University of Bordeaux

Internship Report

Master of Software Engineering (2013 - 2015)

Design and programming of automatic classification methods applied to biological images

---

Student:
LE Van Linh

Supervisor:
Prof. Marie BEURTON-AIMAR

October 27, 2015

Acknowledgements

First of all, I would like to express my deepest gratitude to my supervisor, Prof. Marie BEURTON-AIMAR for her agreement, guide and support during the planning and development of my internship.

I would like to thank Jean-Pierre SALMON for his generous help and comment during my work. I would like to thank the staffs, students in LaBRI, who helped, supported for technique and gave me a professional working environment.

I would also like to thank all professors in University of Bordeaux and PUF-HCM, who imparted a lot of knowledge to learning and researching. I would also like to thank the dean of IT-DLU, who allowed me joined in this course. Finally, I would like to thank my family and colleague for their support and encouragement through my study.

Abstract

Image processing is a field that has many application in life. It can be from the usual application to the application in medicine or cosmology. To obtain the best result, all most of applications must follow two processes: Firstly, we should pre-process the image with some appropriate operations to enhance the interest in also reduce the noises. Secondly, we apply main operations to obtain the result.

The goal of project is built a program with full functions about processing base on the biological images. During my internship at LaBRI, my tasks are developing the algorithm to preprocessing image by removing the unexpected parts. Besides, we also program a method to automatic classification on biological images. The method based on the segmentation and classification.

Finally, I integrated my functions into the IMP tool, which was developed by NGUYEN Hoang Thao. Besides, we also debug the previous code and write the documentation for the next development.

Contents

List of Figures

Chapter 1

Introduction

1.1 Pôle Universitaire Français

The Pôle Universitaire Français (PUF) was created by the intergovernmental agreement of VietNam and France in October 2004. With ambition is building a linking program between the universities in VietNam and the advanced programs of universities in France. There are two PUF's center in VietNam: Pôle Universitaire Français de l'Universite Nationalé du Vietnam - Ha Noi located in Ha Noi capital (PUF-Ha Noi) and Pôle Universitaire Français de l'Universite Nationalé du Vietnam - Ho Chi Minh Ville located in Ho Chi Minh city (PUF-HCM).

1.1.1 PUF-Ha Noi

PUF-Ha Noi is regarded as a nursery for the linking program, it support on administrative procedure and logistics for the early year of program. Besides, PUF-Ha Noi also implement the training program regularly about Master 2 provided by universities and academies in France. About administration, PUF-HN directly under Institut Francophone International (IFI), which was created by VietNam National University at HaNoi in 2012.

1.1.2 PUF-HCM

PUF-HCM[1] is a department of VietNam National Univeristy at Ho Chi Minh city. From the first year of operations, PUF-HCM launched the quality training programs from France in VietNam. With target, bring the programs which designed and evaluated by the international standards for Vietnamese student. PUF-HCM always strive in our training work.
So far, PUF-HCM have five linking programs with the universities in France, and the programs are organized into the subjects: Commerce, Economic, Management and Informatics. In detail:

- Bachelor and Master of Economics : linking program with University of Toulouse 1 Captiole
- Bachelor and Master of Informatics: linking program with University of Bordeaux and University of Paris 6.

The courses in PUF-HCM are provided in French, English and Vietnamese by both Vietnamese and French professors. The highlight of the programs are inspection and diploma was done by the French universities.

[1] http;//pufhcm.edu.vn

1.2 Laboratoire Bordelais de Recherche en Informatique

The Laboratoire Bordelais de Recherche en Informatique (LaBRI)[2] is a research unit associated with the CNRS (URM 5800), the University of Bordeaux and the Bordeaux INP. Since 2002, it has been the partner of Inria. It has significantly increased in staff numbers over recent years. In March 2015, it had a total of 320 members including 113 teaching/research staff (University of Bordeaux and Bordeaux INP), 37 research staff (CNRS and Inria), 22 administrative and technical (University of Bordeaux, Bordeaux INP, CNRS and Inria) and more than 140 doctoral students and post-docs. The LaBRI's missions are: research (pure and applied), technology application and transfer and training.
Today the members of the laboratory are grouped in six teams, each one combining basic research, applied research and technology transfer:

- Combinatorics and Algorithmic
- Image and Sound
- Formal Methods
- Models and Algorithms for Bio-informatics and Data Visualisation
- Programming, Networks and Systems
- Supports and Algorithms for High Performance Numerical Applications

Within these team, research activities are conducted in partnership with Inria. Besides that, LaBRI also collaborate with many other laboratories and companies on French, European and the international.

1.3 The Internship

The internship is considered a duration to apply the knowledge to the real enviroment. It shows the ability synthesis, evaluation and self-research of student. Besides, the student can be study the experience from the real working. My internship is done under the guidance of Prof. Marie BEURTON-AIMAR in a period of six months at LaBRI laboratory.

1.3.1 Objectives and my task

In any fields, constructing and developing a tool to support fully operations need a period of time. With the expect, creating a tool to support the operations about image processing, IMP was created. Begin in 2012, IMP was created by NGUYEN Hoang Thao. In the first version, IMP had basic operations about image processing such as segmentation, smoothly, morphology, transform,.... Besides, it also integrated some algorithm which was processing on image.

As a part of IMP, the general objectives of this internship is developing the operations in the IMP tool, as follows:

- Design and implementation the method to remove the grid on biological images
- Design and program the method to automatic classification on biological images
- Automatic detect the landmarks on biological images
- Maintain some operations in IMP

[2]http://www.labri.fr

1.3.2 Origanization of the document

The all report mainly have five chapters. In the chapter 1, this is the short introduction about my university, mainly information about the laboratory where I do the internship and the objectives of my internship. In chapter 2, we talk about the necessary preliminaries in image processing field which we use to implement the methods. In the chapter 3, I propose the algorithm to preprocessing image, with the aim is decrease the noise in the input and increase the effective of the classification methods. In the chapter 4, I mention method to segmentation image, classification objects and an automatic process detect the landmarks on the biological images. Finally, chapter 5, I present about the implementation of the preprocessing image algorithm and classification methods.

Chapter 2

Background

2.1 Overview about image processing

Nowadays, we have a lot of programs what used to edit the photos (e.g. photoshop, gimp, paint,...). By apply some technique, we can effectively some property to change the image such as: scaling, blurring, rotating image,.... Actually, an image is presented of a set of pixels. Each pixel carry a value which presented for the color at this location. When combine the value of all pixels, we have the image as we can see in the real word. The changing on image really changing the value on each pixel in image. Behind the techniques in these programs are mathematical operations and the field using mathematical operation on an input image, called *image processing*. The output of image processing may be either an image or a set of characteristics related to the image. And most of image processing technique are performed on two-dimensional image. In image processing, we have a lot of operations. In this chapter, we just introduce some basis operations what often useful for the object of this internship.

2.2 Image filtering

Image filtering is a process to modify or enhance the quality of images. This is known as a "neighborhood" operation. The neighborhood is a set of pixels around a selected pixel. In image processing, with a pixel, we can have 4-neighbors or 8-neighbors of it. Image filtering determines the value at the selected pixel by apply some operations with the value of its neighbors. One of the filter operators is smoothing, also called blurring. This technique is used in preprocessing steps, particularly using for noise reduction. With a matrix called kernel. It was sliding over the image. At each position, the output of value at this position is average of its neighborhoods. In image processing, we have many filter techniques. But it can be divided into 2 main types:

Linear filter: The idea behind this filter is replacing the value of every pixel in the image by the average of the gray levels in the neighborhood defined by the filter mask. By this work, this filter sometime are called averaging filter. The result of this process is an image with reduced the sharp edges in gray level, it also reduce the noise because the noise is typically and random in the image. The mask is a matrix useful for blurring, sharpening, or edge-detection, ... The output image is accomplished by convoluting between a mask and an image.

Order-Statistics filter: By ordering the pixels in the image and then replacing the value of the center pixel with the value determined by the ranking result. Median filter is an example of this technique.

2.3 Histogram

Histogram is a representation about distribution of data on the regions (we called bin) in the data range. The bins are the number of sub-range when we divide the entire data range into several small interval (i.e. With the range from 0 - 255 and the size of each sub-range (bin) is 16, the number of bins is 256/16 = 16 bins. The first bin range is 0 - 15, the second range is 15 - 30, and so on). The value at each bin is the numbers of data which have value belong to this bin. Normally, histogram represented by the columns chart with x-axis represented for the number of bins, and y-axis represented for the value of each bin.
Histogram can be used effectively for image enhancement, also useful in many image processing applications, such as image compression and segmentation.

Histogram equation: is a method allow adjust the contrast using the histogram of image. It

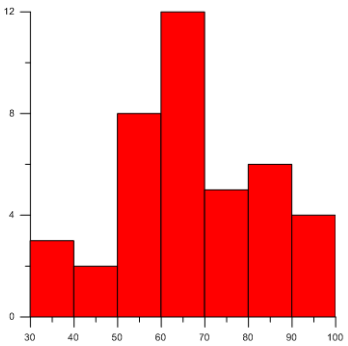


Figure 2.1: An example about histogram

mapping one distribution on a histogram to a wider distribution of intensity values. By this, the image can brighter.

Histogram matching: is a method adjustment of two image using the histogram. This method was finished by calculating the cumulative distribution functions of two histograms and find the histogram matching function. Finally, apply the matching function on each pixel of the image to get the result.

2.4 Segmentation

Segmentation subdivides an image into its regions. The size of regions is depend on the problem being solved. This mean, segmentation should stop when the regions of interest in application have been detected. In the real, the segmentation was applied into many fields such as machine vision, medical imaging, object detection, etc. The most of segmentation algorithms are based on the basic properties of intensity values: discontinuity and similarity. In the first case, the segmentation based on abrupt changes in intensity. The second case, the image segmentation based on a predefined criteria. It means the image was segmented into regions that are similar according to a set of criteria. And, we have many the method to segment an image such as thresholding method, region growing, clustering method, histogram-based method, etc.

Thresholding is a simplest method of image segmentation. Thresholding use a particular threshold value “t”, we split the image into two parts: the first part includes pixels which have the value greater than t, and the second part contains the pixels vice versa. With this technique,

thresholding can be used to create an binary image from a gray scale image. In fact, we have many type of threshold, as follows:

- *Global thresholding*, when t is a constant over an entire image
- *Variable thresholding*, when t changes over an image
- *Local or regional thresholding*, is variable threshoding in a region of an image
- *Dynamic or adaptive thresholding*, if t depends on the spatial coordinates.
- *Multiple thresholding*, thresholding on 3 dominant modes (color image)

Canny algorithm is an edge detection algorithm that uses to detect the structure of image. The process of this algorithm can break into the steps follows [1]:

- Apply the Gaussian filter to smooth the image (remove the noise)
- Find the intensity gradients of the image
- Apply non-maximum suppression to get rid of spurious response to edge detection
- Apply double threshold to determine potential edges
- Track edges

2.5 Color processing

The use of color in image processing do not just identify or extract an objects from scene, it also a factor for image analysis. Color processing can be effect on each component image individually or work directly with pixels based on a color model. The color models is a specification of colors in some standard, generally accept way such as BGR, CMY, HSV or Grayscale model.

- BGR model: using blue, green, red as three primary colors. Image presented in this model consist of three components images for each primary color.
- CMY model: used for hardcopy devices. Based on the BGR mode, each value in CMY mode was computed by integrate between 2 primary color in BGR. Specific, C (cyan) is consist from green and blue, M (magenta) is consist from red and blue and Y (yellow) is consist from red and green.
- HSV model: difference with BGR, HSV using the 3 components are hue, saturation and brightness to present image. Hue is a color attribute which describe a pure color (yellow, orange, red) and saturation give a degree to pick the pure color is diluted by white light. Brightness is a notation of intensity for color sensation.
- Grayscale model: The colors in grayscale just black and white because it just carry the intensity information on each pixel. Because that, the image in grayscale mode was called black and white image. The color of each pixels in image from black, where have weakest intensity to white at the strongest intensity.

[1] https://en.wikipedia.org/wiki/Canny_edge_detector

The most color operations in image processing is transformation. This is a process to the conversion image between the color models by using a transform expression such as BGR to HSV, HSV to BGR, BGR to Grayscale.

Besides that, considering the specific characteristic of each color space, allowing we can classify the pixels in the image. This idea can be used to segment object of an image. HSV model is widely used to compare the color because the range of color (Hue value) is specific.

(a) An image in BGR mode (b) An image in Gray mode

Figure 2.2: The images with color transformation from BGR to Gray

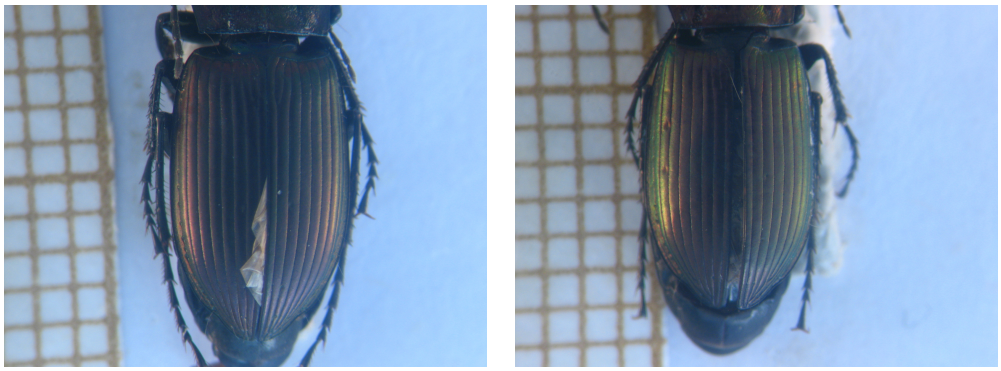
Chapter 3

Preprocessing image

Preprocessing image is a process that reducing the noises or removing the unexpected objects on image. It is used to enhance the quality of input dataset (images) when do a test for a algorithm or a method. Following the requests, we can apply one or more operations to pre-process image. In this chapter, we will discuss about a method to pre-process image within this internship. The method suggested base on the basic knowledge about the operations in image processing.

3.1 Problem

With the input dataset is a set of 293 insect images. Each image contains the parts of insect(body, head,...) and an unexpected object, specifically yellow grid (figure 4.2). To enhance the accuracy of classify method, we need to remove the grid and just keep the insect on each input image.



(a) The yellow gird on the left of insect (b) The insect overlap the yellow grid

Figure 3.1: The input images with yellow grid

3.2 Analysis

Each input image contains the two objects: a part of insect (called insect) and the yellow grid (called grid). About the relative position, the grid always stayed in the left of insect, and insect can either overlap the grid. About the color, image is presented in BGR model with three main color groups: the background color, the yellow color of grid and the color of insect.

The method proposed to remove the grid based on the color processing. If we process the image in BGR model, the algorithm may be complex because the color at each pixel is combined of three values (blue, green, red). While HSV model has a specific channel to present the colors with clear range. We can apply this property for detecting and removing the gird. The proposed process to remove the grid as follows:

1. Find the "*limit*" point of grid: the points stay nearest outside grid.
2. Find the "*replace*" point: location that its value used to replace for grid.
3. Replace the grid by the value at "*replace*" point.

3.2.1 Finding the limiting and the replacing point

Browsing image to check and replace the pixels in gird need a long time. To reduce the time to do that, we should find the limit range of grid. The limit of gird is the points located out of gird and its closest. Instance of checking on all pixel, we just check the pixels stay on the left of limit points.
As we know, the width of grid usually less than a two-thirds of width of image. So, to reduce the time to finding the limit point, we also check from the begin of image to two-thirds of image. The result of this step is the limit points, these used for limiting the length when we check the pixels on yellow grid.
The algorithm to find the limit points are followed:

Data: *inputImage*: The input image (contains the insect and grid)
Result: The coordinate of limit point

```
1  Declare some variables: Mat hsvImage; vector < Mat > hsv_channel;
2  Convert image from BGR to HSV:
   cv : cvtColor(inputImage, hsvImage, COLOR_BGR2HSV) ;
3  Split HSV image into several channel: cv :: split(hsvImage, hsv_channels);
4  Set up initial limit_point and assign with the left-top corner: Point
   limit_point = Point(0, 0);
5  Declare a variable yellow_count to count the number yellow points on each columns
   when processing. An column become a limit line if the number of yellow points on this
   column less than a constant value.;
6  for j ← 10 to hsv_channel[0].cols do
7      if H value at (5, j) > 100 ||(H value at (5, j) > 70 && H value at (5, j) < 100
8      && S value at (5, j) < 10 && V value at (5, j) > 175 ) then
9          limit_point.x ← j;
10         limit_point.y ← 0;
11         yellow_count ← 0;
12         for i ← 1 to hsv_channel[0].rows ∗ 2/3 do
13             if H value at (i, j) <= 38 then
14                 yellow_count + +;
15                 if yellow_count >= 8 then
16                     limit_point.x ← 0;
17                     limit_point.y ← 0;
18                     break;
19                 end
20             end
21         end
22         if limit_point.x! = 0 then
23             break;
24         end
25     end
26 end
27 if limit_point.x == 0 then
28     limit_point.x ← hsv_channel[0].columns/3 + 200;
29     limit_point.y ← 0;
30 end
```

Algorithm 1: Algorithm to find the limiting points

Now, we indicate which is the color used to replace the yellow points. Hence, we choose the points having the value nearest with the background color. The histogram is ideal for choosing the position to replace, but we also have some conditions to obtain a good value.
The algorithm to find the replacing points are followed:

Data: inputImage: the input image
Result: The coordinate of replacing point

```
1  Convert image to gray scale image;
2  Calculate the histogram on gray scale image and mean of histogram;
3  Split the HSV image into channels;
4  for i ← 0 to grayImage.rows do
5  |   for j ← 0 to grayImage.columns do
6  |   |   if value at (i, j) >mean of histogram
7  |   |   && H value (i,j) > 90
8  |   |   && H value (i,j) > 130
9  |   |   && S value at (i,j) > 50
10 |   |   && V value at (i,j) > 215 then
11 |   |   |   return this position ;
12 |   |   end
13 |   end
14 end
```

Algorithm 2: Algorithm to find the replacing point

3.2.2 Replacing the grid

After having the limit points. By processing on all rows of image. At each row, we replace the pixels which have the color value stay in the range of yellow by another value. But the grid is not only created by the yellow point, it contains more the pixel have the value stay in the same range with background. But the brightness of these pixels is less than the background. So, we needs to replace it obtained the good image. In each row, this work repeated until meeting the limit points or a "special point" (called "break" point). It can be a point stayed on the insect or a point belong to background.
For each part of the insect, the color on insect or the background also have the difference value. So, we establish the difference values for each part. Based on the file name of image, we can classify it.

Data: filePath: the file path of image
Result: Which part of insect in image

```
1  QString temp ← filePath.toLower();
2  if temp contains "ely" then
3  |  return ELYTRE;
4  end
5  if temp contains "md" then
6  |  return MDROITE;
7  end
8  if temp contains "mg" then
9  |  return MGAUCHE;
10 end
11 if temp contains "prono" then
12 |  return PRONOTUM;
13 end
14 if temp contains "tete" then
15 |  return TETE;
16 end
17 return ELYTRE;
```

Algorithm 3: Algorithm to get the parts of insect

Data: inputImage: the input image; limit_point: the limit point; part: part of insect; minBrightness: minimum of brightness; rpoint: replacing point
Result: The image after replace the yellow grid

```
for i ← 0 to inputImage.rows do
    for j ← 0 to limit_point.x do
        if part is ELYTRE then
            if value at (i, j + 50) satisfy breaking condition then
                break;
            end
        end
        if part is MDROITE or MGAUCHE then
            if value at (i, j + 50) satisfy breaking condition then
                break;
            end
        end
        if part is PRONOTUM then
            if value at (i, j + 50) satisfy breaking condition then
                break;
            end
        end
        if part is TETE then
            if value at (i, j + 50) satisfy breaking condition then
                break;
            end
        end
        if H value at (i, j + 50) in yellow range then
            replace value at this point by the value at replacing point;
        end
        else if V at (i, j + 50) > minBrightness then
            replace value at this point by the value at replacing point;
        end
        ;
    end
end
Merging three channel of HSV;
Convert the image from HSV to BGR;
```

Algorithm 4: Algorithm to replace the yellow grid

3.3 Summary

In this chapter, we propose a method to remove the grid in the image. In short, the algorithm have steps followed:[1]

1. Converting the input image to HSV model
2. Splitting the image (in HSV) to get the individual channel
3. Finding the limit points

[1]The algorithm is combined from the algorithms in each step, which was described above.

4. Choosing the replace point (calculating the histogram and mean value)
5. Getting the type of input and establish the break conditions.
6. Finding and replacing the yellow points and the “miss brightness” point.
7. Merging the channels of HSV
8. Converting the HSV image to BGR image

Chapter 4

Classification methods

In previous chapter, we introduce a method to remove the unexpected object. In this chapter, we will propose a method to obtain the features what we are interested in and the method to detect the landmarks on the insect. This method was proposed by Palaniswamy[1]. The processes can be discuss in follow steps:

1. Extracting the features:
2. Constructing and comparing the pairwise geometric histogram
3. Estimating the pose by the probabilistic Hough transform
4. Detecting the landmarks by template matching

4.1 Preprocessing image and feature extraction

To obtain the good result, before extracting the features in the image, we need to pre-process the image with a appropriate technique to reduce the noise as well as enhance the features that we care. Feature extraction is a process extracting interested features from digital image. The expect result in this result is list of approximate lines which use to construct the pairwise geometric histogram.

The process mainly separate into two stages: Firstly, we pre-process image. In this stage, we reduce the noise in image by finding a threshold value and apply the thresholding technique to obtain the interested features. Secondly, we extract the features based on the edge segmentation. By applying the appropriate technique to obtain the step edges and broken the edges into approximate lines.

4.1.1 Preprocess image

In this application, we use the thresholding technique to pre-process the image. In thresholding technique, with a threshold value "t", we can decrease the noise and obtain the interested features. The threshold value can be defined by the histogram analysis.

Based on the histogram of the original image, we compute the mean and median of this histogram. With the histogram obtained, we split it into two parts: the first part begin from the bin 0 to the limit value (the limit value is smallest value between mean and median); the second part, starting from the limit value to the end of histogram. For each part, we find the maximum, minimum value and calculating the mean of it. The value "t" obtained by the mean of two mean values in two parts of histogram.

With the threshold value "t", we apply the threshold technique to pre-process image in the CV_THRESH_BINARY mode (keep the pixel has value greater than threshold value).

Data: inputImage: the input image
Result: outputImage: the image after processing

Convert the input image into gray scale image;
Calculate the histogram on gray scale image and store the result in *histogram* variable ;
Compute the *mean* value and *median* value of histogram;
$limit \leftarrow (mean > median \ ? \ median : mean)$;
$limitSub \leftarrow ((limit >= 120) \ ? \ (limit - 25) : (limit - 5))$;
Declare some variables: $int \ imax \leftarrow -1, max \leftarrow -1$;
for $i \leftarrow 0 \ to \ limitSub$ **do**
    if $histogram[i] > max$ **then**
        $max = histogram[i]$;
        $imax = i$;
    end
end
Declare some variables: $int \ imin \leftarrow -1, min \leftarrow max$;
for $k \leftarrow imax \ to \ limit$ **do**
    if $histogram[k] < min$ **then**
        $min = histogram[k]$;
        $imin = k$;
    end
end
Declare some variables: $int \ max2 \leftarrow -1, imax2 \leftarrow -1$;
for $j \leftarrow limit \ to \ end\_of\_histogram$ **do**
    if $histogram[j] > max2$ **then**
        $max2 = histogram[j]$;
        $imax2 = j$;
    end
end
$middle1 \leftarrow (imax1 + imin)/2$;
$middle2 \leftarrow (imax2 + imin)/2$;
$middle \leftarrow (middle1 + middle2)/2$;
Apply the threshold with threshold value is *middle*;

Algorithm 5: Algorithm to preprocess image

4.1.2 Feature extraction

After apply the threshold to pre-process image, we apply the Canny algorithm to detect the step edges, which incorporates non-maximal suppression and hysteresis thresholding. In Canny, the importance parameters are two threshold values and aperture size for the Sobel operator, it decides the pixels kept. The threshold value used in Canny algorithm also the value used in the previous step, and the ratio between lower threshold and upper threshold is 1.5 : 3 (follows the article [1]). In implementation, the Canny operation used from OpenCV library[1], and the parameters need to put into Canny are:

- source: the input image (in grayscale mode)

[1] http://docs.opencv.org/modules/imgproc/doc/feature_detection.html#canny

- destination: the output image
- low_thresh: the first (lower) threshold value
- hight_thresh: the second (upper) threshold value
- kernel_size: size of kernel, aperture for the Sobel operator

The Canny algorithm is not aware of actual edges, the edge detecting was based on the Sobel operator, extracted with non-maximal suppression. So, to obtain the expect result, we need to apply another technique to obtain the step edges. The **findContours** was chosen for this aim, the result is a vector of the edges, and each edge was presented by a vector of the points. Like the Canny, the **findContours** also used from OpenCV library [2] and the parameters used in this operation as follows:

- source: the binary input image
- contours: the output. Each contours is stored in a vector of points.
- hierarchy: optional output vector, containing information about the image topology.
- mode: contours retrieve mode
- method: contours approximation method
- offset: optional offset by which every contour point is shifted.

4.1.3 Edge segmentation

The geometric relation can not constructed from the edges, it always construct from the relation of basic geometric objects, such as the lines. In fact, any arbitrary edge can be represented by a set approximate lines. Instead of representing an edge, we can represent a set of approximate lines of it. This way also useful when we want presentation the edges or describe the relation between it. With the set of step edges was obtained from find contours (the image structure). In this step, we will segment it to approximated lines. The method to segment the edges is a recursive algorithm[2] but it have some change in the "stop condition" of algorithm to easy process, as follows:

- Establish a line *"l"* between two endpoints of edge.
- For each point on edge, we compute the perpendicular distance from it to the line l and keep the point which has the maximum perpendicular distance.
- If the maximum perpendicular distance from a point on edge to the line l is greater than α, then the edge is split at this point. The value chosen for α in the program is 3 ($\alpha = 3$).
- Reprocess both parts which was obtained from step 3.
- The algorithm continues until all edges fragments are represented.

[2] http://docs.opencv.org/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html#findcontours

The algorithm is presented as follows:

Data: listPoints: list of points which presented the edge
Result: Queue of "step" points on the edge

1 Declare the first endpoint: $p0 \leftarrow listPoints[0]$;
2 Declare the second endpoint: $pend \leftarrow listPoints[size - 1]$, *size* is the size of *listPoints*;
3 Set up a straight line between the two endpoints $p0, pend$ (line d);
4 Initialization the max value: $maxDistance \leftarrow 0$;
5 Declare a "split point": $imax \leftarrow 0$;
6 Declare a variable: $distance \leftarrow 0$;
7 **for** *point p in listPoints* **do**
8 | $distance \leftarrow$ from p to line d;
9 | **if** *distance* > *max_distance* **then**
10 | | $maxDistance \leftarrow distance$;
11 | | $imax \leftarrow$ position of p;
12 | **end**
13 **end**
14 **if** *maxDistance* > *3* **then**
15 | split the list of points at *imax* and put into 2 parts $(part1, part2)$;
16 | Pre-process on $part1$;
17 | Pre-process on $part2$;
18 **end**
19 **if** $p0$ *does not exist in result queue* **then**
20 | push $p0$ into queue;
21 | `// queue is a variable of class`
22 **end**
23 **if** *pend does not exist in result queue* **then**
24 | push *pend* into queue;
25 | `// queue is a variable of class`
26 **end**

Algorithm 6: Algorithm to segment an edge

4.2 Pairwise geometric histogram

Pairwise geometric histogram(PGH) is used to encode the relative information between a line and a set of lines in an object. Therefore, an object can represented by a set of PGH. From the set of PGH, we can reconstructed the object or compare with another object. In this section, we will mention the constructing a PGH for an object based on the geometrical relationship and compute the similar distance between two objects.

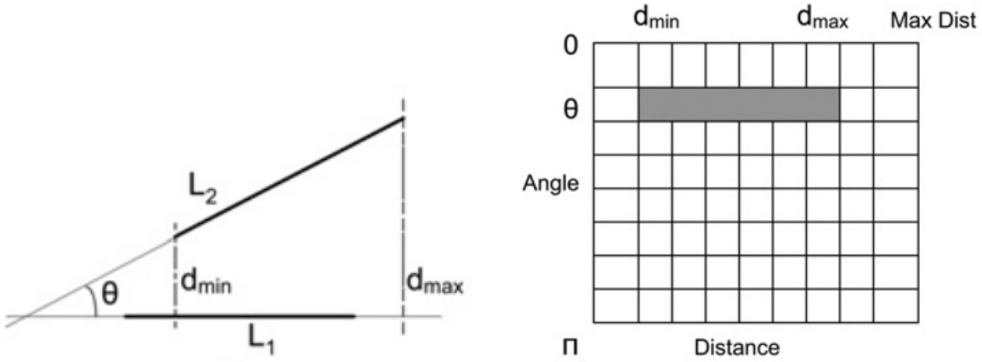
4.2.1 Local pairwise geometric histogram

The PGH is constructed on the geometric features between lines relative. The geometric features are characteristic which can describe the geometric shape such as angle, the length of line, perpendicular between two lines,.... For the shape representation, the relative angle and perpendicular distance is geometrical features useful.
The Local PGH presented the relationship between a reference line with another lines. The proceed to construct the PGH between two lines was described in below:

- Choose the reference line (other lines called object lines)

- Compute the angle between reference line and the object lines
- Calculate the perpendicular distance from two endpoints of object lines to the reference line (assigned dmin and dmax).
- Recording the perpendicular distance and angle relative between reference line and the object lines into the two dimensional histogram.

Example [3]:



(a) The geometric relationship between two lines

(b) The pairwise geometric histogram

Figure 4.1: The geometric features and the PGH

The frequency of the geometric features is recorded as a two dimensional histogram with an angle axis (0 - π) and distance axis (range of perpendicular distance, d_{max} is the maximum distance on all distance of two arbitrary lines). The entries on PGH describe the geometric relationship between the reference line and the object lines. The blurring of entry along the axis regarding the true position and orientation of each object lines for reference line. Following the accuracy, we can indicate the size of histogram and normalize the value to match with size of histogram.

4.2.2 Global pairwise histogram

Based on the constructor of a line in object. The object encoded by recording PGH for all lines within object. If the object is defined by n lines, the full shape representation will composed of n pairwise geometric histograms.
This method still good when we apply some variants on the image, such as translate or rotate the image because the angle and perpendicular distance between a pair of lines is invariant.

4.2.3 Histogram matching

"The histogram matching enables robust classification of shape features by finding similarity between the scene and reference model"[1]. The similar between two models can obtain via the similar distance, which was computed by comparing their probability distribution on geometric histogram. In program, each image was represented by a comprises of many geometric histograms and using the Bhattacharyya metric to determine the similar distance between two

[3]Images extract from the article [1]

models [1]. In general, we have normalize the histograms before comparing. The form of Bhattacharryya metric used to compute the degree of 2 model:

$$d_{Bhattacharyya}(H_iH_j) = \sum_{\theta}^{\pi} \sum_{d}^{d_{max}} \sqrt{H_i(\theta, d)H_j(\theta, d)} \tag{4.1}$$

The significance of parameters in the formula 4.1, as follows:

- θ: angle value, range of θ in angle axis from 0 to π.
- d: the perpendicular distance, range of d in perpendicular distance from 0 to the maximum distance of arbitrary lines of shape.
- $H_i(\theta, d)$ is an entry at row θ and column d in histogram of image i
- $H_j(\theta, d)$ is an entry at row θ and column d in histogram of image j

By the default, the range of angle axis from 0 to 180 degree (correspondence with 180 degree). Based on the accuracy of program, we can increase the range of angle axis. This design allow increase the range of angle axis to several time with default value. Example, the table below show the result when calculating Bhattacharyya distance between image *Md 028.JPG* and some images with difference accuracy:

Reference image	Scene image	180	2 * 180	4 * 180	6 * 180
Md 028.JPG	Md 001.JPG	0.977953	0.964167	0.93861	0.91471
Md 028.JPG	Md 005.JPG	0.96479	0.943657	0.906444	0.871756
Md 028.JPG	Md 010.JPG	0.976241	0.958061	0.925943	0.896445
Md 028.JPG	Md 027.JPG	0.980728	0.968233	0.945442	0.92485

Besides the Bhattacharyya metric, we can also choose another metric to matching the histograms, such as: **Chi-squared** metric and **Intersection** metric. The forms was presented as below:

Chi-squared metric:

$$d_{Chi-squared}(H_iH_j) = \frac{\sum_{\theta}^{\pi} \sum_{d}^{d_{max}} \left(\frac{(H_i(\theta,d)-H_j(\theta,d))^2}{(H_i(\theta,d)+H_j(\theta,d))}\right)}{2} \tag{4.2}$$

Intersection metric

$$d_{Intersection}(H_iH_j) = \sum_{\theta}^{\pi} \sum_{d}^{d_{max}} min(H_i(\theta, d), H_j(\theta, d)) \tag{4.3}$$

The significance of parameters in equation (4.2) and (4.3) is similar with (4.1). For the Bhattacharyya and Intersection metric, the perfect match is 1 and the total mismatch is 0. The result is opposite to Chi-squared metric (0 for perfect match and 1 for total mismatch).

Hence, depend on the purpose of comparison will choose a suitable comparing method. In this program, we want to try on three method to have a general view result when matching the histograms.

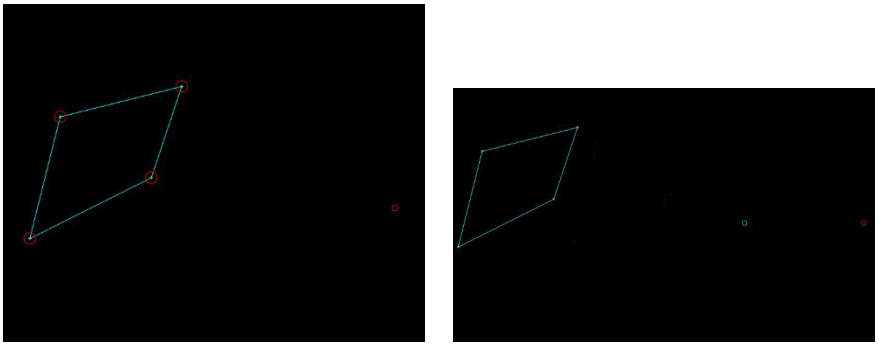
4.2.4 Probabilistic Hough transform

The probabilistic Hough transform (PHT) used to estimate the global shape. Based on a group of features within the scene, identifying the present of a model image in a scene image. The hypothesised location of the model in the scene is indicated based on the conditional probability that any pair scene lines agreement about a position in model.

Estimating the global shape has two main stages. Firstly, training process begin by recording the perpendicular distance and the angle from a reference point to each pair of model lines. Secondly, predicting the pose of scene difference from the model, then we estimate the location of the landmarks. We create a Hough space to store value when exist a pair of scene lines match with the entry in training process. The peak in Hough space is assumed of the reference point of the model in the scene. From this reference point, we can estimate the reference landmarks of reference image on the scene. The process to estimate the global pose is described as follows:

- Choose a arbitrary point in model as reference point
- For each pair lines in model, calculating and recording the perpendicular distance and angle from the reference point to each line.
- Create an two dimensional accumulator, one dimension for the angle and another for the perpendicular distance.
- For each pair lines in scene, finding the entry correspond about position, orientation and scale. Increase the value at correlative cell in the accumulator (indicate by the angle and distance).
- Compute the maximum value in accumulator.
- Indicating the pair of scene lines and the entry with maximal value of accumulator.
- Extending the perpendicular lines of the pair of scene lines at the appropriate position. The intersection of them is the location of reference point in the scene.

In the example below, we apply the PHT to estimate the landmarks of model in the scene. The image in figure 4.2a as the model. In the model, the small red circle and large red circles are the reference point and the landmarks in model, respectively. The image in figure 4.2b as scene. By applying the PHT, we can estimate the reference point (green circle) in the scene and the location of the landmarks (the yellow circles).



(a) The model image (b) The scene image

Figure 4.2: The landmarks estimated by probabilistic Hough transform

Training process

In this step, recording the perpendicular distance and angle from each pair model lines to a reference point (called reference table). The reference point can be chosen at arbitrary position on model image. To save the time to process at the next step, we just consider the "closet pair lines". In this application, the reference point chosen at the center of image and the closet pair lines are pair of lines have all three conditions: the length of each line greater than 60 pixels, the lines are not parallel and the distance between it less than 5 pixels. The algorithm to consider a pair closet lines and construct the reference table as follows:

Data: line1 (the first line), line2 (the second line)
Result: Two line closet or not (bool)
1 $distance1 \leftarrow$ distance from the first endpoint of $line1$ to $line2$;
2 $distance2 \leftarrow$ distance from the second endpoint of $line1$ to $line2$;
3 **if** $line1.length() > 60$ *and* $line2.length() > 60$
4 *and* $line1$ *not parallel with* $line2$
5 *and (*$distance1 <= 5$ *or* $ditance2 <= 5$ *)* **then**
6 | *return true*;
7 **end**
8 return $false$;

Algorithm 7: Algorithm to consider the closet lines

Data: lines (a list of lines), refPoint (the reference point)
Result: The reference table
1 Declare the reference table $refTable$;
2 **for** *line i in* $lines.size()$ **do**
3 | **for** *line j in* $lines.size()$ **do**
4 | | **if** $i != j$ *and* $line(i)$ *closet with* $line(j)$ **then**
5 | | | Compute the angle and perpendicular distance from $line(i)$ to $refPoint$;
6 | | | Compute the angle and perpendicular distance from $line(j)$ to $refPoint$;
7 | | | Create an entry to store pair of lines and its information ;
8 | | | Add the entry into reference table ;
9 | | **end**
10 | **end**
11 **end**
12 return reference table ;

Algorithm 8: Algorithm to construct the reference table

Estimating process

The estimating process is duration estimate the reference landmarks on the scene image. Firstly, we need to find the reference point on scene image. Secondly, we estimate the reference landmarks on the scene image from the reference point.

By finding a pair of scene lines agree with a pair of model lines, we can detect the position of the reference point on scene image. We create an accumulator to store each agreement between pair of scene lines and pair of model lines. For each pair of scene lines, we find its exist in the reference table and increase the value at correspondence position in accumulator. At the end, we can obtain the pair of scene lines and pair of model lines correspondence with the maximum value in accumulator. By extending the perpendicular lines of the pair of scene lines at the

appropriate position, we can meet the reference point at the intersection of them.

In this program, two pair lines called agreement if the angle difference between them less than one degree and the length scale less than two. Two algorithm followed describe the definition between two pair lines and finding the position of reference point in scene image.

Data: line1 (the first reference line), line2 (the second reference line), sline1 (the first scene line), sline2 (the second scene line)
Result: Two pair lines similar or not (boo)

```
1 angle1 ← angle between line1 and line2;
2 angle2 ← angle between sline1 and sline2;
3 if abs(ange1 - angle2) < 1 and abs(line1.length()/sline1.length() -
  line2.length()/sline2.length()) < 2 then
4 |   return true;
5 end
6 return false ;
```

Algorithm 9: Algorithm to check the agreement between two pair lines

Data: refTable (the reference table), slines (pair of scene lines)
Result: The entry in reference table

```
1 Declare the entry in reference table entry ;
2 for  entry et in refTable do
3 |   if agree between lines in entry and slines then
4 |   |   entry ← et;
5 |   end
6 end
7 return entry ;
```

Algorithm 10: Algorithm to find the agreement of pair scene lines in model

Data: lines (a list of scene lines), refTable (reference table)
Result: The reference table

```
1  Create an accumulator, acc;
2  Declare the reference table refTable ;
3  for  line i in lines.size() do
4  |   for line j in lines.size() do
5  |   |   if i != j and line(i) closet with line(j) then
6  |   |   |   Find the agreement of pair scene lines in mode;
7  |   |   |   Increase the value in acc with correspondence position;
8  |   |   |   Marked the maximum value, pair of scene lines and entry in reference table;
9  |   |   end
10 |   end
11 end
12 Find the intersection (intersect) between two perpendicular lines with pair scene lines at
   appropriate position;
13 // The appropriate position is correct with the distances in reference
   table.
14 return intersect ;
```

Algorithm 11: Algorithm to find the reference point in scene

By finding the reference point, the landmarks in scene image can be estimated by calculating the relative between the reference point and the reference landmarks. Besides, we also record

the difference about rotation, orientation and scale between model image and scene image.

4.2.5 Template matching

Template matching is duration to refine the estimated landmarks on the scene image with an appropriate method.

Cross-correlation

Cross-correlation is a method of estimating the similarity between two signals. By computing the sum of product between 2 signals when sliding, and choose the maximal value. It is used for searching a short signal in a longer signal. In image processing, it used to detect the present of an object (template) in a large object (image). The equation of cross-correlation as follows (equation 4.6):

$$R_{ccorr}(x,y) = \sum_{x',y'}[T(x'.y').I(x+x',y+y')] \tag{4.4}$$

Where:

- T is template which use to slide and find the exist in other image.
- I is image which we expect to find the template image
- (x', y') are coordinates in template where we get the value to compute.
- $(x+x', y+y')$ are coordinates in image where we get the value to compute when template T sliding.

By sliding the template on image by each pixel from left to right and top to down. At each position, we compute the $R_{ccorr}(x,y)$. The position have maximal $R_{ccorr}(x,y)$ is position that best similar of template in image.

However, if we use the original image to compute and find the similarity, the brightness of template and image can change the conditions and the result. So, we can be normalize the image before apply the cross-correlation to reduce the effect of lighting difference between them. The normalization coefficient is:

$$Z(x,y) = \sqrt{\sum_{x',y'} T(x'.y')^2 . \sum_{x',y'} I(x+x',y+y')^2} \tag{4.5}$$

The value of this method when we normalized computation as below:

$$R_{ccorr\_norm}(x,y) = \frac{R_{ccorr}(x,y)}{Z(x,y)} = \frac{\sum\limits_{x',y'}[T(x'.y').I(x+x',y+y')]}{\sqrt{\sum\limits_{x',y'} T(x'.y')^2 . \sum\limits_{x',y'} I(x+x',y+y')^2}} \tag{4.6}$$

Template matching

Now, back to the our problem. With a reference image and its set of landmarks. We use the cross-correlation to refine the landmarks. In this case, the template is a region around each landmark in reference image and the image is also a region around the Hough landmark detection in scene image. Hence, to save the time to process, before applying the cross-correlation, the scene image rotated to match with model using Hough estimate.

For each landmarks in reference image, we create a bounding box around the landmarks with an arbitrary size and use landmark as center point. When create the bounding box, we also need to keep the distance between left corner to the landmarks, because sometime, with the landmark position, the size of bounding box can be over the size of image. Use this box as template and do the cross-correlation with each scene image. The results obtained store the location where template match with image. From these position, we can indicate the position of each landmark of reference image on scene image. The algorithm to create the bounding box around a landmark described follows:

Data: image (reference image), landmark (location of a reference landmark), tsize (size of bounding box), distance (to keep the distance from the landmark to bounding box)
Result: A matrix represented for bounding box of landmark

Get the matrix of image (image presented by matrix):
$MatmatImg = image.getMatrix()$;
`// Indicate the top left-corner of bounding box:`
$int\ lx = (landmark.x - tsize/2) < 0\ ?\ 0\ :\ (landmark.x - tsize/2)$;
$int\ yx = (landmark.y - tsize/2) < 0\ ?\ 0\ :\ (landmark.y - tsize/2)$;
`// Keep the distance from the landmark to bounding box`
$distance.x = landmark.x - lx$;
$distance.y = landmark.y - ly$;
`// Indicate the low right-corner of bounding box`
$int\ lx2 = (landmark.x + tsize/2) > matImg.cols\ ?\ matImg.cols\ :$
$(landmark.x + tsize/2)$;
$int\ yx2 = (landmark.y + tsize/2) < matImg.rows\ ?\ matImg.rows\ :$
$(landmark.y + tsize/2)$;
`// Create the bounding box around landmark`
$Mat\ box(matImg, Rect(lx, ly, lx2 - lx, ly2 - ly))$;
return the box;

Algorithm 12: Algorithm to create a bounding box around a landmark

The below algorithm describe the method to estimate the reference landmarks on scene image by using cross-correlation. Before applying the cross-correlation, the scene image rotated to match with the model. The angle used to rotate is sum of the difference between the scene line and model line to which it matched and the difference between two pairs of similar lines. To apply the cross correlation, we have used the function $matchTemplate$[4] in OpenCV library with matching method is CV_CCORR_NORMED (cross-correlation normalize). This function allow we compare the template overlap the image and it support many different methods to match. When the template slide over each pixel on image, the coefficient between them was calculated and stored in a array.

After finished, to get the value and position of maximum value when we compute the coefficient,

[4]http://docs.opencv.org/modules/imgproc/doc/object_detection.html?highlight=matchtemplate#matchtemplate

we use a function in OpenCV, $minMaxLoc$[5]. This method used to detect the minimum and maximum value in an array. Beside that, it also output the location where having the minimum and maximum value.

Data: refImage (reference image), sceneImage (the scene image), lmpath (file path store the reference landmarks)

Result: A list of landmarks on scene image

1 Get the reference landmarks from file and store in list $refLandmarks$;
2 Create a variable to store the new landmarks: $sceneLandmarks$;
3 Estimate the reference landmarks ($refLandmarks$) in scene image using probabilistic Hough transform and save into a variable: $esLandmarks$;
4 `// Get the matrix of scene image`
5 $sceneMatrix = sceneImage.getMatrix()$;
6 Rotate the scene matrix with appropriate angle;
7 **for** *variable i in* $esLandmarks.size()$ **do**
8 `// Get the reference landmark`
9 $Point\ refPoint = refLandmarks.at(i)$;
10 `// Create a bounding box of reference landmark` $refPoint$
11 $Mat\ template = createTemplate(refImage, refPoint, size)$;
12 `// Get the estimate landmark`
13 $Point\ esPoint = esLandmarks.at(i)$;
14 `// Create a bounding box of estimate landmark` $esPoint$
15 $Mat\ sceneImg = createTemplate(sceneImage, esPoint, size)$;
16 Create the matrix to store the value when do the cross-correlation: $result$;
17 `// Apply the matching and store the result into matrix` $result$
18 $cv :: matchTemplate(sceneMatrix, template, result, CV\_TM\_CCORR\_NORMED)$;
19 `// Get the maximum value and position in` $result$ `matrix`
20 $double\ maxValue, minValue$;
21 $Point\ maxLoc, minLoc$;
22 $cv :: minMaxLoc(result, \&minValue, \&maxValue, \&minLoc, \&maxLoc, Mat())$;
23 Compute the position of landmark from maximum position;
24 Push the landmark into the list $sceneLandmarks$;
25 **end**
26 Return the list of landmarks;

Algorithm 13: Algorithm to get the position of reference landmarks in scene image

[5] http://docs.opencv.org/modules/core/doc/operations_on_arrays.html?highlight=minmaxloc#minmaxloc

Chapter 5

Implementation

5.1 Software architecture

Continue the IMP tool, all the functions for this task was saved in the **impls_2015** package of program. Besides the method was created by myself, I also use some methods from the OpenCV (library for image processing) and Qt framework (framework for C++).

The class diagram[1] in 5.1 show mainly classes of my task. The *mainly* methods located in the

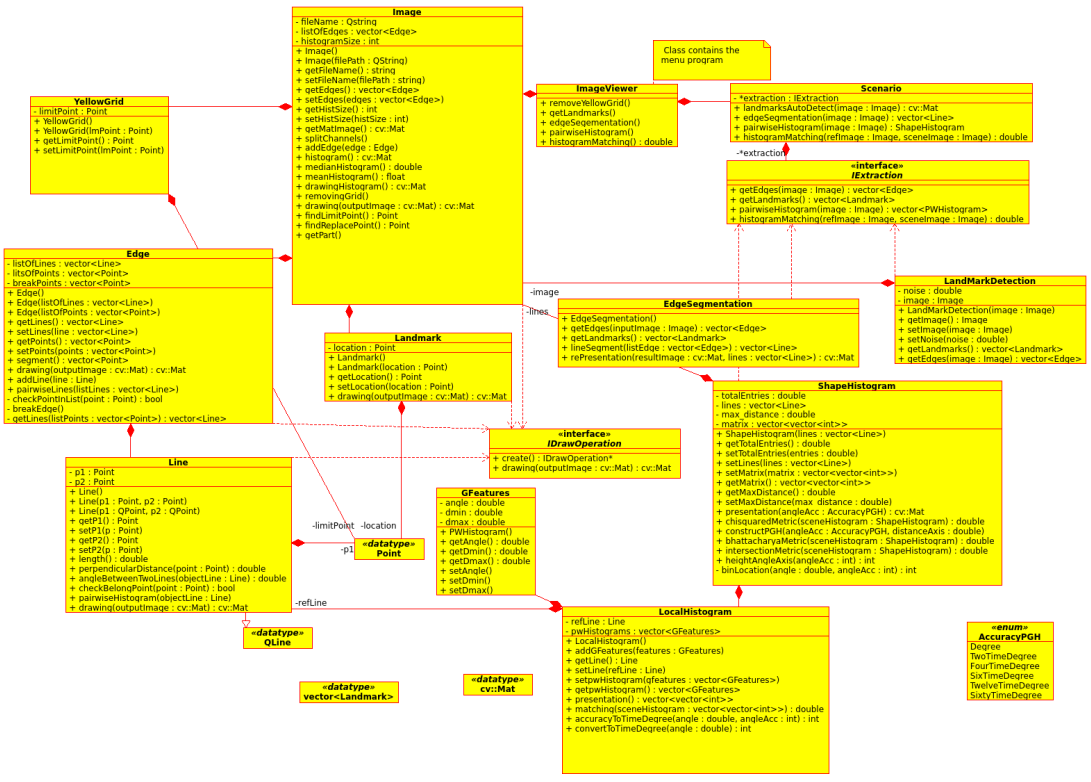


Figure 5.1: The class diagram of program

ImageViewer class, where contains all functions of the software. To represent the information of image and preprocessing about clear the yellow grid, we use the classes such as *Line*, *Edge*, *Landmark*, *YellowGird*, *Image* class. For the edge segmentation, construct the pairwise geometric

[1]See the full image in Appendix

histogram, probabilistic hough transform and landmarks detection, we have *GFeatures*, *LocalHistogram*, *ShapeHistogram*, *EdgeSegmentation*, *PHTransform LandmarkDection* class. The main functions were inherated from the abstract classes *HistogramMethod*, *SegmentMethod*, *HoughMethod*, *LandmarkMethod* respective and used in *main* class (*ImageViewer*) via the *Sceneario* class. The description, properties and methods in each class was discussed in below sections.

5.2 Image preprocessing

The *Image processing* section describes the information about the classes which describe the geometric objects can be represent the image and the method to remove the yellow grid on the images.

5.2.1 Line class

Line class describe the information of a straight line and the methods can do with a line.

The attributes

- **p1**: the first endpoint of line.
- **p2**: the second endpoint of line.

The methods

- **Line()**: Constructor an empty line.
- **Line(p1: Point, p2: Point)**: Constructor a line with two endpoints p1, p2. With type of the endpoints is **Point** (in OpenCV)
- **getP1()**: Getter the first endpoint
- **setP1(p:Point)**: Setter the first endpoint
- **getP2()**: Getter the second endpoint
- **setP2(p:Point)**: Setter the second endpoint
- **length()**: Calculate the length of line
- **perpendicularDistance(point:Point)**: Compute the perpendicular distance from point "**point**" to the line
- **angleBetweenTwoLines(objectLine: Line)**: Compute the angle between two lines.
- **checkBelongPoint(point: Point)**: Check a point stay on the line or not. If the point stay on the line, the return value is true; otherwise, return false.
- **intersection(objectLine: Line)**: Finding the intersection point of two lines. If two lines have intersect, the method will return the coordinate of intersection point; otherwise, method will return a negative point.
- **interection(equation1:vector<double>, equation2: vector<double>)**: Finding the intersection of two lines. In this case, the lines presented under $y = ax + b$, and vector input contains three coefficient of line.

- **pairwiseHistogram(objectLine: Line)**: Finding the geometric features between two lines. The return value is an **GFeatures** object, it includes the information of angle between two lines, the distance from two endpoints of *objectLine* to the reference line.
- **equationOfLine()**: Calculate the equation of line.
- **drawing(outputImage: Mat)**: Drawing the line on the output image.
- **operator==(line:Line)**: Defining the equal lines.
- **parallelLine(distance: double)**: Finding the lines parallel with it and the distance between them is *distance*. If found, the method return a list of equation of the lines.
- **interParallel(line1: Line, line2: Line, distance1: double, distance2: double, width: int, height: int)**: Finding the intersection between two lines parallel with $line1$ and $line2$ in an image have the size $widthxheight$. The distance from $line1$ to its parallel line is $distance1$, the distance between $line2$ and its parallel is $distance2$.
- **isNull()**: Checking a line is null. Method return $true$ if two endpoints of line are equal, otherwise return $false$.

5.2.2 Edge class

The **Edge** class used to presented for a curve and the methods with edge. An edge can be presented by a list of lines or a list of points.

The attributes

- **listOfLines**: list of approximate lines of edge.
- **listOfPoints**: list of point represented edge.
- **breakPoints**: list of break points on edge when edge broken to the approximate lines.

5.2.3 The methods

- **Edge()**: Construct an empty edge.
- **Edge(lines: vector<Line>)**: Constructor an edge from a list of lines.
- **Edge(point: vector<Point>)**: Constructor an edge from a list of points.
- **getLines()**: Get the lines of edge.
- **getLines(listPoints: vector<Point>)**: Get the lines from a list of points.
- **setLines(lines: vector<Line >)**: Set the lines of edge.
- **getPoints()**: Get the points of edge.
- **addLine(line: Line)**: Add a line into the list of lines of edge.
- **readFile(filePath: QString)**: Get the list lines of edge from a file.
- **breakEdge(),segment()**: Break edge into approximate lines from the list points of edge.
- **drawing(outputImage Mat)**: Draw edge

5.2.4 Image class

Image class presented the information of an image such as file name, list of edge extracted from it,... and the methods with image.

The attributes

- **fileName**: File name of image
- **listOfEdges**: List of edges extracted from image
- **listOfLandmarks**: List of landmark of image
- **matrixImage**: Matrix represented for image
- **pghHistogram**: Pairwise geometric histogram (PGH) presented for image
- **histogramSize**: Histogram size of image (256)

The methods

- **Image()**: Construct an empty image
- **Image(filePath: QString)**: Construct an image with a full path
- **getFileName()**: Get the full path of image
- **setFileName(filePath: QString)**: Set the path of image
- **getEdges()**: Get the edges from image
- **setEdges(edges: vector <Edge>)**: Set the edges of image
- **getMatrixImage()**: Get the matrix presentation of image
- **addEdge(edge: Edge)**: Add an edge into list edges of image
- **histogram()**: Get the histogram of image
- **medianHistogram()**: Compute the median of histogram of image
- **meanHistogram()**: Compute the mean of histogram of image
- **drawingHistogram()**: Draw the histogram of image
- **drawing(outputImage: Mat)**: Draw image
- **getPart()**: Get the kind of image (elytre, tete, mandibule,...)
- **splitChannels()**: Split the image into several channel colors, if image in BGR mode.
- **findLimitPoint()**: Find the "limit point" of yellow grid
- **findReplacePoint()**: Find the "replace point"
- **removingGrid(minBrightness: int)**: Remove the yellow grid on image
- **lineSegment()**: Segment the edges of image into set of approximate lines.

- **setShapeHistogram(shapeHist: ShapeHistogram)**: Set PGH of image.
- **getShapeHistogram()**: Get PGH histogram of image
- **readLandmarksFile(filePath: string)**: Read the landmarks of image from a file
- **readImagesFolder(folderPath: QString)**: Read the images from a folder.
- **getName()**: Get name only of image.

5.3 The abstract classes

The abstract classes contains the abstract methods get the actions on image such as segmentation, PGH construction,.... The methods were implemented by the inherit classes, respective and provide the way access to action for other classes. The abstract classes include: *HistogramMethod* class, *SegmentMethod* class, *HoughMethod* class, *LandmarkMethod* class.

5.4 Edge segmentation

This section describes the classes used for segment an image. Besides the classes construct the edge which described in previous section such as *Line, Edge,...*, we also provide the access method for another classes.

The **EdgeSegmentation** class provides the methods such as obtain the lines from an image, presentation the result after segmentation or applying the segmentation on an images folder. The detail methods as below:

- **EdgeSegmentation()**: Constructor of edge segmentation class
- **lineSegment(image: Image)**: Extract the lines in an image. This method return a list of approximate lines of object in image.
- **rePresentation(outputImage: Mat, lines: vector <Line>)**: Present the list of lines.
- **segmentDirectory(inputFolder: QString, outputFolder: QString)**: Apply the segmentation on *inputFolder* and save the result into *outputFolder*

5.5 Construct pairwise geometric histogram

This section describes the classes used in process of PGH construction.

5.5.1 GFeatures class

GFeatures class contains the relative information of the objects in PGH such as angle, minimal distance and maximal distance.

The attributes

- **angle**: The angle between two lines
- **dmin**: The minimum distance between two lines
- **dmax**: The maximum distance between two lines

The methods

- **GFeatures()**: Constructor of class
- **getAngle()**: Get the angle
- **getDmin()**: Get the minimum distance
- **getDmax()**: Get the maximum distance
- **setAngle(angle: double)**: Set the angle
- **setDmin(dmin: double)**: Set the minimum distance
- **setDmax(dmax: double)**: Set the maximum distance

5.5.2 LocalHistogram class

LocalHistogram class constructed for containing the informations when computing the PGH of a line in object. Besides, it also have the methods help the user change the accuracy.

The attributes

- **pwHistogram**: List of relative information between this line and another lines.
- **maxDistance**: The maximal distance in PGH of this line.

The methods

- **LocalHistogram()**: Constructor of a local histogram
- **convertAngleToMinute(angle: double)**: Convert the angle from degree to minute
- **getPWHistogram()**: Get the list PGH relative information
- **setPWHistogram(gfeatures: vector<GFeatures>)**: Set the list PGH relative information
- **getMaxDistance()**: Get the maximum distance in PGH.
- **addGFeatures(features: GFeatures)**: Add a relative information
- **constructorMaxtrix(angleAcc: AccuracyPGH, distanceAxis: int, totalEntries: int)**: Construct the matrix contains the PGH
- **bhattacharyyaMetric(sceneHist: LocalHistogram, distanceAxis: int, angleAcc: AccuracyPGH)**: Compute the measure distance between this PGH and another local histogram with an angle accuracy and distance accuracy.
- **accuracyToTimeDegree(angle: double, angleAcc: AccuracyPGH)**: Compute the location of angle in PGH matrix with an angle accuracy.
- **heightOfAngleAxis(angleAcc: AccuracyPGH)**: Compute the length of angle axis in PGH matrix
- **distanceOffset(maxDistance: double, distance: double, cols: int)**: Compute the location of distance in PGH matrix.
- **AccuracyPGH**: Angle accuracy enum

5.5.3 ShapeHistogram class

ShapeHistogram class contains the information about PGH of an object. It was constructing based on combine all PGH of the lines in object.

The attributes

- **listLocalHistogram** : List of local histogram (The local histogram constructed for each line in object)
- **max_distance**: The maximum distance in shape's PGH
- **matrix**: The matrix present for shape's PGH
- **totalEntires**: Total entries in matrix of shape's PGH

The methods

- **ShapeHistogram()**: Constructor of a shape PGH
- **getTotalEntries()**: Get the total entries
- **setTotalEntries(entries: double)**: Set the total entries
- **getMaxDistance()**: Get the maximum distance
- **setMaxDistance(distance: double)**: Set the maximum distance
- **getMatrix()**: Get the matrix
- **setMatrix(matrix: vector<vector<int>>)**: Set the matrix
- **getListLocalHistogram()**: Get the list of local histograms
- **constructPGH(prLines: vector <Line>)**: Construct the PGH of a shape from a list lines.
- **constructMatPGH(angleAcc: AccuracyPGH, cols: int)**: Construct the matrix contains PGH with an angle accuracy and distance accuracy
- **writeMatrix(fileName: QString)**: Save the PGH matrix into a file
- **distanceOffset(distance: double, cols: int)**: Compute the offset of *distance* in the matrix *cols* columns.
- **bhattacharyyaMtric(sceneHist: ShapeHistogram)**: Compute the measure distance between it and another shape PGH using Bhattacharyya
- **chiSquaredMetric(sceneHist: ShapeHistogram)**: Compute the measure distance between it and another shape PGH using Chi-squared
- **intersectionMetric(sceneHist: ShapeHistogram)**: Compute the measure distance between it and another shape PGH using intersection

5.5.4 GeometricHistogram class

This class provides the access ways for another classes. By this class, the user can compute the pairwise geometric histogram of an image and calculate the distance between the PGH histograms. The methods in **GeometricHistogram** class include:

- **GeometricHistogram()**: Constructor of **GeometricHistogram** class
- **shapeHistogram(image: Image, angleAcc: AccuracyPGH, distanceAcc: int, matresult: Mat)**: Construct the PGH of an image with an accuracy about angle and distance.
- **pairwiseHistogramDirectory(folderPath: QString, angleAcc: LocalHistogram, distanceAcc: int)**: Construct the PGH for all image in a folder based on the angle and distance accuracy.
- **getDistanceMetric(refHist: ShapeHistogram, sceneHist: ShapeHistogram, matching: MatchingMethod)**: Compute the measure distance between two PGHs based on a method matching (Bhattacharyya, Chi-squared or intersection).
- **pghHistogramMatching(refImage: Image, sceneImage: Image, matching: MatchingMethod, angleAcc: AccuracyPGH, distanceAcc: int)**: Compute the measure distance between two images based on a method matching (Bhattacharyya, Chi-squared or intersection), an accuracy about angle and distance.
- **pghHistogramDirectoryMatching(refImage: Image, folderPath: QString, matching: MatchingMethod, angleAcc: AccuracyPGH, distanceAcc: int)**: Compute the measure distance between an reference image and the images in a folder.

5.6 Estimate the global pose

5.6.1 HoughSpace class

HoughSpace class contains the angle and distance from a line to a reference point. It was used when we construct the probabilistic hough transform

The attributes

- **distance**: Distance from line to reference point
- **angle**: Angle between line and reference point(line via reference point and parallel with x axis)

The methods

- **HoughSpace()**: Constructor of *HoughSpace* class
- **getDistance()**: Get the distance
- **setDistance(distance: double)**: Set the distance
- **getAngle()**: Get the angle
- **setAngle(angle:double)**: Set the angle

- **computeDistance(line: Line, refPoint: Point)**: Compute the distance from a line to a reference point
- **computeAngle(line: Line, refPoint: Point)**: Compute the angle between a line and a reference point
- **closestPoint(line: Line, origin: Point)**: Compute the coordinate of a line via the **origin** point and perpendicular with *line*

5.6.2 PHTEntry class

The **PHTEntry** class present for each entry when constructing the reference table in training process. Each entry contains the pair of lines and its information about angle and distance to a reference point.

The attributes

- **line1**: The first line
- **line2**: The second line
- **listHoughSpace**: The list of information about angle and distance from two lines to the reference point.

The methods

- **PHTEntry()**: Constructor of **PHTEntry** class
- **setLine1(line: Line)**: Set the first line
- **setLine2(line: Line)**: Set the second line
- **getLine1()**: Get the first line
- **getLine2()**: Get the second line
- **getHoughSpaces()**: Get the information of pair lines
- **addHoughSpace(houghSpace: HoughSpace)**: Add a pair angle and distance from a line to reference point into list.

5.6.3 PHoughTransform class

The **PHoughTransform** class describe the main process when we apply the probabilistic hough transform to estimate the landmarks. It includes the methods to construct the reference table, find the reference point in scene image and estimate the landmarks.

The methods

- **PHoughTransform()**: Constructor of **PHoughTransform** class
- **closetLine(line1: Line, line2: Line)**: Check two lines are closet or not.
- **similarPairLines(ref1: Line, ref2: Line, scene1: Line, scene2: Line)**: Check the similar of two pair of lines. Pair of scene lines *(scene1, scene2)* and pair of model lines *(ref1, ref2)*.

- **constructTable(lines: vector <Line>, refPoint: Point)**: Construct the reference table from a list of model lines and reference point.
- **findHoughSpace(refTable: vector <PHTEntry>,line1: Line, line2: Line)**: Find the exist(agreement) of a pair of scene lines (line1, line2) in reference table.
- **matchingInScene(refTable: vector <PHTEntry>, sceneLines: vector <Line>, width: int, height: int, maxVector: vector <Line>)**: Find the pair in scene image have the maximum agreement in model. The method return the matching entry in reference table and keep the pair of scene lines has the best vote.
- **refPointInScene(entry: PHTEntry, matchLines: vector <Line>, angleDiff: double, width: int, height: int)**: Find the reference point in the scnene.
- **findLandmarks(refPoint: Point, esPoint: Point, angleDiff: double, refLandmarks: vector <Point>, width: int, height: int)**: Estimate the reference landmarks in the scene image.
- **estimateLandmarks(mImage: Image, sImage: Image, mlmPath: string, angleDiff: double, ePoint: Point)**: Estimate the landmark of a model image (mImage) on a scene image (sImage). The method also keep the angle difference between two images and positon of reference point in scene.
- **angleDifference(refLine: Line, sceneLine: Line)**: Find the angle difference between two lines
- **phtPresentation(mImage: Image, sImage: Image, mlmPath: string)**: Represent the estimate landmarks result of model image on scene image.
- **phtDirectory(mImage: Image, mlmPath: QString, sceneDir: QString, saveDir: QString)**: Estimate the landmarks of a model image on all image in a folder and save the result into another folder.

5.7 Refine the landmarks

In this step, we use template matching to refine the estimated landmarks of previous step. The **LandmarkDetection** class provides the methods to refine the landmarks. Besides, we also can compute the centroid point of object. The methods in this class described as follows:

- **createTemplate(matImage: Mat, landmark: Point, tsize: int, location: Point, distance: Point)**: Create a bounding box around a point (landmark) with the size (*tsize*) and keep the left-corner of box, distance between the point and the left-corner
- **rotateImage(image: Mat, angle: double, center: Point)**: Rotate *image* around *center* point with *angle*
- **matchingTemplate(refImage: Image, sceneImage: Image, lmPath: QString, templSize: int, sceneSize: int, angleDiff: double, mcResult: vector <Point>)**: Refine the landmarks of *refImage* on *sceneImage*. The size of bounding boxes around the reference landmarks and estimated landmarks are *templSize* and *sceneSize*.
- **measureDistance(landmarks: vector <Point>, bary: Point)**: Compute the centroid of a list of landmarks and keep the center point of it.

- **centroidMatching(refImage: Image, sceneImage: Image, lmPath: QString, templSize: int, sceneSize: int, ebary: Point)**: Compute the centroid of a model landmarks in a scene image.

5.8 Result

result...

Chapter 6

Conclusion

about conclusion

Bibliography

[1] Sasirekha Palaniswamy, Neil A Thacker, and Christian Peter Klingenberg. Automatic identification of landmarks in digital images. *IET Computer Vision*, 4(4):247–260, 2010.

[2] PA Riocreux, Neil A Thacker, and RB Yates. An analysis of pairwise geometric histograms for view-based object recognition. In *BMVC*, pages 1–10, 1994.