Harbin Institute of Technology (Shenzhen)

Image Processing Project Report

Experiment Name: Wafer Sawing Line Detection

Student Name: 潘延麒、刘俊杰

Student ID: 22S051053、22S151187

Report Date: 2023/5/3

Content

1. Project Introduction

Wafer is a circular silicon disk used for the production of semiconductor/optical devices. Generally, it is made of silicon material, and undergoes several rigorous processing and surface modification stages throughout the manufacturing process. Wafer meets its wide applications in industries, including computer hardware, mobile phones, televisions, cars, and other electronic devices. It serves as the foundation for the manufacturing of semiconductor devices and remains one of the important pillars of modern industry.

Wafer sawing is an important step for the further integration. During wafer sawing, the wafer is cut into multiple individual chips. Once the sawing process is complete, the cut chips are inspected for quality and electrical performance and the good chips are then assembled into final products. To ensure a successful sawing, it is necessary to inspect the **sawing lines** for precision and reliability of the sawing.

Generally, sawing line lies in the middle of several individual chips, as shown in Figure 1.1 (the green line). To detect the sawing line, three steps are required:

(1) Detect the edges of the sawing line (see ①②).
(2) Detect the center line of the edges, which is served as the sawing line (see ③).
(3) Calculate the angle of the sawing line (see ④).

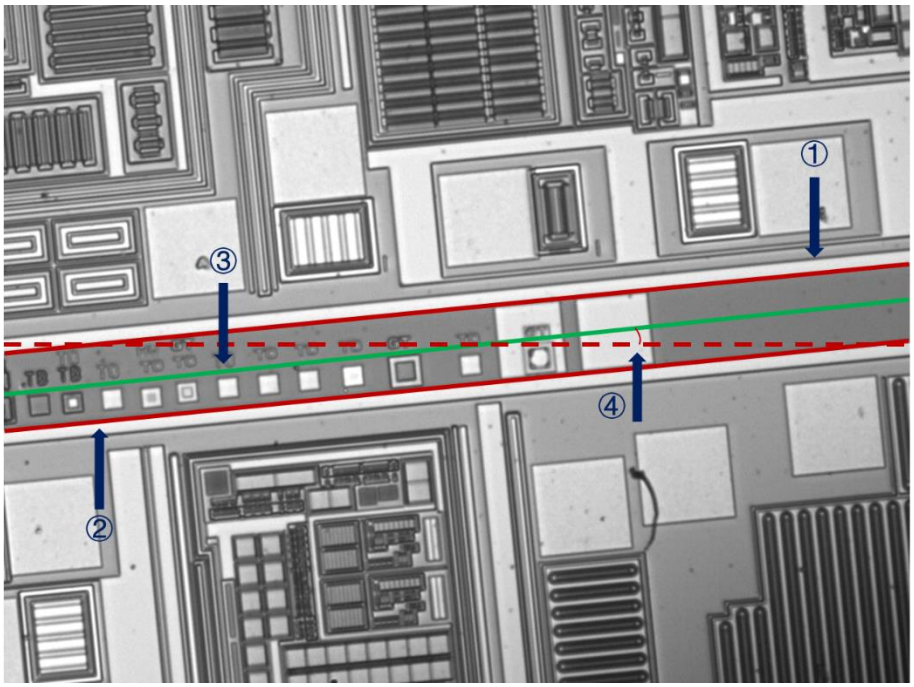
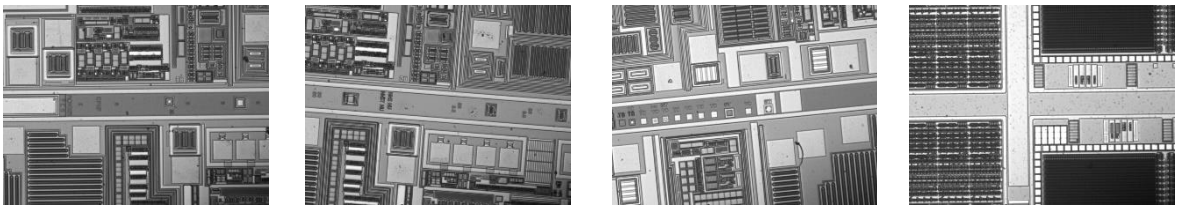


Figure 1.1. Illustration of wafer sawing line detection.

In this project, we are required to detect sawing lines on four given real-world wafers' imaging, as shown in Figure 1.2. Among these wafers, the first three wafers contain only one sawing line (Figures 1.2(a)-(c)) while the last wafer contains two crossed sawing lines (Figure 1.2(d)). To achieve the **goal**, we first make the **key observation** that all the edges of the sawing lines go through the whole wafers. Thus, the most naïve approach to extract the sawing line is to detect the longest edges in the given images (referred to as *Naïve Detector, ND*). However,

such the **length-based** approach might suffer bad robustness. For example, the extracted edges might be discontinuous (due to mis-chosen parameters or imaging noisy, see Section 5.3), leading to the case that the detected longest edges might not be the edges of the sawing lines.



(a) Wafer-1 (single) **(b)** Wafer-2 (single) **(c)** Wafer-3 (single) **(d)** Wafer-4 (crossed)

Figure 1.2. Four real-world wafers' imaging.

To address the problem of *ND*, we further propose a **border point-based** approach, *Line Through Detector (LTD)* that guarantees the detected edges go through the whole wafer, in other words, the edges are guaranteed to be continuous through the whole wafer. Specifically, *LTD* evaluates all the possible lines connected by the points on the border of the given wafer imaging, selects the edges that could be sawing lines' edge (by measuring whether they are nearly parallel), and finally verifies the combination of edges by determining if they belong to the edges of "clean area".

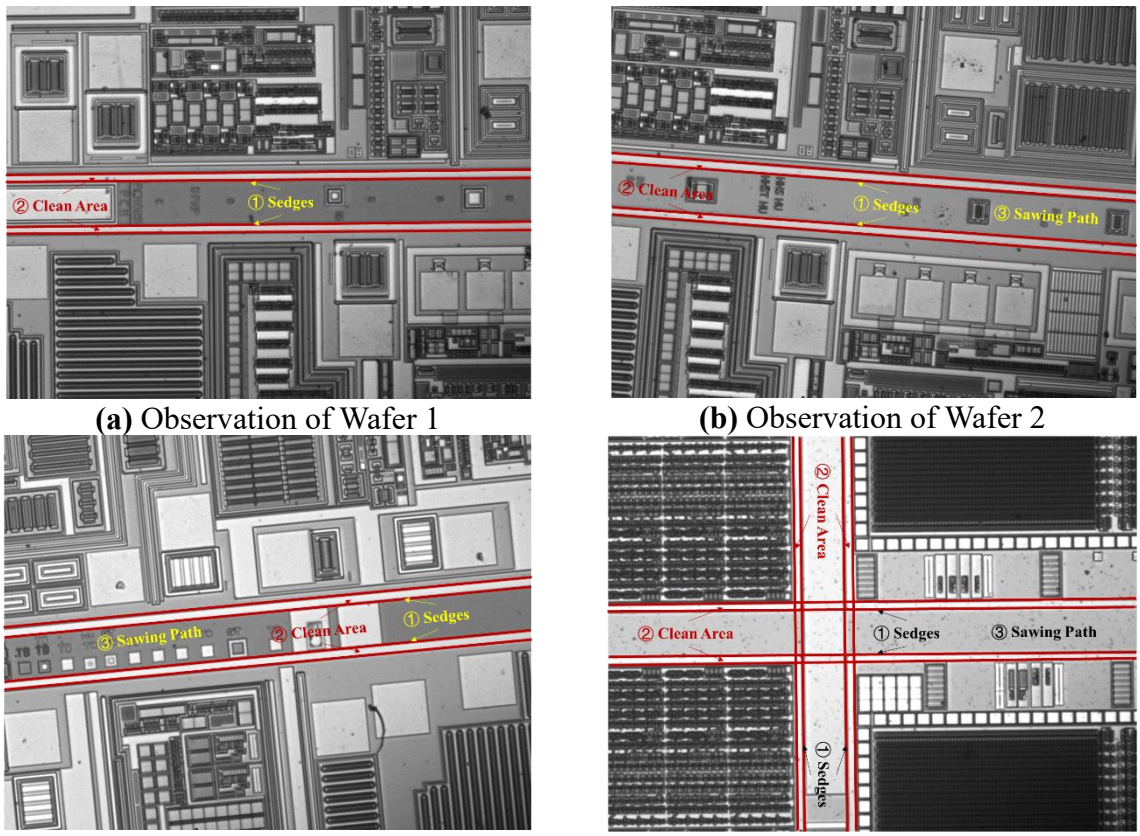
The experimental results show that *ND* and *LTD* can both detect the sawing lines in the given wafers. However, *LTD* can achieve more precise detection due to its guarantees of the continuousness of the extracted edges and the efforts on edges verifications. Moreover, we show that *LTD* can be more robust by adding noisy edges to the wafers.

In summary, our works can be concluded as the following four folds:

- A detailed analysis of four real-world wafers' imaging and obtain our key observation (Section 2): **the edges of the sawing lines intend to go through the whole wafers**. Based on the observation, we propose two algorithms, length-based *Naïve Detector* and border point-based *Line Through Detector* to detect the sawing lines.
- A *Naïve Detector (ND)* that extracts the longest edges of the given wafers' imaging (**LOC≈550**). To alleviate the problem of discontinuous detected edges, we propose a three-step processing algorithm that links the line segments, extracts the edges of sawing lines, and recursively searches other possible sawing lines (Section 3).
- A *Line Through Detector (LTD)* that extracts the boarder points belong to edges (**LOC≈1300**). *LTD* elegantly guarantees the continuousness of the extracted edges, and make every effort to verify the correctness of the possible edge pairs. Specifically, *LTD* insightfully notices the "clean area" alongside the edges of sawing lines, and verifies the edge by leveraging that area (Section 4).
- Comprehensive experiments are made to evaluate the efficiency of *ND* and *LTD*, including precision, robustness, and performance. The results show that *ND* and *LTD* can both detect the sawing lines in the given four real-world wafers' imaging. *LTD* shows a more precise result and can be more robust to resist noisy. However, *ND* is able to perform more faster than *LTD* since *LTD* introduces heavy calculations on verifications while *ND* is much simpler (Section 5).
- We'll release our code at https://github.com/Hoit-23o2/WaferDetector.git after the deadline of this project.

2. Observation and Motivation

In this section, we analyze the characteristics of the edges of sawing lines (we refer to *sedges*). As Figure 2.1 shows, the *sedges* go through the whole wafer, i.e., the end points of *sedges* are at the boundary of each imaging. For brevity, we define the area between *sedges* as *sawing path*. We further notice that there is a "clean area" alongside the *sedges* (denoted by *clean area*). We conjecture that the *clean area* is mainly used for better split individual chips on the wafer since *clean area* makes the *sedges* more distinguishable. Thus, *clean area* should be a common object for wafers.



(a) Observation of Wafer 1 **(b)** Observation of Wafer 2

(c) Observation of Wafer 3 **(d)** Observation of Wafer 4

Figure 2.1. Observation of four real-world wafers' imaging.

According to the above observation, we obtain our three motivations.

Motivation 1: Length-based Method. Since *sedges* go through the whole wafer, they intend to be the longest edges in the wafers' imaging. Thus, the basic idea here is to first detect all edges in the given imaging and extract the longest edges as *sedges*. This motivates us to implement *Naïve Detector (ND)* to illustrate such the naivest, length-based method.

Motivation 2: Point-based Method. Traditional edge detection algorithms might suffer discontinuous edges, which might affect the results of length-based method. We notice that the end points of *sedges* (we refer to these points as *spoints*) are always located at the boundary of the imaging; thus, we can detect *spoints* first, and then link the points to obtain the continuous *sedges*. Such point-based method eliminates the discontinuous problem of length-based method. However, determining the *sedges* is not straightforward since detection algorithm might detect many potential *spoints*. We propose *Line Through Detector (LTD)* to address these challenges.

Motivation 3: Verification by *Clean Area*. Correct combination of *sedges* intends to have a *clean area* around. Thus, the chosen potential *sedge* can be verified by determining the intersection between potential *sedge* and the *clean area*. However, detecting the *clean area* is not straightforward. *LTD* combines dilation and subtraction to elegantly address the challenge.

We then present the design and implementation of *ND* and *LTD* in the following sections.

3. Design and Implementation of *Naïve Detector*

The key idea of *Naïve Detector (ND)* is to detect the longest edges in the given wafer imaging as *sedges*, and then use these *sedges* to determine the sawing line. The challenge here is that the *sedges* intend to be noisy and discontinuous; thus, we need a way to connect edges. To address the challenges, we introduce a three-step-processing algorithm by running on Wafer 1 (if not otherwise specify), as described below.

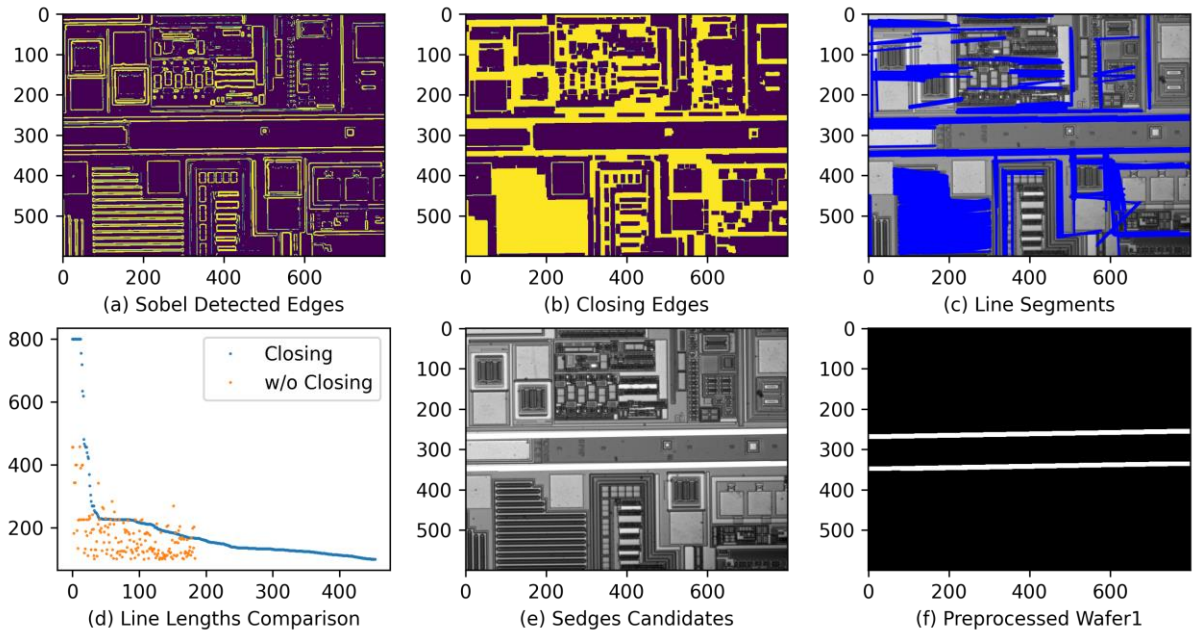
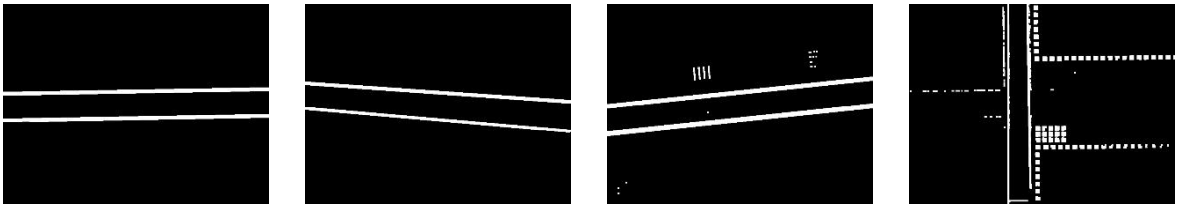


Figure 3.1. Illustration of Preprocessing step of *ND*.

Pre-processing: Edge Detection & Linking. Figure 3.1 illustrates the process of pre-processing. Pre-processing is mainly used to detect the edges and extract *sedges* in a coarse granularity. Specifically, *ND* first uses Sobel Operator to extract possible edges from the wafer (Figure 3.1(a)). However, these edges can be discontinuous, as shown in Figure 3.1(d) (w/o Opening), the longest of which cannot exceed 500px, while the width of the imaging is longer than 600px. To address the problem, *ND* applies a **closing** operation to "link" those line segments (Figure 3.1(b)), and then *ND* performs Hough Line detection algorithm to obtain the lines that longer than 100px while the maximum gap is 10px (i.e., with the parameters minLineLength=100 and maxLineGap=10). We illustrate the detected lines in Figure 3.1(c) and measure the length of them in Figure 3.1(d) (Opening). In the figures, we can find that line segments are intuitively linked together. According to the detected line length statistics, *ND* selects top K lines with length in 800±100px as **candidate *sedges*** and highlights these lines (with white colors) in the original wafer imaging, as shown in Figure 3.1(e). To further reduce the interference of background, *ND* uses a simple thresholding to extract the candidate *sedges*, i.e., we set the pixels that not 255 to 0; thus, the white candidate *sedges* remain and we finish the preprocessing (Figure 3.1(f)).

Figure 3.2 shows the preprocessed results of all four wafers' imaging. Note that the results of preprocessing are coarse since we select many candidates in this step but only two *sedges* are actually required. The subsequent step, i.e., Processing, is to extract the exact two *sedges*.



(a) Pre Wafer 1 (a) Pre Wafer 2 (a) Pre Wafer 3 (a) Pre Wafer 4

Figure 3.2. Preprocessed wafers by *ND*. *ND* removes most noisy background information, leaving candidate *sedges* that can be easily detected and extracted.

Processing: Sedges Extraction. Processing is mainly used to extract *sedges* from preprocessed results in a fine-grained way. We first make an observation on the preprocessed result, and then introduce how processing works. As Figure 3.3 shows, the gap between *sedges* (i.e., the width of *sawing path*, denoted by *outer group gap*) is far larger than the width of *clean area* (denoted by *inner group gap*), which can be leveraged to split candidate *sedges* into two groups. In addition, we assume that the closest candidate *sedges* of each group are the right *sedges*; thus, we can obtain *sedges* by comparing the centering distance of candidate *sedges* in each group.

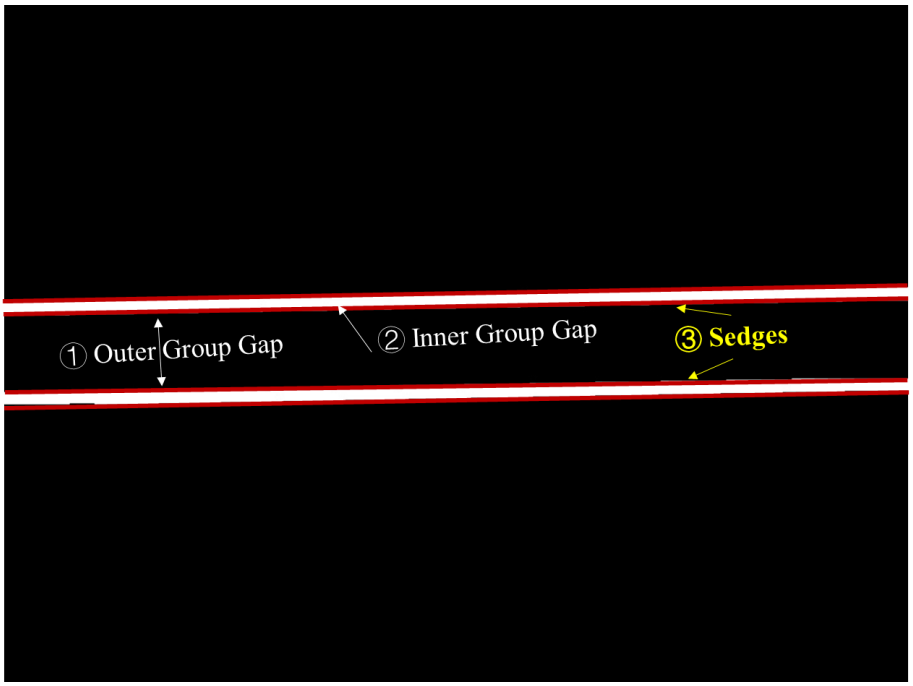


Figure 3.3. The observation on preprocessed result (Wafer 1).

Based on the above observations, we present the processing steps. As Figure 3.4 shows, *ND* first performs Sobel edge detection followed by a Hough Line detection to extract candidate *sedges* of preprocessed wafer imaging. We draw these candidates in the original wafer imaging for better observation, as shown in Figure 3.4(a). Their lengths are calculated in Figure 3.4(b). In the figure, the number of line segments decrease dramatically after the preprocessing (see Figure 3.1(d)), reducing the interference of unnecessary edges. Next, we run KNN algorithm to group these candidates into two groups, and leverage a double-for loop to select the closet two

candidate from these two groups as *sedges* (based on our assumption). At last, we fine-tune the end points of these two *sedges* to ensure that they go through the whole wafer imaging (by adjust x-axis) and nearly parallel (by adjust y-axis). Specifically, assume the slope of one *sedge* is K, then we rotate another *sedge* until its slope also equals to K.

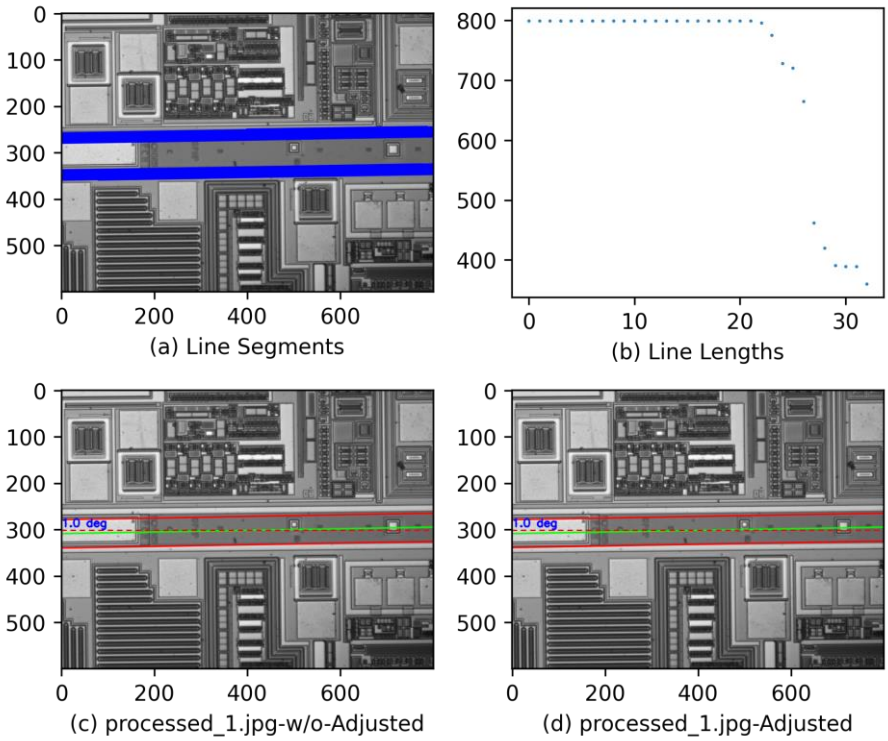


Figure 3.4. Illustration of Processing step of *ND*. The *sedges* detected in Wafer 1 go through the whole wafer imaging and nearly parallel; thus, the adjustment shows marginal effects.

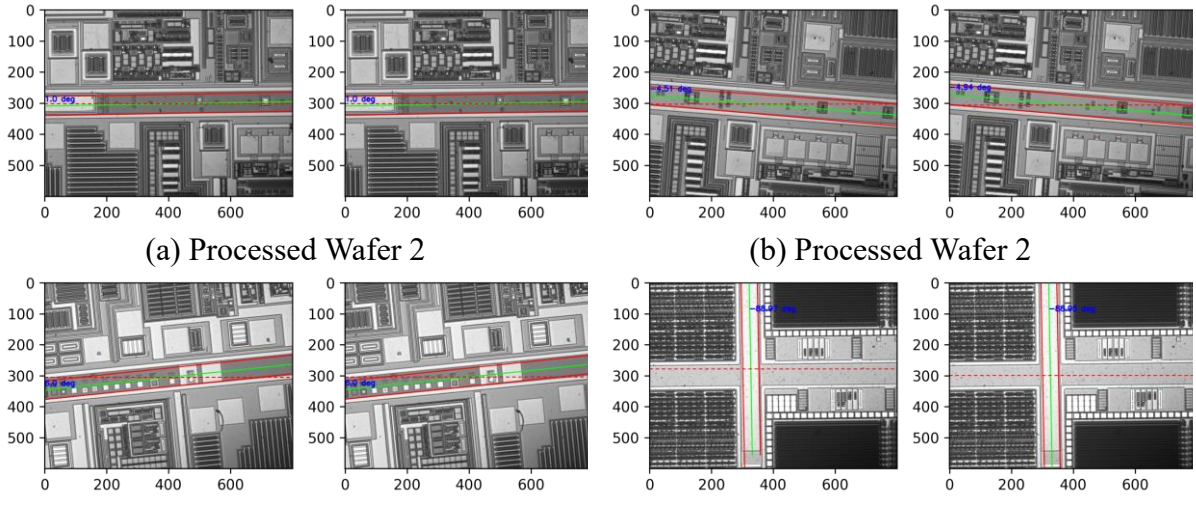


Figure 3.5. Processed wafers by *ND*. In each subfigure, the left figure indicates the sawing line without adjustment, while the right one is performed with fine-grained adjustment.

Figure 3.5 presents the final results after processing. Fine-grained adjustment can improve continuousness of *sedges* and ensure they go through the whole wafer imaging, as illustrated in

Figure 3.5(d). However, such the adjustment might introduce unacceptable error since *ND* forces the parallelism of selected *sedges*. For example, in Figure 3.5(b), the angle of the sawing line is changed from -4.51° to -4.94° after the adjustment. We attempt to redesign the strategy of the adjustment in the future.

Post-processing: Sawing Line Candidates Detection. The most challenging task for *ND* is to detect multiple sawing lines, such as Wafer 4. The above preprocessing and processing steps only help us detect the *sedges* that has clear edge. However, the horizontal *sedges* in Wafer 4 are intercepted by the vertical *sedges*, and thus they are not continuous. To address the problem, the key idea is to **link the horizontal *sedges***. However, determining the position of horizontal *sedges* and linking them are not straightforward.

We take a tricky approach, i.e., post-processing of *ND*, which fills the area between *sedges* by several white and black edges to address the problem, as shown in Figure 3.6. In this way, we can link the horizontal *sedges* without determining the precise position of them. To make the filling processing automatically, we can iteratively move the white edges forward until one longer edge is found or no more edges can be found. For example, assume the width of white/black edge is 5px, then we move each white edge forward 1px per iteration, and 5 iterations in total to determine whether there is another horizontal *sedge*. Note that the determine process is handled by rerunning processing algorithm. The post processing is terminated when there is no new *sedge* detected.

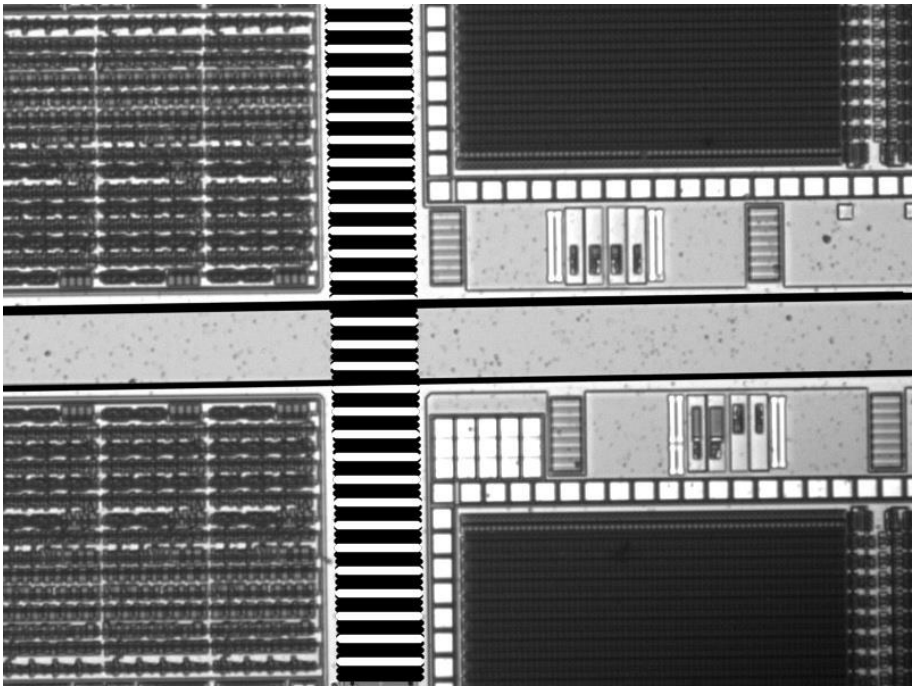
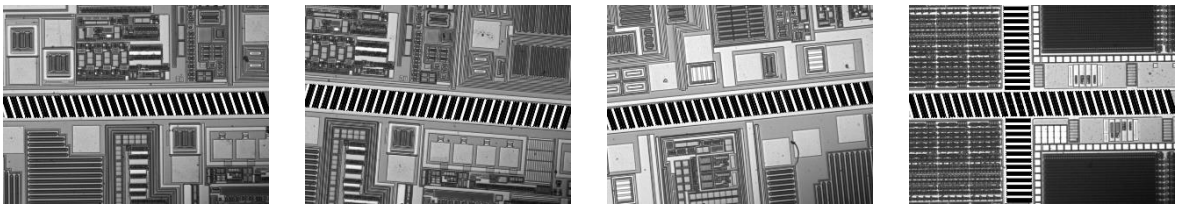


Figure 3.6. Filling the area between vertical *sedges* for detecting the horizontal *sedges*. The new *sedges* are highlighted by horizontal black lines.

Figure 3.7 illustrates the final results of wafers after post-processing. The algorithm can stop automatically when no new *sedges* can be found.



(a) Post Wafer 1 (a) Post Wafer 2 (a) Post Wafer 3 (a) Post Wafer 4

Figure 3.7. Post-processed wafers by *ND*. *ND* fills the area between *sedges*, and terminates post-processing automatically when there is no new *sedge* can be found.

Put It All Together. We summarize *ND*'s three-step-processing algorithm in List 1.

List 1. Pseudo code for three-step-processing algorithm in *ND*

```
Input: wafer imaging
Output: wafers

// Preprocessing and Processing the wafer imaging

img ← ND.load(wafer imaging)
pre_res ← ND.preprocess(img)
res, wafer ← ND.process(pre_res)

// Prepare Post-processing

wafers ← []
wafers.append(wafer)
terminated ← False
tmp_img ← img

// Automatically run Post-processing.

while not terminated:
    // Run Processing in Post-processing to determine if new sedges can be found.
    // If found, terminated is assigned to false, and vice versa.
    post_res, terminated ← ND.postprocess(tmp_img, wafer)
    if terminated then:
        for wafer in wafers
            ND.draw(img, wafer)
    else:
        res, wafer ← ND.process(tmp_img)
        wafers.append(wafer)

return wafers
```

4. Design and Implementation of *Line Through Detector*

Since *ND* heavily relies on the continuousness of detected edges, it suffers robustness problem if the sedges of wafer imaging are corrupted (see Section 5.3) or the edge detection algorithm cannot return enough continuous results (due to mis-chosen parameters). To address the problem of *ND*, we further propose a **border point-based** approach, *Line Through Detector (LTD)*. Based on our observations in Section 2, *LTD* assumes that the sawing path of the wafer runs through the entire image. Specifically, *LTD* first detects all possible **through-lines (the line goes through the whole image)** in the image, then removes some meaningless through-lines. Next, the through-lines are paired to form a pair of lines, each pair forming a sawing path. And then, by traversing the combination of sawing paths and checking the rationality of this combination, the appropriate combination of sawing paths is finally detected.

By analyzing the provided sample images (see Figure 2.1), it can be found that there are no edges within a certain area outside the sawing path (excluding noise points), which we refer to as the "*clean area*". So if the candidate sawing path composed of two through-lines is the valid sawing path, there must be a *clean area* on the outside of this sawing path, otherwise, this sawing path is invalid, as shown in Figure 4.1.

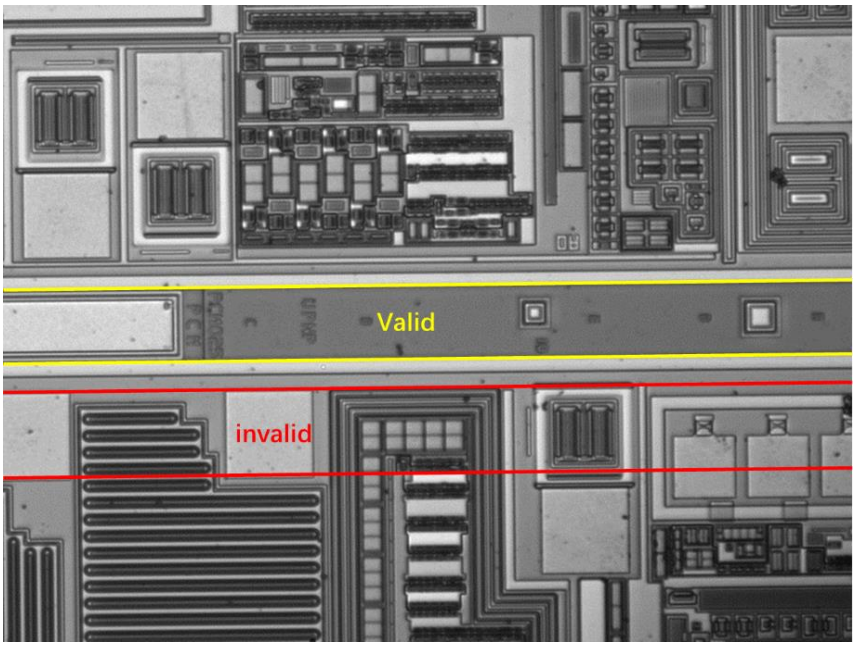


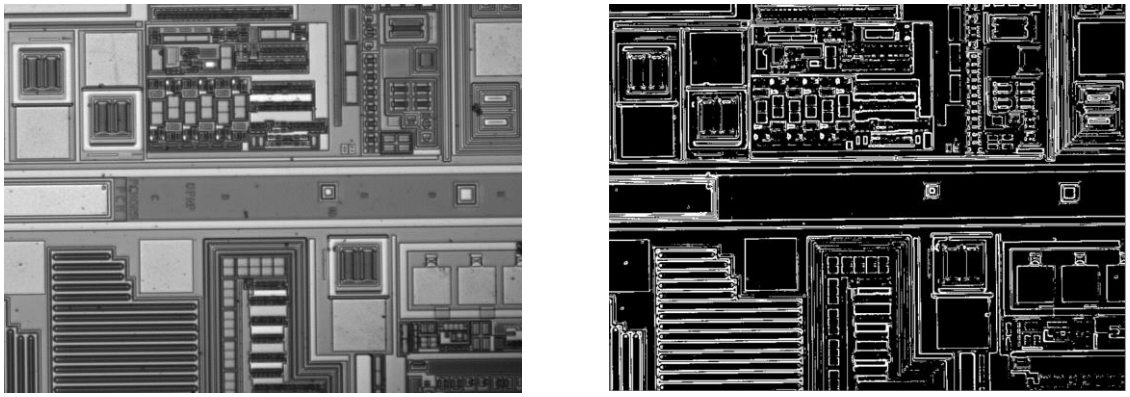
Figure 4.1. Valid and invalid sawing paths.

Therefore, the simplest implementation of this solution is to first find all possible through-lines from the graph, and then test the legality of each sawing path according to the above rules (the sawing path is composed of two through-lines) until the most suitable sawing path is found.

However, due to the presence of many through-lines on the wafer, there may be many invalid through-lines in the graph. Detecting the sawing path composed of these invalid through-lines may lead to a decrease in the speed of the algorithm. Therefore, the complete idea of this algorithm is to: (1) find all possible through-lines; (2) Eliminate invalid through-lines; (3) Pairwise matching of the through-line to form a sawing path; (4) Due to the possibility of multiple sawing paths in the figure, it is necessary to combine the sawing paths obtained in (3) and then check the legality of this combination (combined with a "*clean area*"); (5) Obtain the

most suitable combination of sawing paths.

Step 1: Detect edges in the image. *LTD* uses *Sobel* operator to calculate gradient map and uses threshold to extract image edges. The original image and edge image of the first sample are shown in Figure 4.2.



(a) Original image of Wafer 1 **(b)** Edge image of Wafer 1

Figure 4.2. Detect edge image for Wafer 1.

Step 2: Detect possible *clean areas*. The sawing path has a certain width and runs through the entire image. Therefore, the sawing path must pass through at least two boundaries in the upper, lower, left, and right boundaries, and the *clean area* near the sawing path also has this characteristic. Therefore, if an area is a *clean area*, it must pass through at least two image boundaries (as shown in Figure 4.3).

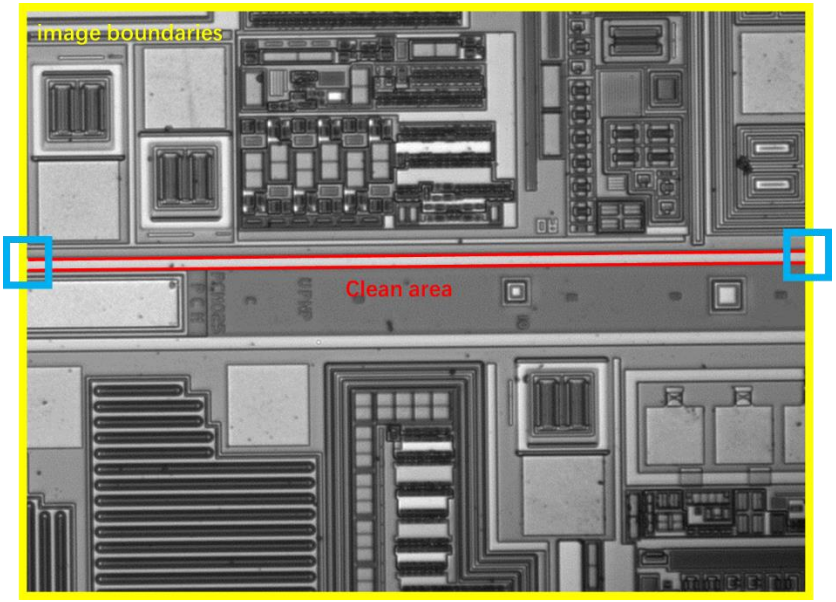
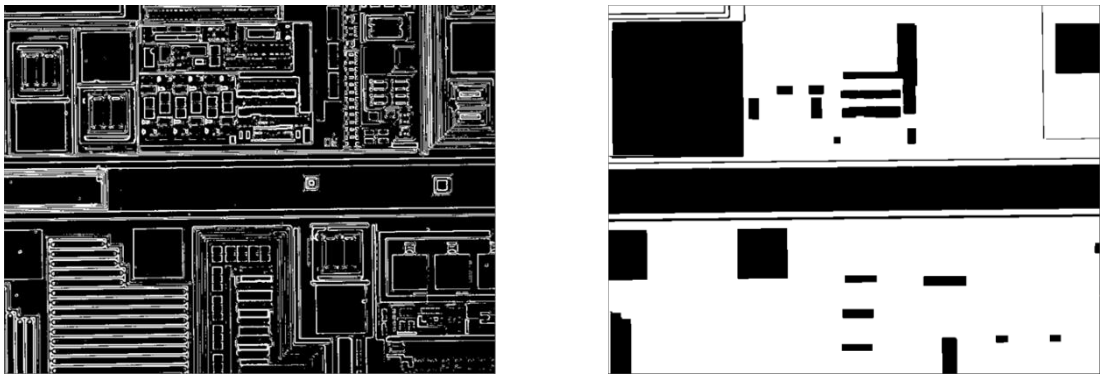


Figure 4.3. *Clean area* passing through two boundaries, i.e., left and right boundaries.

Therefore, using the edges obtained in **Step 1** as separators, extract all connected domains in the image, and then test whether these connected domains have a "degree" greater than two (passing through at least two different image boundaries). Only connected domains with two or

more degrees can be *clean areas*, and only near these connected domains can there be sawing paths. By removing all connected domains with a degree less than 2, a more simplified image can be obtained. At the same time, for each connected domain, multiple dilation operations are used, followed by the same number of erosion to reduce the number of empty holes in these connected domains (In Figure 4.4(b), the white area indicates areas that may be *clean areas*).

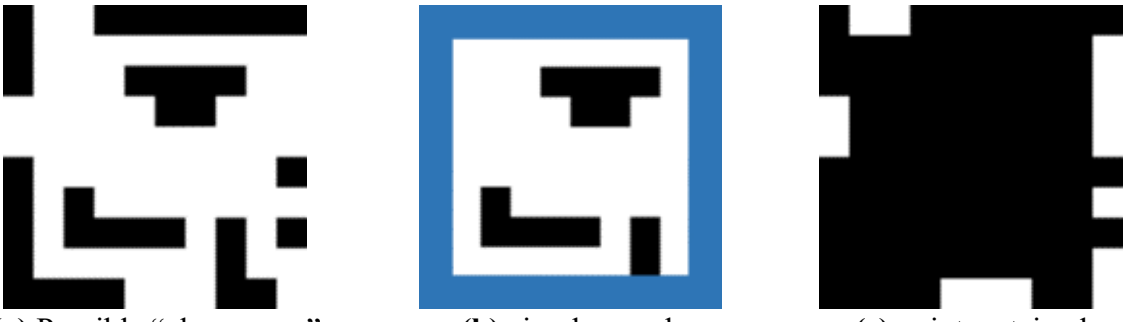


(a) Edge image of Wafer 1 **(b)** Possible *clean areas* of Wafer 1

Figure 4.4. Detect possible *clean areas* for Wafer 1.

Step 3: Obtain the candidate set of through-lines. Based on the results of **Step 2**, we can use the Hough transform to obtain the possible lines in the image, and then extend these lines to generate a set of through-lines. However, *LTD* takes another approach to extract through-lines: find points located at the top, bottom, left, and right boundaries of the image, and connect these points to form a set of through-lines.

Firstly, for the result in **Step 2**, use a circular mask with a thickness of 1 to obtain the points located at the boundary of the image. The following Figure 4.5 is a simple example. The white points in the figure are the points that may form a *clean area*, and the blue circle is the circular mask. Only white points under the circular mask can be retained.



(a) Possible "clean areas" **(b)** circular mask **(c)** points retained

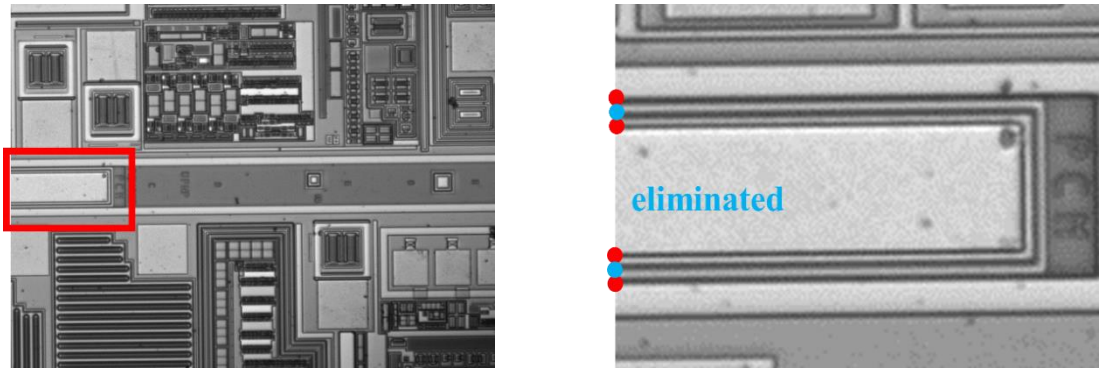
Figure 4.5. Use ring mask to obtain points on the edge.

White dots in the Figure 4.5(c) can form different connected domains. As the thickness of these connected domains is 1, the area of the connected domain is approximately equal to the "length" of the connected domain. *LTD* uses dilation and erosion operations to connect connected domains with gaps less than a certain threshold, and handle connected domains with a "length" greater than a certain threshold, retaining only endpoints, as shown in Figure 4.6.

(a) Connected domains **(b)** Long connected domains **(c)** Extract endpoints

Figure 4.6. Extracting endpoints of excessively long connected domains.

Due to the absence of interference lines in the *clean area*, when there are a large number of horizontal lines in the sawing path, merging points with smaller gaps into a connected domain and then extracting the endpoints of this connected domain can prevent the extraction of some useless points, such as reducing two points in the Figure 4.7(a) , and ensuring that one of the points is on the through-line (blue points belong to boundary point, but they are eliminated due to the merge operation).



(a) Areas with parallel lines **(b)** Extract only red points

Figure 4.7. Reduce unnecessary points through merge operations.

Extract the connected domains from the obtained result image (see Figure 4.6(c)), and the center point of each connected domain is the point used to form the through-line. When selecting two points that form a through-line, if the line formed by these two points is only at the boundary of the image and does not pass through the "interior" of the image, then the through-line formed by these two points will not be considered, as shown in Figure 4.8.

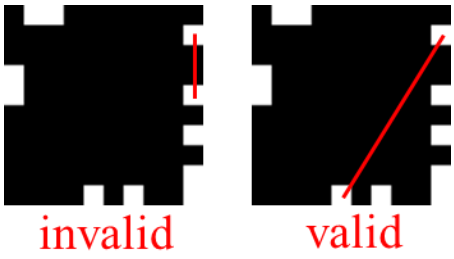
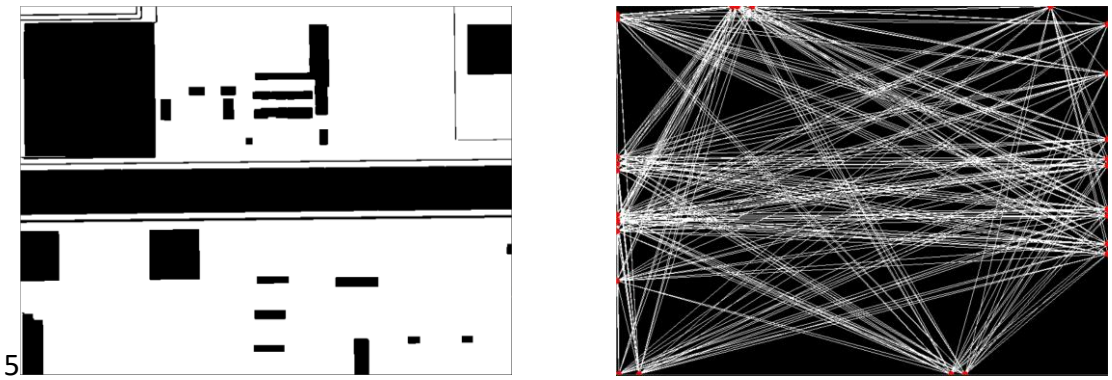


Figure 4.8. Valid and invalid through-lines.

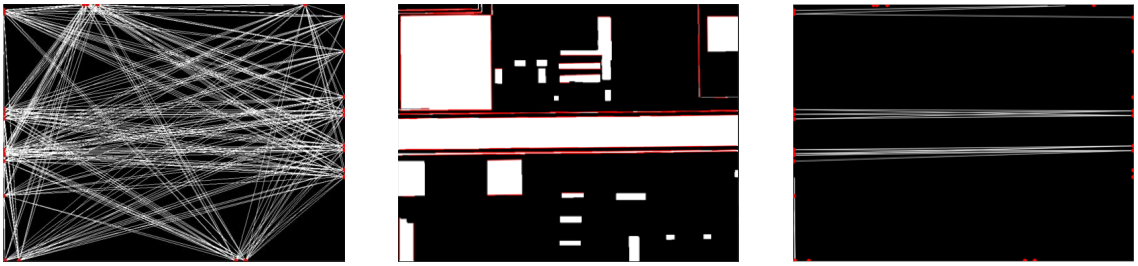
The candidate set for the first sample's through-line is shown in Figure 4.9.



(a) Possible "clean areas" of Wafer 1 **(b)** Candidate set

Figure 4.9. Candidate set for sample 1's through-line.

Step 4: Filter out invalid through-lines. In order to reduce unnecessary overhead, it is necessary to eliminate potentially meaningless through-lines in the candidate set. If a through-line is meaningful, it must intersect with a segment that already exists in the image, that is, the through-line must pass through a segment that actually exists in the image. So *LTD* first uses the *Hough Transform* to extract possible line segments in the image, and then compares all through-lines and these line segments in pairs to check their degree of intersection. If a through-line does not overlap with any real line segments, then the through-line is considered invalid. In the following figure 4.10, the red line segment is the actual line segment in the image.



(a) Candidate set **(b)** Line segments present in the graph **(c)** Filtered results

Figure 4.10. Filter out through-lines that do not pass through any line segments.

Step 5: Form through-line pairs (sawing path). The sawing path is composed of two through-lines. *LTD* will pair the through-lines obtained from **Step 4** (see Figure 4.10(c)) in pairs to form a sawing path. If the distance between two through-lines is too far or too close, the sawing path they form will not be considered, that is, setting a threshold to indicate the minimum and maximum width of the sawing path. At the same time, when forming a sawing path, *LTD* will also consider whether the two through-lines are approximately parallel. If the two through-lines are not parallel to each other, the sawing path they form will not be considered.

Figure 4.11 shows a pair of through-lines and the sawing path they formed:

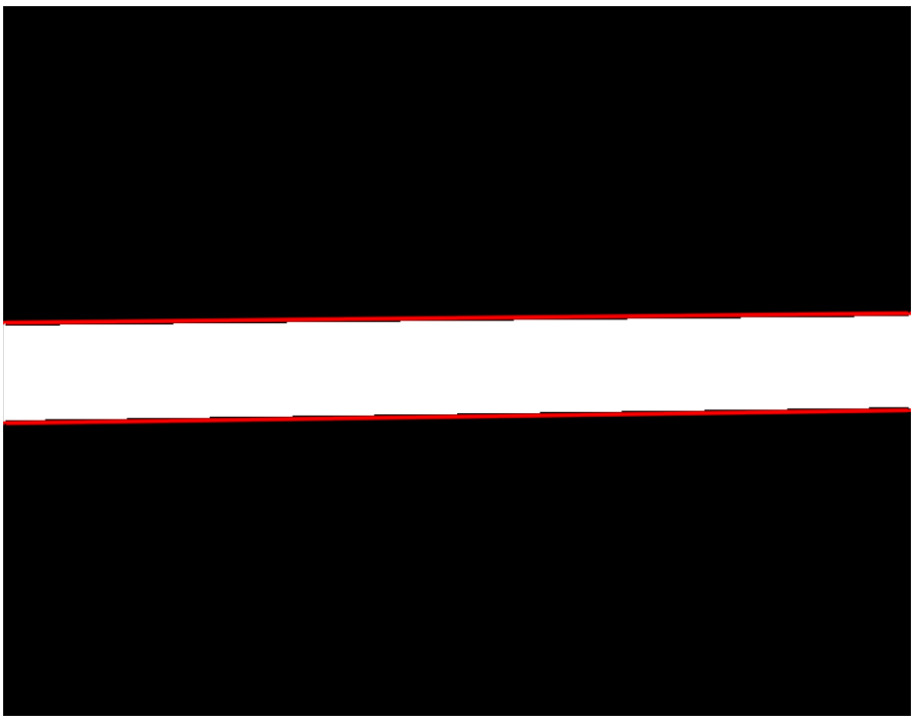


Figure 4.11. A pair of through-lines and the sawing path they formed.

The reason for setting the sawing path width threshold is to deal with the situations where the wafer after edge detection has too many parallel lines, as shown in Figure 4.12.

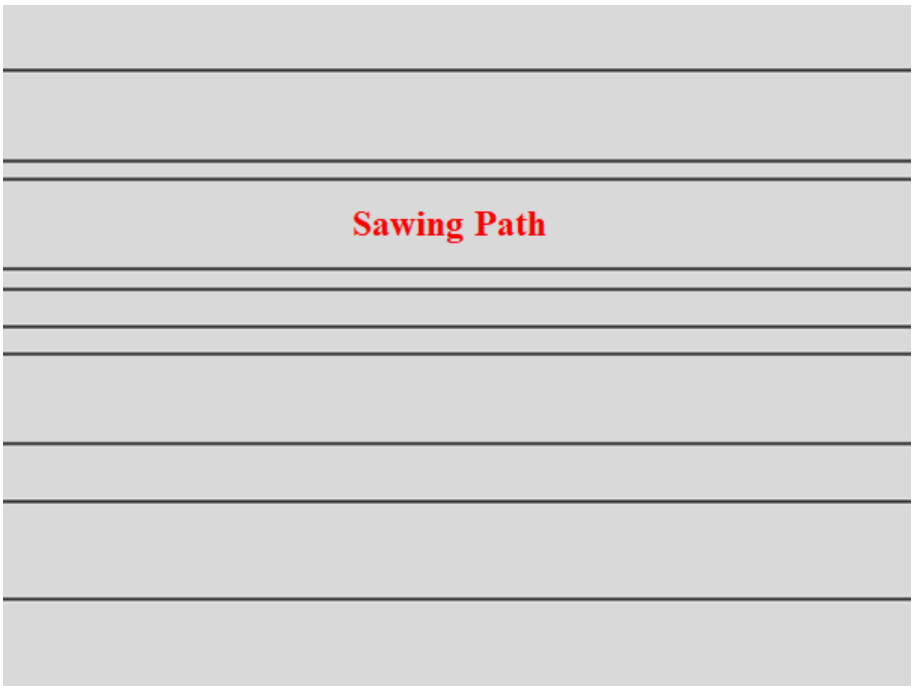


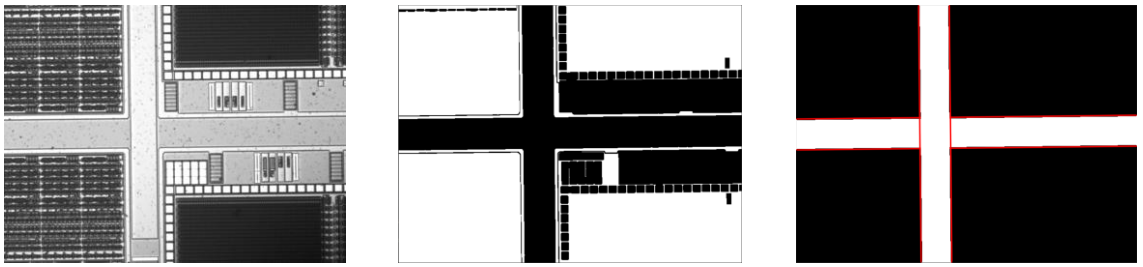
Figure 4.12. Exceptional case.

Step 6: Build a combination and check its rationality. For the Wafer 4, there are two sawing paths, they form a pair of sawing paths. *LTD* will detect the compatibility of each pair

of sawing paths. The process is: firstly, detect if there is any overlap between the two sawing paths, and if so, the two sawing paths are incompatible. Afterwards, the angles of the two cutting channels are detected. If the difference in angles between the two sawing paths exceeds the threshold, these two sawing paths are compatible; Otherwise, check the distance between these two sawing paths. If the distance is less than the threshold, then these two sawing paths are incompatible. Note that the distance threshold is used to deal with situations where there are some parallel sawing paths in the image.

Next, we define *Combination* as a combination of sawing paths. A *Combination* contains one or more sawing paths that are compatible with each other. Thus, based on all the sawing paths obtained in **Step 5**, we can arrange and combine them, and perform compatibility testing to obtain all possible *Combinations*. And then, each *Combination* can be drawn to obtain the *sawing region* (white points in the Figure 4.13(c)). The number of white points is the area of the *sawing region* of the *Combination*.

Finally, *LTD* will check the rationality of the *Combination*. According to the previous description, there must be a clean area around the cutting region, so the algorithm dilates the drawing results of the *Combination*. If there are no too "long" black area in the area covered by the expanded points, the *Combination* is reasonable. The area of the black area during the detection process serves as the *Error* of this *Combination*.



(a) Original image **(b)** Possible "*clean area*" **(c)** Sawing region

Figure 4.13. Use possible *clean areas* and dilation operations to check the rationality of the Combination.

So far, *LTD* has obtained a series of reasonable *Combinations*, and then *LTD* will sorts these *Combinations* by combining the *Error/NumOfSawingPath* (*EPSP*) of each *Combination* to obtain a sorted list of *Combinations*. *LTD* selects the *Combination* with the smallest *EPSP* as a reference to determine the number of sawing paths in the wafer imaging. For example, if the determined *NumOfSawingPath* is 2, then *LTD* will select the *Combination* with 2 sawing paths and the smallest area of *sawing region* from all *Combinations* as the final output.

The reason for choosing the *Combination* with the smallest area of *sawing region* is to guarantee the selection of the correct sawing path (i.e., selecting the right image instead of left image in the Figure 4.14).

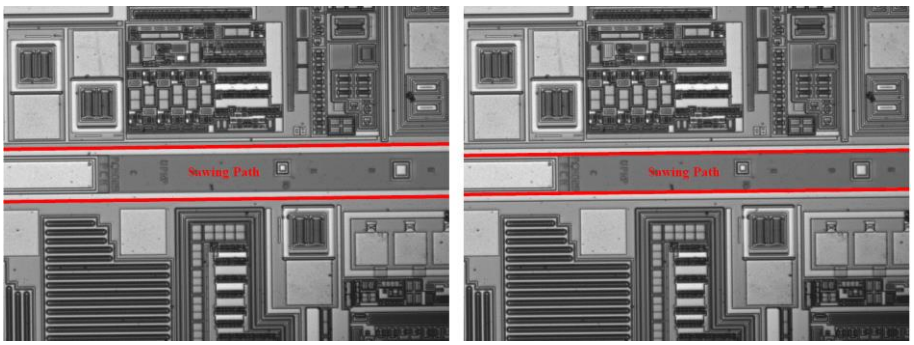


Figure 4.14. Choose a Combination with the smallest area (right image).

Additional. There are still some anti-noise tricks during the implementation of *LTD*, and the relevant process can be seen in the source code.

Put It All Together. We summarize *LTD*'s algorithm in List 2.

List 2. Pseudo code for algorithm in *LTD*

```
Input: waferImage
Output: Combination

img ← imread(waferImage)
oriEdgeImage ← LTD.DetectEdge(img)
cuttingImage ←LTD.DetectCleanArea(oriEdgeImage)
edgeImage ← 255 – cuttingImage      // Pixel value inversion
pointImage ← LTD.DetectPoint(edgeImage)

// Props is the set of edge point
props, lineList ←LTD.DetectLineThrough(pointImage)
lineList ← LTD. FilterOutLine(edgeImage, props, lineList)

combinationList ← LTD.CheckCuttingLine(oriEdgeImage, props, lineGroup)
combination←LTD.OptCombination(image, props, combinationList)

return combination
```

5. Experimental Results and Analysis

To measure the detection results objectively, we hand-craft golden *sedges* for the given wafers' imaging for comparison. Specifically, we manually mark *sedges* on the original wafer imaging by using a four-element tuple (x_1, y_1, x_2, y_2), where (x_1, y_1) and (x_2, y_2) denote the end points of one *sedge*. The golden *sedges* are listed in Table 5.1.

Table 5.1. Golden rules for comparison

Wafers	Direction	Sedge1	Sedge2	Sawing Angle	Sawing Center
Wafer 1	horizontal	(0, 269, 799, 260)	(0, 337, 799, 328)	0.65	(399.5, 298.5)
Wafer 2	horizontal	(0, 239, 799, 300)	(0, 306, 799, 368)	4.40	(399.5, 303.25)
Wafer 3	horizontal	(0, 307, 799, 229)	(0, 376, 799, 297)	5.61	(399.5, 302.25)
Wafer 4	horizontal	(0, 276, 799, 258)	(0, 335, 799, 326)	89.37	(326.25, 299.5)
	vertical	(289, 0, 295, 599)	(357, 0, 364, 599)	0.97	(399.5, 298.75)

Metrics. We measure the error of angle and center of the detected sawing line by using Euclidean distance, as shown below. In the equation, sAngle denotes the **absolute** angle of the golden sawing line while $(\mathrm{sCenterX}, \mathrm{sCenterY})$ denotes its center point.

$$\mathrm{Error} = \sqrt{(\mathrm{angle} - \mathrm{sAngle})^2 + (\mathrm{centerX} - \mathrm{sCenterX})^2 + (\mathrm{centerY} - \mathrm{sCenterY})^2} \quad (1)$$

Testbed. We run the experiments on a Windows 10 machine with an Intel i7-11800H CPU clocked at 2.30GHz and 16GB memory.

5.1 Detection Results of *ND*

We run *ND* on the given four real-world wafers' imaging and the preprocessed, post-processed, and final results are present in Figure 5.1. We compare the detected sawing lines in final results with our golden rules, the results are presented in Table 5.2. To further study the effects of fine-grained adjustment (i.e., extend and rotate the detected *sedges*), we compare the sawing lines without adjustment with the golden rules in Table 5.3. The results indicate that the adjustment in Processing step can reduce the error and improves the precision in most cases.

Table 5.2. Comparison between the results of *ND* and golden rules, **with adjustment** ("+" indicates the improvement over Table 5.3, i.e., *ND* without adjustment).

Wafers	Direction	Angle	Center	sAngle	sCenter	Error
Wafer 1	horizontal	1.00	(399.5, 302.0)	0.65	(399.5, 298.5)	3.52
Wafer 2	horizontal	4.94	(399.5, 306.5)	4.40	(399.5, 303.25)	3.29
Wafer 3	horizontal	6.00	(399.5, 304.0)	5.61	(399.5, 302.25)	1.79 (+)
Wafer 4	horizontal	88.95	(326.5, 299.0)	89.37	(326.25, 299.5)	0.70 (+)
	vertical	1.00	(399.5, 298.0)	0.97	(399.5, 298.75)	0.75 (+)

However, the strategy also might change the sawing line angle significantly. In Wafer 2, the golden angle is 4.4°, while adjustment rotates the detected sawing line from more precise 4.51° to 4.94°. This could be further optimized by applying adaptive adjustment, e.g., if the degree of the rotation exceeds a pre-defined threshold, then we give up this adjustment. We leave this

as our future work.

Table 5.3. Comparison between the results of *ND* and golden rules, **without adjustment** ("-" indicates the sawing line cannot be extracted).

Wafers	Direction	Angle	Center	sAngle	sCenter	Error
Wafer 1	horizontal	1.00	(399.5, 302.0)	0.65	(399.5, 298.5)	3.52
Wafer 2	horizontal	4.51	(400.0, 306.5)	4.40	(399.5, 303.25)	3.29
Wafer 3	horizontal	6.00	(399.5, 305.0)	5.61	(399.5, 302.25)	2.78
Wafer 4	horizontal	88.97	(327.0, 278.0)	89.37	(326.25, 299.5)	21.5
	vertical	-	-	0.97	(399.5, 298.75)	MAX

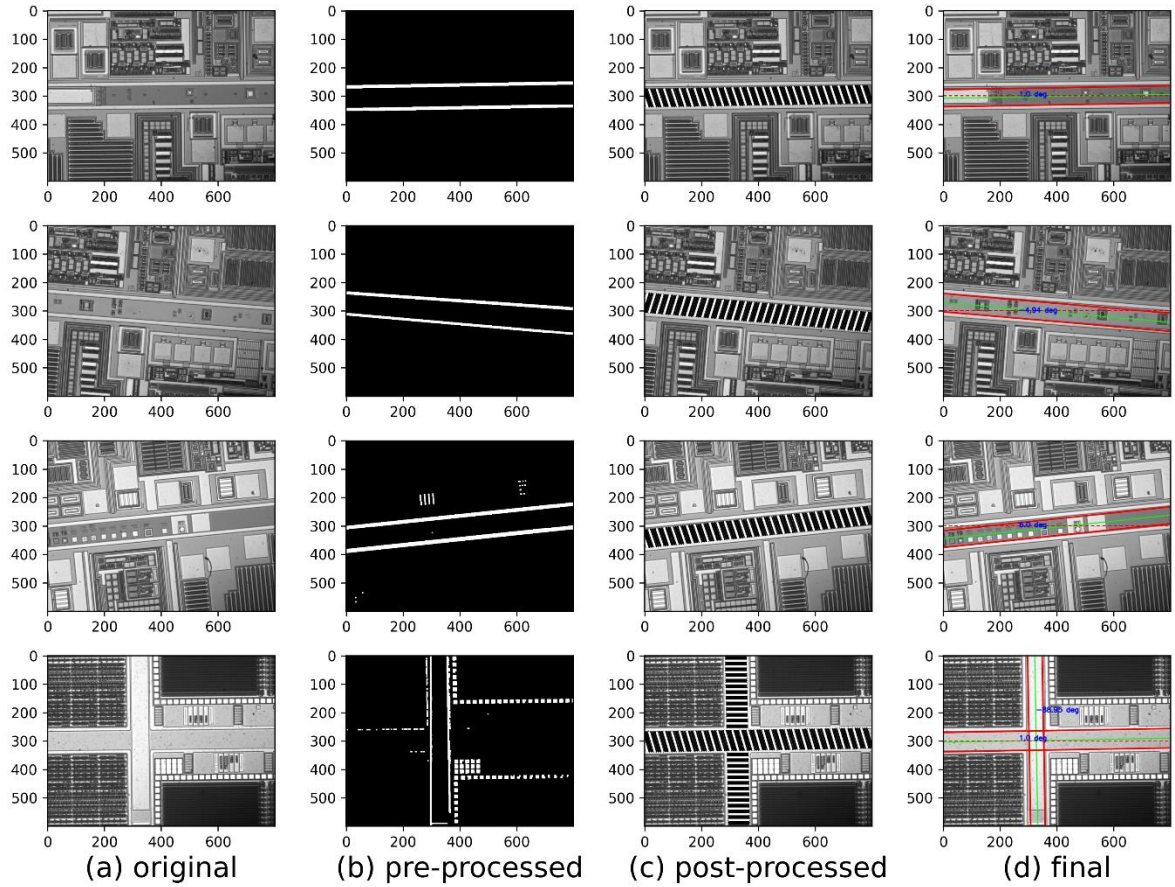
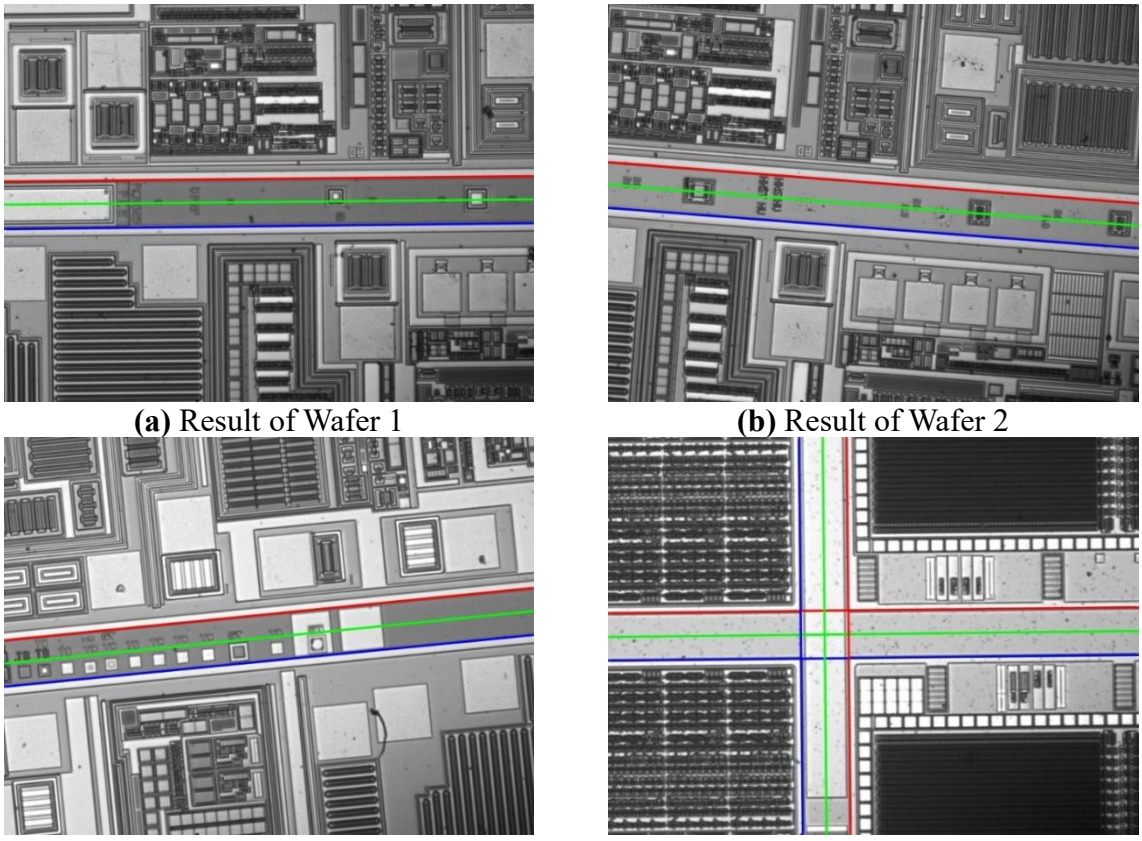


Figure 5.1. Detection results by *ND*.

Discussion. Note that the error of *ND* comes from several operations, such as closing operation in Preprocessing step (for linking line segments), the assumption that *sedges* are closet in the candidates (for finding the *sedges*), and the fine-grained adjustment operation (for ensuring the *sedges* go through the wafer and nearly parallel) as mentioned above. Besides, *ND* also relies on the accuracy of edge detection and segments algorithm. Although *ND* is not robust, it is simple to implement and performs efficiently since it does not involve heavy computations (caused by complex verifications).

5.2 Detection Results of *LTD*

Results. *LTD* will visualize the obtained sawing lines and store the visualized results as images (see Figure 5.2). In the figure, the blue line represents the lower edge of the sawing line, the red line represents the upper edge of the sawing line, and the green line represents the through-line. For samples with multiple sawing lines, the order of sawing lines is not clearly defined.



(a) Result of Wafer 1 **(b)** Result of Wafer 2

(c) Result of Wafer 3 **(d)** Result of Wafer 4

Figure 5.2. Results of *LTD*.

We run *LTD* on the given four real-world wafers' imaging and the Mid-Output are present in Figure 5.3. We compare the detected sawing lines in final results with our golden rules, the results are presented in Table 5.4. According to the data in the table, the accuracy of *LTD* has been relatively improved, with most errors less than 0.5.

Table 5.4. Comparison between the results of *LTD* and golden rules ("+" indicates the improvement over Table 5.2, i.e., *ND*).

Wafers	Direction	Angle	Center	sAngle	sCenter	Error
Wafer 1	horizontal	0.57	(399.5, 298.75)	0.65	(399.5, 298.5)	0.26 (+)
Wafer 2	horizontal	4.40	(399.5, 303.63)	4.40	(399.5, 303.25)	0.375 (+)
Wafer 3	horizontal	5.61	(399.5, 302.13)	5.61	(399.5, 302.25)	0.125 (+)
Wafer 4	horizontal	89.43	(326.0, 299.5)	89.37	(326.25, 299.5)	0.25 (+)
	vertical	0.65	(399.5, 296.5)	0.97	(399.5, 298.75)	2.27

Mid-Output. Mid-Output of *LTD* is shown as Figure 5.3. The figure shows the intermediate processing results of four samples. In the step of detect possible *clean areas*, the number of connected domains in all four samples has been reduced, and the empty holes inside the connected domains have also been reduced, which decreases the information that *LTD* needs to pay attention to and can accelerate the algorithm's operation. In this step, the extraction of connected domains depends on the results of edge detection. Therefore, if the extracted edges fail to divide areas that originally did not belong to the same connected domain, this optimization step will actually have adverse effects.

By using possible *clean areas*, combined with edge detection and circular masking, edge points located on the image boundaries can be extracted. Afterwards, *LTD* combines the center coordinates of the connected domain where these edge points are located to generate a candidate set of through-lines. From the figure, it can be seen that the candidate set for the through-line of Sample 3 is relatively large. After analysis, we think that this is due to the complexity of Sample 3 at the image boundaries. Therefore, the running time of this algorithm also depends on the complexity of the image at the boundaries. If *LTD* use the *Hough Transform* to extract line segments from the image and generate a candidate set of through-lines based on these line segments, the complexity of the algorithm will depend more on the internal complexity of the image, and the subsequent process will also need to connect multiple line segments into a through-line, which will also bring some overhead. At the same time, generating a through-line based on edge points can ensure that this line is "straight" and by extending it, the error between it and the actual through-line can be smaller. Overall, the two schemes for extracting candidate sets of through-lines each have their own advantages and disadvantages.

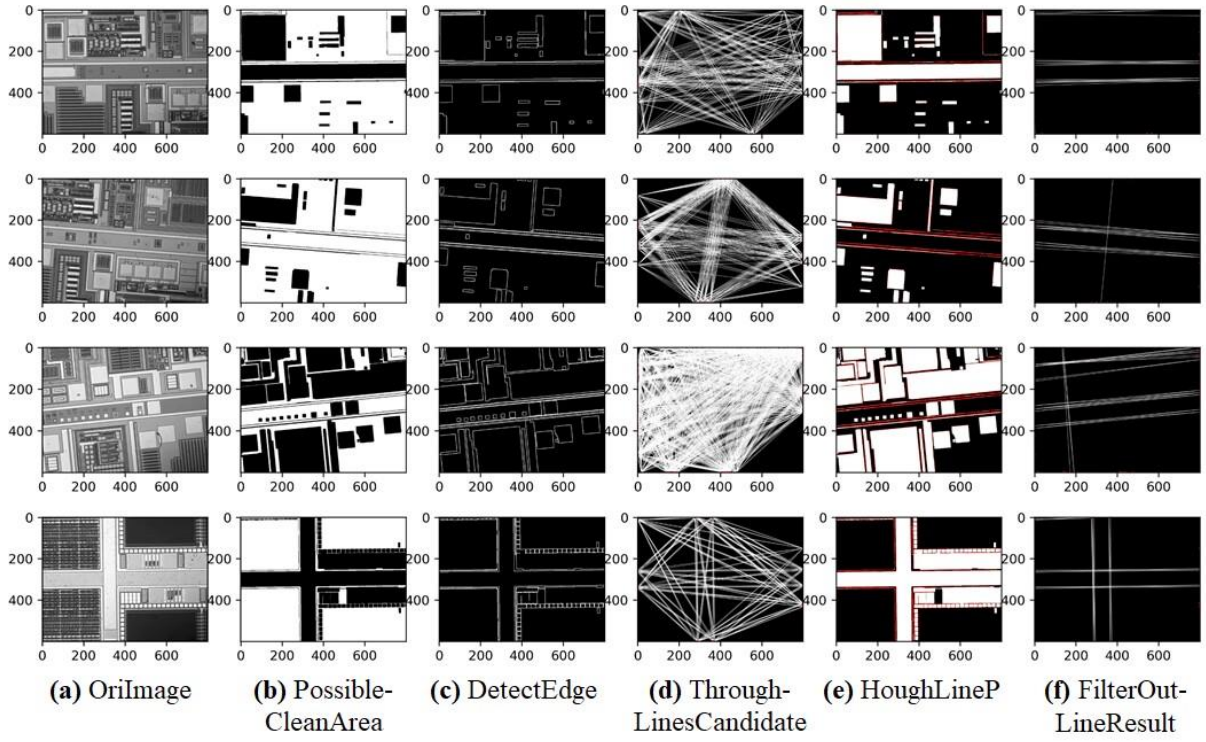
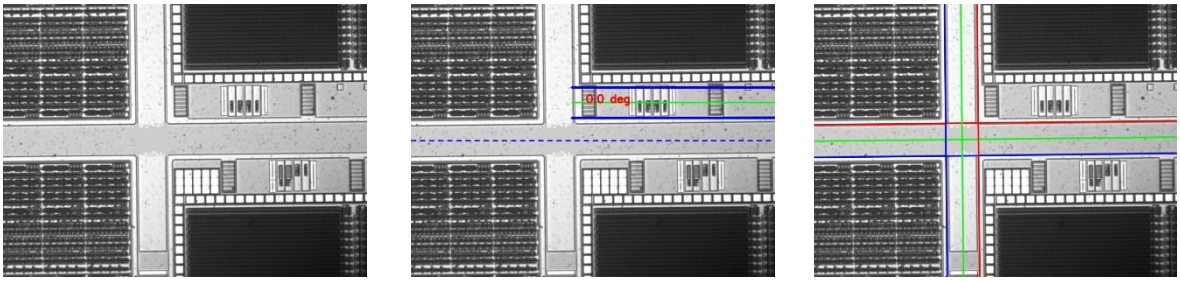


Figure 5.3. Mid-Output of *LTD*.

Discussion. *LTD* only relies on edge points on the image boundary to generate through-lines, thus reducing the impact of internal complexity in the image. *LTD*'s detection of the rationality of *Combinations* only relies on the *clean area*, so as long as the *clean area* is "clean", the through-lines on both sides of this area can be extracted, and then the corresponding Combinations can be extracted. However, the complexity of *LTD* is high. When the images are complex, a large number of candidates through-lines may be generated and *LTD* will require more time to detect the appropriate *Combination*. Under certain special circumstances, there may still be detection abnormalities.

5.3 Robustness Evaluation

In this section, we show that *LTD* can be more robust compared to *ND* since *LTD* does not depend on the length of the detected edges. We reuse Wafer 4 as an example to show this. We manually blur the vertical *sedges* of Wafer 4, as shown in Figure 5.4(a). The final results of *ND* and *LTD* are presented in Figures 5.4(b) and 5.4(c). In the figure, we find that *ND* detects the wrong *sedges* since it only extracts the longest line as *sedges*; while for *LTD*, it successfully detects both horizontal and vertical sawing lines thanks to its border point-based approach and robust *clean area*-based verification algorithm.



(a) Corrupted wafer (b) Detected by *ND* (c) Detected by *LTD*

Figure 5.4. Robustness evaluation.

5.4 Performance Evaluation

We study the performance of *ND* and *LTD* in this section. Specifically, we use time command to measure the real time of processing Wafer 1-4 and the corrupted Wafer 4. *ND* spends around 3.459s while *LTD* spends more than 30s since *LTD* performs complex verifications to ensure the correction of detected *sedges*. We plan to optimize the performance of *LTD* by replacing several loops with matrix batch calculations and leveraging GPU ability. Note that we have replaced the calculation of the angle between straight lines with matrix calculation, resulting in a certain improvement in speed.

6. Summary

In this project, we implemented two detectors, *ND* and *LTD*, to complete the detection of wafer sawing lines, each with its own characteristics and the ability to handle different situations. Specifically, *ND* extracts the longest edges from the wafer to gain the sawing lines while *LTD* leverages a boarder point-based method to ensure the continuousness of the detected edges and uses a *clean area*-based approach to carefully verify the edges of sawing lines. Extensive evaluation results demonstrate that *LTD* can achieve the most precise results and retain good robustness while *ND* can perform more efficient with a slightly higher error.

During the implementation, we use several digital image processing techniques to extract wafer sawing line from the image, such as gradient operation (e.g., Sobel Operator), edge detection, image morphology processing (e.g., Dilation, Erosion, and Closing operations), image connected domain extraction, and line segments extraction (e.g., Hough Transform), etc. Meanwhile, mathematical knowledge is used to calculate the distance and angle of straight lines.

This project has taught us knowledge related to wafer cutting, while also allowing us to put the knowledge learned in class into practice. The difficulties encountered during the completion of this project mainly include: (1) excessive computational overhead of the algorithm; (2) Unable to effectively utilize matrix operations to accelerate algorithms; (3) There are interference points in the image, resulting in poor algorithm performance. Our future work will focus on addressing the above issues.

7. Contributions & Acknowledgement

Table 7.1 summarizes the contributions of this project. The overall Line of Codes (LOC) of this project is approximately 1850. We thank our lecturer, Guangming Lu, for insightfully providing us an opportunity to solve the real-world problems (i.e., wafer sawing line detection) by using DIP techniques.

Table 7.1. Contributions of this project.

Name&ID	Membership	Work Description	LOC (Python)	Contrib.
Yanqi Pan (潘延麒) 22S051053	Captain	1. Deciding & writing the primary Logical Flow of the Report. 2. Design and Implement Naïve Detector. 3. Experiments on Naïve Detector. 4. Error Estimation.	≈550	48%
Junjie Liu (刘俊杰) 22S151187	Member	1. Design and Implement Line Through Detector. 2. Writing Implementation part of Line Through Detector. 3. Experiments on Naïve Detector and Line Through Detector.	≈1300	52%