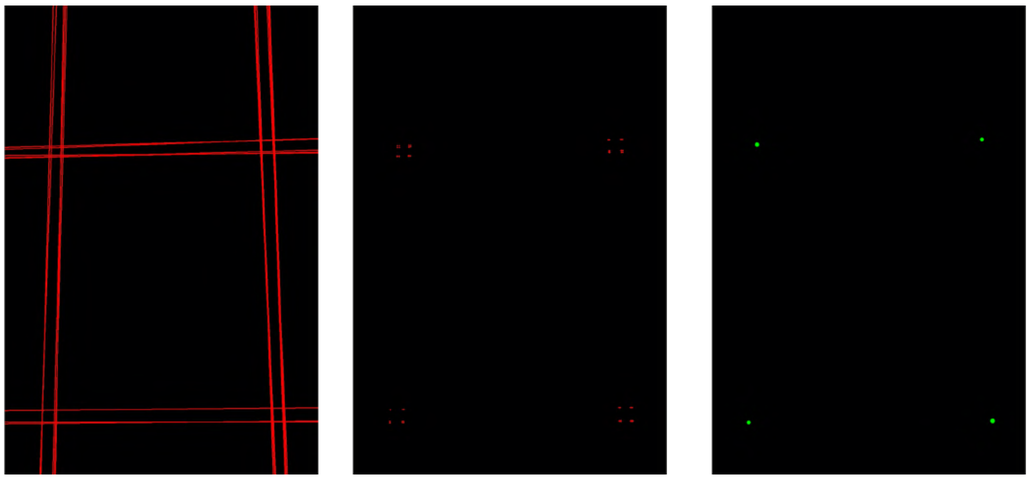

(d) Hough transform (e) Intersection points (f) Convex hull

Figure 22: Step-by-step examples of using trail corners extraction algorithm.

This approach uses a lot of gradient methods, that is why it is obviously better work on the photos without blocks (blocks add additional gradients to the image and it could be hardly processed automatically).
But since we know that each series of photos includes first "calibration" photo — photo of the the tray without blocks and surfaces, shooted from the same angle as all other photos in this series, we can apply this part of the algorithm only once. We can find 4 corners of the tray only using the "calibration" photo of the series, and then use found points for each photo from the series.

This approach has shown good performance on the most of given photos. You can see some examples of correct tray localization on the fig. 23. The tetragons with corners placed at 4 found by the algorithm points are drawn on the images by green frame. You can see that these tetragons good approximate tray on the photos.

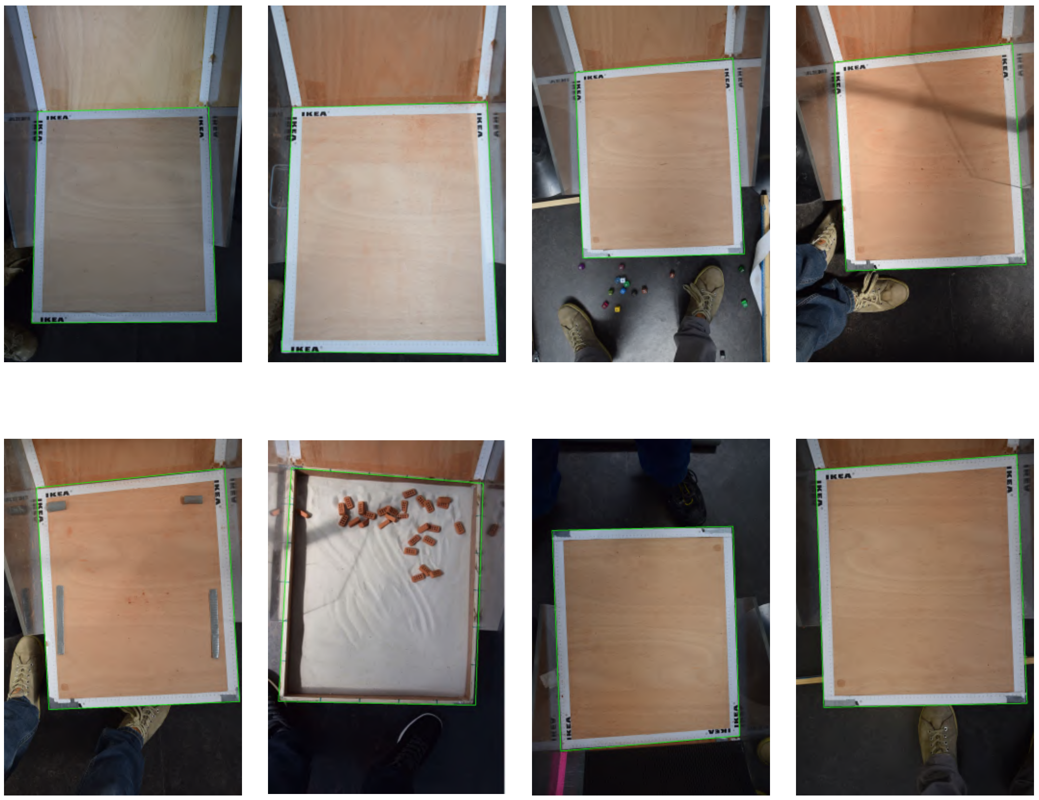


Figure 23: Some example of correctly found corners of the tray.

But on some photos it works not so good (see fig. 24). It is caused by presence of some excess objects on the photo, such that surfaces on the slope, some pretty textured fibers around the experimental setup and so on.

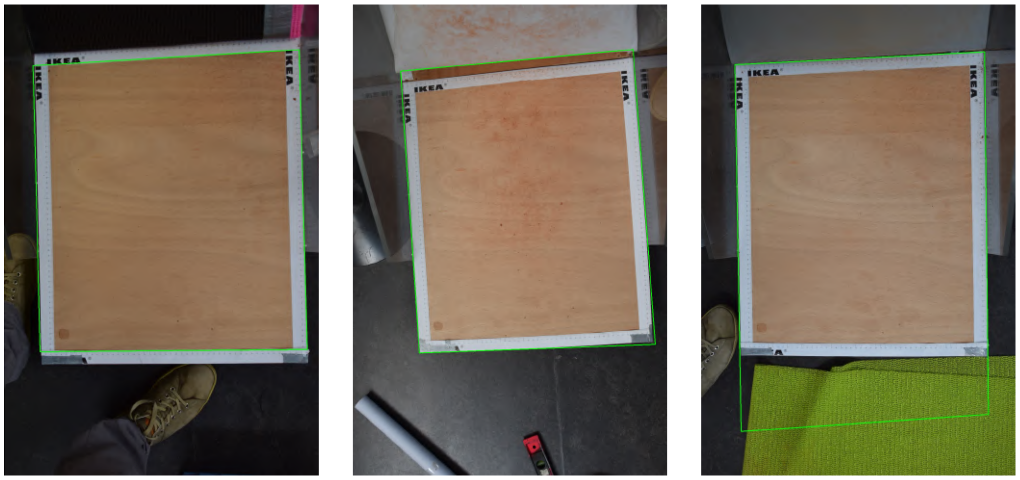


Figure 24: Some example of incorrectly found corners of the tray.

Our algorithm works well in the cases when on the photo absent extra objects and when there is contrast between the tray and the background, for example on some images on the borders were glued by silver tape, which made it impossible to distinguish background from border. Since algorithm works only with one photo from the folder and we had small variability of photos (about 20 folders) and only 20% mistakes on it, that is why this statistic can not be representative.

### 4.1.2 Homography

Since we for a given photo, we know the 4 angles of the tray the last step remained consists in doing an homography to extract the tray and suppress the projective effect. An example is given on the fig. 25.

## 4.2 Blocks detection

The work is now to use the corrected picture to find and extract the blocks. The adopted strategy is the next, we want to find a mask which allows us to know if a pixel belongs to a block or to the tray. A mask (for the blocks) is a binary picture in which white pixels correspond to a block and black pixels correspond to tray. Such a mask, if it is well computed provides us many important pieces of information. Indeed, if we find the contours of such a mask, and if for each contour we draw the minimum enclosing circle of this contour, we are supposed to get one circle for one block. Let's now describe the precise procedure.

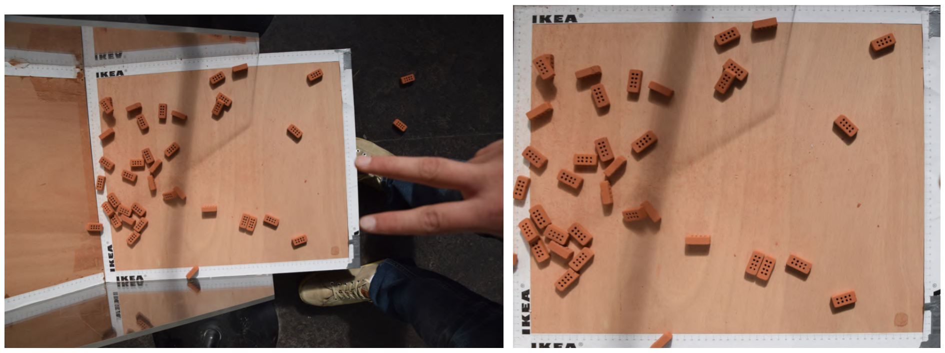


Figure 25: Example of homography algorithm performance (from left to right): before applying homography; after applying homography.

### 4.2.1 Pursuit of the mask

As a reminder, we are searching for a mask for the blocks. More precisely, we want to create a picture with black pixels for the tray and white pixels for the blocks.

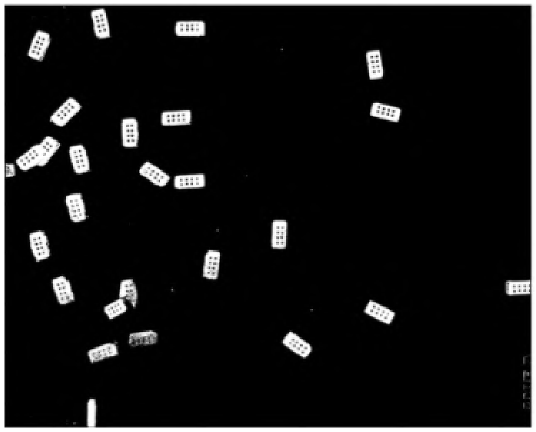


Figure 26: Example of mask.

To determine such a mask, many strategies exist.
The simplest way consists in considering the 3 components of the RGB space. For each component, we can fix a low threshold and a high threshold. If for each component, its value is between the 2 corresponding threshold, we consider that the pixel belongs to a block.
In practice, these 6 threshold are determined as follow :
We click manually on several blocks (at least 10). This operation provides 3

sets of values (one set corresponds to one RGB component and contains the values of this component for the clicked pixels). Then we can take as low threshold the first quartile and as high threshold the third quartile.

This technique is particularly efficient when the tray is well illuminated, without shadow and with blocks of color different from the background.
Another way to obtain a such binary picture is to use the Otsu's algorithm described earlier in this report.

**An other approach**.
Sometimes we dispose of a photo of the trail before the rocks fall, it's possible to use this picture to do a better segmentation of the picture.This image can be proceed in the same manner than the previous photos. That is to say that we can extract the trail, this will be particularly useful as we will see soon.

One idea that we can exploit is to substrate the empty trail with the full one. This give us the result shown in the fig. 27.

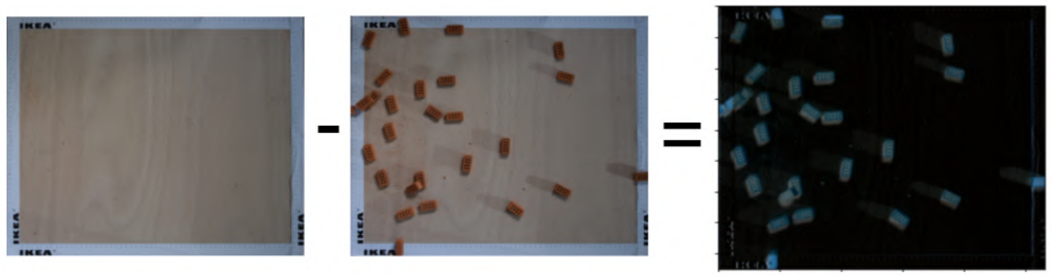


Figure 27: Example of substraction the empty trail from the full one.

This operation is particularly interesting because it delete non-essential information for the extraction of blocks such as: 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. Be careful, the substraction of 2 color images gives a color image but in our case, such an operation increases a lot the contrast between the tray and the blocks that is way the previous methods of mask would be more efficient.

For instance, an Otsu threshold used on the image shown in the fig.27 (after passing the image to gray scale) will give us the result can be seen in the fig. 28.

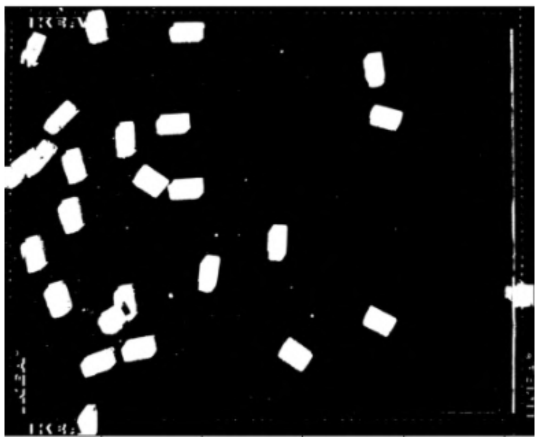


Figure 28: Example of Otsu tresholding.

All the previous methods can provide a mask, but as illustrated in fig. 28 some default still exist on the image. These defaults may be probably due to little movements of the camera in the case of substraction. In the other cases, some defaults also exist, indeed one can imagine that some pixels of the tray are seen as part of the blocks because the color is sometimes similar.

This issue is easily solved thanks to an opening operation (see morphological transformations before). If we apply successively an erosion and a dilation on our image, we get the results shown in the fig. 29.

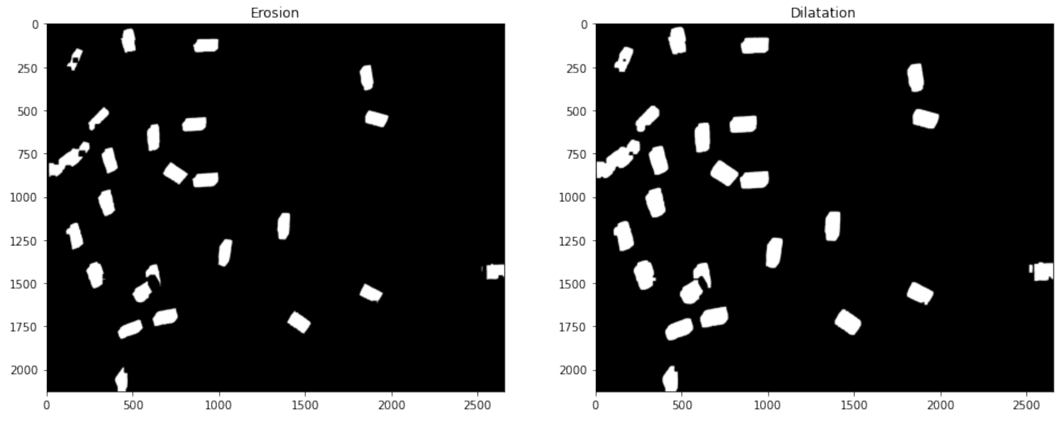


Figure 29: Example of erosion and dilatation. The right image is the mask

Now we have got a mask, the first idea is to extract contours of this mask and construct the more little circles that contains each contours. Such an opera-

tion allows us to get contours around the white parts. Result you can see in the fig. 30.
This result is interesting because we are now able to find the isolated blocks.

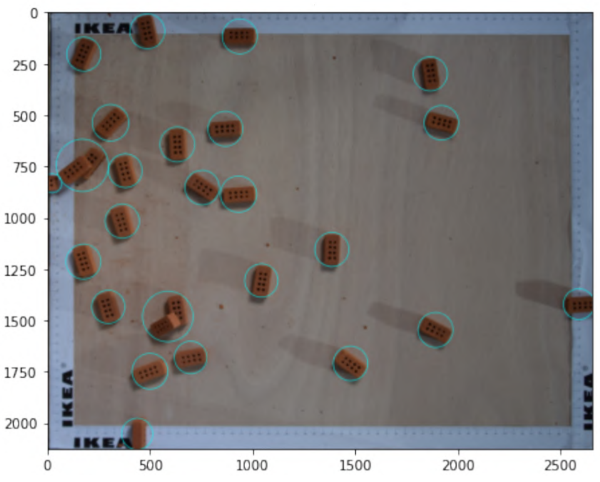


Figure 30: Resulting extraction of blocks using first mask.

However, when several blocks are regrouped, a single circle covers further blocks. This is not yet the expected result but some peaces of information can be extracted from it. Indeed the radius of the circle is bigger when the number of blocks included in it increases. But this is insufficient and we have to continue the process on the image. The next part is dedicated to the improvement of the mask.

### 4.2.2 Refining of the 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 original image, and then use a Canny edge detector to find the separation between the blocks. In the fig. 31 can be seen an example of application of this procedure.

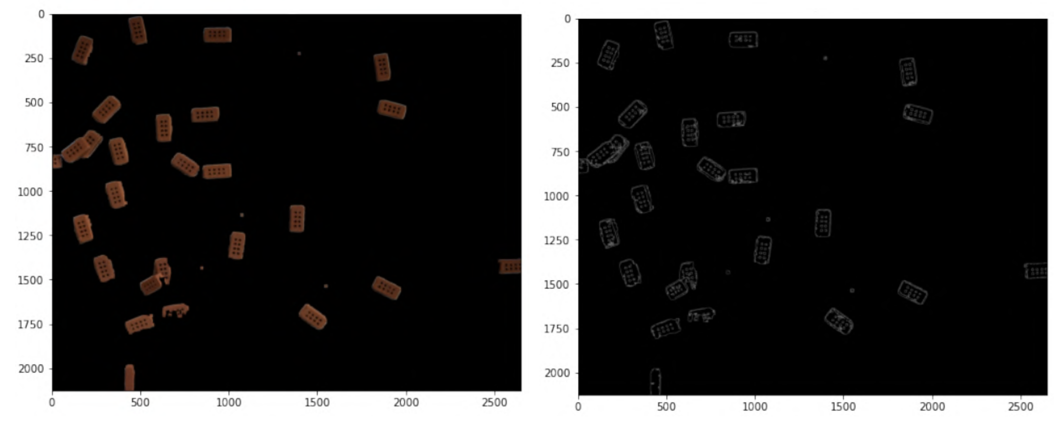


Figure 31: Example of applying Canny algorithm after applying the first mask.

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 are the circles corresponding to the black hole of the blocks. Nevertheless, we know a method to localize circles on an image, this is the Hough transform for circles that we already talk about in section 2.2.3. 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, (we first detect all the circles and then we select only circles of this size) the result could be seen in the fig.32.

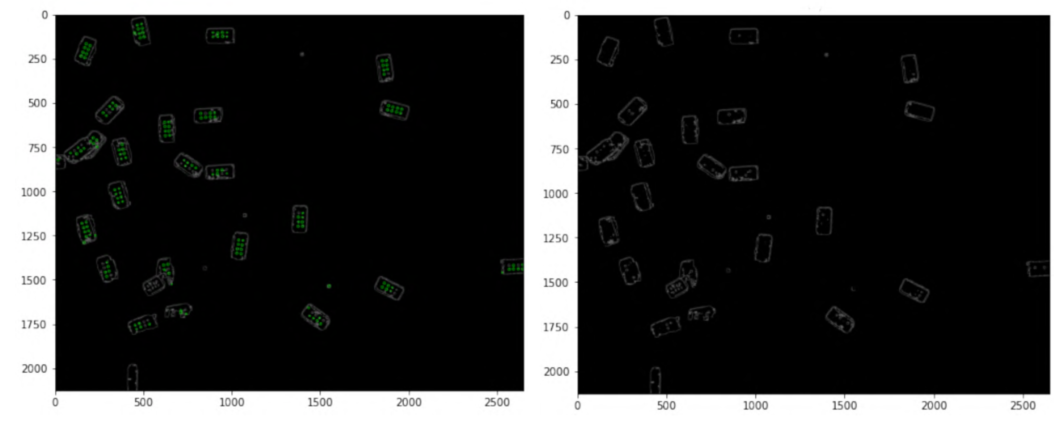


Figure 32: Example of removing small circles by Hough transform (from left to right): detected circles; after removing circles.

In green appears all the detected circles with the good size (this size has to be determined in a real image), 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 (the mask of fig. 29, right). In addition, all the detected circles are colored in white. We get a new mask, as you can see in the fig. 33.

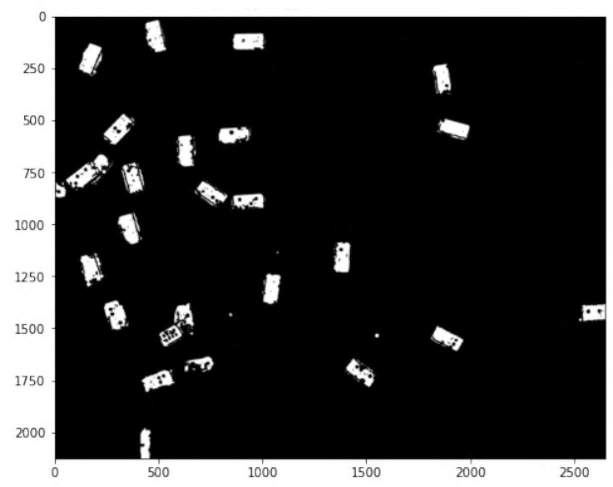


Figure 33: Refined mask.

If we draw the circles around each contours the new result will be really different from the previous one. This is due to the refined mask which is more complex than the first one. As a example, we get for two different images the results shown in the fig. 40.

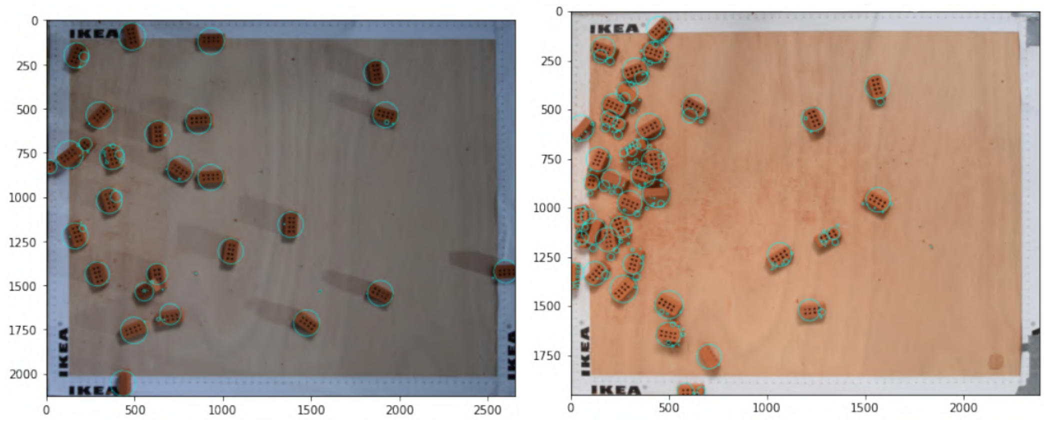


Figure 34: Examples of found circles using refined mask

### 4.2.3 Circles filtering

The problem is now clear, we have too many circles. To solve this issue we have applied 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 suppress 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 corresponds to two blocks. As a result we suppress 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 results can bee seen in the fig. 35.

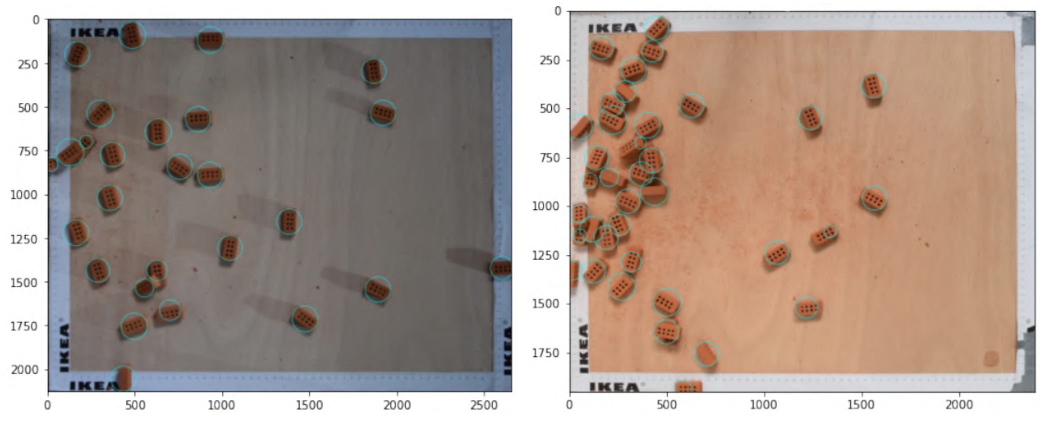


Figure 35: Results of the circles filtering.

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

## 4.3 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 or to low brightness as can seen in the to below images.

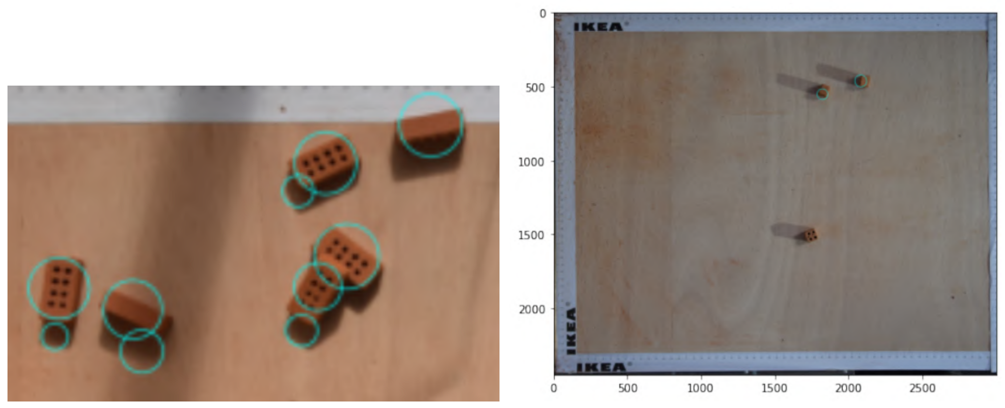


Figure 36: Some examples of limitations.

# 5 Statistics over the results

Our application has been created to process images with large blocks, that's why this application has been tested on the folder this application has been tested on the folder **14 briques/grosses briques/bois/Photos**. This folder contains 50 pictures, on each picture there are 14 blocks on wooden tray.

## 5.1 Number of blocks outside the tray

Firstly it's necessary to know if our algorithm finds the good number of blocks on the tray.

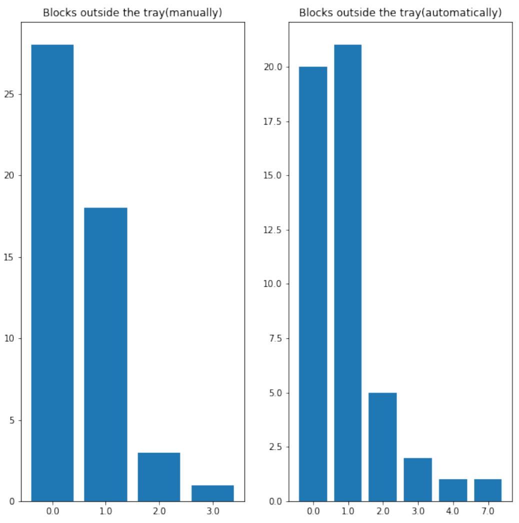


Figure 37: Graphics showing the number of blocks outside the tray.

Two graphics shown in the fig. 37 seem to be similar.This idea is confirmed by an accuracy test which says that the algorithm finds the right number of blocks in 66% of the cases. However it's all possible to see that our application did a huge mistake on one picture, it's the picture where the number of blocks found is seven.

## 5.2 Influence on the distance of the automatic measurement

This part talks about the effects of the automatic measurement on the distance.

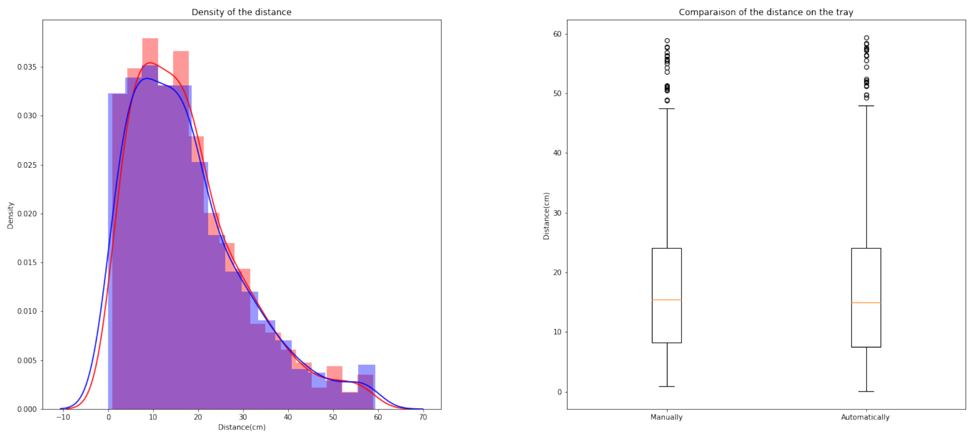


Figure 38: Graphics showing the density distribution of the distance

We can see on this two graphics that the distribution seems to be same.Moreover the number of outliers also seems to be the same. This idea can be confirm with statistical test called Kolmogorov-Smirnov. The p-value obtains between this two samples is 0.64 that's why it's not possible to reject the hypothesis that the distributions of our two samples are the same.

## 5.3 Analysis of the error

For the error analysis we proceeded as follows, we compared only those images in which we found the right number of blocks. We made this choice because it is difficult to compare the error between two images if the number of blocks is too different.
For the error analysis we will proceed in the following way, we will compare only those images in which we have found the right number of blocks. First of all we will look at the general error, i.e. the error on all the merged images.

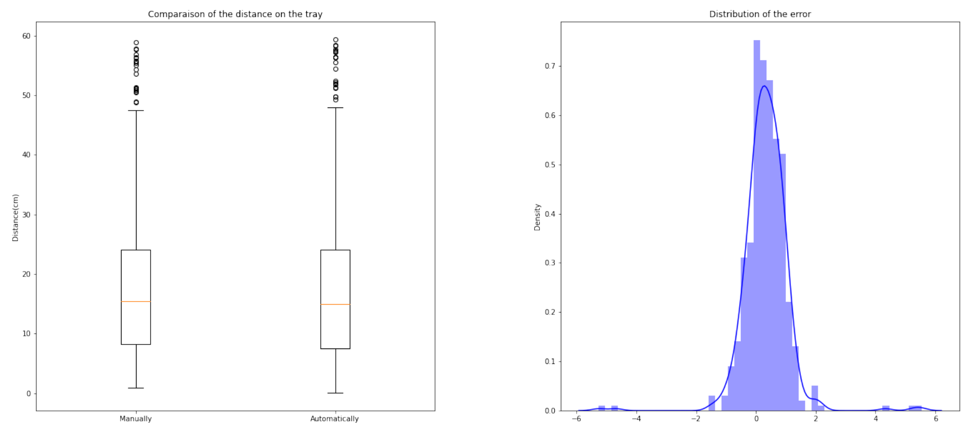


Figure 39:  Analysis of the error

We can see that these results were expected. Indeed the error is supposed to be a normal law centred in 0 . The shapiro-wilk test allows to confirm the hypothesis of normality, and this sample has an empirical mean of 0.33 and an empirical variance of 0.58.
Now it's necessary to talk about error for each picture.

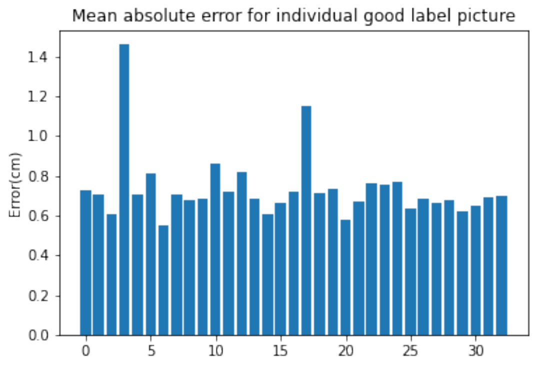


Figure 40: Mean absolute error for individual good label picture

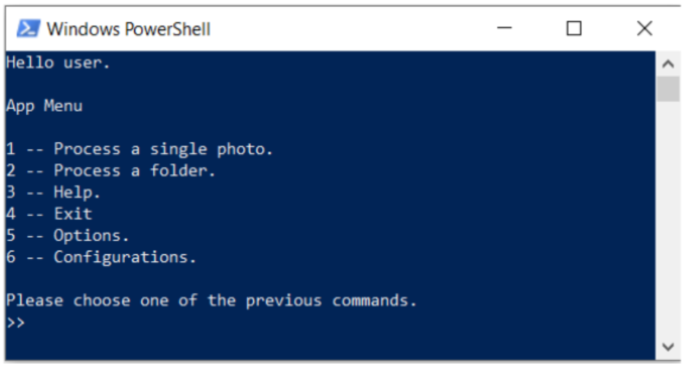
In this picture, the error per image is very stable and the mean absolute error between each picture is 0.73 cm.

# 6 Python Application

## 6.1 Instructions manual

Our app can be launch as following, the user has to go in the app folder and to writhe the next command :
**python3 ./main.py**

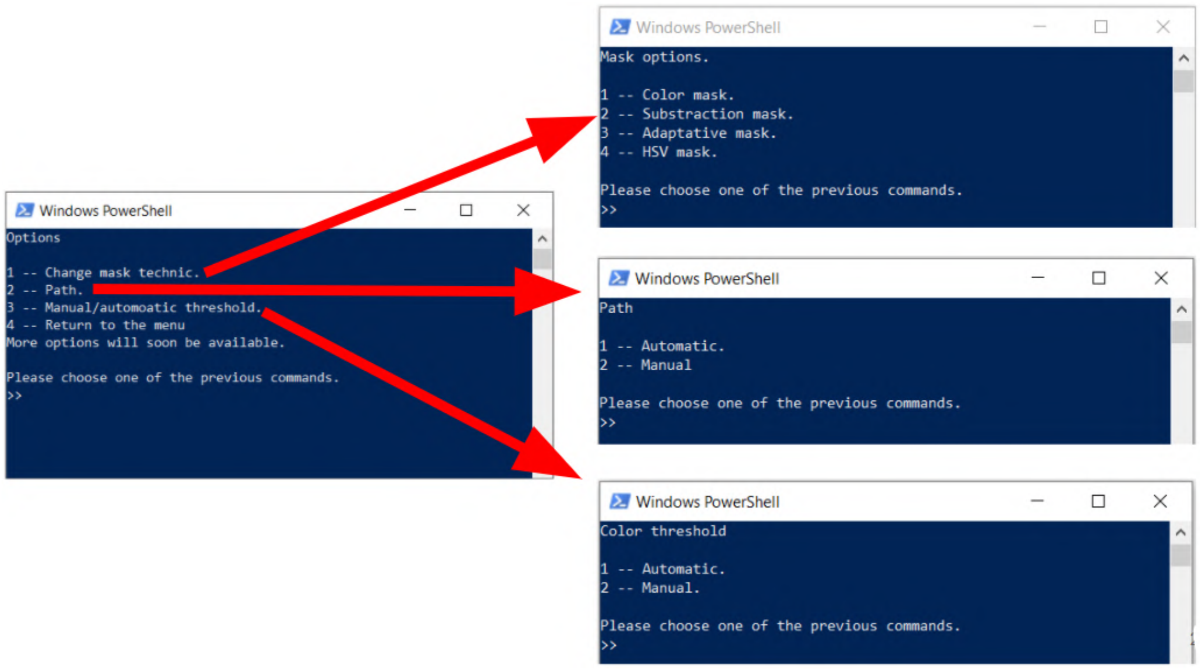
The the menu of the app should appear.



```
Hello user.

App Menu

1 -- Process a single photo.
2 -- Process a folder.
3 -- Help.
4 -- Exit
5 -- Options.
6 -- Configurations.

Please choose one of the previous commands.
>>
```

Figure 41: Menu of the application.

First, let's have a look to the options :



```
Options

1 -- Change mask technic.
2 -- Path.
3 -- Manual/automoatic threshold.
4 -- Return to the menu
More options will soon be available.

Please choose one of the previous commands.
>>
```

```
Mask options.

1 -- Color mask.
2 -- Substraction mask.
3 -- Adaptative mask.
4 -- HSV mask.

Please choose one of the previous commands.
>>
```

```
Path

1 -- Automatic.
2 -- Manual

Please choose one of the previous commands.
>>
```

```
Color threshold

1 -- Automatic.
2 -- Manual.

Please choose one of the previous commands.
>>
```

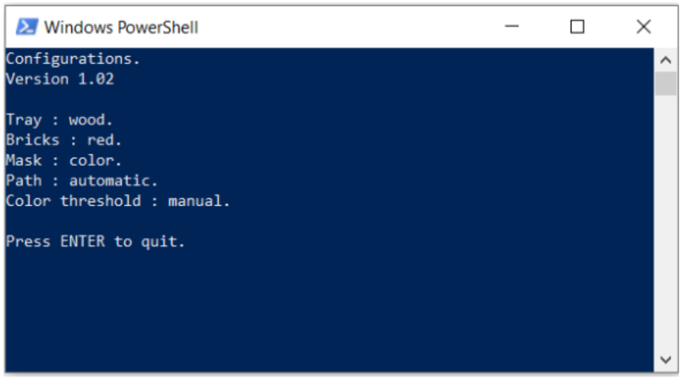
Figure 42: Options of the application.

The first option allows the user to change the technique for finding the mask. The different kinds of mask available in the app.

- Color mask. The mask described in the report.
- Substraction mask. The app will use the substraction trick.
- HSV mask. The app pass from RGB space to HSV space apply technique similar to color mask.

In the case of a color mask, the app makes possible to choose manually the colors to detect. (with the clicks and quantiles procedure). This can be changed in the option : **Manual/automatic threshold**

Finally the option **Path** make possible to generate automatically the name of the csv files.
Once options are fixed, the user can check them in the **Configurations** as it appears in the next image.



```
Configurations.
Version 1.02

Tray : wood.
Bricks : red.
Mask : color.
Path : automatic.
Color threshold : manual.

Press ENTER to quit.
```

Figure 43: Configurations of the application.

Coming back to the menu, the 2 main procedures are :
**Process a single photo**
**Process a photo**

**Process a single photo** : This will process a single photo. The application will ask to the user the path to the photo to process. First, the app will search for the four angulars of the tray and will display its results at the screen. Then the app ask the user to confirm or infirm these angulars. (the user must answer by yes (y) or no (n) in the terminal. If the answer is no, the user must click by hand on the four angulars and close the windows. Now the app will extract to tray and save it in the folder results. After that, it continue and detect all the bricks on the tray. The are encircle in yellow and the result photography is again saved in the folder results. Now the user have to press ENTER to return to the menu.
**Process a photo** : This command requires some conditions. The main one is that the folder to process contains in first position a photography of the empty tray, and next it must contains at least one photography with blocks. The algorithm will ask to the user the path of the folder to process. The user must write it and press ENTER. Moreover, the application can (that depends on the options) ask for the number of blocks and for the name of the csv file to save. Like it has been done in the single photography process, the algorithm will search for the four angulars of the tray and ask to the user if they are well positionate. The user must write 'y' or 'n' in the terminal and press ENTER. If the answer is no, the user must select by hand the four angulars, and close the window. Now the algorithm is able to process the folder. For each photography (except the empty one), it will extract the tray, save it in the folder results, next it will extract the blocks and save the new photography in

the same folder. In addition the application will create a new CSV file in the folder CSV. Each line of the file corresponds to one photography, it contains the number of the photo, the total number of blocks, the number of block(s) outside the tray, all the 'x' coordinates, and all the 'y' coordiantes. The user must press ENTER to come back to the menu.

## 7 Recommendation for the shooting

The methods used in our application are very sensitive to the quality of the provided photos. 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 precisely. 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, desirable the colour of scotch tape used to glue the measuring tape significantly differs from the floor colour. Additionally, using of non-reflective material for the experimental setup walls can improve stability of the image processing algorithms performance. 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.

## 8 Conclusion

The problem was quite complicated, for its solution we have studied many articles and tried various approaches. As a result, we got an algorithm that solves our problem quite well, including the results for the huge blocks on a wooden or sand support is very encouraging for the future. For more confident work, user should follow the recommendations during the collection of the dataset. For ease of use, we have implemented an application. The result can be obtained both in the photo and as a CSV file.

## 9 Acknowledgment

We would like to address our thanks to our tutor for this project, Frederique Leblanc for her continous involvement in the project.

## References

[1] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986.

[2] I. Sobel and G. Feldman. An isotropic 3×3 image gradient operator. 1990.

[3] Hough Paul V C. Method and means for recognizing complex patterns.

[4] R. Duda and P. Hart. Use of the hough transformation to detect lines and curves in pictures. *Commun. ACM*, 15:11–15, 1972.

[5] N. Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, 1979.

[6] Open source computer vision library. `https://opencv.org`.

[7] Satoshi Suzuki and Keiichi Abe. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1):32–46, 1985.

[8] Si-Yu Guo, Ya-Guang Kong, Qiu Tang, and Fan Zhang. Probabilistic hough transform for line detection utilizing surround suppression. In *2008 International Conference on Machine Learning and Cybernetics*, volume 5, pages 2993–2998, 2008.